
ESMValTool User's and Developer's Guide

Release 2.7.1.dev0+g97e395e57.d20221028

ESMValTool Development Team

Oct 28, 2022

ESMVALTOOL

I	Introduction	1
1	About	3
2	Support	5
3	License	7
II	What ESMValTool can do for you	9
4	Data finding	13
5	Data selection	15
6	Data fixing	17
7	Variable derivation	19
8	Run the preprocessor	21
9	Run the diagnostics	23
III	Getting started	25
10	Installation	27
11	Configuration	39
12	Running	41
13	Output	43
IV	Gallery	47
V	Recipes	65
14	Atmosphere	67
15	Climate metrics	173

16	Future projections	185
17	IPCC	267
18	Land	305
19	Ocean	343
20	Other	395
VI	Obtaining input data	435
21	Models	439
22	Observations	441
23	Datasets in native format	447
VII	Making a recipe or diagnostic	449
24	Introduction	451
25	Recipe	453
26	Diagnostic	455
27	Writing a CMORizer script for an additional dataset	457
VIII	Contributing to the community	463
28	Contributing code and documentation	467
29	Making a new diagnostic or recipe	475
30	Broken recipe policy	485
31	Making a new dataset	487
32	Support for multiple versions of a dataset	491
33	Review of pull requests	493
34	Maintaining a recipe	497
35	Upgrading a namelist (recipe) or diagnostic to ESMValTool v2	499
36	GitHub Workflow	505
37	Moving work from the private to the public repository	511
38	Release schedule and procedure for ESMValCore and ESMValTool	513

IX	Utilities	523
39	Pre-commit	527
40	nclcodestyle	529
41	Colormap samples	531
42	Running multiple recipes	533
43	Comparing recipe runs	535
44	Testing recipe settings	537
45	draft_release_notes.py	539
46	Converting Version 1 Namelists to Version 2 Recipes	541
47	Recipe filler	543
48	Extracting a list of input files from the provenance	545
X	ESMValTool Code API Documentation	547
49	Shared Diagnostic Code	551
50	Diagnostic Scripts	571
XI	Frequently Asked Questions	783
51	Is there a mailing list?	785
52	What is YAML?	787
53	Re-running diagnostics	789
54	Enter interactive mode with iPython	791
55	Use multiple config-user.yml files	793
56	Create a symbolic link to the latest output directory	795
57	Can ESMValTool plot arbitrary model output?	797
XII	Changelog	799
58	v2.7.0	801
59	v2.6.0	805
60	v2.5.0	809
61	v2.4.0	813
62	v2.3.0	817

63	v2.2.0	821
64	v2.1.1	825
65	v2.1.0	827
66	v2.0.0	829
67	v2.0.0b4	833
 XIII Indices and tables		835
Python Module Index		839
Index		841

Part I

Introduction

ABOUT

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

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

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

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

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.

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.

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](#).

2.3 Core development team

- Deutsches Zentrum für Luft- und Raumfahrt (DLR), Institut für Physik der Atmosphäre, Germany (Co-PI)
 - ESMValTool Core Co-PI and Developer: contact for requests to use the ESMValTool and for collaboration with the development team, access to the PRIVATE GitHub repository.
- Met Office, United Kingdom (Co-PI)
- Alfred Wegener institute (AWI) Bremerhaven, Germany
- Barcelona Supercomputing Center (BSC), Spain
- Netherlands eScience Center (NLLeSC), The Netherlands
- Ludwig Maximilian University of Munich, Germany

- Plymouth Marine Laboratory (PML), United Kingdom
- Swedish Meteorological and Hydrological Institute (SMHI), Sweden
- University of Bremen, Germany
- University of Reading, United Kingdom

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.

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](https://doi.org/10.5281/zenodo.3401363)) and ESMValCore ([doi:10.5281/zenodo.3387139](https://doi.org/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.

Part II

What ESMValTool can do for you

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

- establish the correct input and output parameters and the structured workflow;
- acquire the correct data;
- execute the workflow; and
- output the desired collective data and media.

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

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

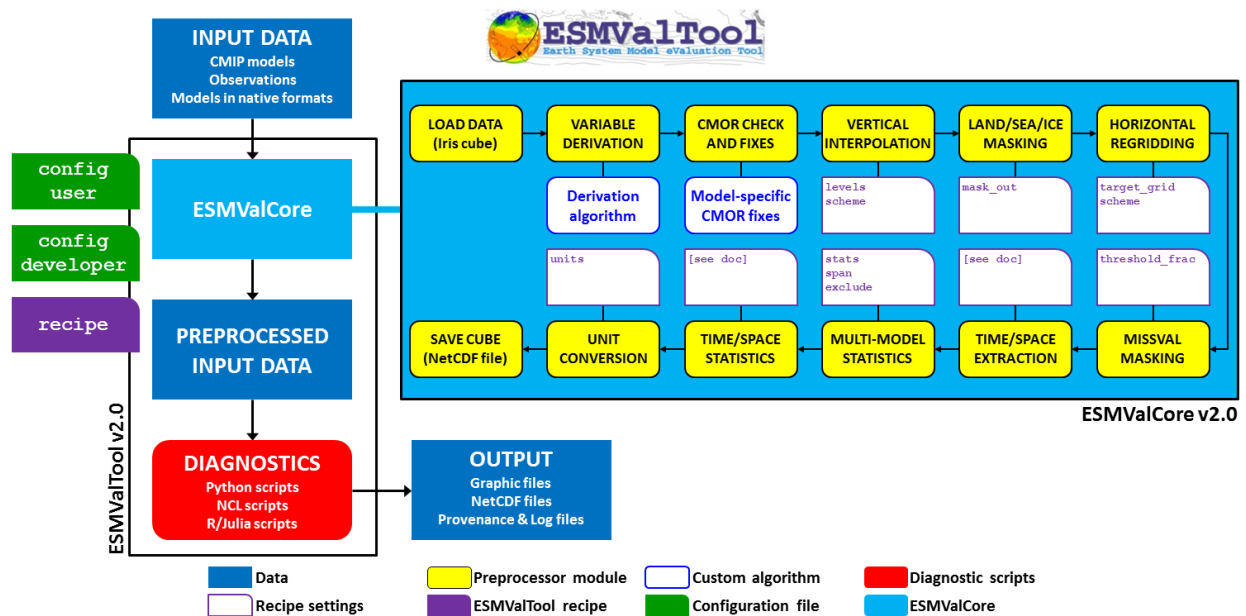


Fig. 1: Schematic of the system architecture.

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

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](#));

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;

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).

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`).

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.

RUN THE DIAGNOSTICS

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

Part III

Getting started

INSTALLATION

Note: ESMValTool now uses *mamba* instead of *conda* for the recommended installation. For more information about the change, have a look at *Move to Mamba*.

ESMValTool 2.0 requires a Unix(-like) operating system and Python 3.8+.

The ESMValTool can be installed in multiple ways.

Recommended installation methods:

- On Linux, please install via the *mamba package manager* (see <https://anaconda.com>);
- For MacOSX, please follow separate instructions for *installation on MacOSX*.

Further options for installation are:

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

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

Note that there is also a [Tutorial](#) available with more explanations and guidance if the installation instructions here are too concise. See *common installation issues* if you run into trouble.

10.1 Mamba/Conda installation

In order to install the *conda* package, you will need mamba pre-installed.

For a minimal mamba installation (recommended) go to <https://mamba.readthedocs.io/en/latest/installation.html>. It is recommended that you always use the latest version of mamba, as problems have been reported when trying to use older versions.

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 Mambaforge-Linux-x86_64.sh
```

and follow the instructions on your screen. Immediately update mamba after installing it:

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

You can check that mamba installed correctly by running

```
which mamba
```

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

10.2 ESMValTool installation

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

```
mamba create --name esmvaltool esmvaltool 'python=3.10'
```

Here mamba is the executable calling the mamba package manager to install esmvaltool. The reason why we are also specifying 'python=3.10' is that it will make it easier for mamba to find a working combination of all required packages, see *Mamba fails to solve the environment* in *common installation issues* for an in-depth explanation. Python 3.8 and 3.9 are also supported, in case you prefer to work with an older version of Python.

This will create a new [conda environment](#) and install ESMValTool into it with a single command.

```
conda activate esmvaltool
```

Of course it is also possible to choose a different name than esmvaltool for the environment.

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: Creating a new conda environment is often much faster and more reliable than trying to update an existing conda environment.

10.2.1 Julia installation

If you want to use the ESMValTool Julia functionality, you will also need to install Julia. If you are just getting started, we suggest that you come back to this step later when and if you need it. To perform the Julia installation, make sure that your conda environment is activated and then execute

```
mamba install julia
```


10.2.2 Installation of subpackages

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

There are three language specific packages available:

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

The main `esmvaltool` package contains all four subpackages listed above. If you only need to run a recipe with diagnostics in some of these languages, it is possible to install only the dependencies needed to do just that. The diagnostic script(s) used in each recipe, are documented in [Recipes](#). The extension of the diagnostic script can be used to see in which language a diagnostic script is written.

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.

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](#).

10.3 Installation on MacOSX

The Python diagnostics of the ESMValTool are supported on MacOSX, but Julia, 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 pre-installed (see [Mamba/Conda installation](#) for more details).

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

```
mamba create --name esmvaltool esmvaltool-python 'python=3.10'
```

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 MacOSX. 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](#).

10.4 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. By far the easiest way to install these dependencies is to use *mamba*. For a minimal mamba installation (recommended) go to <https://mamba.readthedocs.io/en/latest/installation.html>.

After installing mamba, download the file with the list of dependencies:

```
wget https://raw.githubusercontent.com/ESMValGroup/ESMValTool/main/environment.yml
```

and install these dependencies into a new conda environment with the command

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

Finally, activate the newly created environment

```
conda activate esmvaltool
```

and install ESMValTool as well as any remaining Python dependencies with the command:

```
pip install esmvaltool
```

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

```
esmvaltool install Julia
```

10.5 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.

10.6 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
```

10.7 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.

10.7.1 Obtaining the source code

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

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

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

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.

10.7.2 Installation Using Mamba from Source

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

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

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

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

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

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

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

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

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

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

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

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

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

```
cd ESMValTool
```

and (when on Linux) create a new environment called `esmvaltool` containing just Python with the command

```
mamba create --name esmvaltool 'python=3.10'
```

if needed, older versions of Python can also be selected. Next, install many of the required dependencies, including the ESMValCore package and Python, R, and NCL interpreters, into this environment by running

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

MacOSX note: ESMValTool functionalities in Julia, NCL, and R are not supported on MacOSX, due to conflicts in the conda environment. To install a conda environment on MacOSX, use the dedicated environment file:

```
mamba env create --name esmvaltool --file environment_osx.yml
```

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

```
conda activate esmvaltool
```

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

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

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

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

10.7.3 Software installation

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

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

Using the `--editable` flag will cause the installer to create a symbolic link from the installation location to your source code, so any changes you make to the source code will immediately be available in the installed version of the tool. This command will also install extra development dependencies needed for building the documentation, running the unit tests, etc.

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

```
esmvaltool install Julia
```

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.

MacOSX note: some recipes may depend on the OpenMP library, which does not install via mamba on MacOSX. 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
```

10.7.4 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 conda environment.

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

First follow all steps above. Next, go to the place where you would like to keep the 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.

Go into the folder you just downloaded

```
cd ESMValCore
```

and then install ESMValCore in development mode

```
pip install --editable '[develop]'
```

To check that the installation was successful, run

```
python -c 'import esmvalcore; print(esmvalcore.__path__[0])'
```

this should show the directory of the source code that you just downloaded.

If the command above shows a directory inside your conda environment instead, e.g. `~/mamba/envs/esmvaltool/lib/python3.8/site-packages/esmvalcore`, you may need to manually remove that directory and run `pip install -e '[develop]'` again.

10.8 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.
- 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.

10.9 Installation from the conda lock file

A fast conda environment creation is possible using the provided conda lock file. This is a secure alternative to the installation from source, whenever the conda environment can not be created for some reason. A conda lock file is an explicit environment file that contains pointers to dependency packages as they are hosted on the Anaconda cloud; these have frozen version numbers, build hashes, and channel names, parameters established at the time of the conda lock file creation, so may be obsolete after a while, but they allow for a robust environment creation while they're still up-to-date. We regenerate these lock files every 10 days through automatic Pull Requests (or more frequently, since the automatic generator runs on merges on the main branch too), so to minimize the risk of dependencies becoming obsolete. Conda environment creation from a lock file is done just like with any other environment file:

```
conda create --name esmvaltool --file conda-linux-64.lock
```

The latest, most up-to-date file can always be downloaded directly from the source code repository, a direct download link can be found [here](#).

Note: *pip* and *conda* are NOT installed, so you will have to install them in the new environment: use `conda-forge` as channel): `conda install -c conda-forge pip` at the very minimum so we can install *esmvalcore* afterwards.

Note: For instructions on how to manually create the lock file, see [these instructions](#).

10.10 Common installation problems and their solutions

10.10.1 Mamba fails to solve the environment

If you see the text `Solving environment:` with the characters `-\\|/` rotating behind it for more than 10 minutes, mamba may be having problems finding a working combination of versions of the packages that the ESMValTool depends on. Because the ESMValTool is a community tool, there is no strict selection of which tools can be used and installing the ESMValTool requires installing almost any package that is available for processing climate data. To help mamba solve the environment, you can try the following.

Always use the latest version of mamba, as problems have been reported by people using older versions, to update, run:

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

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.

It can help mamba if you let it know what version of certain packages you want, for example by running

```
mamba create -n esmvaltool esmvaltool 'python=3.10'
```

you ask for Python 3.10 specifically and that makes it much easier for mamba to solve the environment, because now it can ignore any packages that were built for other Python versions. Note that, since the esmvaltool package is built with Python>=3.8, asking for an older Python version, e.g. `python=3.7`, in this way, it will result in installation failure.

10.10.2 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 -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.

10.10.3 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.

10.10.4 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.

10.10.5 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`.

10.10.6 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. See also [Mamba fails to solve the environment](#).

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.

10.11 Move to Mamba

Mamba is a much faster alternative to `conda`, and environment creation and updating benefits from the use of a much faster (C++ backend) dependency solver; tests have been performed to verify the integrity of the `esmvaltool` environment built with `mamba`, and we are now confident that the change will not affect the way ESMValTool is installed and run, whether it be on a Linux or OSX platform. From the user's perspective, it is a straightforward use change: the CLI (command line interface) of `mamba` is identical to `conda`: any command that was run with `conda` before will now be run with `mamba` instead, keeping all the other command line arguments and flags as they were before. The only place where `conda` should not be replaced with `mamba` at command line level is at the environment activation point: `conda activate` will still have to be used.

CONFIGURATION

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

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

```
esmvaltool config get_config_user
```

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

There is a lesson available in the [ESMValTool tutorial](#) that describes how to personalize the configuration file. It can be found [at this site](#).

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.

12.1 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 `~/climate_data`, run

```
esmvaltool run examples/recipe_python.yml --offline=False
```

The `--offline=False` option tells ESMValTool to search for and download the necessary climate data files, if they cannot be found locally. The data only needs to be downloaded once, every following run will re-use previously downloaded data. If you have all required data available locally, you can run the tool without the `--offline=False` argument (the default). Note that in that case the required data should be located in the directories specified in your user configuration file. Recall that the chapter [Configuring ESMValTool](#) provides an explanation of how to create your own `config-user.yml` file.

See [running esmvaltool](#) in the ESMValCore documentation for a more complete introduction to the `esmvaltool` command.

12.2 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.

12.3 Running multiple recipes

Have a look at *[Running multiple recipes](#)* if you are interested in running multiple recipes in parallel.

OUTPUT

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

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

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

This directory will contain 4 further subdirectories:

1. *Diagnostic output* (work): A place for any diagnostic script results that are not plots, e.g. files in NetCDF format (depends on the diagnostics).
2. *Plots*: The location for all the plots, split by individual diagnostics and fields.
3. *Run*: This directory includes all log files, a copy of the recipe, a summary of the resource usage, and the *settings.yml* interface files and temporary files created by the diagnostic scripts.
4. *Preprocessed datasets* (preproc): This directory contains all the preprocessed netcdfs data and the *metadata.yml* interface files. Note that by default this directory will be deleted after each run, because most users will only need the results from the diagnostic scripts.

13.1 Preprocessed datasets

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

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

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

13.2 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.

13.3 Diagnostic output

The `work/` directory will contain all files that are output at the diagnostic stage. I.e., 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.

13.4 Plots

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

13.5 Settings.yml

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

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

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

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

13.6 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_T00M_mfo_2002-2004.nc
: cmor_table: CMIP5
dataset: HadGEM2-ES
diagnostic: diag_timeseries_scalars
end_year: 2004
ensemble: r1i1p1
exp: historical
field: T00M
filename: [...]/recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/mfo/
↪ CMIP5_HadGEM2-ES_Omon_historical_r1i1p1_T00M_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
```

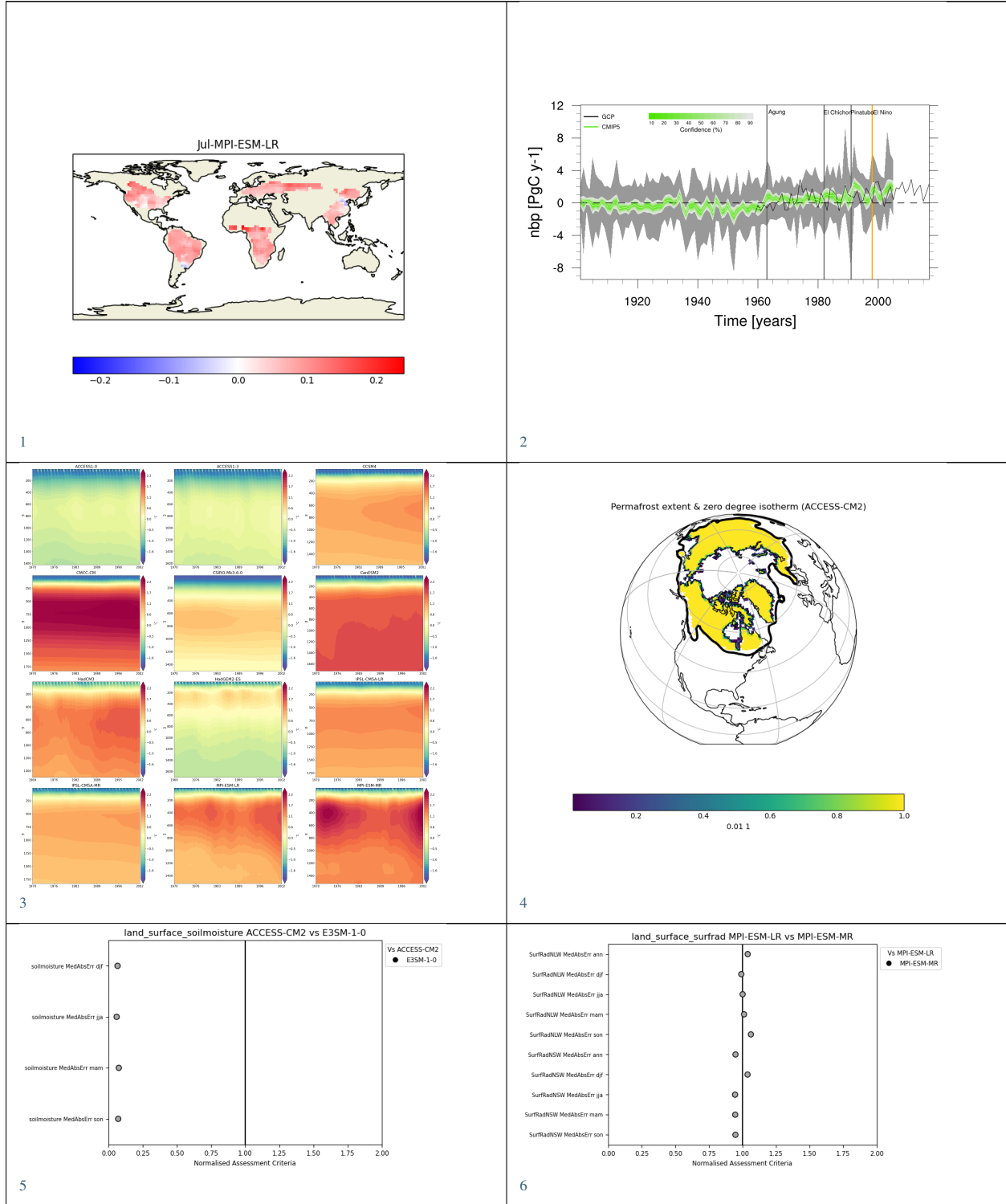
As you can see, this is effectively a dictionary with several items including data paths, metadata and other information.

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.

Part IV

Gallery

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



continues on next page

continues on next page

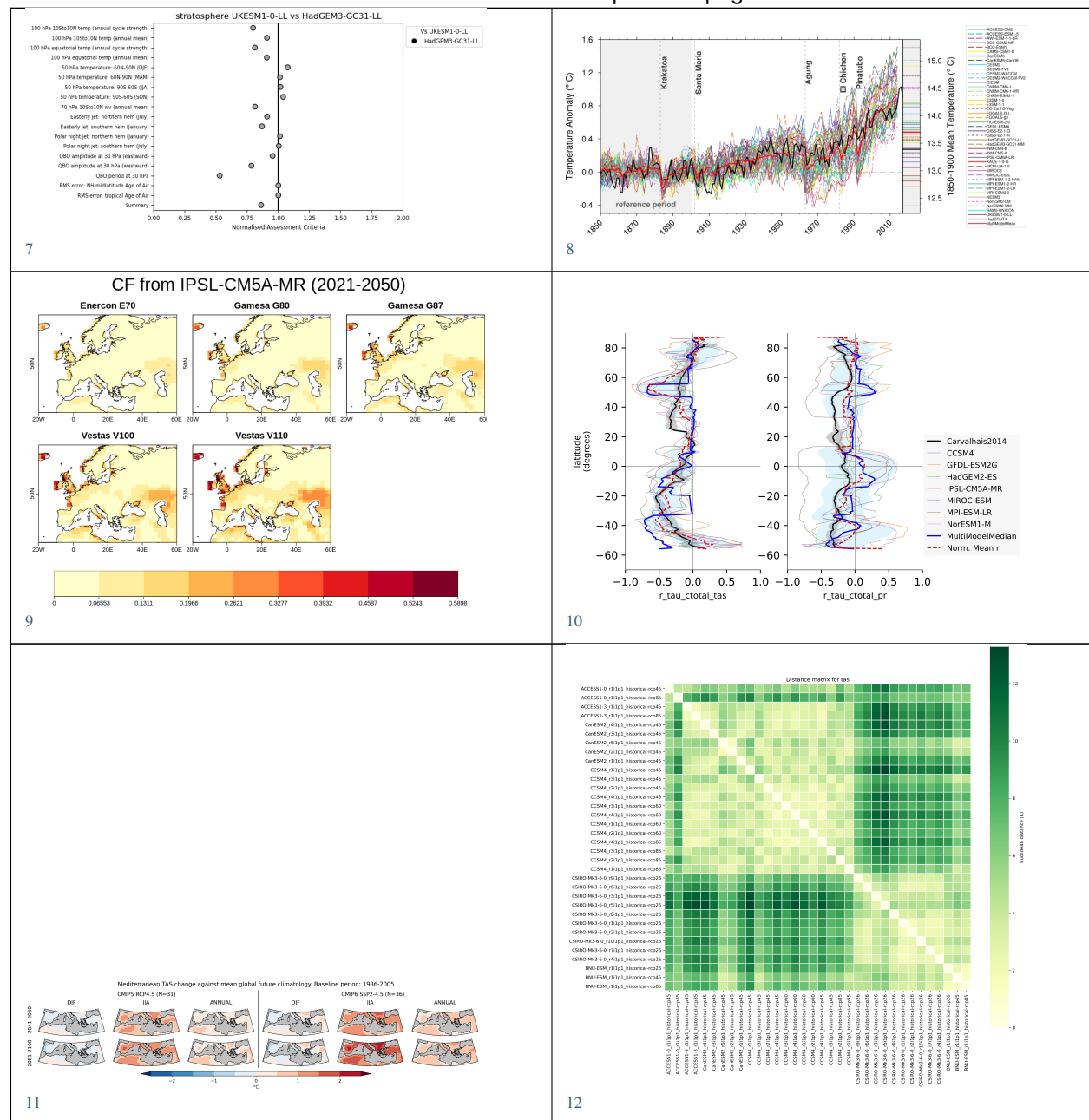
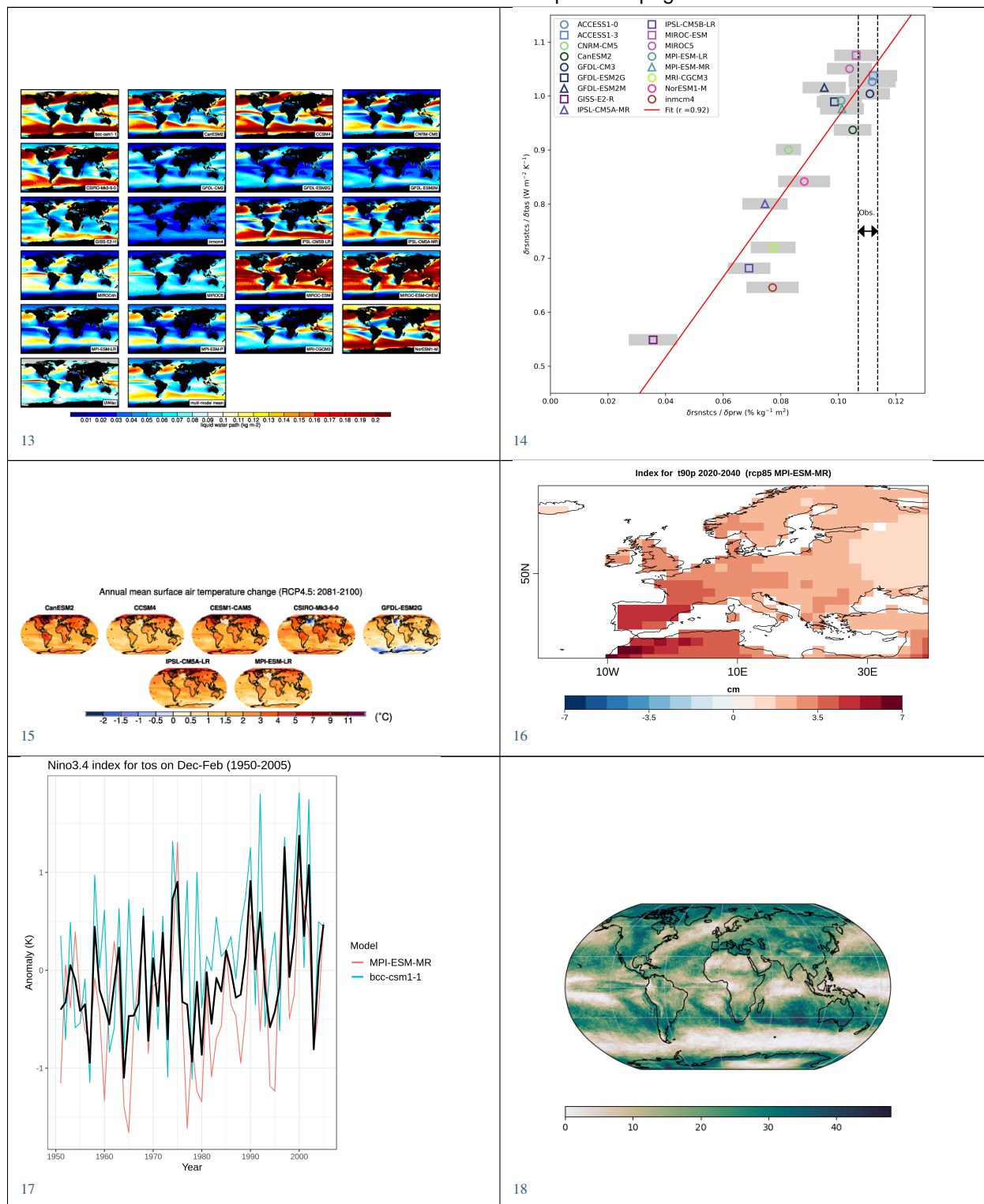
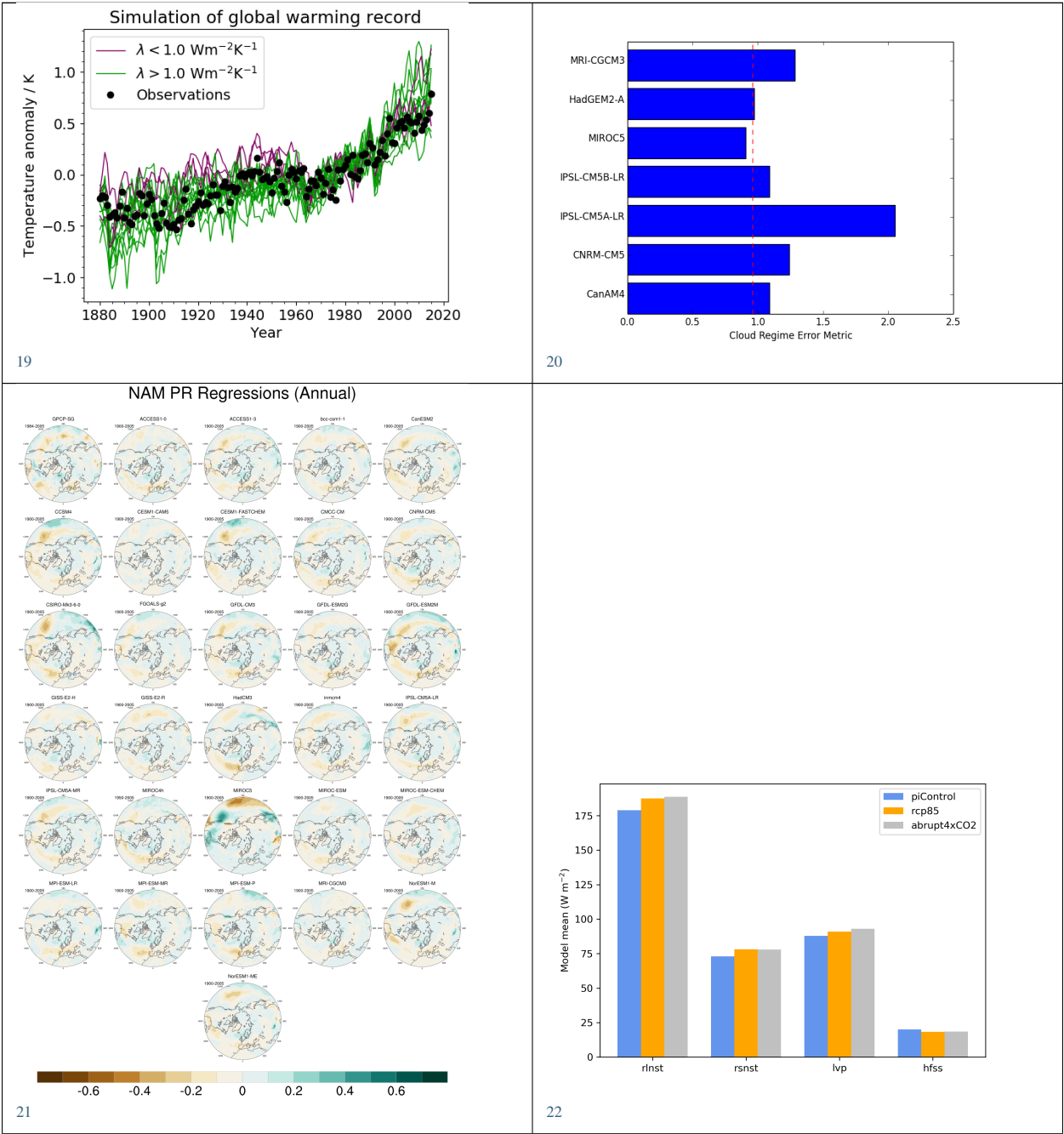


Table 1 – continued from previous page



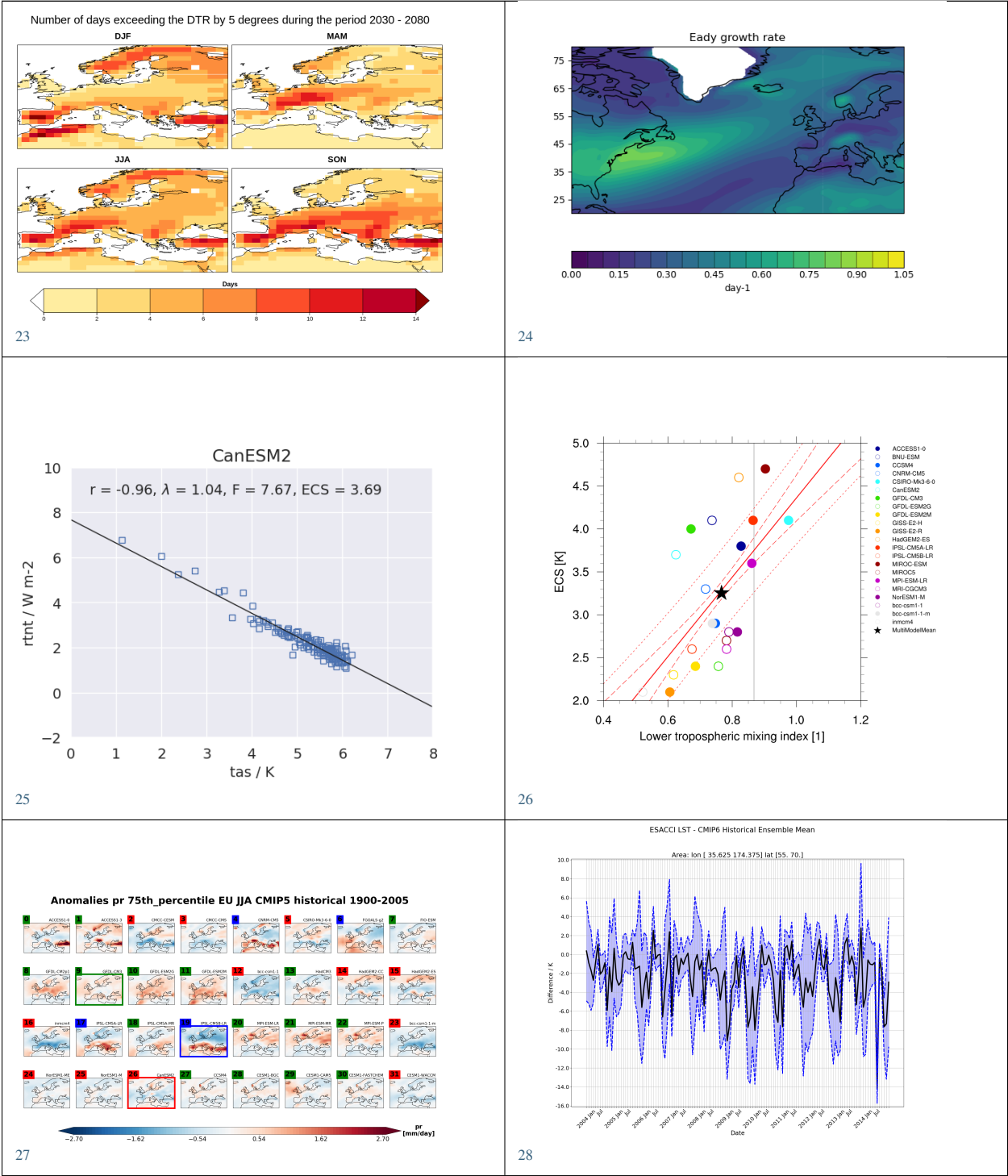
continues on next page

Table 1 – continued from previous page



continues on next page

Table 1 – continued from previous page



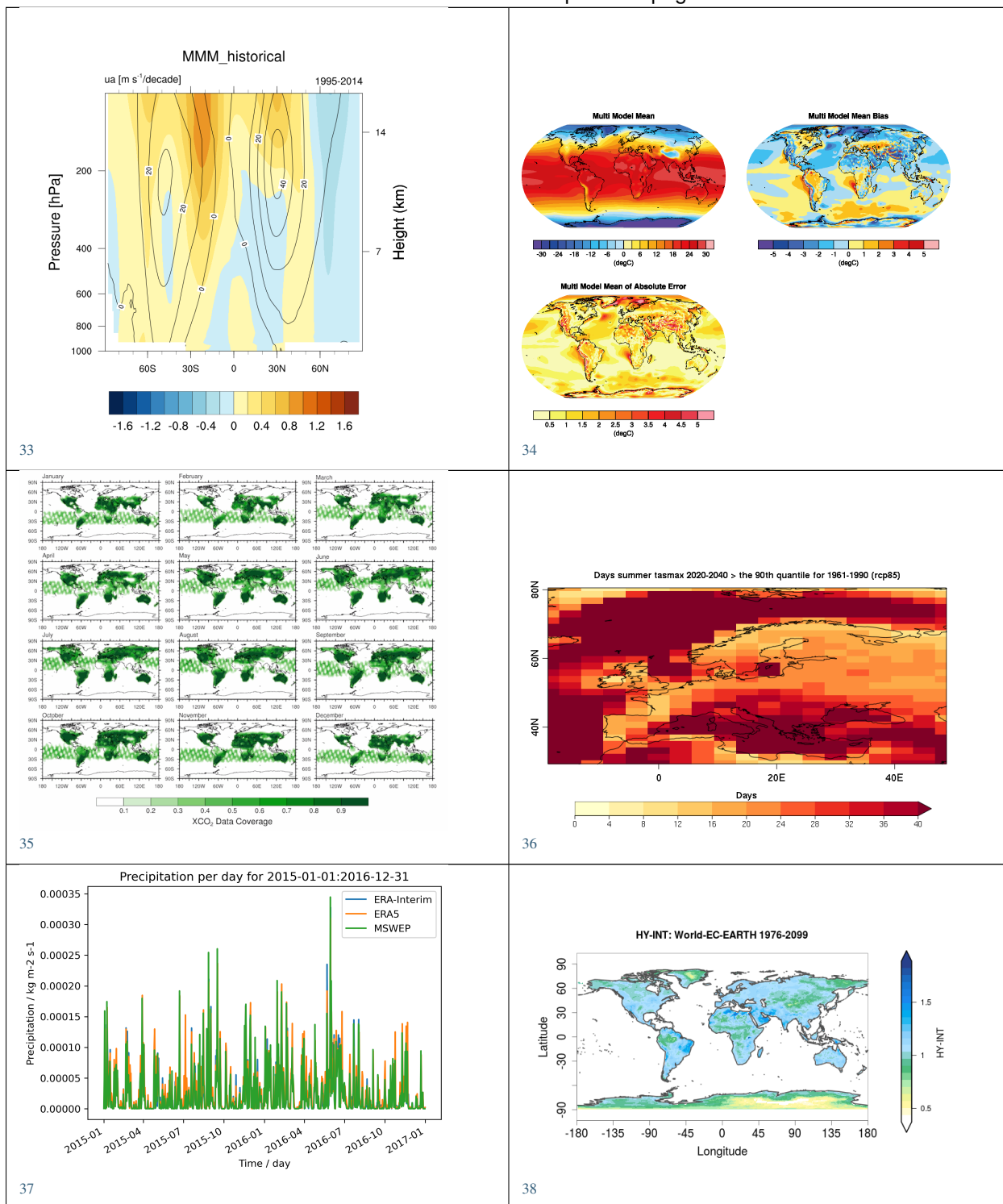
continues on next page

Table 1 – continued from previous page

<p>Mass Concentration of Total Phytoplankton Expressed as Chlorophyll in Sea Water [mg m⁻³]</p> <p>29</p>	<p>Air temperature</p> <p>30</p>
<p>31</p>	<p>32</p>

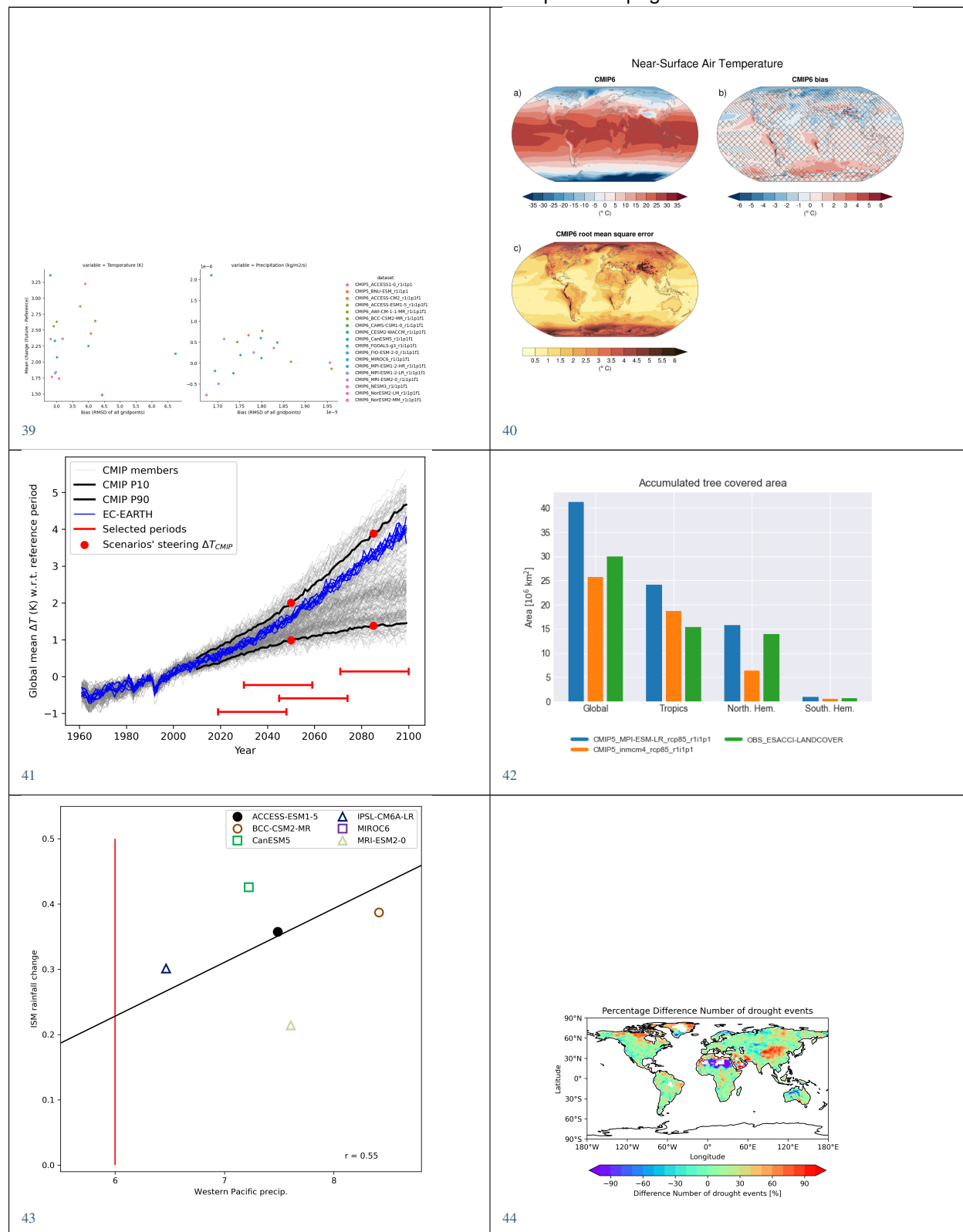
continues on next page

Table 1 – continued from previous page



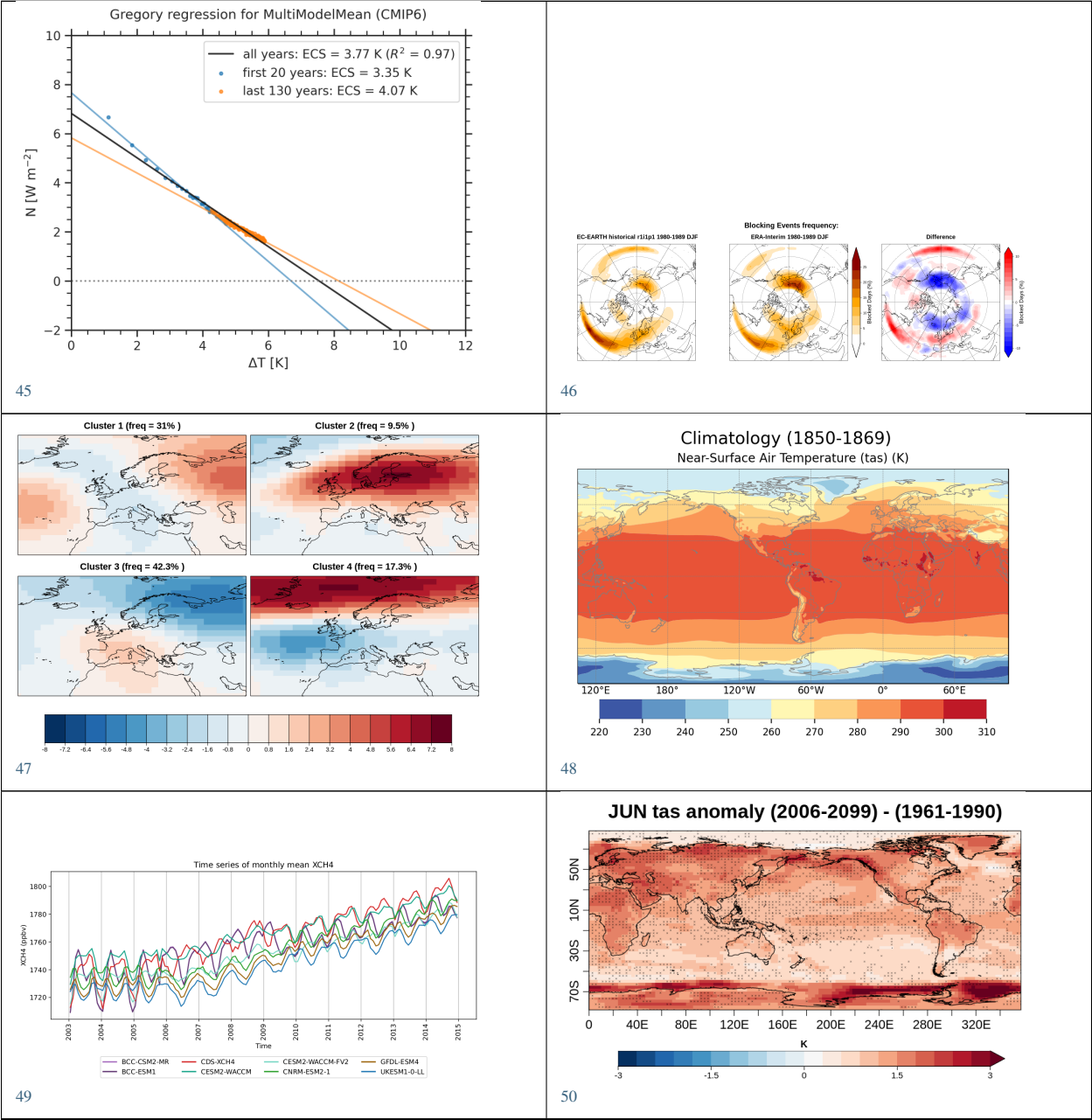
continues on next page

Table 1 – continued from previous page



continues on next page

Table 1 – continued from previous page

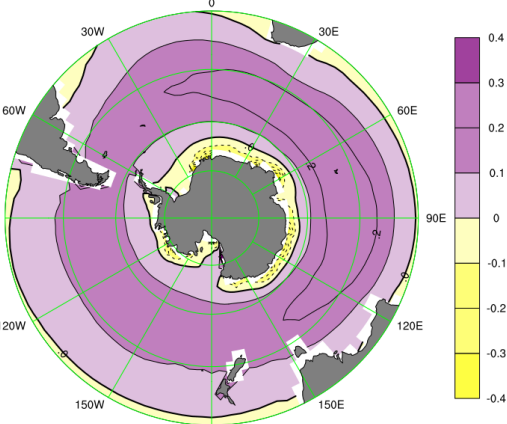
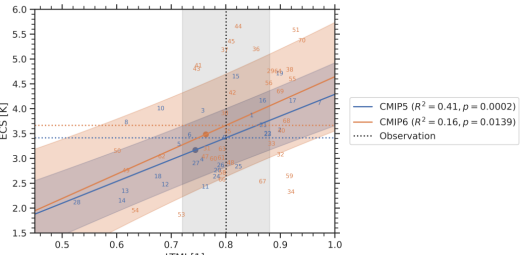
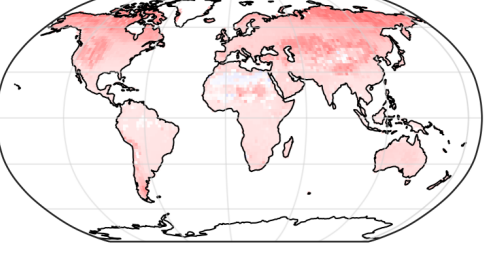
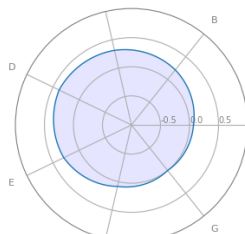
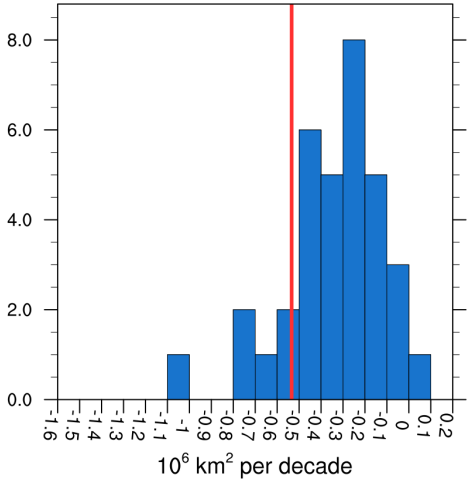
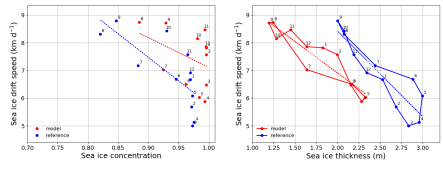


continues on next page

continues on next page

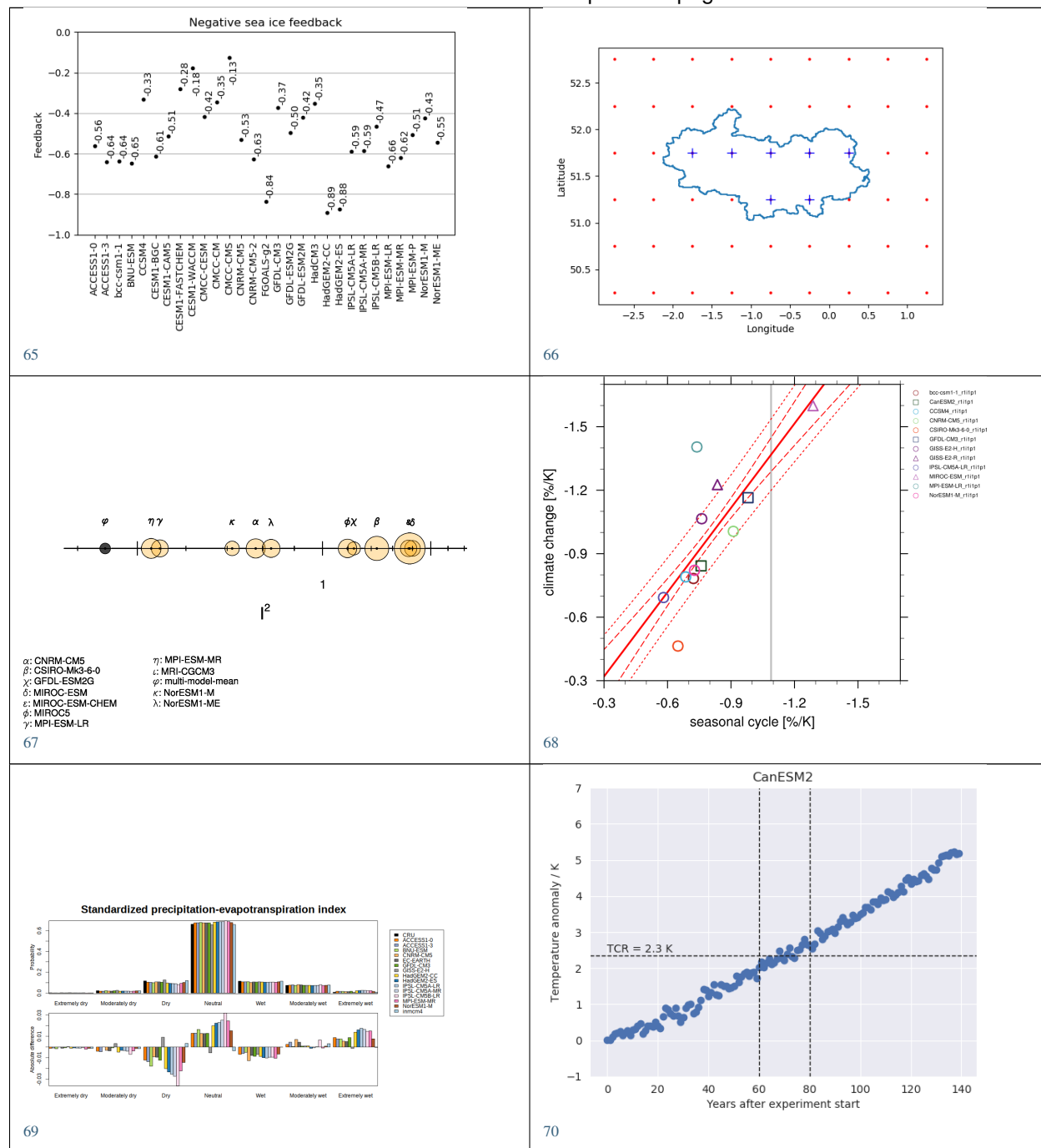


Table 1 – continued from previous page

<p>CanESM2</p> <p>annual mean 1986 - 2005</p> <p>tauu (Pa)</p>  <p>59</p>	<p>Sherwood et al. (2014) constraint (SHL)</p>  <p>60</p>
<p>GBRT</p>  <p>61</p>	<p>sos bias</p> <p>MPI-ESM1-2-HR vs ESACCI-SEA-SURFACE-SALINITY_V1</p>  <p>62</p>
<p>September Arctic sea ice extent trends 1960-2005</p>  <p>63</p>	<p>Seasonal cycle CMIP5_GFDL-ESM2G_historical_r1i1p1_1979_2005</p>  <p>64</p>

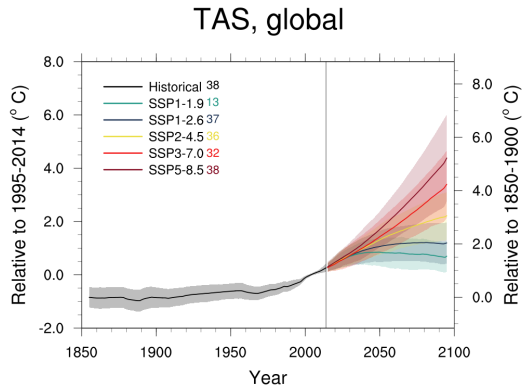
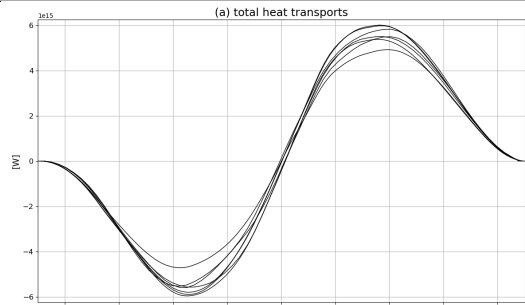
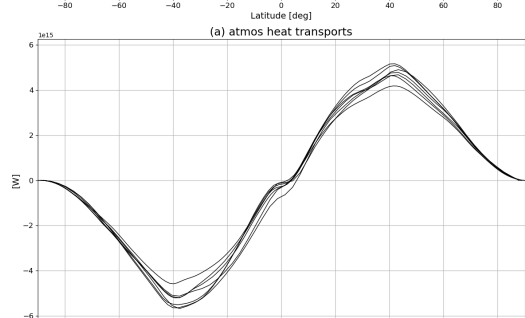
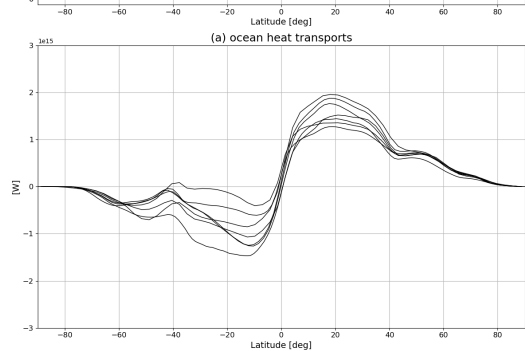
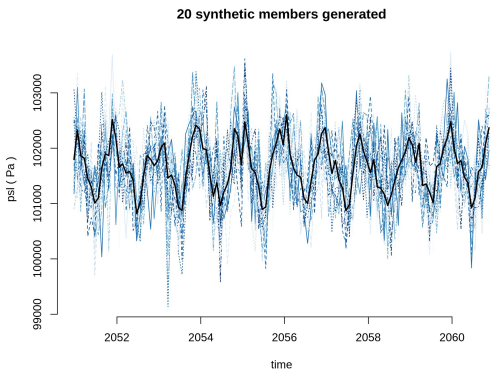
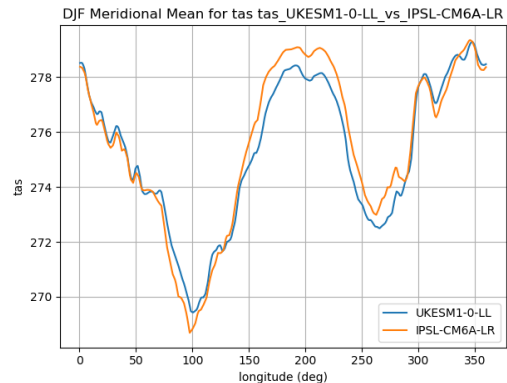
continues on next page

Table 1 – continued from previous page



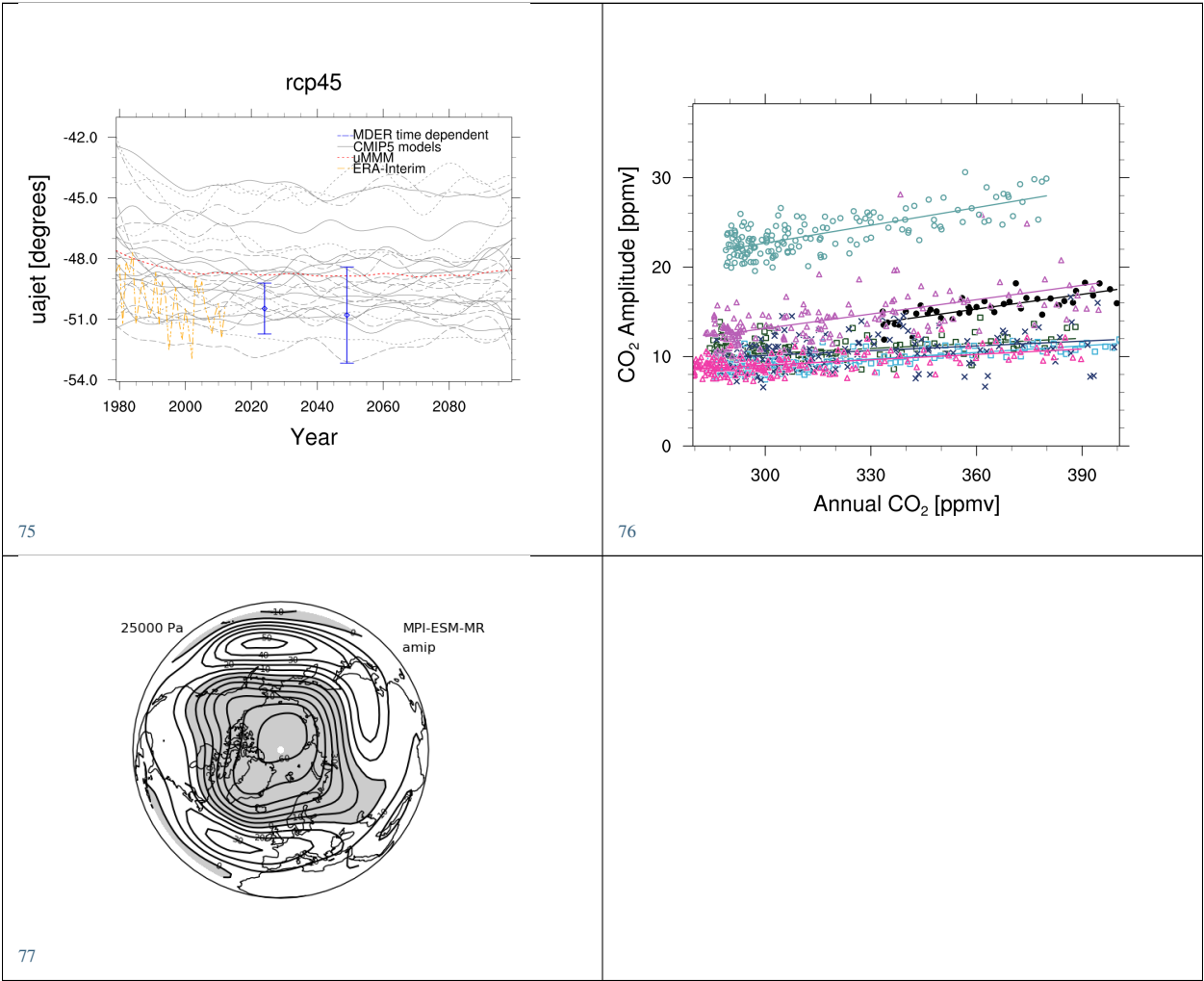
continues on next page

Table 1 – continued from previous page

<div data-bbox="207 709 727 1094"><p>TAS, global</p></div> <div data-bbox="207 1197 227 1218">71</div>	<div data-bbox="828 222 1349 1188"><p>(a) total heat transports</p><p>(a) atmos heat transports</p><p>(a) ocean heat transports</p></div> <div data-bbox="828 1197 847 1218">72</div>
<div data-bbox="207 1249 695 1619"><p>20 synthetic members generated</p></div> <div data-bbox="207 1638 227 1659">73</div>	<div data-bbox="828 1239 1331 1619"><p>DJF Meridional Mean for tas_tas_UKESM1-0-LL_vs_IPSL-CM6A-LR</p></div> <div data-bbox="828 1638 847 1659">74</div>

continues on next page

Table 1 – continued from previous page



- 1 *Landcover - Albedo*
- 2 *Land and ocean components of the global carbon cycle*
- 3 *Recipe for evaluating Arctic Ocean*
- 4 *Land-surface Permafrost - Autoassess diagnostics*
- 5 *Land-surface Soil Moisture - Autoassess diagnostics*
- 6 *Land-surface Surface Radiation - Autoassess diagnostics*
- 7 *Stratosphere - Autoassess diagnostics*
- 8 *Quantifying progress across different CMIP phases*
- 9 *Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power*
- 10 *Turnover time of carbon over land ecosystems*
- 11 *Climate Change Hotspot*
- 12 *Climate model Weighting by Independence and Performance (ClimWIP)*
- 13 *Clouds*
- 14 *Evaluate water vapor short wave radiance absorption schemes of ESMs with the observations, including ESACCI data.*
- 15 *IPCC AR5 Chapter 12 (selected figures)*
- 16 *Combined Climate Extreme Index*
- 17 *Nino indices, North Atlantic Oscillation (NAO), Souther Oscillation Index (SOI)*
- 18 *Consecutive dry days*
- 19 *Emergent constraint on equilibrium climate sensitivity from global temperature variability*
- 20 *Cloud Regime Error Metric (CREM)*
- 21 *Climate Variability Diagnostics Package (CVDp)*
- 22 *Evaluate water vapor short wave radiance absorption schemes of ESMs with the observations.*
- 23 *Diurnal temperature range*
- 24 *Eady growth rate*
- 25 *Equilibrium climate sensitivity*
- 26 *Emergent constraints for equilibrium climate sensitivity*
- 27 *Ensemble Clustering - a cluster analysis tool for climate model simulations (EnsClus)*
- 28 *ESA CCI LST comparison to Historical Models*
- 29 *Ocean chlorophyll in ESMs compared to ESA-CCI observations.*
- 30 *Example recipes*
- 31 *Extreme Events Indices (ETCCDI)*
- 32 *Diagnostics of stratospheric dynamics and chemistry*
- 33 *Ozone and associated climate impacts*
- 34 *IPCC AR5 Chapter 9 (selected figures)*
- 35 *Spatially resolved evaluation of ESMs with satellite column-averaged CO₂*
- 36 *Heat wave and cold wave duration*
- 37 *Hydro forcing comparison*
- 38 *Hydroclimatic intensity and extremes (HyInt)*
- 39 *Quick insights for climate impact researchers*
- 40 *IPCC AR6 Chapter 3 (selected figures)*
- 41 *KNMI Climate Scenarios 2014*
- 42 *Landcover diagnostics*
- 43 *Constraining future Indian Summer Monsoon projections with the present-day precipitation over the tropical western Pacific*
- 44 *Drought characteristics following Martin (2018)*
- 45 *Context for interpreting equilibrium climate sensitivity and transient climate response from the CMIP6 Earth system models*
- 46 *Blocking metrics and indices, teleconnections and weather regimes (MiLES)*
- 47 *Modes of variability*
- 48 *Monitor*
- 49 *Diagnostics of integrated atmospheric methane (XCH₄)*
- 50 *Multi-model products*
- 51 *Ocean diagnostics*
- 52 *Performance metrics for essential climate parameters*
- 53 *Psyplot Diagnostics*
- 54 *Capacity factor for solar photovoltaic (PV) systems*
- 55 *Precipitation quantile bias*
- 56 *Radiation Budget*
- 57 *RainFARM stochastic downscaling*
- 58 *Runoff, Precipitation, Evapotranspiration*
- 59 *Ocean metrics*
- 60 *Emergent constraints on equilibrium climate sensitivity in CMIP5: do they hold for CMIP6?*
- 61 *Constraining uncertainty in projected gross primary production (GPP) with machine learning*
- 62 *Sea Surface Salinity Evaluation*
- 63 *Sea Ice*
- 64 *Seaice drift*
- 65 *Seaice feedback*
- 66 *Shapeselect*
- 67 *Single Model Performance Index (SMPI)*
- 68 *Emergent constraint on snow-albedo effect*
- 69 *Standardized Precipitation-Evapotranspiration Index (SPEI)*
- 70 *Transient Climate Response*
- 71 *Climate model projections from the ScenarioMIP of CMIP6*
- 72 *Thermodynamics of the Climate System - The Diagnostic Tool TheDiaTo v1.0*
- 73 *Toymodel*
- 74 *Zonal and Meridional Means*

Part V

Recipes

ATMOSPHERE

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

14.1.1 Overview

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

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

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

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

1D Atmospheric Blocking

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

2D Atmospheric blocking

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

Z500 Empirical Orthogonal Functions

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

North Atlantic Weather Regimes

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

14.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

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

Diagnostics are stored in diag_scripts/miles/

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

and subroutines

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

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

14.1.3 User settings

1. miles_block.R

Required settings for variables

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

Required settings for script

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

2. miles_eof.R

Required settings for variables

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

Required settings for script

- seasons: Selected season('DJF','MAM','JJA','SON','ALL') or your period as e.g. 'Jan_Feb_Mar'
- teles: Select EOFs ('NAO','AO','PNA') or specify custom area as "lon1_lon2_lat1_lat2"

3. miles_regimes.R

Required settings for variables

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

Required or optional settings for script

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

14.1.4 Variables

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

14.1.5 Observations and reformat scripts

- ERA-INTERIM

14.1.6 References

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

14.1.7 Example plots

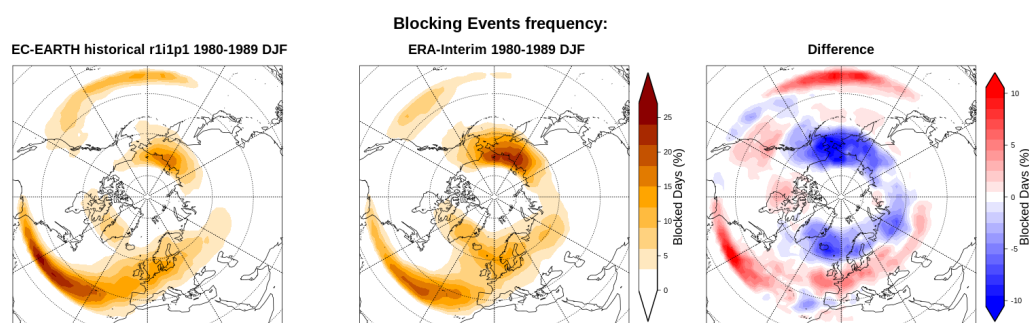


Fig. 1: Blocking Events frequency for a CMIP5 EC-Earth historical run (DJF 1980-1989), compared to ERA-Interim. Units are percentage of blocked days per season.

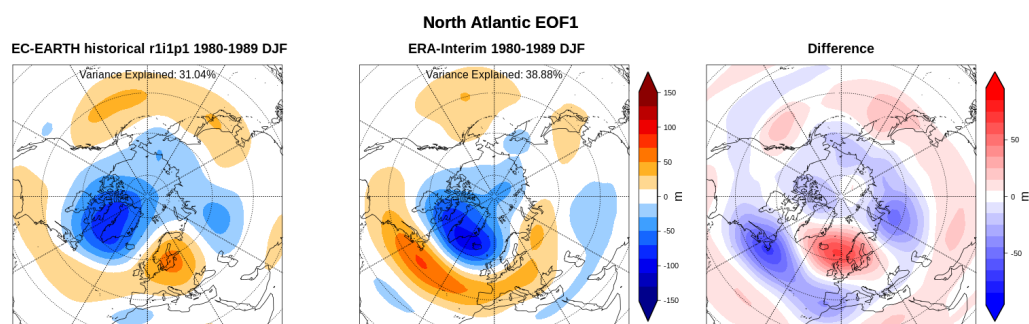


Fig. 2: North Atlantic Oscillation for a CMIP5 EC-Earth historical run (DJF 1980-1989) compared to ERA-Interim, shown as the linear regression of the monthly Z500 against the first Principal Component (PC1) of the North Atlantic region.

14.2 Clouds

14.2.1 Overview

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

14.2.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_lauer13jclim.yml`

Diagnostics are stored in `diag_scripts/clouds/`

- `clouds.ncl`: global maps of (multi-year) annual means including multi-model mean
- `clouds_bias.ncl`: global maps of the multi-model mean and the multi-model mean bias
- `clouds_interannual`: global maps of the interannual variability
- `clouds_isccp`: global maps of multi-model mean minus observations + zonal averages of individual models, multi-model mean and observations
- `clouds_taylor.ncl`: taylor diagrams

14.2.3 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 filenames
- `panel_labels`: label individual panels (true, false)
- `PanelTop`: manual override for “@gnsPanelTop” used by panel plot(s)
- `projection`: map projection for plotting (default = “CylindricalEquidistant”)
- `showdiff`: calculate and plot differences model - reference (default = false)
- `rel_diff`: if `showdiff` = true, then plot relative differences (%) (default = False)

- `ref_diff_min`: lower cutoff value in case of calculating relative differences (in units of input variable)
- `region`: show only selected geographic region given as `latmin`, `latmax`, `lonmin`, `lonmax`
- `timemean`: time averaging - “seasonal” = DJF, MAM, JJA, SON), “annual” = annual mean
- `treat_var_as_error`: treat variable as error when averaging (true, false); true: $\text{avg} = \sqrt{\text{mean}(\text{var} * \text{var})}$, false: $\text{avg} = \text{mean}(\text{var})$

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 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`

3. Script `clouds_interannual.ncl`

Required settings (scripts)

none

Optional settings (scripts)

- `colormap`: e.g., WhiteBlueGreenYellowRed, rainbow
- `explicit_cn_levels`: use these contour levels for plotting
- `extrafiles`: write plots for individual models to separate files (true, false)
- `projection`: map projection, e.g., Mollweide, Mercator

Required settings (variables)

none

Optional settings (variables)

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

Color tables

- variable “lwp”: diag_scripts/shared/plots/rgb/qcm3.rgb

1. Script 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

2. 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 = “”)

- `mask_ts_sea_ice`: true = mask $T < 272$ K as sea ice (only for variable “ts”); false = no additional grid cells masked for variable “ts”
- `styleset`: “CMIP5”, “DEFAULT” (if not set, `clouds_taylor.ncl` will create a color table and symbols for plotting)
- `timemean`: time averaging; `annualclim` (default) = 1 plot annual mean; `seasonalclim` = 4 plots (DJF, MAM, JJA, SON)
- `valid_fraction`: used for creating sea ice mask (`mask_ts_sea_ice` = true): fraction of valid time steps required to mask grid cell as valid data

Required settings (variables)

- `reference_dataset`: name of reference data set

Optional settings (variables)

none

14.2.4 Variables

- `clwvi` (atmos, monthly mean, longitude latitude time)
- `clivi` (atmos, monthly mean, longitude latitude time)
- `clt` (atmos, monthly mean, longitude latitude time)
- `pr` (atmos, monthly mean, longitude latitude time)
- `rlut`, `rlutcs` (atmos, monthly mean, longitude latitude time)
- `rsut`, `rsutcs` (atmos, monthly mean, longitude latitude time)

14.2.5 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 reformat scripts for non-obs4MIPs data for download instructions.

- CERES-EBAF (obs4MIPs) - CERES TOA radiation fluxes (used for calculation of cloud forcing)
- GPCP-SG (obs4MIPs) - Global Precipitation Climatology Project total precipitation
- MODIS (obs4MIPs) - MODIS total cloud fraction
- UWisc - University of Wisconsin-Madison liquid water path climatology, based on satellite observations from TMI, SSM/I, and AMSR-E, reference: O'Dell et al. (2008), J. Clim.

Reformat script: `esmvaltool/cmorizers/data/formatters/datasets/uwisc.ncl`

14.2.6 References

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

- Lauer A., and K. Hamilton (2013), Simulating clouds with global climate models: A comparison of CMIP5 results with CMIP3 and satellite data, *J. Clim.*, 26, 3823-3845, doi: 10.1175/JCLI-D-12-00451.1.
- O'Dell, C.W., F.J. Wentz, and R. Bennartz (2008), Cloud liquid water path from satellite-based passive microwave observations: A new climatology over the global oceans, *J. Clim.*, 21, 1721-1739, doi:10.1175/2007JCLI1958.1.
- Pincus, R., S. Platnick, S.A. Ackerman, R.S. Hemler, Robert J. Patrick Hofmann (2012), Reconciling simulated and observed views of clouds: MODIS, ISCCP, and the limits of instrument simulators. *J. Climate*, 25, 4699-4720, doi: 10.1175/JCLI-D-11-00267.1.

14.2.7 Example plots

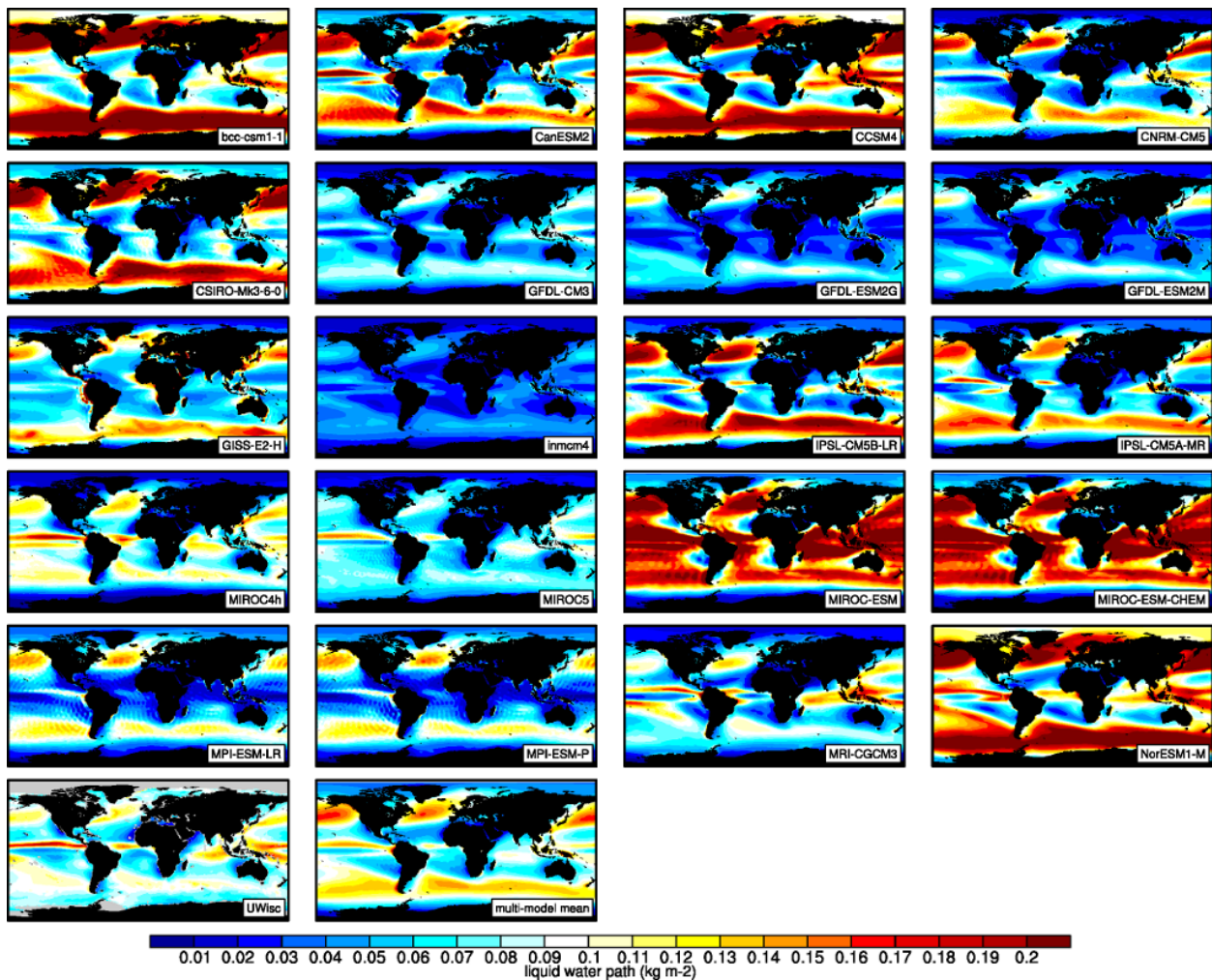


Fig. 3: The 20-yr average LWP (1986-2005) from the CMIP5 historical model runs and the multi-model mean in comparison with the UWisc satellite climatology (1988-2007) based on SSM/I, TMI, and AMSR-E (O'Dell et al. 2008).

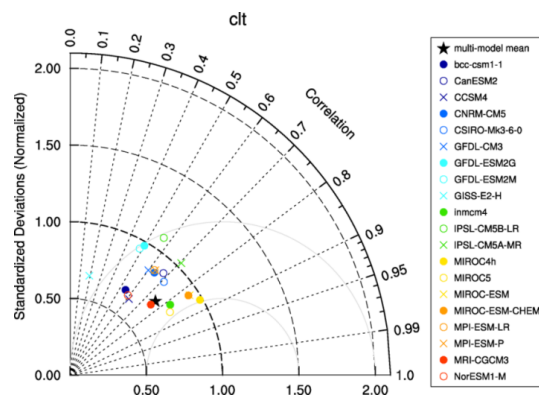


Fig. 4: Taylor diagram showing the 20-yr annual average performance of CMIP5 models for total cloud fraction as compared to MODIS satellite observations.

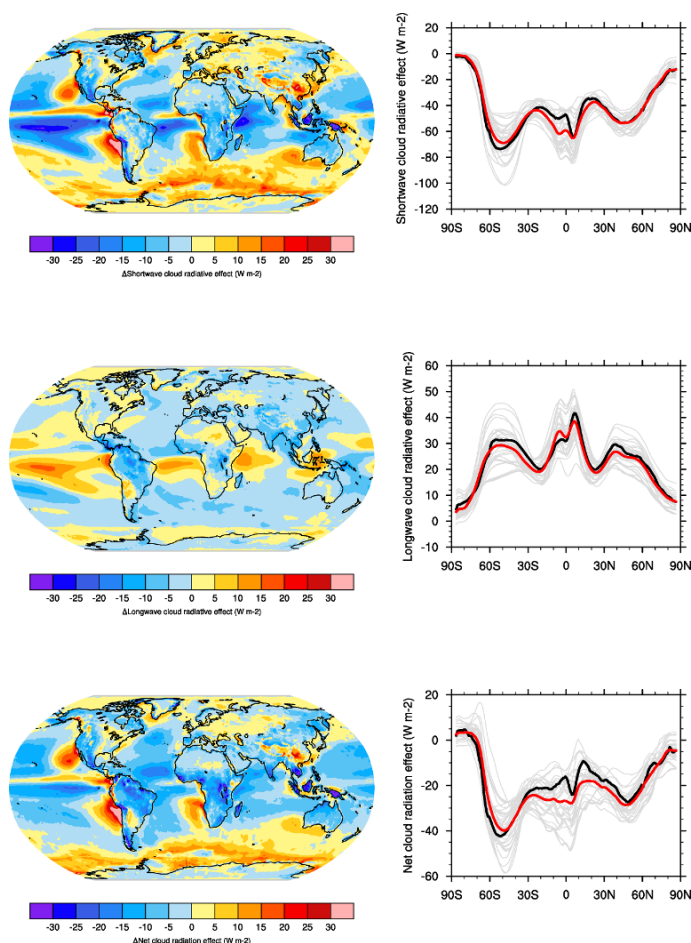


Fig. 5: 20-year average (1986–2005) annual mean cloud radiative effects of CMIP5 models against the CERES EBAF (2001–2012). Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model mean biases against CERES EBAF are shown on the left, whereas the right panels show zonal averages from CERES EBAF (thick black), the individual CMIP5 models (thin gray lines) and the multi-model mean (thick red line). Similar to Figure 9.5 of Flato et al. (2013).

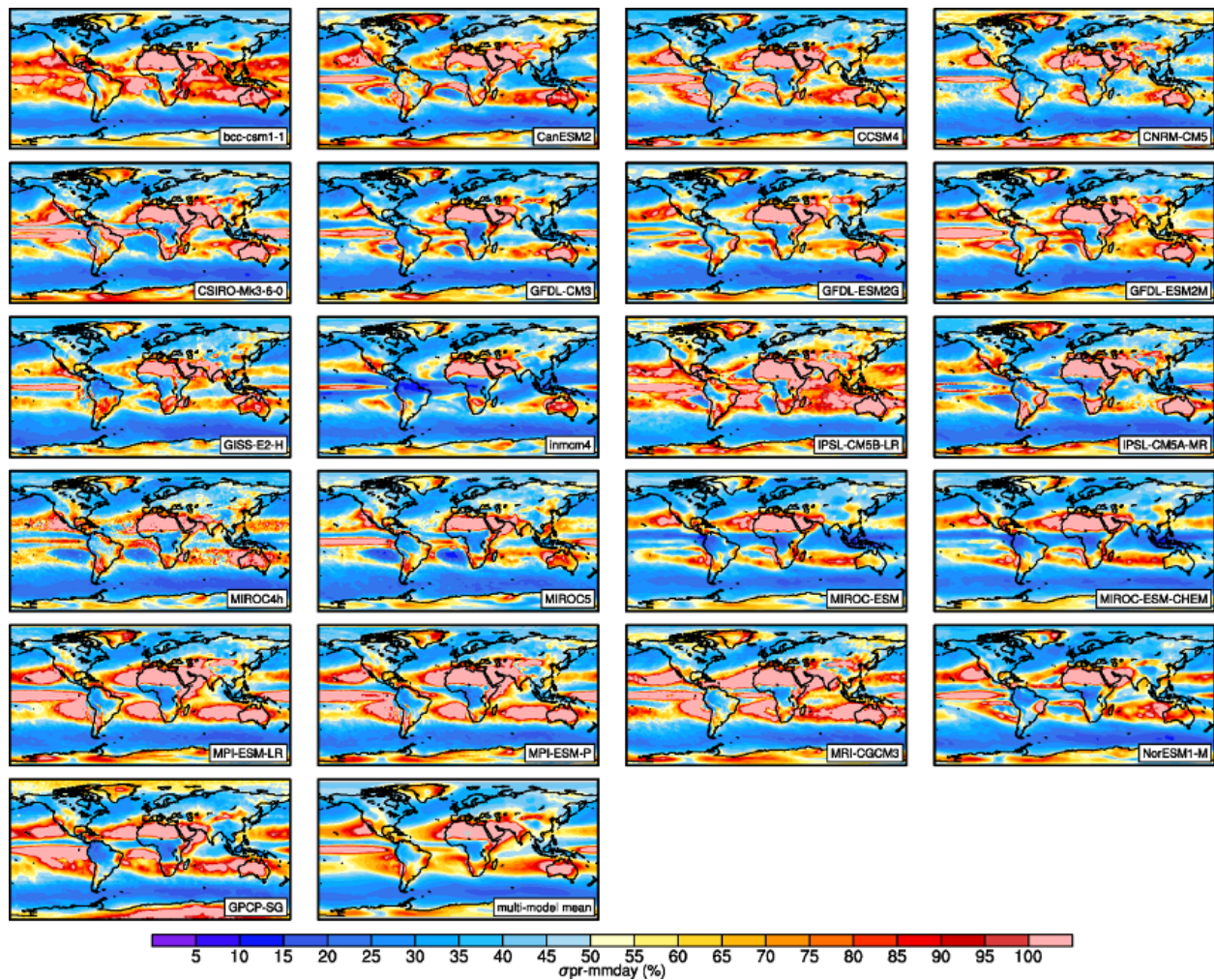


Fig. 6: Interannual variability of modeled and observed (GPCP) precipitation rates estimated as relative temporal standard deviation from 20 years (1986-2005) of data. The temporal standard deviations are calculated from monthly anomalies after subtracting the climatological mean seasonal cycle.

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

14.3.1 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 planned to consist of three-dimensional vertically resolved monthly mean water vapour data (in ppmv) with spatial resolution of 100 km, covering 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 planned 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.

14.3.2 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`

14.3.3 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 CO₂. 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.

14.3.4 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)

14.3.5 Observations and reformat scripts

deangelisf3f4.py:

- ***rsnstcs*:**
CERES-EBAF
- ***prw***
HOAPS, planned 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

14.3.6 References

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

14.3.7 Example plots

14.4 Cloud Regime Error Metric (CREM)

14.4.1 Overview

The radiative feedback from clouds remains the largest source of uncertainty in determining the climate sensitivity. Traditionally, cloud has been evaluated in terms of its impact on the mean top of atmosphere fluxes. However it is quite possible to achieve good performance on these criteria through compensating errors, with boundary layer clouds being too reflective but having insufficient horizontal coverage being a common example (e.g., Nam et al., 2012). Williams and Webb (2009) (WW09) propose a Cloud Regime Error Metric (CREM) which critically tests the ability of a model to simulate both the relative frequency of occurrence and the radiative properties correctly for a set of cloud regimes determined by the daily mean cloud top pressure, cloud albedo and fractional coverage at each grid-box. WW09 describe in detail how to calculate their metrics and we have included the CREMpd metric from their paper in ESMValTool, with clear references in the lodged code to tables in their paper. This has been applied to those CMIP5 models who have submitted the required diagnostics for their AMIP simulation (see Figure 8 below). As documented by WW09, a perfect score with respect to ISCCP would be zero. WW09 also compared MODIS/ERBE to ISCCP in order to provide an estimate of observational uncertainty. This was found to be 0.96 and this is marked on Figure 8,

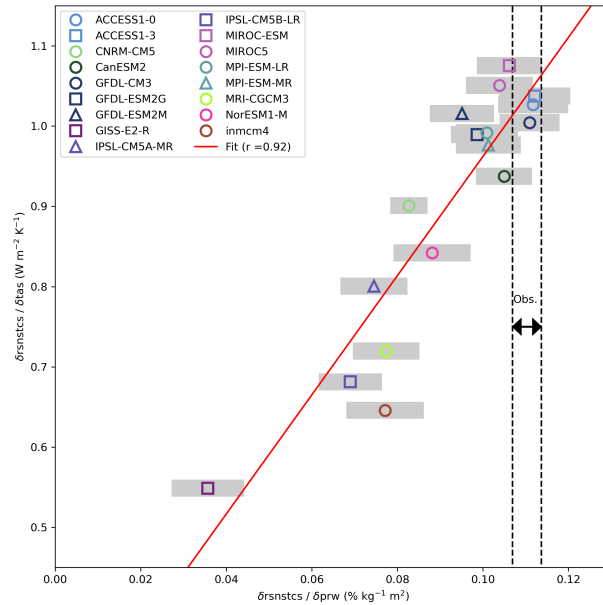


Fig. 7: Scatter plot and regression line computed between the ratio of the change of net short wave radiation (rsnst) and the change of the Water Vapor Path (prw) against the ratio of the change of net short wave radiation for clear sky (rsnstcs) and the change of surface temperature (tas). The width 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.

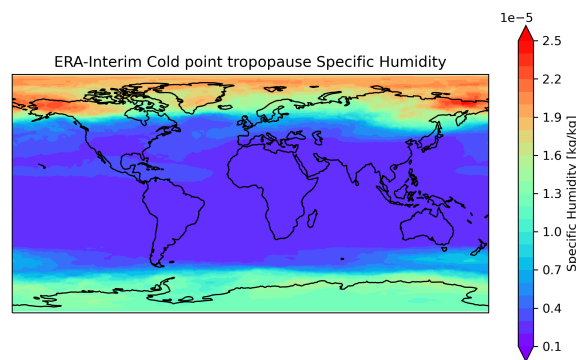


Fig. 8: Map of the average Specific Humidity (hus) at the cold point tropopause from ERA-Interim data. The diagnostic averages the complete time series, here 2010-2014.

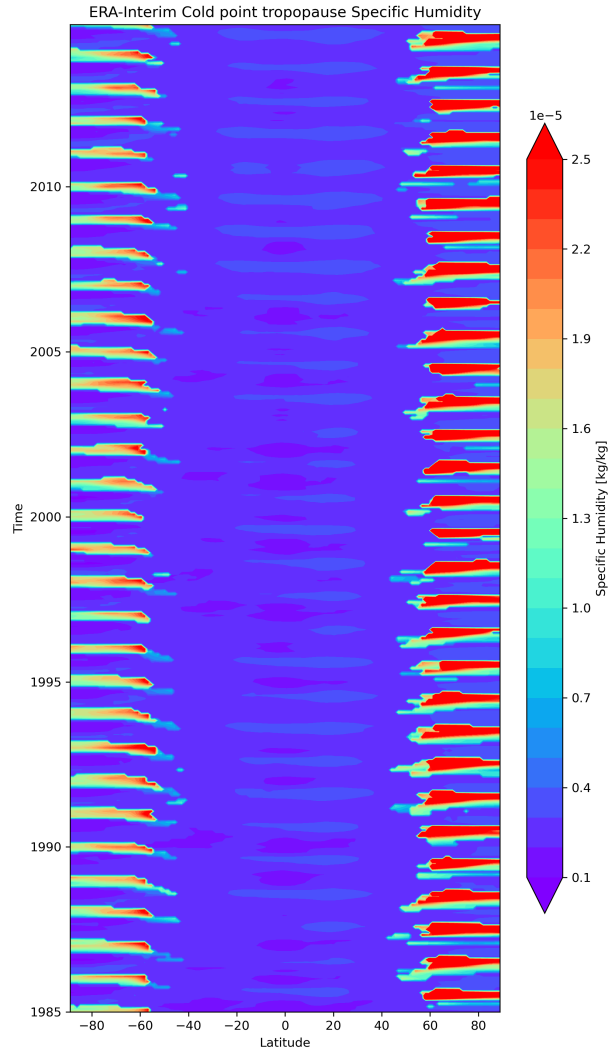


Fig. 9: Latitude versus time plot of the Specific Humidity (hus) at the cold point tropopause from ERA-Interim data.

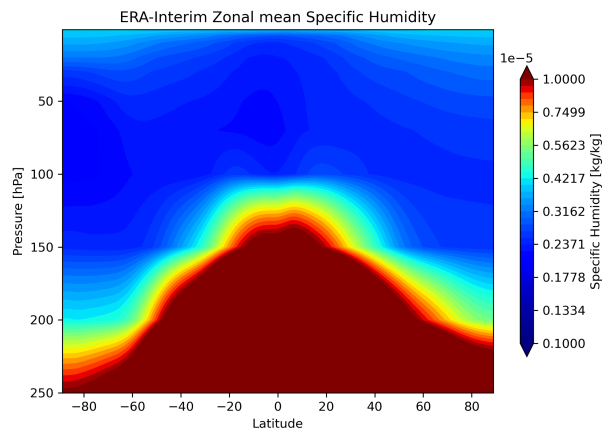


Fig. 10: Zonal average Specific Humidity (hus) between 250 and 1 hPa from ERA-Interim data. The diagnostic averages the complete time series, here 1985-2014.

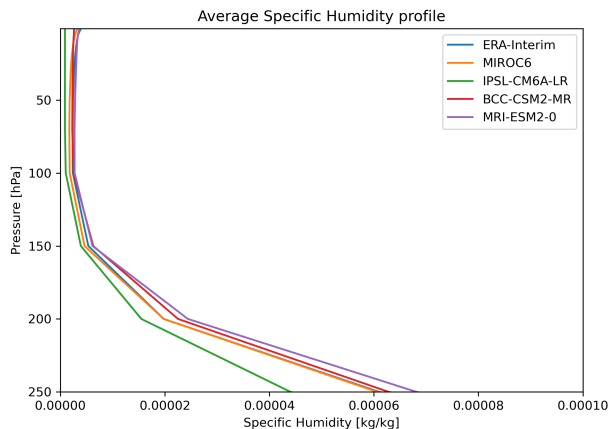


Fig. 11: Average Specific Humidity (hus) profile between 250 and 1 hPa from ERA-Interim and CMIP6 model data. The diagnostic averages the complete time series, here 1985-2014.

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.

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.

14.4.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_williams09climdyn_CREM.yml`

Diagnostics are stored in `diag_scripts/crem/`

- `ww09_esmvaltool.py`

14.4.3 User settings

None.

14.4.4 Variables

- `albiscpp` (atmos, daily mean, longitude latitude time)
- `cltiscpp` (atmos, daily mean, longitude latitude time)
- `pctiscpp` (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.

14.4.5 Observations and reformat scripts

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

14.4.6 References

- Nam, C., Bony, S., Dufresne, J.-L., and Chepfer, H.: The ‘too few, too bright’ tropical low-cloud problem in CMIP5 models, *Geophys. Res. Lett.*, 39, L21801, doi: 10.1029/2012GL053421, 2012.
- Williams, K.D. and Webb, M.J.: A quantitative performance assessment of cloud regimes in climate models. *Clim. Dyn.* 33, 141-157, doi: 10.1007/s00382-008-0443-1, 2009.

14.4.7 Example plots

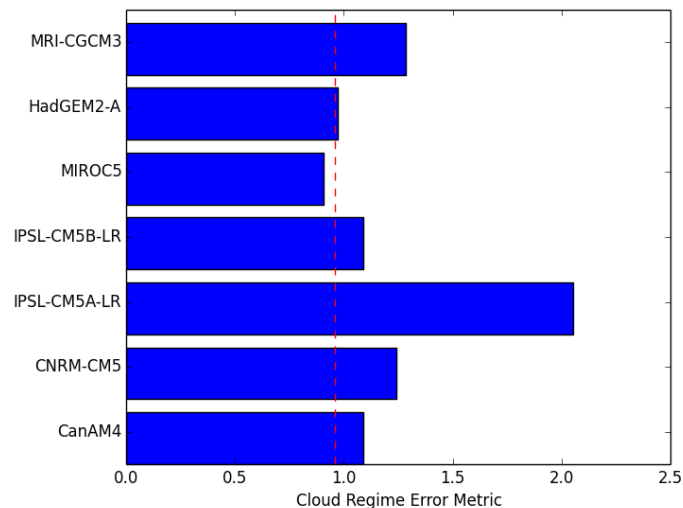


Fig. 12: Cloud Regime Error Metrics (CREMpd) from William and Webb (2009) applied to selected CMIP5 AMIP simulations. A perfect score with respect to ISCCP is zero; the dashed red line is an indication of observational uncertainty. Note: as daily sea ice concentration (sic) is not available for all models shown, the regridded fields from CanAM4 have been used for all models.

14.5 Combined Climate Extreme Index

14.5.1 Overview

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

In `recipe_extreme_index.yml`, after defining the area and reference and projection period, the weights 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.

14.5.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_extreme_index.yml`

Diagnostics are stored in `diag_scripts/magic_bsc/`

- `extreme_index.R`

14.5.3 User settings

User setting files are stored in `recipes/`

1. `recipe_extreme_index.yml`

Required settings for script

- `weight_t90p`: 0.2 (from 0 to 1, the total sum of the weight should be 1)
- `weight_t10p`: 0.2 (from 0 to 1, the total sum of the weight should be 1)
- `weight_Wx`: 0.2 (from 0 to 1, the total sum of the weight should be 1)
- `weight_rx5day`: 0.2 (from 0 to 1, the total sum of the weight should be 1)
- `weight_cdd`: 0.2 (from 0 to 1, the total sum of the weight should be 1)
- `running_mean`: 5 (depends on the length of the future projection period selected, but recommended not greater than 11)

14.5.4 Variables

- tasmax (atmos, daily, longitude, latitude, time)
- tasmin (atmos, daily, longitude, latitude, time)
- sfcWind (atmos, daily, longitude, latitude, time)
- pr (atmos, daily, longitude, latitude, time)

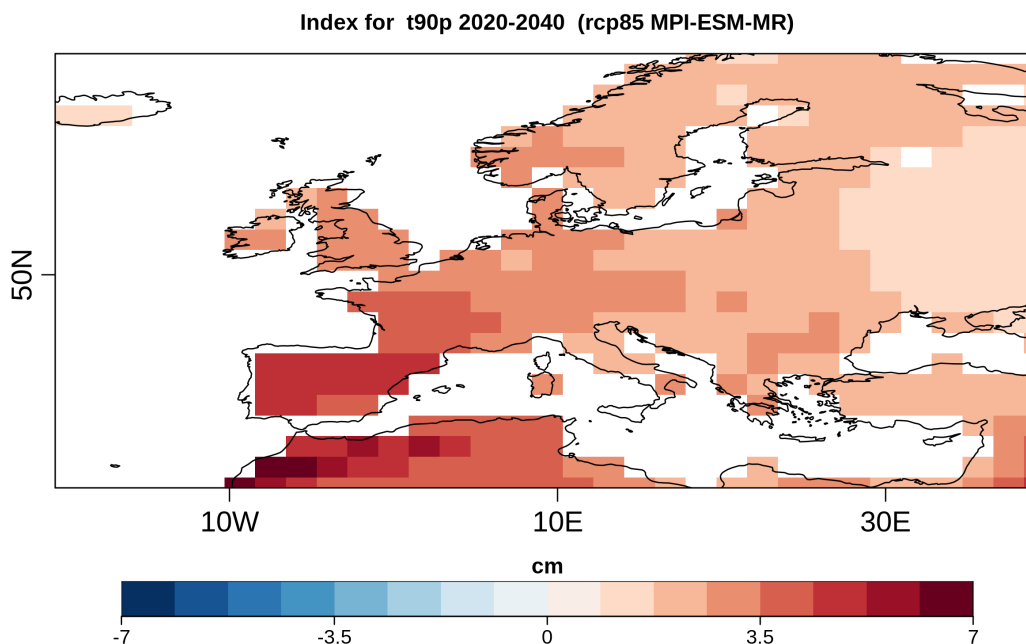
14.5.5 Observations and reformat scripts

None

14.5.6 References

- Alexander L.V. and Coauthors (2006). Global observed changes in daily climate extremes of temperature and precipitation. *J. Geophys. Res.*, 111, D05109. <https://doi.org/10.1029/2005JD006290>
- American Academy of Actuaries, Canadian Institute of Actuaries, Casualty Actuarial Society and Society of Actuaries. Actuaries Climate Index. <http://actuariesclimateindex.org> (2018-10-06).
- Donat, M., and Coauthors (2013). Updated analyses of temperature and precipitation extreme indices since the beginning of the twentieth century: The HadEX2 dataset. *J. Geophys. Res.*, 118, 2098–2118, <https://doi.org/10.1002/jgrd.50150>.
- Fouillet, A., Rey, G., Laurent, F., Pavillon, G., Bellec, S., Guihenneuc-Jouyaux, C., Clavel J., Jouglu, 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 in extreme 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>

14.5.7 Example plots



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.

14.6 Consecutive dry days

14.6.1 Overview

Meteorological drought can in its simplest form be described by a lack of precipitation. First, a wet day threshold is set, which can be either a limit related to measurement accuracy, or more directly 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.

14.6.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_consecdrydays.yml`

Diagnostics are stored in `diag_scripts/droughtindex/`

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

14.6.3 User settings in recipe

1. Script `diag_cdd.py`

Required settings (script)

- `plim`: limit for a day to be considered dry [mm/day]
- `frlim`: the shortest number of consecutive dry days for entering statistic on frequency of dry periods.

Optional settings (script)

Under `plot`:

- `cmap`: the name of a colormap. cmoceanc colormaps are also supported.
- other keyword arguments to `esmvaltool.diag_scripts.shared.plot.global_pcolormesh()` can also be supplied.

14.6.4 Variables

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

14.6.5 Example plots

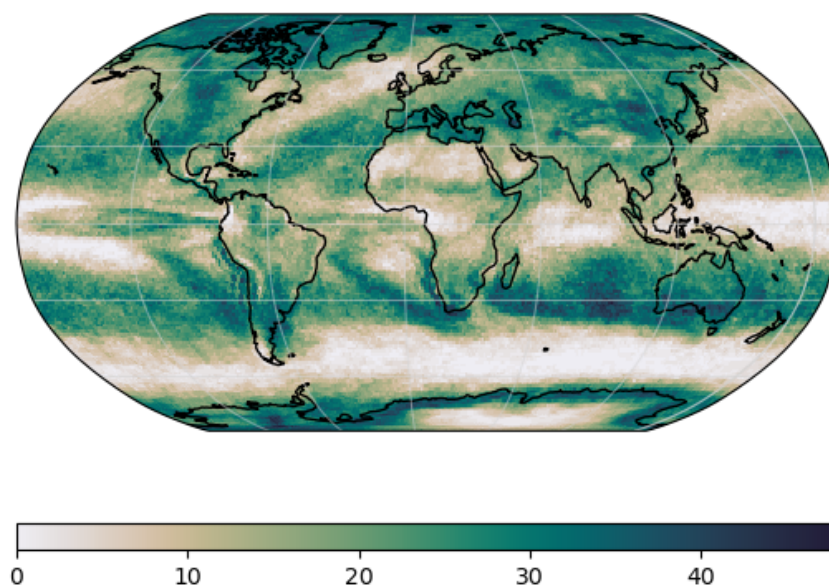


Fig. 13: Example of the number of occurrences with consecutive dry days of more than five days in the period 2001 to 2002 for the CMIP5 model bcc-csm1-1-m.

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

14.7.1 Overview

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

14.7.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_deangelis15nat.yml`

Diagnostics are stored in `diag_scripts/`

- `deangelis15nat/deangelisf1b.py`
- `deangelis15nat/deangelisf2ext.py`
- `deangelis15nat/deangelisf3f4.py`

14.7.3 User settings in recipe

The recipe can be run with different CMIP5 and CMIP6 models. `deangelisf1b.py`: Several flux variables (W m^{-2}) and up to 6 different model experiments can be handled. 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.

14.7.4 Variables

`deangelisf1b.py`: Tested for:

- *rsnst* (atmos, monthly, longitude, latitude, time)
- *rlnst* (atmos, monthly, longitude, latitude, time)
- *lvp* (atmos, monthly, longitude, latitude, time)
- *hfss* (atmos, monthly, longitude, latitude, time)

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

`deangelisf2ext.py`:

- *rsnst* (atmos, monthly, longitude, latitude, time)
- *rlnst* (atmos, monthly, longitude, latitude, time)
- *rsnstcs* (atmos, monthly, longitude, latitude, time)
- *rlnstcs* (atmos, monthly, longitude, latitude, time)

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

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

14.7.5 Observations and reformat scripts

deangelisf1b.py: * None

deangelisf2ext.py: * None

deangelisf3f4.py:

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

14.7.6 References

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

14.7.7 Example plots

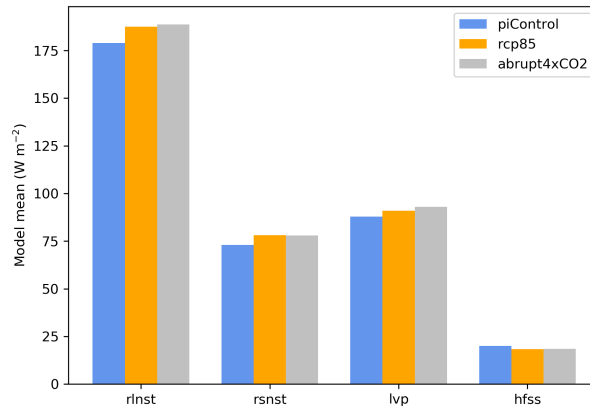


Fig. 14: Global average multi-model mean comparing different model experiments for the sum of upward long wave flux at TOA and net downward long wave flux at the surface (rlnst), heating from short wave absorption (rsnst), latent heat release from precipitation (lvp), and sensible heat flux (hfss). The 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 CO₂ scenario averaged over the years 141-150 after CO₂ quadrupling in all models except CNRM-CM5-2 and IPSL-CM5A-MR, where the average is calculated over the years 131-140 (gray). The figure shows that energy sources and sinks readjust in reply to an increase in greenhouse gases, leading to a decrease in the sensible heat flux and an increase in the other fluxes.

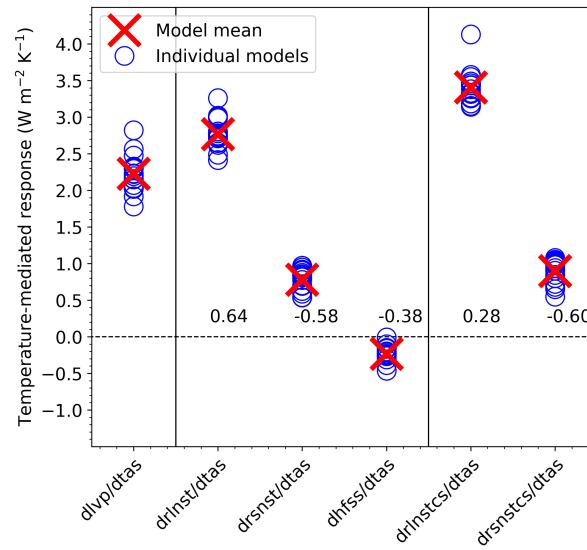


Fig. 15: The temperature-mediated response of each atmospheric energy budget term for each model as blue circles and the model mean as a red cross. The numbers above the abscissa are the cross-model correlations between $dlvp/dtas$ and each other temperature-mediated response.'

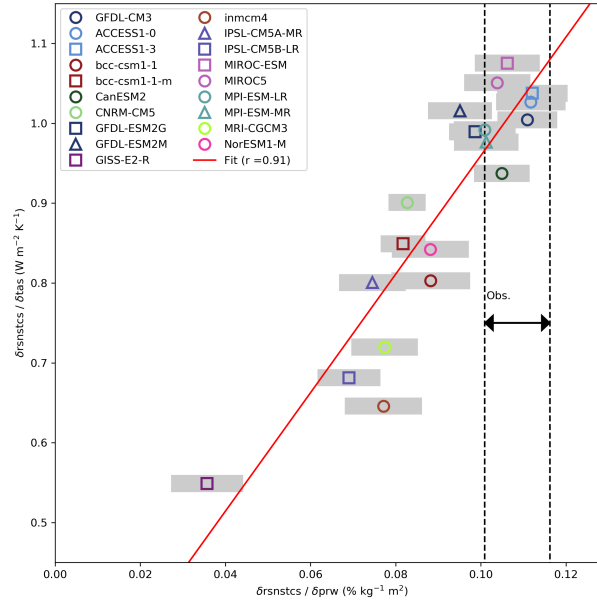


Fig. 16: Scatter plot and regression line the between the ratio of the change of net short wave radiation (rsnst) and the change of the Water Vapor Path (prw) against the ratio of the change of netshort wave radiation for clear sky (rsnstcs) and the the change of surface temperature (tas). The width 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.

14.8 Diurnal temperature range

14.8.1 Overview

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

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

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

14.8.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_diurnal_temperature_index.yml`

Diagnostics are stored in `diag_scripts/magic_bsc/`

- `diurnal_temp_index.R` : calculates the diurnal temperature vulnerability index.

14.8.3 User settings

User setting files are stored in `recipes/`

1. `recipe_diurnal_temperature_index.yml`

Required settings for script

- None

14.8.4 Variables

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

14.8.5 Observations and reformat scripts

None

14.8.6 References

- Amiri, S. (2013). Economic and Environmental Benefits of CHP-based District Heating Systems in Sweden. Retrieved from <http://www.sgc.se/ckfinder/userfiles/files/sokmotor/LiU67.pdf>
- Braganza, K., Karoly, D. J., & Arblaster, J. M. (2004). Diurnal temperature range as an index of global climate change during the twentieth century. *Geophysical Research Letters*, 31(13), n/a – n/a. <https://doi.org/10.1029/2004GL019998>
- Déandreis, C. (IPSL), Braconnot, P. (IPSL), 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>

14.8.7 Example plots

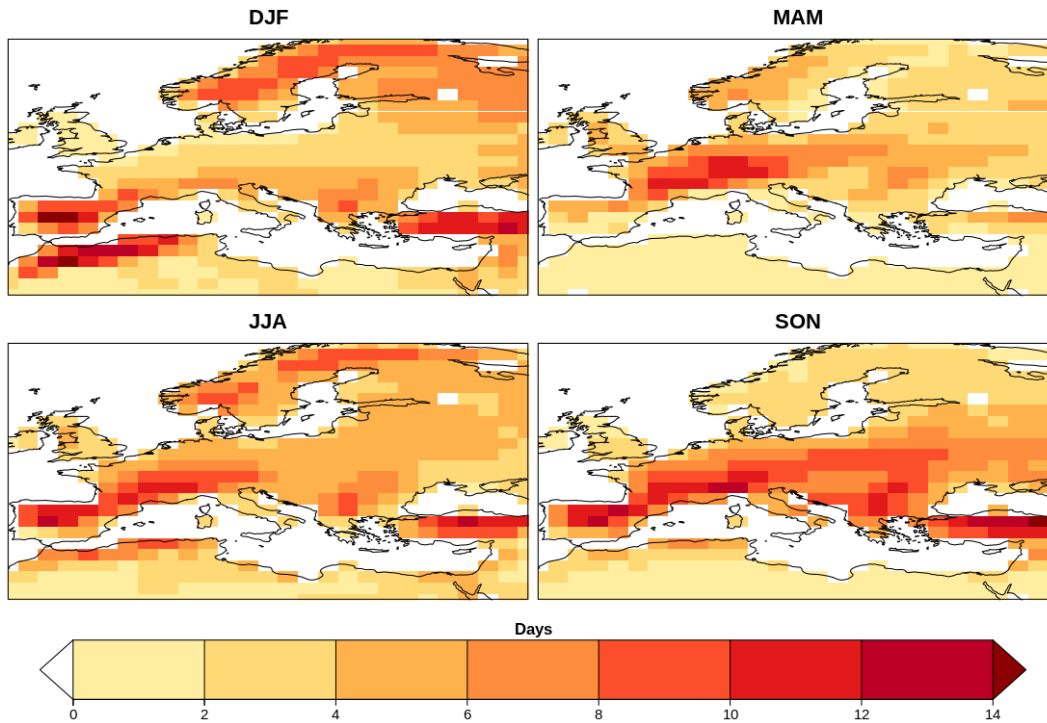
Mean number of days exceeding the Diurnal Temperature Range (DTR) simulated during the historical period (1961-1990) by 5 degrees during the period 2030-2080. The result is derived from one RCP 8.5 scenario simulated by MPI-ESM-MR.

14.9 Eady growth rate

14.9.1 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.

Number of days exceeding the DTR by 5 degrees during the period 2030 - 2080



14.9.2 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.

14.9.3 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.

14.9.4 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)

14.9.5 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.

14.9.6 Example plots

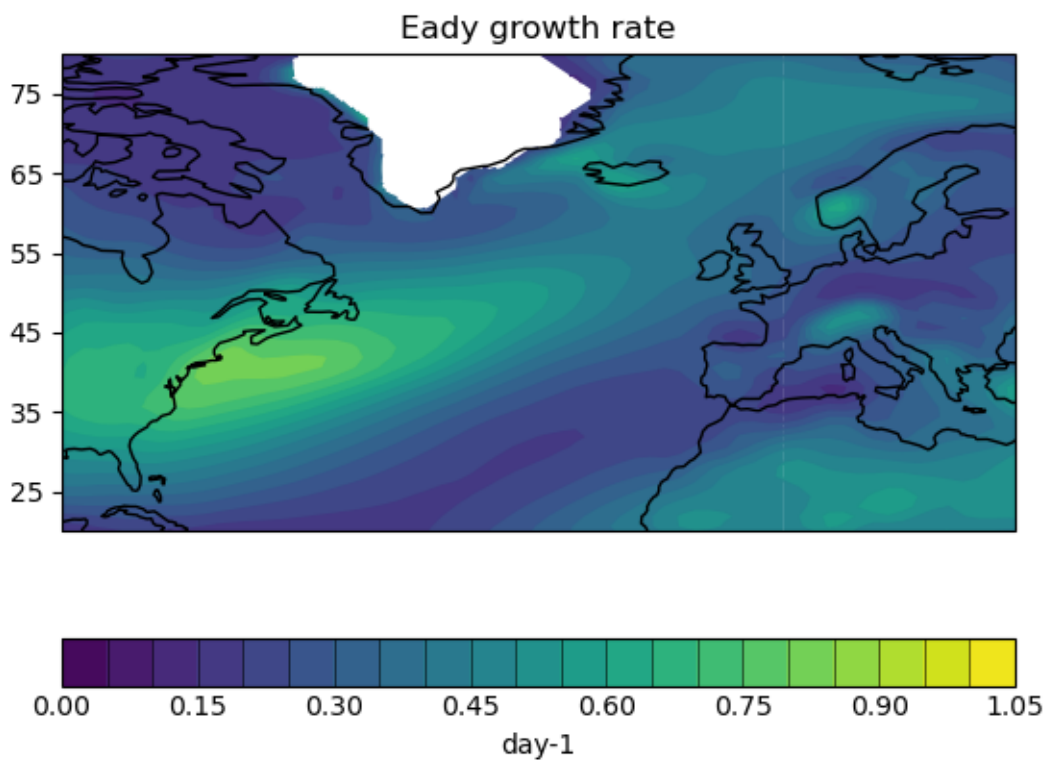


Fig. 17: Eady Growth Rate values over the North-Atlantic region at 70000 Pa.

14.10 Extreme Events Indices (ETCCDI)

14.10.1 Overview

This diagnostic uses the standard `climindex.pcic.ncdf` R library to compute the 27 climate change indices specified by the joint CCI/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.

14.10.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_extreme_events.yml`

Diagnostics are stored in `diag_scripts/extreme_events/`

- `ExtremeEvents.r`

and subroutines

- `common_climindex_preprocessing_for_plots.r`
- `make_Gleckler_plot2.r`
- `make_timeseries_plot.r`
- `cfg_climindex.r`
- `cfg_extreme.r`

14.10.3 User settings

Required settings for script

- `reference_datasets`: list containing the reference datasets to compare with
- `timeseries_idx`: list of indices to compute for timeseries plot. The syntax is “XXXETCCDI_TT”, where “TT” can be either “yr” or “mon” (yearly or monthly indices are computed) and “XXX” can be one of the following: “altcdd”, “altcsdi”, “altcwi”, “altwsdi”, “cdd”, “csdi”, “cwi”, “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 climindex (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 time-series plot. When this option is used the data for the following indices are detrended and normalized in the time-series plots: "altcdd", "altcsdi", "altcwg", "altwsdi", "cdd", "cwg", "dtr", "fd", "gsl", "id", "preptot", "r10mm", "r1mm", "r20mm", "r95p", "r99p", "rx1day", "rx5day", "sdii", "su", "tnn", "tnx", "tr", "txn", "txp", "txx" (default: false)

Additional optional setting controlling the plots:

- Timeseries plots:
 - `ts_png_width`: width for png figures (default: 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_rmsspacer`: 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 position of coloured colorbar (default: 0.05)
 - `gl_xscale_spacer_rmsestd`: horizontal position 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)

14.10.4 Variables

- `tas` (atmos, daily mean, longitude latitude time)
- `tasmin` (atmos, daily minimum, longitude latitude time)
- `tasmax` (atmos, daily maximum, longitude latitude time)
- `pr` (atmos, daily mean, longitude latitude time)

14.10.5 Observations and reformat scripts

None.

14.10.6 References

- Zhang, X., Alexander, L., Hegerl, G. C., Jones, P., Klein Tank, A., Peterson, T. C., Trewin, B., Zwiers, F. W., Indices for monitoring changes in extremes based on daily temperature and precipitation data, *WIREs Clim. Change*, doi:10.1002/wcc.147, 2011
- Sillmann, J., V. V. Kharin, X. Zhang, and F. W. Zwiers, Climate extreme indices in the CMIP5 multi-model ensemble. Part 1: Model evaluation in the present climate. *J. Geophys. Res.*, doi:10.1029/2012JD018390, 2013

14.10.7 Example plots

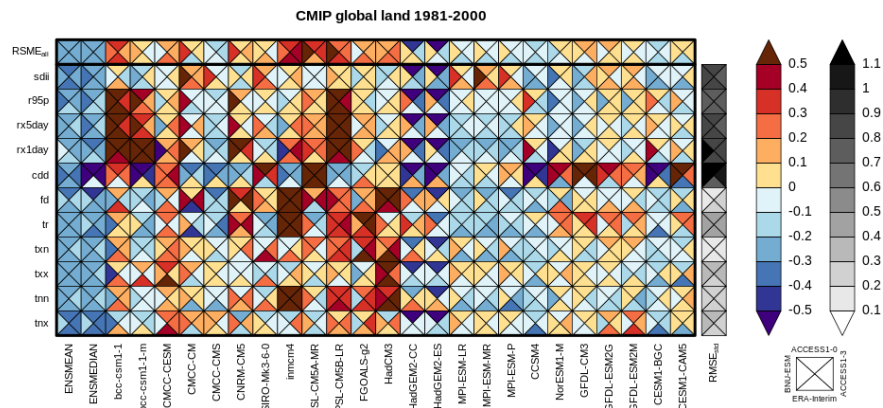


Fig. 18: Portrait plot of relative error metrics for the CMIP5 temperature and precipitation extreme indices evaluated over 1981-2000. Reproduces Fig. 9.37 of the IPCC AR5 report, Chapter 9.

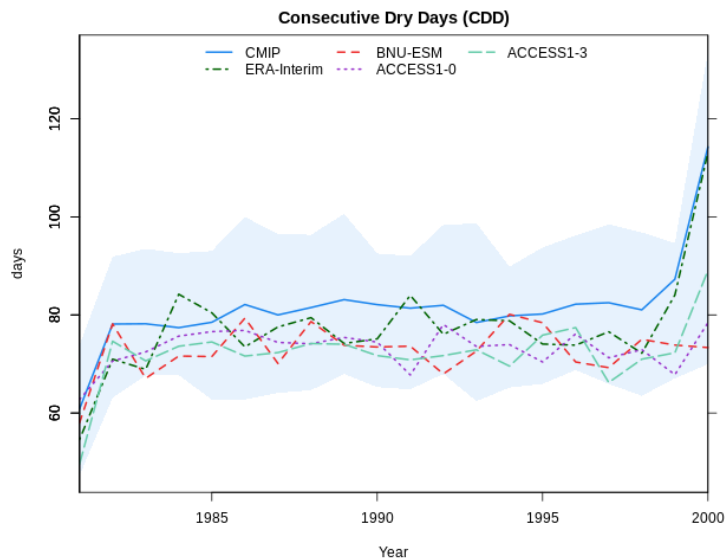


Fig. 19: Timeseries of the Consecutive Dry Days index over 1981-2000 for a selection of CMIP5 models, the CMIP5 multi-model mean (CMIP) and ERA-Interim. Shading is used to reproduce the multi-model spread.

14.11 Diagnostics of stratospheric dynamics and chemistry

14.11.1 Overview

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

- Vertical profile climatological mean bias of climatological mean for selected seasons and latitudinal region.
- Vertical and latitudinal profile of climatological mean for selected seasons. This figure and setting is valid for figure 5 (CH₄) figure 6 (H₂O) figure 11 (HCL) figure 13 (tro3).
- Total ozone anomalies at different latitudinal band and seasons.

14.11.2 Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/`

- `recipe_eyring06jgr.yml`

Diagnostics are stored in `esmvaltool/diag_scripts/eyring06jgr/`

- `eyring06jgr_fig01.ncl`
- `eyring06jgr_fig05a.ncl`
- `eyring06jgr_fig05b.ncl`
- `eyring06jgr_fig15.ncl`

14.11.3 User settings in recipe

1. Preprocessor

- **regrid_interp_lev_zonal:** Regridding and interpolation reference_dataset levels used by eyring06jgr_fig01 and eyring06jgr_fig05
- **zonal :** Regridding and zonal mean used by eyring06jgr_fig15

2. Script <eyring06jgr_fig01.ncl>

Required settings for script

- **latmin:** array of float, min lat where variable is averaged, i.e. [60., 60., -90., -90.]
- **latmax:** array of float, and max lat where variable is averaged, i.e. [90., 90., -60., -60.]
- **season:** array of string., season when variable is averaged, i.e. ["DJF", "MAM", "JJA", "SON"]
- **XMin:** array of float, min limit X axis [-30., -30., -30., -30.]
- **XMax:** array of float, max limit X axis [20., 20., 20., 20.]
- **levmin:** array of float, min limit Y axis [1., 1., 1., 1.]
- **levmax:** array of float, max limit Y axis [350., 350., 350., 350.]

Optional settings for script

- **start_year:** int, year when start the climatology calculation [1980] (default max among the models start year).
- **end_year:** int, year when end the climatology calculation [1999] (default min among the models end year).
- **multimean:** bool, calculate multi-model mean, (i.e. False/True) (default False).

Required settings for variables

- **preprocessor:** regrid_interp_lev.
- **reference_dataset:** name of the reference model or observation for regridding and bias calculation (e.g. ERA-Interim").
- **mip:** Amon.

14.11.4 Variables

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

14.11.5 Example plots

14.12 Ozone and associated climate impacts

14.12.1 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

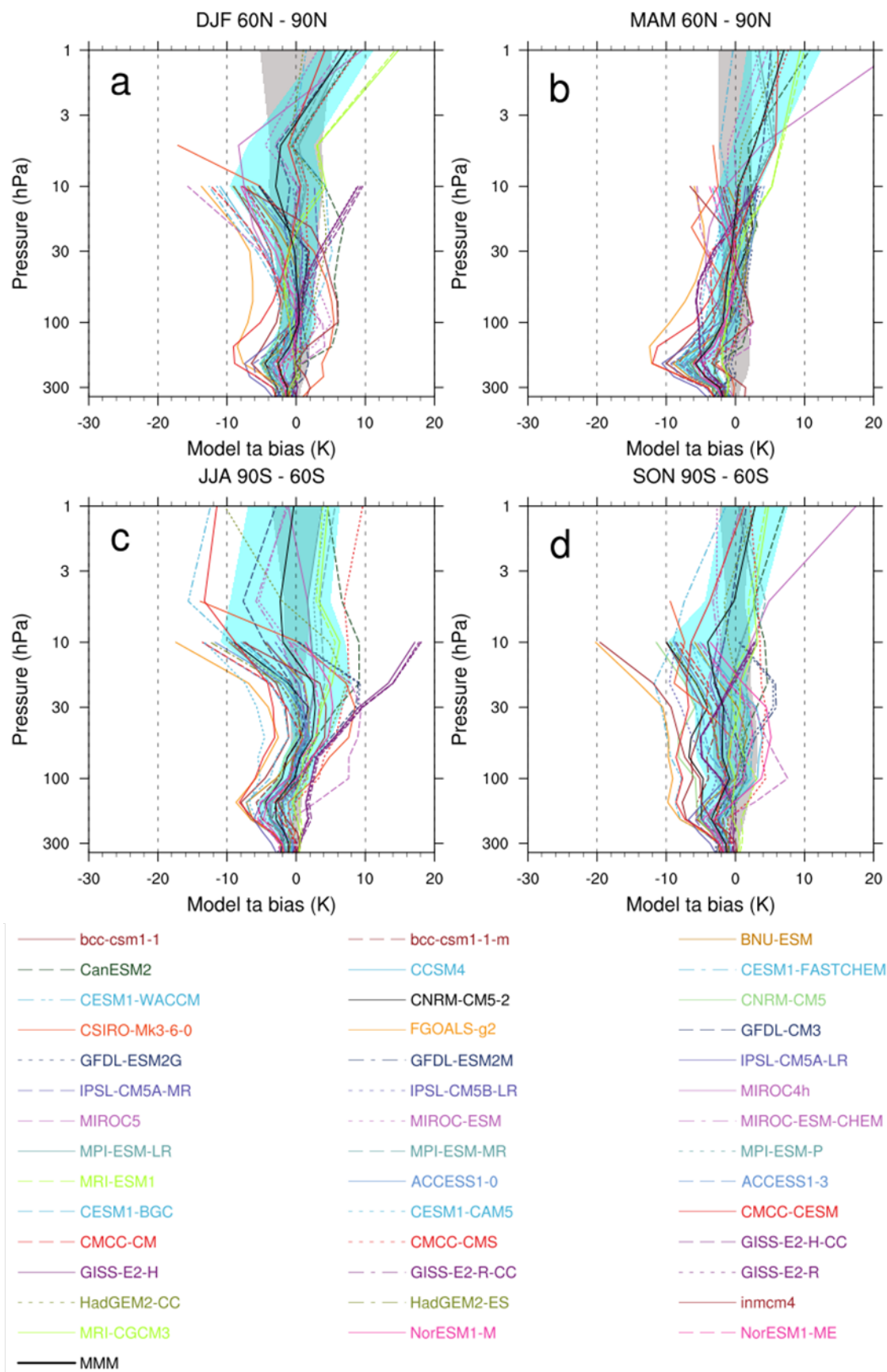


Fig. 20: Climatological mean temperature biases for (top) 60–90N and (bottom) 60–90S for the (left) winter and (right) spring seasons. The climatological means for the CCMs and ERA-Interim data from 1980 to 1999 are included. Biases are calculated relative to ERA-Interim reanalyses. The grey area shows 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.

14.12.2 Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/

- recipe_eyring13jgr_12.yml

Diagnostics are stored in esmvaltool/diag_scripts/eyring13jgr/

- eyring13jgr_fig12.ncl

14.12.3 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.

14.12.4 Variables

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

14.12.5 Observations and reformat scripts

- ERA5 *Reformatting with*: recipes/cmorizers/recipe_era5.yml

14.12.6 Example plots

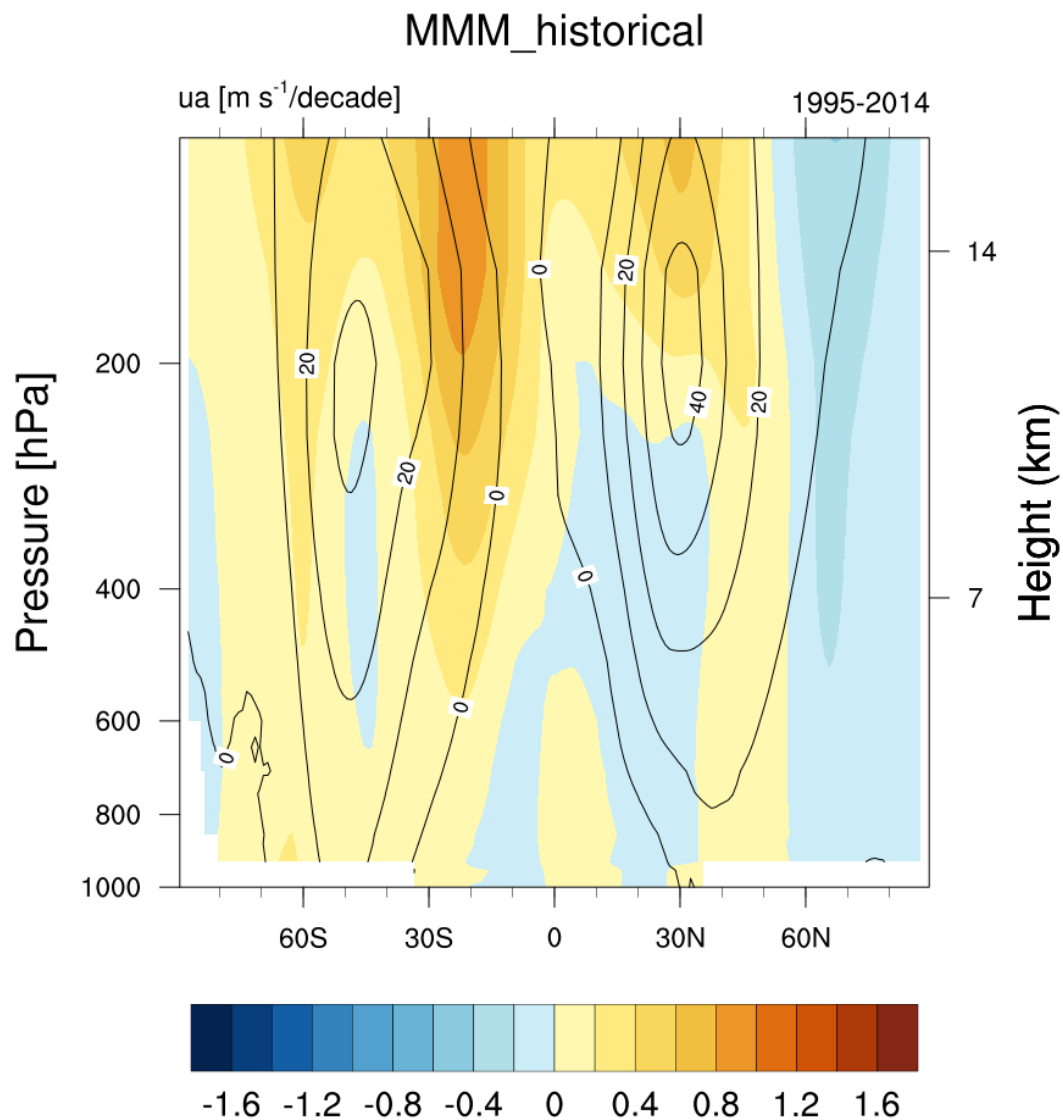


Fig. 21: Long-term mean (thin black contour) and linear trend (colour) of zonal mean DJF zonal winds for the multi-model mean CMIP6 over 1995-2014

14.13 Spatially resolved evaluation of ESMs with satellite column-averaged CO₂

14.13.1 Overview

This recipe reproduces the figures of Gier et al. (2020). It uses satellite column-averaged CO₂ data to evaluate ESMs by plotting several quantities such as timeseries, seasonal cycle and growth rate in different areas.

14.13.2 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

14.13.3 User settings in recipe

1. Preprocessor

- `conv_units`: converts units to plot-units
- `mmm_ref`: calculates multi-model mean and regrid 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:**
 - `stylesheet`: stylesheet for color coding panels
 - `output_file_type`: output file type for plots, default: `config_user -> 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: `config_user -> 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: `config_user -> 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: `config_user -> 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: `config_user -> 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: `config_user -> 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.

14.13.4 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)

14.13.5 Observations and reformat scripts

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

14.13.6 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.

14.13.7 Example plots

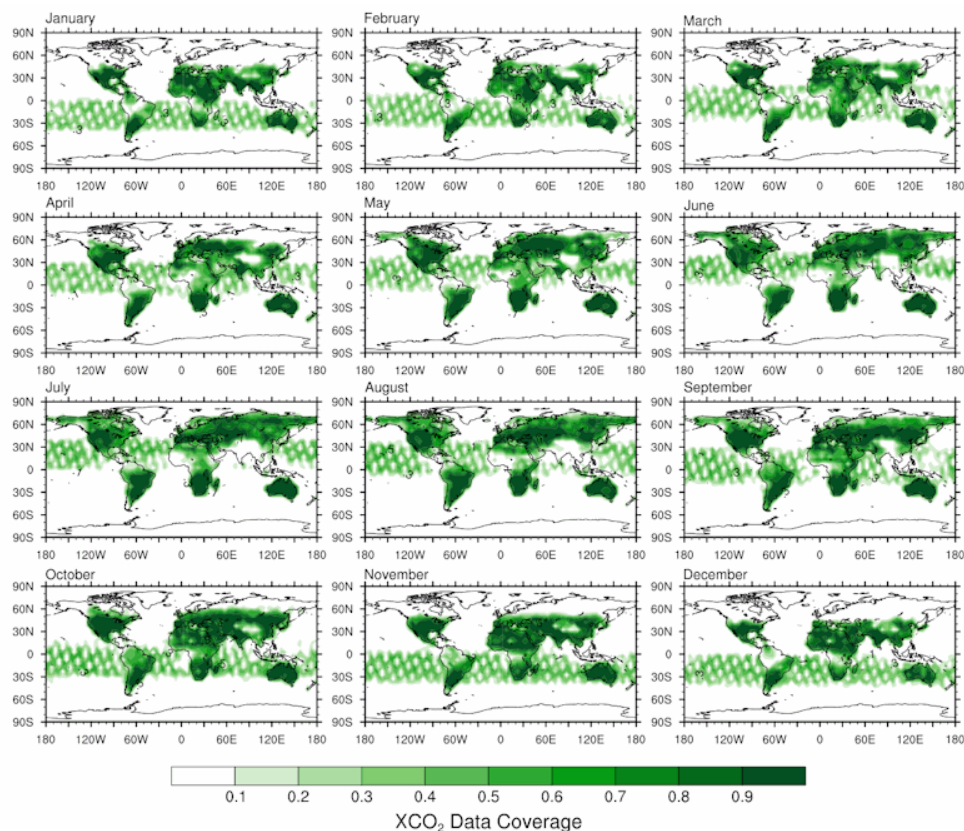


Fig. 22: Mean fractional coverage of monthly satellite data.

14.14 Heat wave and cold wave duration

14.14.1 Overview

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

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

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



Fig. 24: Timeseries with panels depicting growth rate and seasonal cycle.

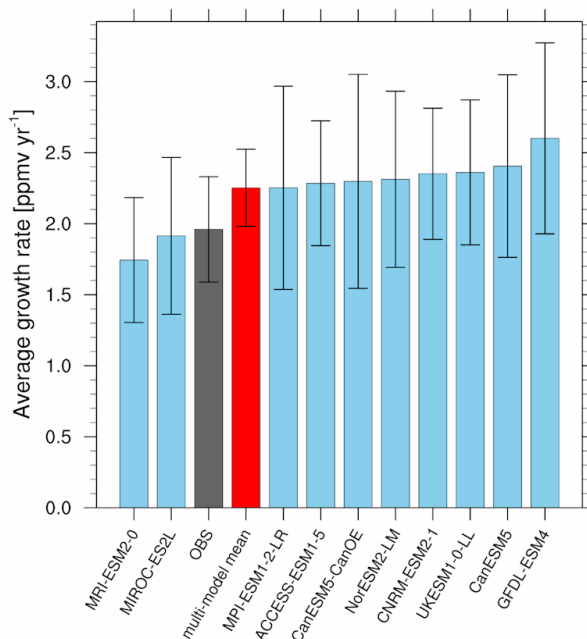


Fig. 25: Barplot of the growth rate, averaged over all years, with standard deviation of interannual variability.

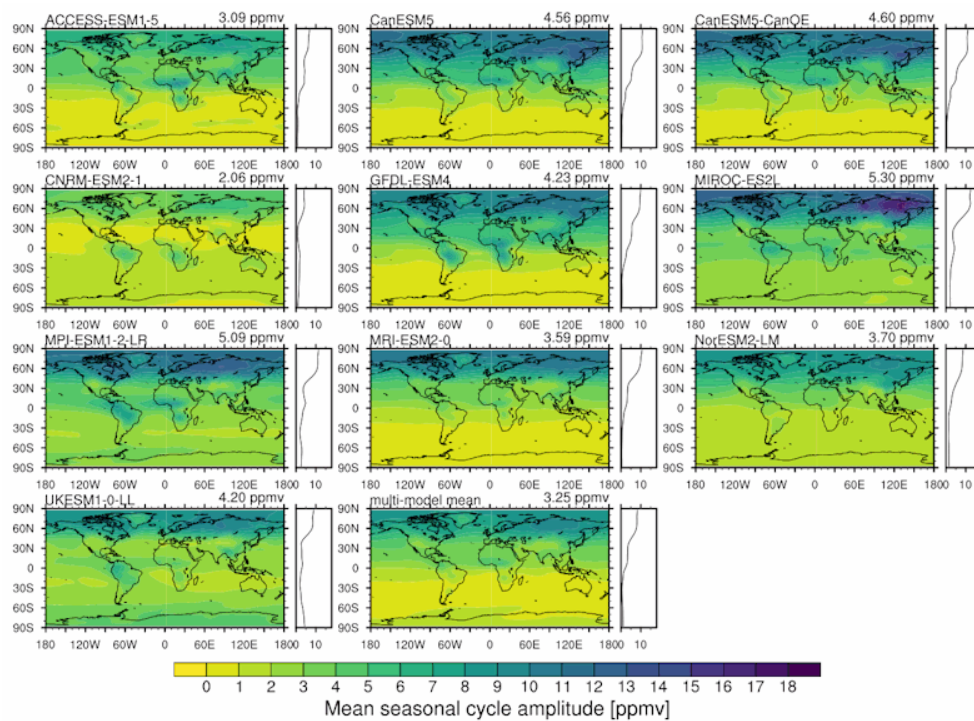


Fig. 26: Panel plot of spatially resolved seasonal cycle amplitude for all models, including a zonal average sidepanel.

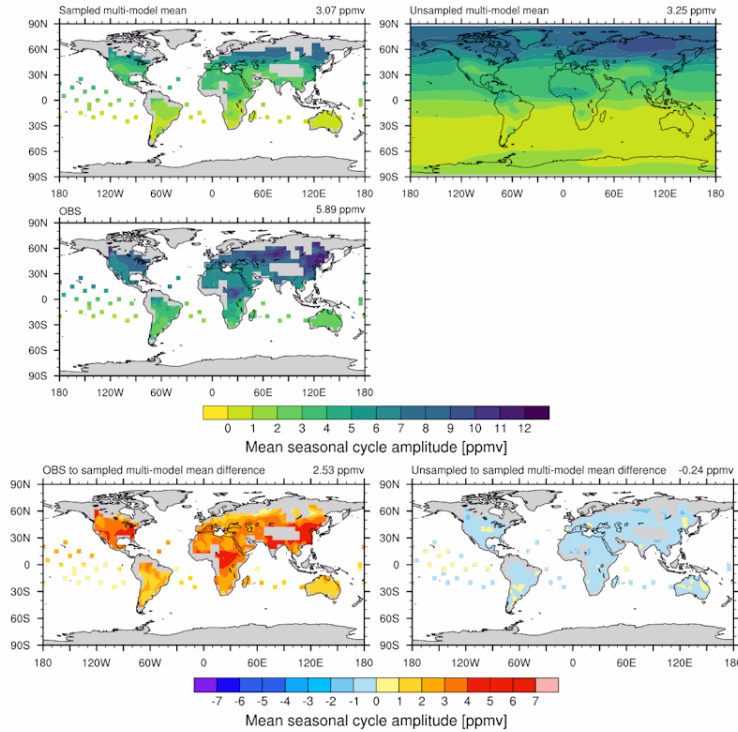


Fig. 27: Seasonal cycle amplitude map comparing influence of sampling, and difference to observations.

14.14.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_heatwaves_coldwaves.yml

Diagnostics are stored in diag_scripts/magic_bsc/

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

14.14.3 User settings

User setting files are stored in recipes/

1. recipe_heatwaves_coldwaves.yml

Required settings for script

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

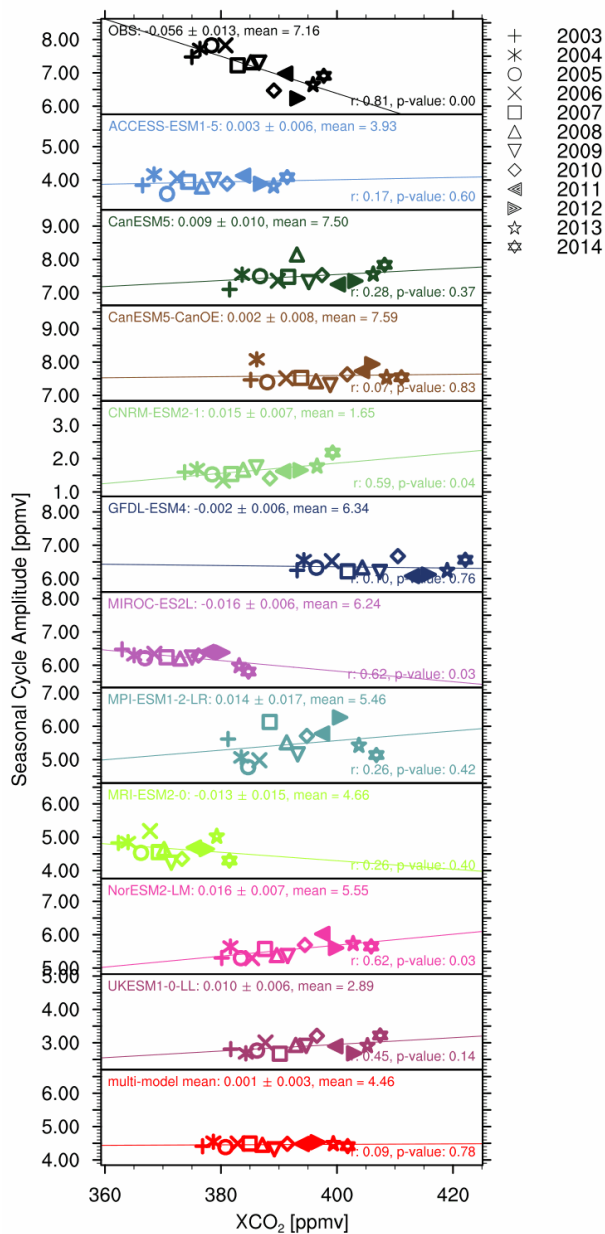


Fig. 28: Panel plots showing seasonal cycle amplitude against XCO₂, includes regression line and p-value.

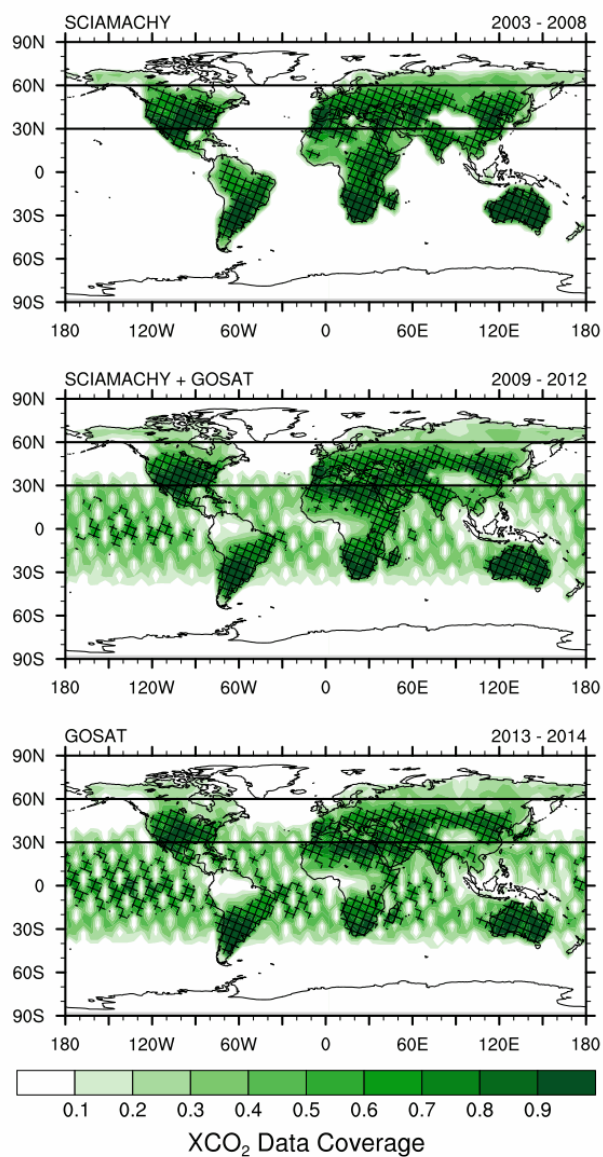


Fig. 29: Mean spatial data coverage for different satellites.

14.14.4 Variables

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

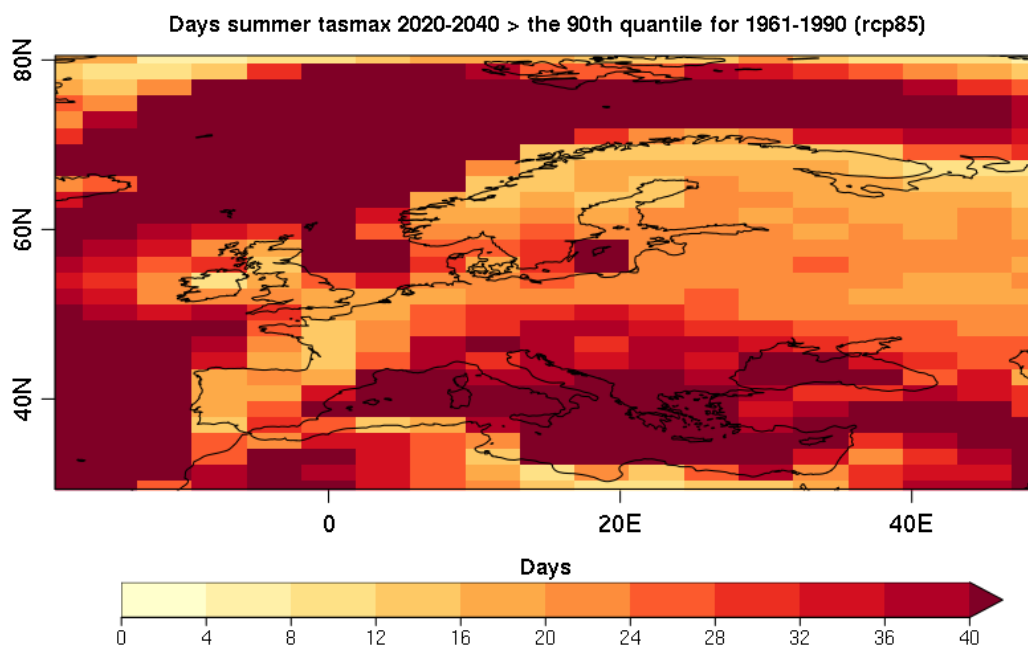
14.14.5 Observations and reformat scripts

None

14.14.6 References

- Cardoso, S., Marta-Almeida, M., Carvalho, A.C., & Rocha, A. (2017). Heat wave and cold spell changes in Iberia for a future climate scenario. *International Journal of Climatology*, 37(15), 5192-5205. <https://doi.org/10.1002/joc.5158>
- Ouzeau, G., Soubeyroux, J.-M., Schneider, M., Vautard, R., & Planton, S. (2016). Heat waves analysis over France in present and future climate: Application of a new method on the EURO-CORDEX ensemble. *Climate Services*, 4, 1-12. <https://doi.org/10.1016/J.CLISER.2016.09.002>
- Wang, Y., Shi, L., Zanobetti, A., & Schwartz, J. D. (2016). Estimating and projecting the effect of cold waves on mortality in 209 US cities. *Environment International*, 94, 141-149. <https://doi.org/10.1016/j.envint.2016.05.008>
- Zhang, X., Hegerl, G., Zwiers, F. W., & Kenyon, J. (2005). Avoiding inhomogeneity in percentile-based indices of temperature extremes. *Journal of Climate*, 18(11), 1641-1651. <https://doi.org/10.1175/JCLI3366.1>

14.14.7 Example plots



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

14.15 Hydroclimatic intensity and extremes (HyInt)

14.15.1 Overview

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

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.

14.15.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_hyint.yml` (evaluating the 6 hydroclimatic indices, performing trend analysis and plotting)
- `recipe_hyint_extreme_events.yml` (similar to the `recipe_hyint.yml` but with an additional call to the *Extreme Events Indices (ETCCDI)* diagnostic for calculation of ETCCDI indices and inclusion of them in the trend analysis and plotting)

Diagnostics are stored in `diag_scripts/hyint/`

- `hyint.R`

and subroutines

- `hyint_diagnostic.R`

- `hyint_functions.R`
- `hyint_parameters.R`
- `hyint_plot_trends.R`
- `hyint_etccdi_preproc.R`
- `hyint_metadata.R`
- `hyint_plot_maps.R`
- `hyint_preproc.R`
- `hyint_trends.R`

See details of the `extreme_events` diagnostics under `recipe_extreme_events.yml`.

14.15.3 Known issues

recipe_hyint_extreme_events.yml

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

- `url <- "https://cran.r-project.org/src/contrib/Archive/ncdf4.helpers/ncdf4.helpers_0.3-3.tar.gz"`
- `pkgFile <- "ncdf4.helpers_0.3-3.tar.gz"`
- `download.file(url = url, destfile = pkgFile)`
- `install.packages(pkgs=pkgFile, type="source", repos=NULL)`
- `unlink(pkgFile)`

14.15.4 User settings

Required settings for script

- `norm_years`: first and last year of reference normalization period to be used for normalized indices
- `select_indices`: indices to be analysed and plotted. Select one or more fields from the following list (order-sensitive): `"pa_norm"`, `"hyint"`, `"int_norm"`, `"r95_norm"`, `"wsl_norm"`, `"dsl_norm"`, `"int"`, `"dsl"`, `"wsl"`
- `select_regions`: Select regions for timeseries and maps from the following list: `GL=Globe`, `GL60=Global 60S/60N`, `TR=Tropics (30S/30N)`, `SA=South America`, `AF=Africa`, `NA=North America`, `IN=India`, `EU=Europe`, `EA=East-Asia`, `AU=Australia`
- `plot_type`: type of figures to be plotted. Select one or more from: 1=lon/lat maps per individual field/exp/multi-year mean, 2=lon/lat maps per individual field exp-ref-diff/multi-year mean, 3=lon/lat maps multi-field/exp-ref-diff/multi-year mean, 11=timeseries over required individual region/exp, 12=timeseries over multiple regions/exp, 13=timeseries with multiple models, 14=summary trend coefficients multiple regions, 15=summary trend coefficients multiple models

Additional settings for `recipe_hyint_extreme_events.yml`

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

- `etccdi_list_import`: specify the list of ETCCDI indices to be imported, e.g.: “tn10pETCCDI”, “tn90pETCCDI”, “tx10pETCCDI”, “tx90pETCCDI”
- `select_indices`: this required settings should here be revised to include the imported indices, e.g.: “pa_norm”, “hyint”, “tn10pETCCDI”, “tn90pETCCDI”, “tx10pETCCDI”, “tx90pETCCDI”

Optional settings for script (with default setting)

1. Data

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

2. Plotting

- `npancol` (2): number of columns in timeseries/trends multipanel figures
- `npanrow` (3): number of rows in timeseries/trends multipanel figures
- `autolevels` (true): select automated (true) or pre-set (false) range of values in plots
- `autolevels_scale` (1): factor multiplying automated range for maps and timeseries
- `autolevels_scale_t` (1.5): factor multiplying automated range for trend coefficients

3. Maps

- `oplot_grid` (false): plot grid points over maps
- `boxregion` (false): !=0 plot region boxes over global maps with thickness = abs(boxregion); white (>0) or grey (<0).
- `removedesert` (false) remove (flag as NA) grid points with mean annual pr < 0.5 mm/day (deserts, Giorgi2014). This affects timeseries and trends calculations too.

4. Timeseries and trends

- `weight_tseries` (true): adopt area weights in timeseries
- `trend_years` (false): (a) false = apply trend to all years in dataset; (b) [year1, year2] to apply trend calculation and plotting only to a limited time interval
- `add_trend` (true): add linear trend to plot
- `add_trend_sd` (false): add dashed lines of stdev range to timeseries
- `add_trend_sd_shade` (false): add shade of stdev range to timeseries
- `add_tseries_lines` (true): plot lines connecting timeseries points
- `add_zeroline` (true): plot a dashed line at y=0
- `trend_years_only` (false): limit timeseries plotting to the time interval adopted for trend calculation (excluding the normalization period)
- `scale100years` (true): plot trends scaled as 1/100 years
- `scalepercent` (false): plot trends as percent change

14.15.5 Variables

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

Additional variables for recipe_hyint_extreme_events.yml

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

14.15.6 Observations and reformat scripts

None.

14.15.7 References

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

14.15.8 Example plots

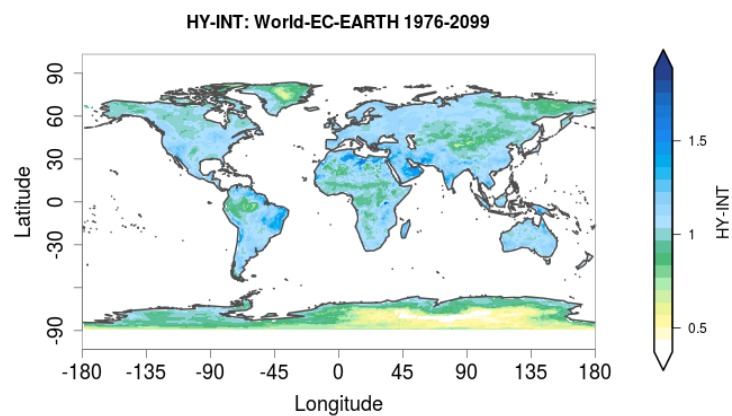


Fig. 30: Mean hydroclimatic intensity for the EC-EARTH model, for the historical + RCP8.5 projection in the period 1976-2099

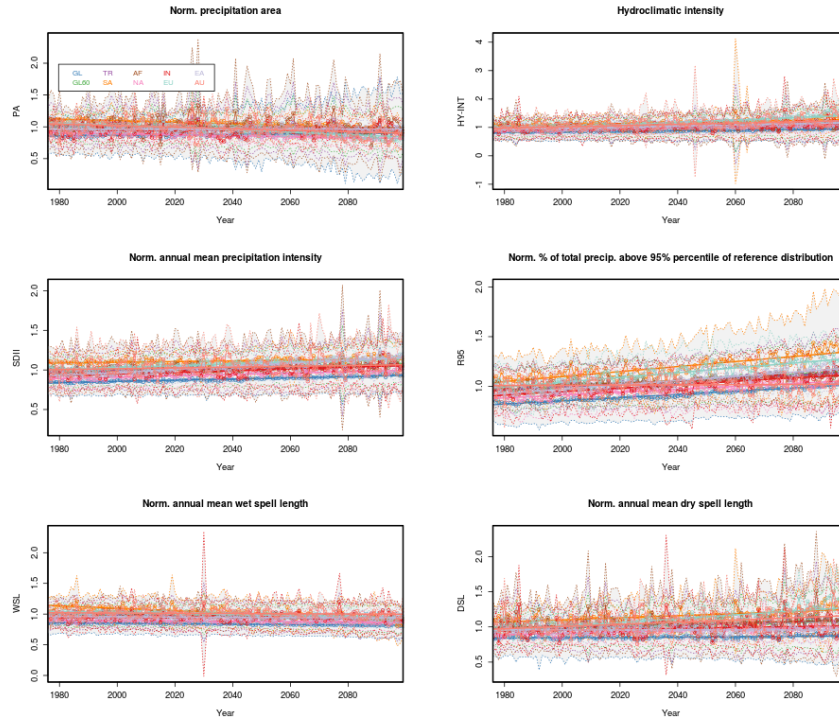


Fig. 31: Timeseries for multiple indices and regions for the ACCESS1-0 model, for the historical + RCP8.5 projection in the period 1976-2099, normalized to the 1976-2005 historical period.

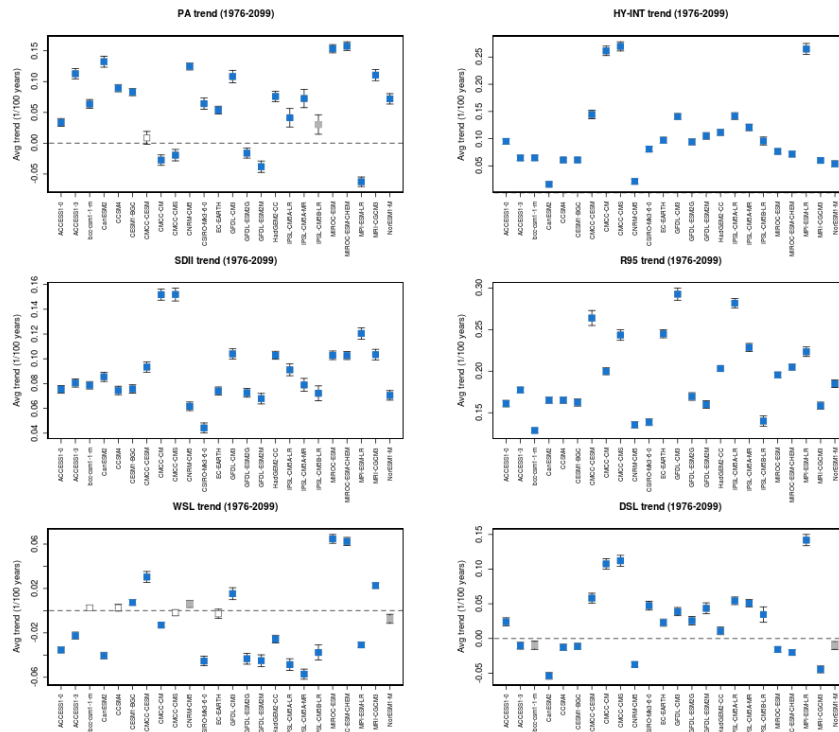


Fig. 32: Multi-model trend coefficients over selected indices for CMIP5 models in the RCP8.5 2006-2099 projection, normalized to the 1976-2005 historical period.

14.16 Quick insights for climate impact researchers

14.16.1 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.

14.16.2 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.

14.16.3 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.

14.16.4 Variables

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

14.16.5 Observations and reformat scripts

- ERA5 data can be used via the native6 project.

14.16.6 References

- None

14.16.7 Example plots

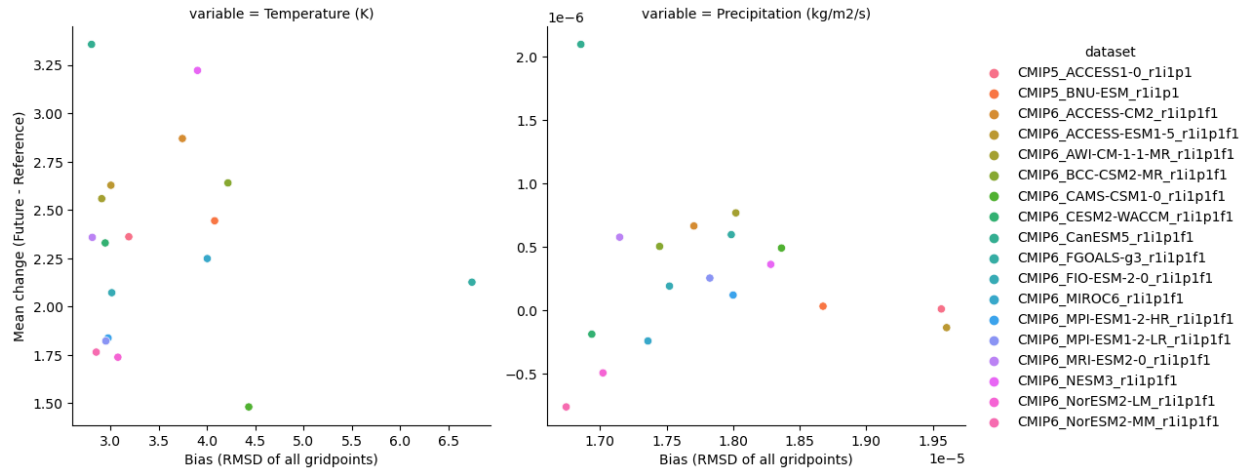


Fig. 33: “Bias and change for each variable”

14.17 Modes of variability

14.17.1 Overview

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

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

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

14.17.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_modes_of_variability.yml

Diagnostics are stored in diag_scripts/magic_bsc/

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

14.17.3 User settings

User setting files are stored in recipes/

1. recipe_modes_of_variability.yml

Required settings for script

- plot type: rectangular or polar
- ncenters: number of centers to be computed by the clustering algorithm (maximum 4)
- cluster_method: kmeans (only psl variable) or hierarchical clustering (for psl or sic variables)
- detrend_order: the order of the polynomial detrending to be applied (0, 1 or 2)
- EOFs: logical indicating whether 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).

14.17.4 Variables

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

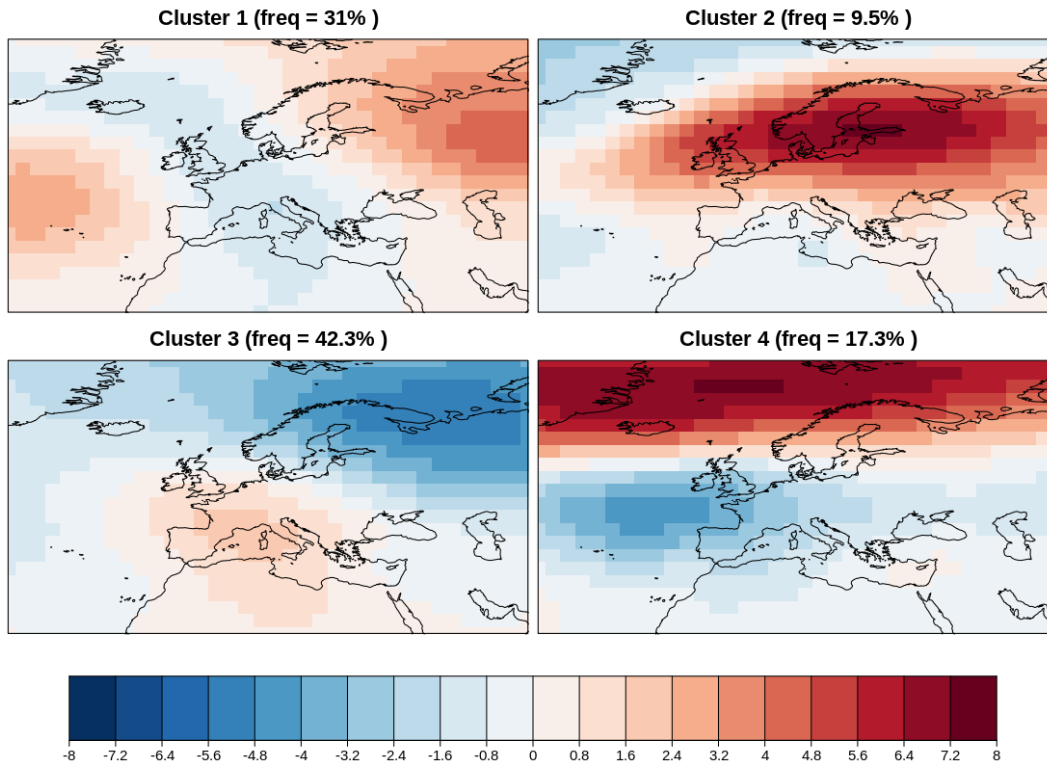
14.17.5 Observations and reformat scripts

None

14.17.6 References

- Dawson, A., T. N. Palmer, and S. Corti, 2012: Simulating regime structures in weather and climate prediction models. *Geophysical Research Letters*, 39 (21), <https://doi.org/10.1029/2012GL053284>.
- Ferranti, L., S. Corti, and M. Janousek, 2015: Flow-dependent verification of the ECMWF ensemble over the Euro-Atlantic sector. *Quarterly Journal of the Royal Meteorological Society*, 141 (688), 916-924, <https://doi.org/10.1002/qj.2411>.
- Grams, C. M., Beerli, R., Pfenninger, S., Staffell, I., & Wernli, H. (2017). Balancing Europe's wind-power output through spatial deployment informed by weather regimes. *Nature climate change*, 7(8), 557, <https://doi.org/10.1038/nclimate3338>.
- Hannachi, A., D. M. Straus, C. L. E. Franzke, S. Corti, and T. Woollings, 2017: Low Frequency Nonlinearity and Regime Behavior in the Northern Hemisphere Extra-Tropical Atmosphere. *Reviews of Geophysics*, <https://doi.org/10.1002/2015RG000509>.
- Michelangeli, P.-A., R. Vautard, and B. Legras, 1995: Weather regimes: Recurrence and quasi stationarity. *Journal of the atmospheric sciences*, 52 (8), 1237-1256, doi: 10.1175/1520-0469(1995)052<1237:WRRAS>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>.

14.17.7 Example plots



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

14.18 Diagnostics of integrated atmospheric methane (XCH4)

14.18.1 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)

14.18.2 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`

14.18.3 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.nc1`).

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.

14.18.4 User settings in recipe

1. Preprocessor

- `pp_lineplots_xx_mon`: Regridding, masking all missing values from all used datasets, area-mean ('xx' can be 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 be 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 be 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

14.18.5 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.

14.18.6 Example plots

14.19 Precipitation quantile bias

14.19.1 Overview

Precipitation is a dominant component of the hydrological cycle, and as such a main driver of the climate system and human development. The reliability of climate projections and water resources strategies therefore depends on how well precipitation can be reproduced by the models used for simulations. While global circulation models from the CMIP5 project observations can reproduce the main patterns of mean precipitation, they often show shortages and biases in the ability to reproduce the strong precipitation tails of the distribution. Most models underestimate precipitation over arid regions and overestimate it over regions of complex topography, and these shortages are amplified at high quantile precipitation. The quantilebias recipe implements calculation of the quantile bias to allow evaluation of the precipitation bias based on a user defined quantile in models as compared to a reference dataset following Mehran et al. (2014). The quantile bias (QB) is defined as the ratio of monthly precipitation amounts in each simulation to that of the

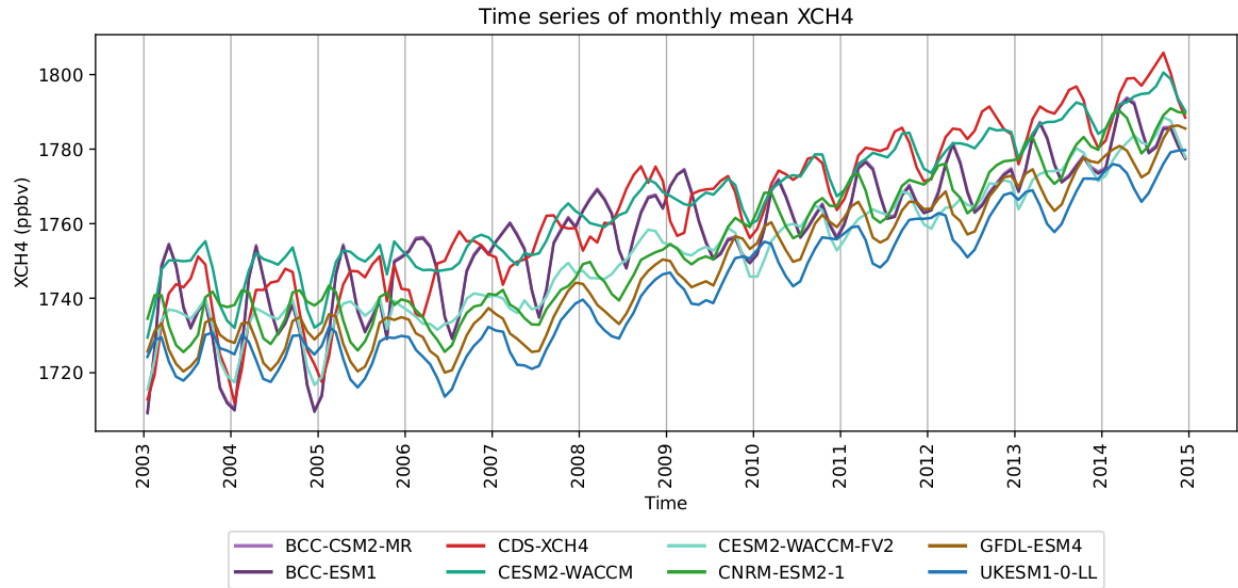


Fig. 34: Monthly mean time series of XCH₄, calculated over the whole globe, for individual CMIP6 model simulations.

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.

14.19.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_quantilebias.yml`

Diagnostics are stored in `diag_scripts/quantilebias/`

- `quantilebias.R`

14.19.3 User settings

Required settings for script

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

14.19.4 Variables

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

14.19.5 Observations and reformat scripts

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

14.19.6 References

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

14.19.7 Example plots

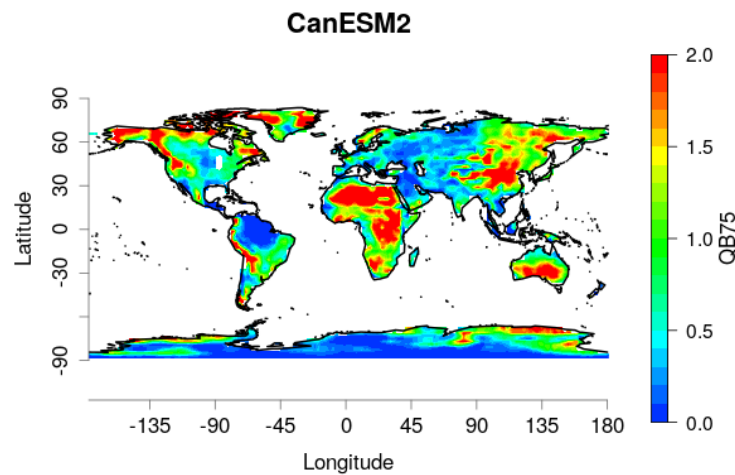


Fig. 35: Quantile bias, as defined in Mehran et al. 2014, with threshold $t=75$ th percentile, evaluated for the CanESM2 model over the 1979-2005 period, adopting GPCP-SG v 2.3 gridded precipitation as a reference dataset. The optimal reference value is 1. Both datasets have been regridded onto a 2° regular grid.

14.20 Quantifying progress across different CMIP phases

14.20.1 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.

14.20.2 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`

14.20.3 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()` (affects all plots).

14.20.4 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)

14.20.5 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 - reanalysis of surface temperature

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

- PATMOS-x - total cloud cover

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

14.20.6 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://cds.climate.copernicus.eu/cdsapp#!/home>
- 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>

14.20.7 Example plots

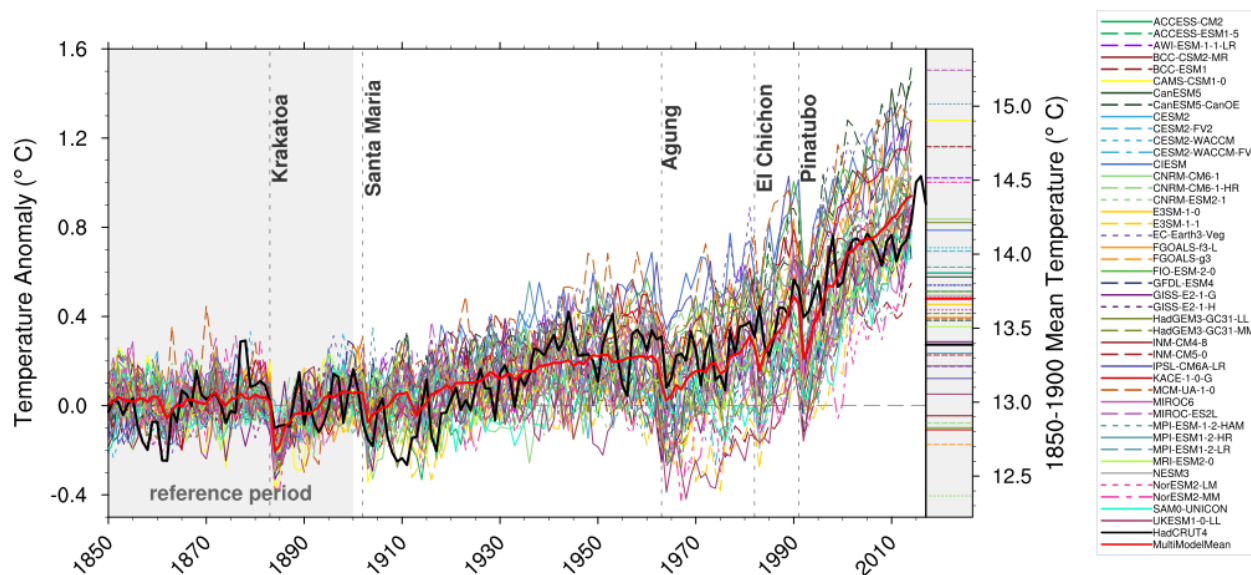


Fig. 36: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1850-1900 time mean of each individual time series (Fig. 1).

14.21 Standardized Precipitation-Evapotranspiration Index (SPEI)

14.21.1 Overview

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

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

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

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

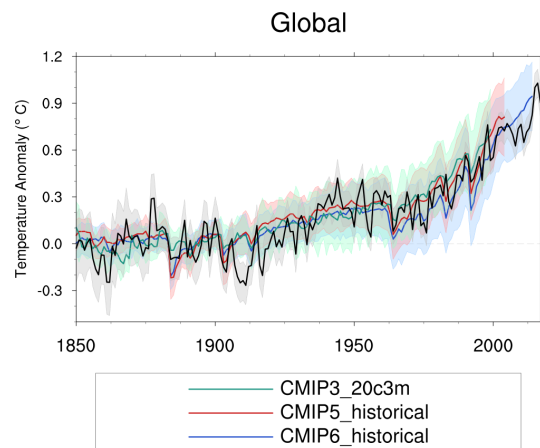


Fig. 37: Observed and simulated time series of the anomalies in annual and global mean surface temperature as in Figure 1; all anomalies are calculated by subtracting the 1850-1900 time mean from the time series. Displayed are the multimodel means of all three CMIP ensembles with shaded range of the 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).

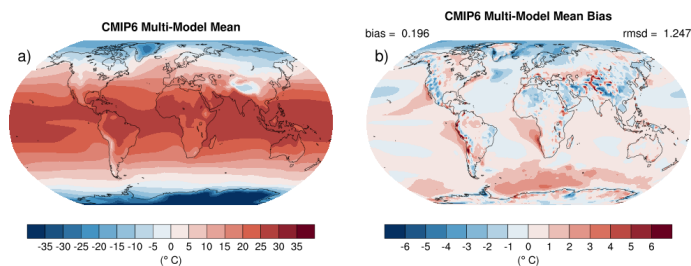


Fig. 38: Annual mean near-surface (2 m) air temperature ($^{\circ}\text{C}$). (a) Multimodel (ensemble) mean constructed with one realization of CMIP6 historical experiments for the period 1995-2014. Multimodel-mean bias of (b) CMIP6 (1995-2014) compared to the corresponding time period of the climatology from ERA5 (Copernicus Climate Change Service (C3S), 2017). (Fig. 3)

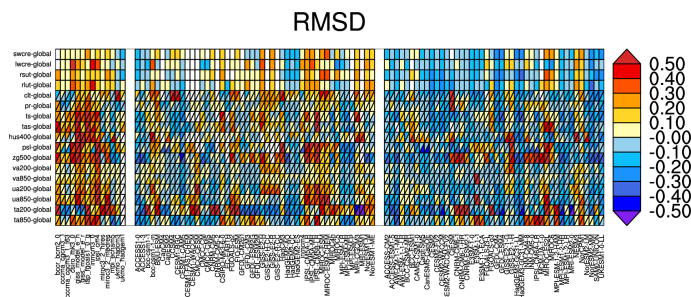


Fig. 39: Relative space-time root-mean-square deviation (RMSD) calculated from the climatological seasonal cycle of the CMIP3, CMIP5, and CMIP6 simulations (1980-1999) compared to observational data sets (Table 5). A relative performance is displayed, with blue shading being better and red shading worse 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).

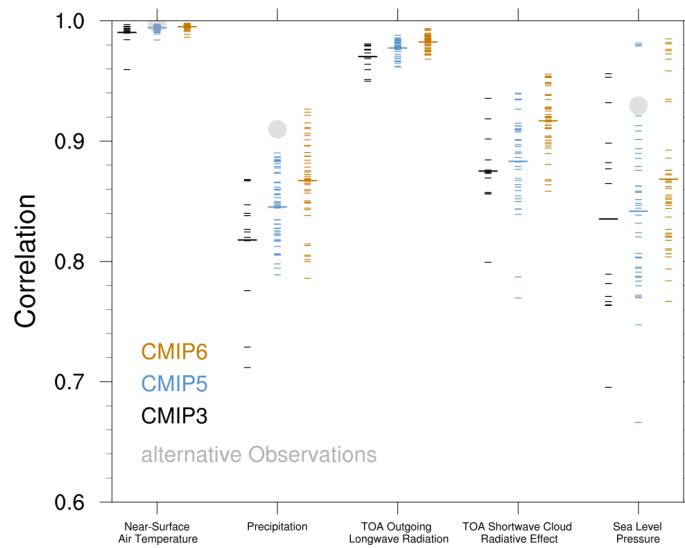


Fig. 40: Centered pattern correlations between models and observations for the annual mean climatology over the period 1980–1999 (Fig. 7).

14.21.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_spei.yml`

Diagnostics are stored in `diag_scripts/droughtindex/`

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

14.21.3 User settings

1. Script `diag_spei.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.

14.21.4 Variables

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

14.21.5 References

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

14.21.6 Example plots

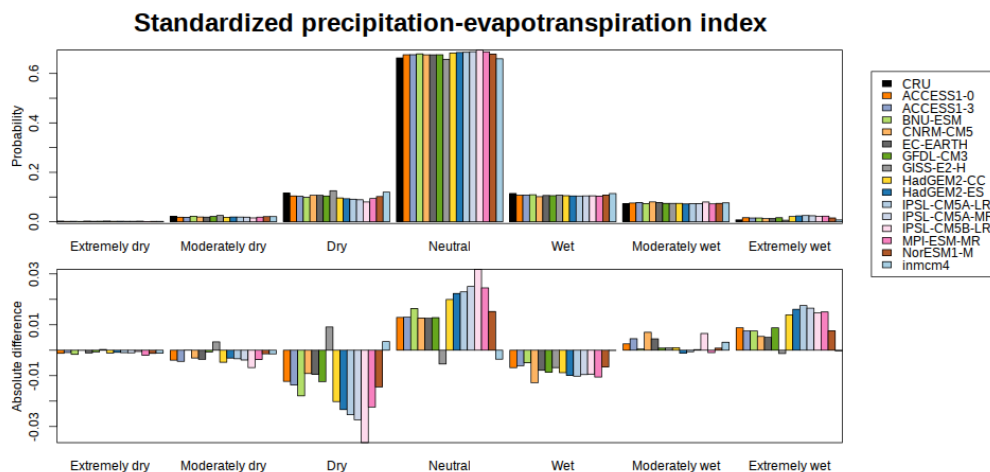


Fig. 41: (top) Probability distribution of the standardized precipitation-evapotranspiration index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.

14.22 Drought characteristics following Martin (2018)

14.22.1 Overview

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

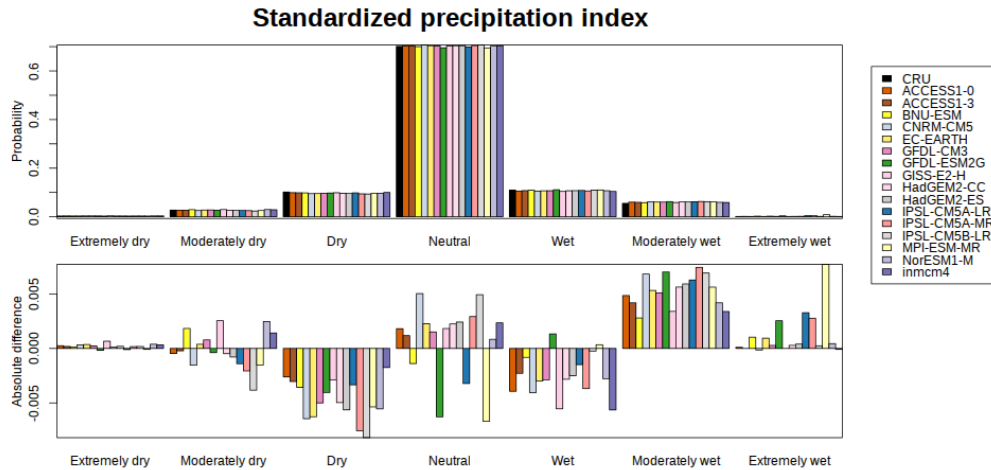


Fig. 42: (top) Probability distribution of the standardized precipitation index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.

14.22.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_martin18grl.yml`

Diagnostics are stored in `diag_scripts/`

- `droughtindex/diag_save_spi.R`
- `droughtindex/collect_drought_obs_multi.py`
- `droughtindex/collect_drought_model.py`
- `droughtindex/collect_drought_func.py`

14.22.3 User settings in recipe

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

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

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

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

by `droughtindex/diag_save_spi.R` as well as write captions and filenames. At the moment only indexname: 'SPI' is supported. * `threshold`: Threshold for this index below which an event is considered to be a drought, the setting for SPI should be usually `threshold: -2.0` but any other value will be accepted. Values should not be < -3.0 or > 3.0 for SPI (else it will identify none/always drought conditions). * `start_year`: Needs to be equal or larger than the `start_year` for `droughtindex/diag_save_spi.R`. * `end_year`: Needs to be equal or smaller than the `end_year` for `droughtindex/diag_save_spi.R`. * `comparison_period`: should be $< (\text{end_year} - \text{start_year})/2$ to have non overlapping time series in the comparison.

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

14.22.4 Variables

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

14.22.5 Observations and reformat scripts

None

14.22.6 References

- Martin, E.R. (2018). Future Projections of Global Pluvial and Drought Event Characteristics. *Geophysical Research Letters*, 45, 11913-11920.
- McKee, T. B., Doesken, N. J., & Kleist, J. (1993). The relationship of drought frequency and duration to time scales. In *Proceedings of the 8th Conference on Applied Climatology* (Vol. 17, No. 22, pp. 179-183). Boston, MA: American Meteorological Society.

14.22.7 Example plots

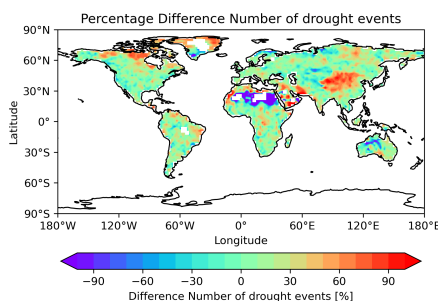


Fig. 43: Global map of the percentage difference between multi-model mean of 15 CMIP models and the CRU data for the number of drought events [%] based on SPI.

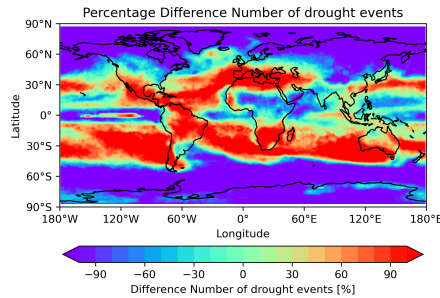


Fig. 44: Global map of the percentage difference between multi-model mean for RCP8.5 scenarios (2050-2100) runs and historical data (1950-2000) for 15 CMIP models for the number of drought events [%] based on SPI.

14.23 Stratosphere - Autoassess diagnostics

14.23.1 Overview

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

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

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

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

14.23.2 Prior and current contributors

Met Office:

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

ESMValTool:

- Since April 2018: Porting into ESMValTool by Valeriu Predoi

14.23.3 Developers

Met Office:

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

ESMValTool:

- Since April 2018: Valeriu Predoi

14.23.4 Review of current port in ESMValTool

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

14.23.5 Metrics and Diagnostics

Performance metrics:

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

Diagnostics:

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

14.23.6 Model Data

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

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

14.23.7 Inputs and usage

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

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

Note: Polar Night/Easterly Jets Metrics

Polar Night Jets (PNJ) metrics require data available at very low air pressures ie very high altitudes; both Polar Night Jet and Easterly Jets computations should be performed 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
```

(continues on next page)

(continued from previous page)

```
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
```

14.23.8 References

- Andrews, A. E., and Coauthors, 2001: Mean ages of stratospheric air derived from in situ observations of CO₂, CH₄, and N₂O. J. Geophys. Res., 106 (D23), 32295-32314.
- Dee, D. P., and Coauthors, 2011: The ERA-Interim reanalysis: configuration and performance of the data assimilation system. Q. J. R. Meteorol. Soc, 137, 553-597, doi:10.1002/qj.828.
- Engel, A., and Coauthors, 2009: Age of stratospheric air unchanged within uncertainties over the past 30 years. Nat. Geosci., 2, 28-31, doi:10.1038/NGEO388.

14.23.9 Observations Data sets

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

14.23.10 Sample Plots and metrics

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

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

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

Another reference run comparing UKESM1-0-LL to the physical model HadGEM3-GC31-LL can be found [here](#) .

14.24 Land-surface Permafrost - Autoassess diagnostics

14.24.1 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)

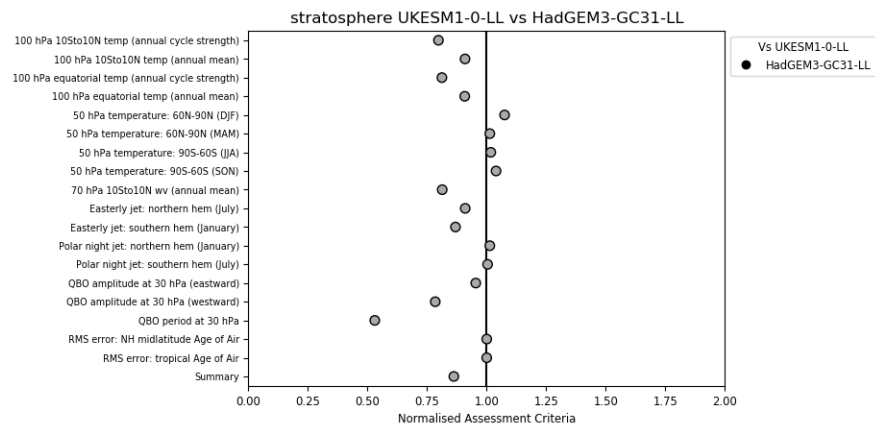


Fig. 45: Standard metrics plot comparing standard metrics from UKESM1-0-LL and HadGEM3-GC31.

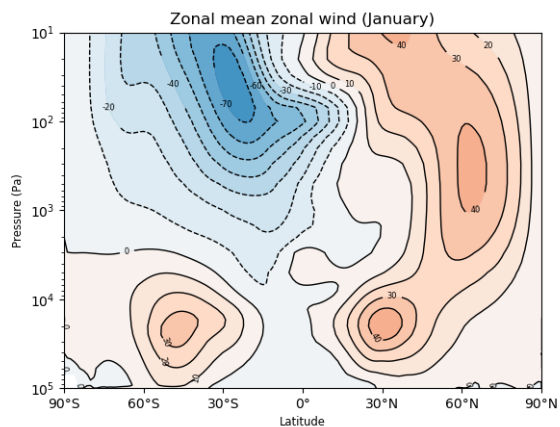


Fig. 46: Zonal mean zonal wind in January for UKESM1-0-LL.

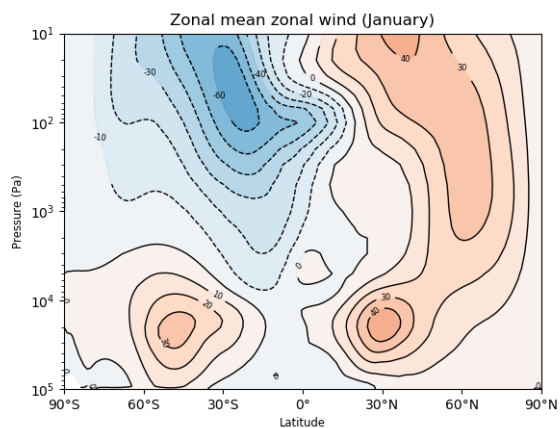


Fig. 47: Zonal mean zonal wind in January for HadGEM3-GC31-LL.

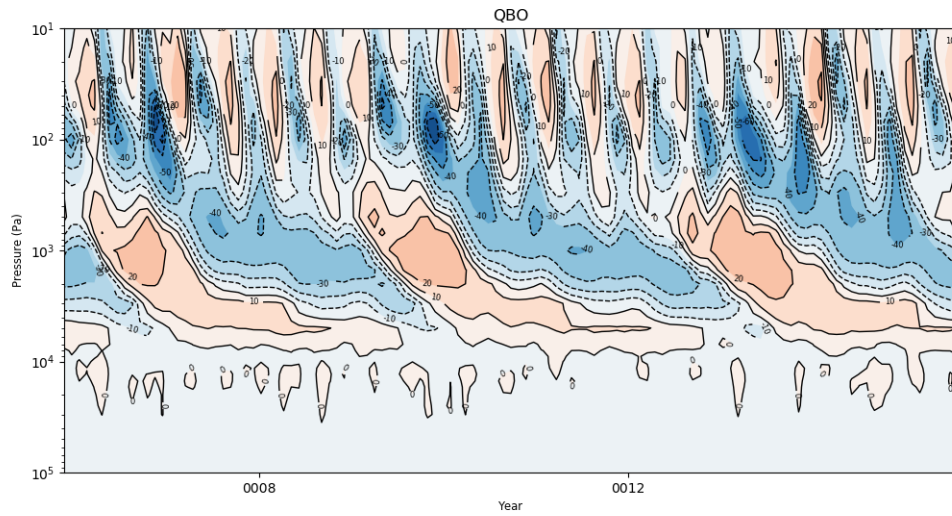


Fig. 48: QBO for UKESM1-0-LL.

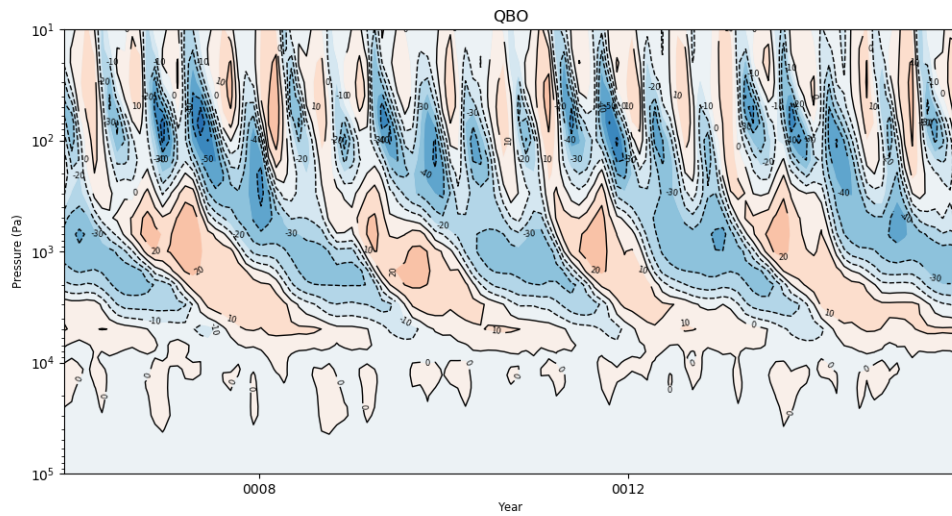


Fig. 49: QBO for HadGEM3-GC31-LL.

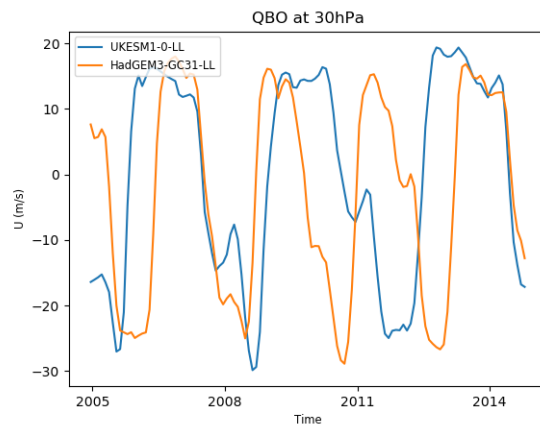


Fig. 50: QBO at 30hPa comparison between UKESM1-0-LL and HadGEM3-GC31-LL.

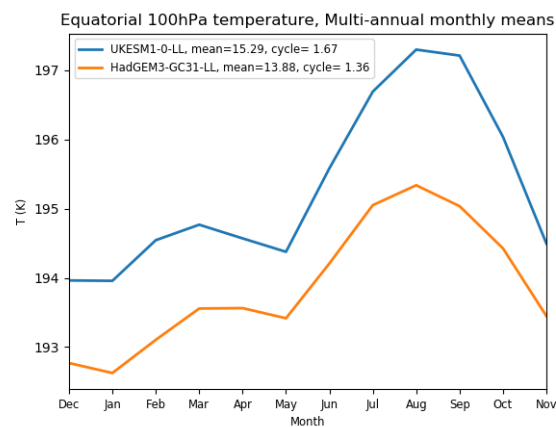


Fig. 51: Equatorial temperature at 100hPa, multi annual means.

- 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.

14.24.2 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

14.24.3 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)
- `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_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

14.24.4 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)

14.24.5 Observations and reformat scripts

None

14.24.6 References

- Observed permafrost extent is from <http://nsidc.org/data/ggd318.html>: 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 calculated 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.

14.24.7 Example plots

14.24.8 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.

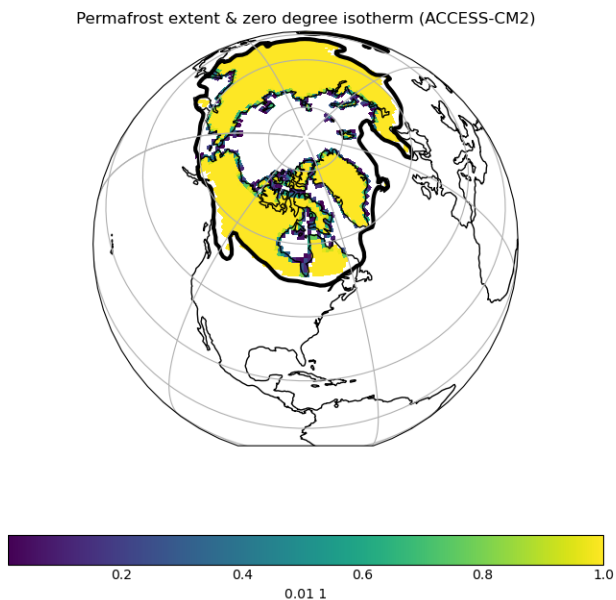


Fig. 52: Permafrost extent and zero degC isotherm, showing North America

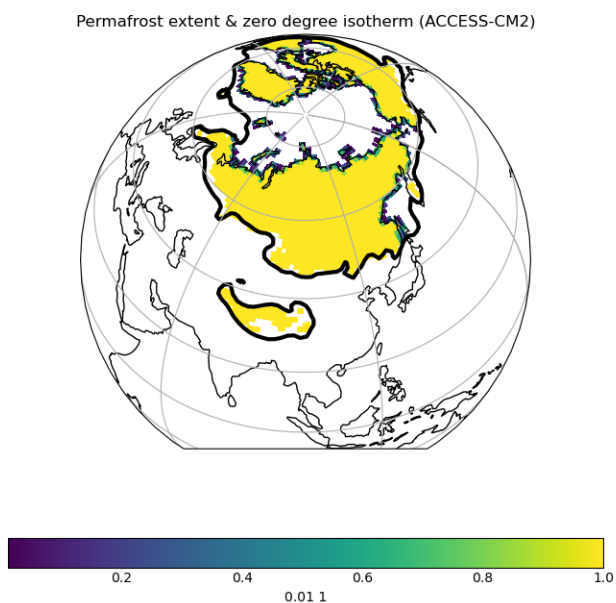


Fig. 53: Permafrost extent and zero degC isotherm, showing Asia and Europe

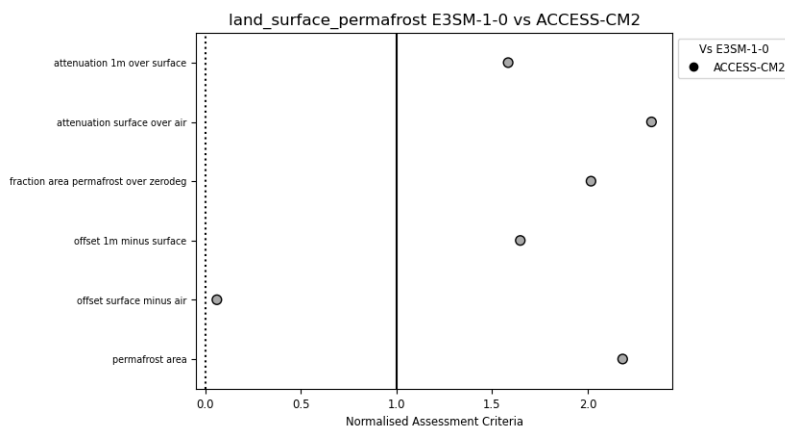


Fig. 54: Normalised metrics plot comparing a control and experiment simulation

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:
  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
```


14.25 Land-surface Surface Radiation - Autoassess diagnostics

14.25.1 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.

14.25.2 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

14.25.3 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

14.25.4 Variables

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

14.25.5 Observations and reformat scripts

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

14.25.6 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

14.25.7 Example plots

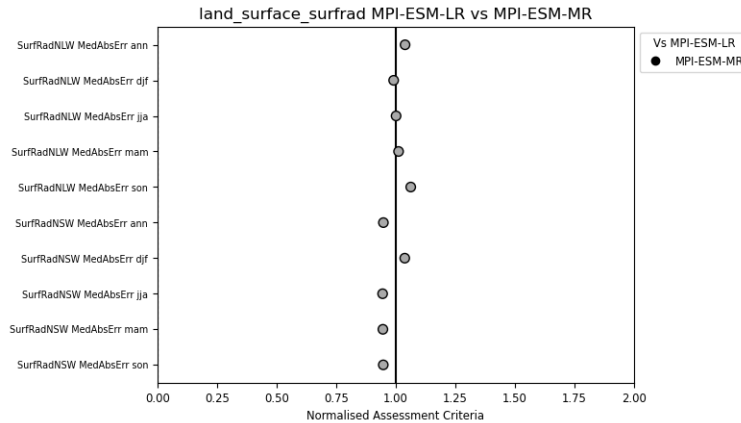


Fig. 55: Normalised metrics plot comparing a control and experiment simulation

14.25.8 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
```

(continues on next page)

(continued from previous page)

start: 1997/12/01
end: 2002/12/01

14.26 Land-surface Soil Moisture - Autoassess diagnostics

14.26.1 Overview

Soil moisture is a critical component of the land system, controlling 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.

14.26.2 Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/`

- `recipe_autoassess_landsurface_soilmoisture.yml`

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

- `autoassess_area_base.py`: wrapper for autoassess scripts
- `land_surface_soilmoisture/soilmoisture.py`: script to calculate soil moisture metrics
- `plot_autoassess_metrics.py`: plot normalised assessment metrics

14.26.3 User settings in recipe

1. Script `autoassess_area_base.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
- `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_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

14.26.4 Variables

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

14.26.5 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>

14.26.6 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>

14.26.7 Example plots

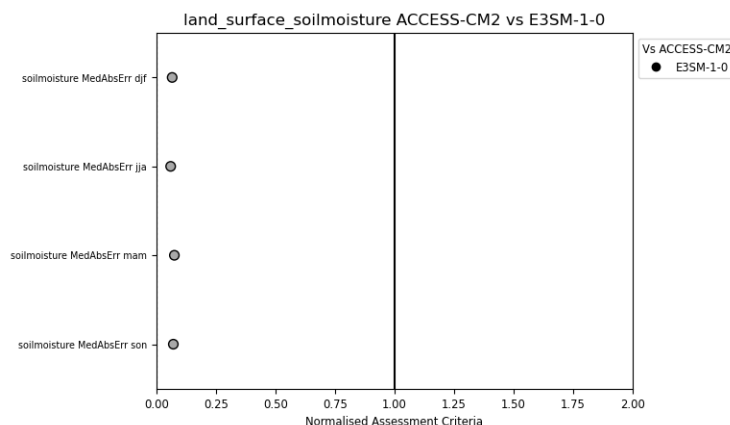


Fig. 56: Normalised metrics plot comparing a control and experiment simulation

14.26.8 Additional notes on 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_soilmoisture: &autoassess_landsurf_soilmoisture_settings
    script: autoassess/autoassess_area_base.py
    title: "Autoassess Land-Surface Soilmoisture Diagnostic"
    area: land_surface_soilmoisture
    control_model: IPSL-CM5A-LR
    exp_model: inmcm4
    obs_models: []
    start: 1997/12/01
```

(continues on next page)

(continued from previous page)

`end: 2002/12/01``climfiles_root: '/gws/nopw/j04/esmeval/autoassess_specific_files/files' # on JASMIN`

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

14.27.1 Overview

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

The recipe `recipe_zmnam.yml` can be used to evaluate the representation of 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.

14.27.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_zmnam.yml`

Diagnostics are stored in `diag_scripts/zmnam/`

- `zmnam.py`

and subroutines

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

14.27.3 User settings

Hemisphere of interest (NH or SH)

14.27.4 Variables

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

14.27.5 Observations and reformat scripts

None.

14.27.6 References

- Baldwin, M. P. and Thompson, D. W. (2009), A critical comparison of stratosphere–troposphere coupling indices. *Q.J.R. Meteorol. Soc.*, 135: 1661–1672. doi:10.1002/qj.479.
- Baldwin, M. P. and Dunkerton, T. J. (2001), Stratospheric Harbingers of Anomalous Weather Regimes. *Science* 294 (5542): 581–584. doi:10.1126/science.1063315.
- Wallace, J. M. (2000), North Atlantic Oscillation/annular mode: Two paradigms-one phenomenon. *Q.J.R. Meteorol. Soc.*, 126 (564): 791–805. doi:10.1002/qj.49712656402.

14.27.7 Example plots

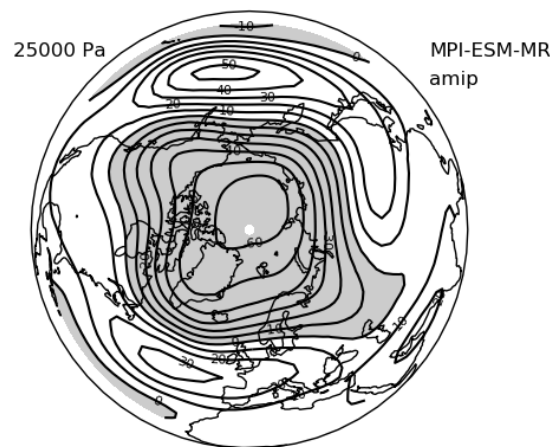


Fig. 57: Regression map of the Northern Hemisphere zonal mean AM index onto geopotential height, for a selected pressure level (250 hPa) for the MPI-ESM-MR model (CMIP5 AMIP experiment, period 1979–2008). Negative values are shaded in grey.

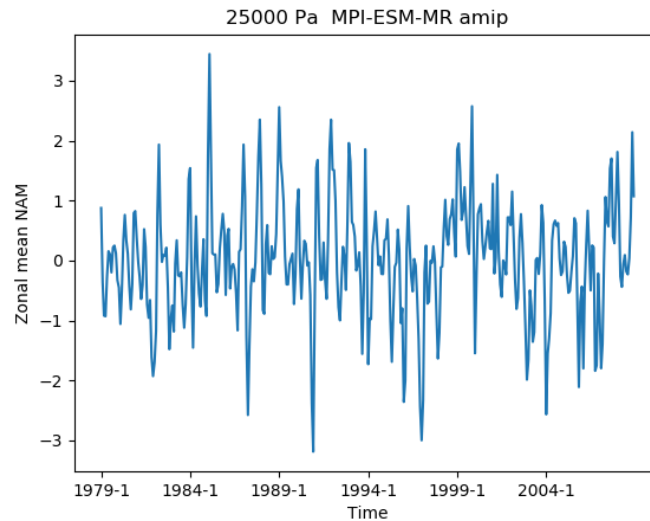


Fig. 58: Time series of the Northern Hemisphere zonal mean AM index for a selected pressure level (250 hPa) for the MPI-ESM-MR model (CMIP5 AMIP experiment, period 1979-2008).

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

14.28.1 Overview

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

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

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

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

14.28.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_thermodyn_diagtool.yml`

Diagnostics are stored in `diag_scripts/thermodyn_diagtool/`

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

14.28.3 User settings

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

- `wat`: if set to 'true', computations are performed of the water mass and latent energy budgets and transports
- `lsm`: if set to true, the computations of the energy budgets, meridional energy transports, water mass and latent energy budgets and transports are performed separately over land and oceans
- `lec`: if set to 'true', computation of the LEC are performed
- `entr`: if set to 'true', computations of the material entropy production are performed
- `met` (1, 2 or 3): the computation of the material entropy production must be performed with the indirect method (1), the direct method (2), or both methods. If 2 or 3 options are chosen, the intensity of the LEC is needed for the entropy production related to the kinetic energy dissipation. If `lec` is set to 'false', a default value is provided.

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

14.28.4 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'):

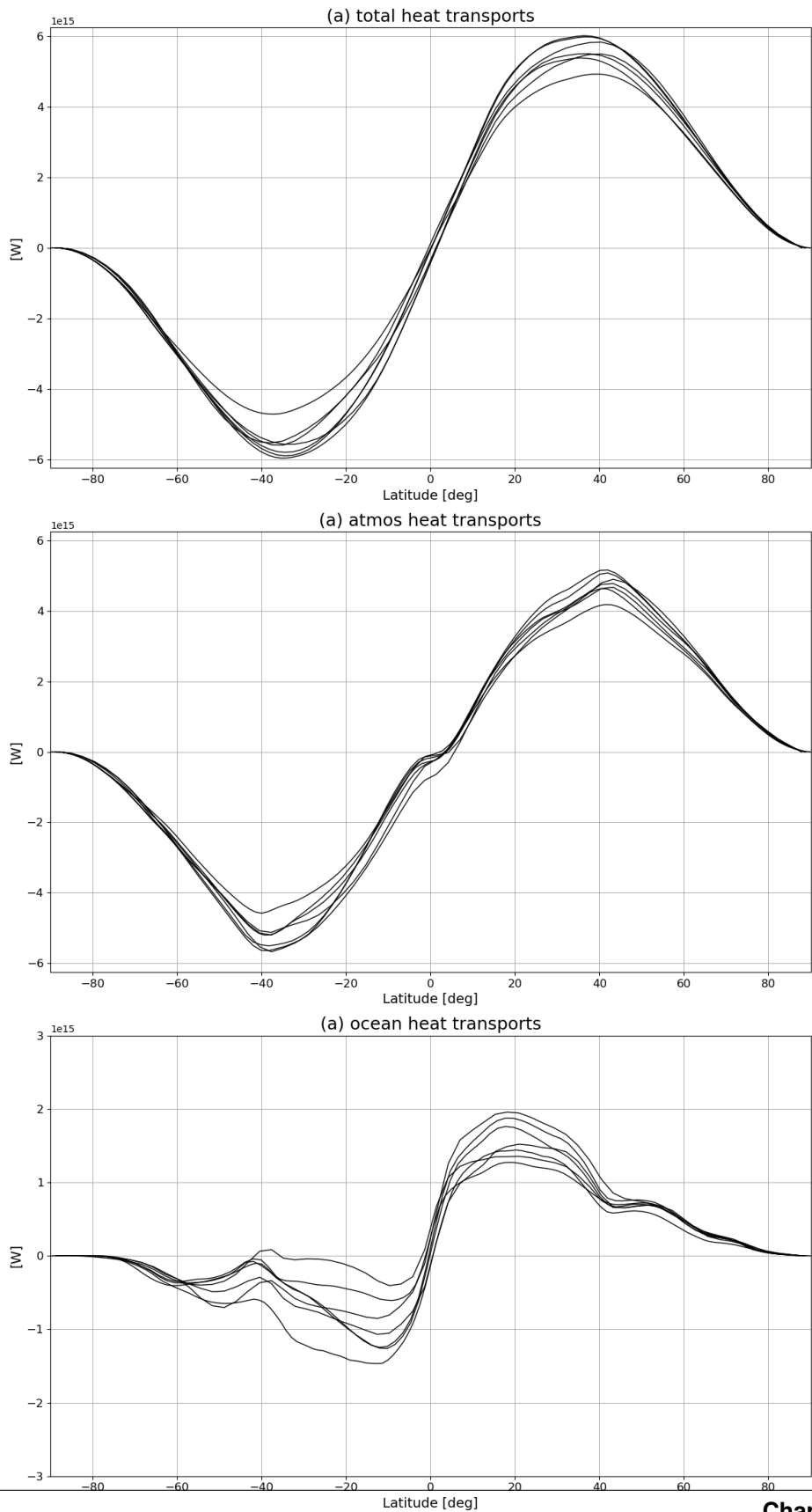
- tas (atmos, daily mean, time latitude longitude)
- uas (atmos, daily mean, time latitude longitude)
- vas (atmos, daily mean, time latitude longitude)

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

14.28.5 References

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

14.28.6 Example plots



takes two models designated by CONTROL and EXPERIMENT and compares them via a number of analyses. Optionally a number of observational datasets can be added for processing.

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

At least re-gridding on a common grid for all model and observational datasets should be performed in preprocessing (if datasets are on different grids). Also note that 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).

14.29.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_validation.yml (CMIP5)
- recipe_validation_CMIP6.yml (CMIP6)

Diagnostics are stored in diag_scripts/

- validation.py
- shared/_validation.py

14.29.3 User settings

1. validation.py

Required settings for script

- title: title of the analysis, user defined;
- control_model: control dataset name e.g. UKESM1-0-LL;
- exper_model: experiment dataset name e.g. IPSL-CM6A-LR;
- ob-ser-va-tional_datasets: list of at least

one element;
if no OBS
wanted comment
out;
e.g. ['ERA-
Interim'];

- `analysis_type`:
use any of:
`lat_lon`, `merid-
ional_mean`,
`zonal_mean`;
- `sea-
sonal_analysis`:
boolean, if sea-
sonal means
are needed e.g.
`true`;
- `save_cubes`:
boolean, save
each of the
plotted cubes
in `/work`;

14.29.4 Variables

- any variable

14.29.5 Observations and re- format scripts

*Note: (1)
`obs4MIPs`
or `OBS` or
`ana4mips` can
be used.*

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

14.29.6 References

- none, basic technical analysis

14.29.7 Example plots

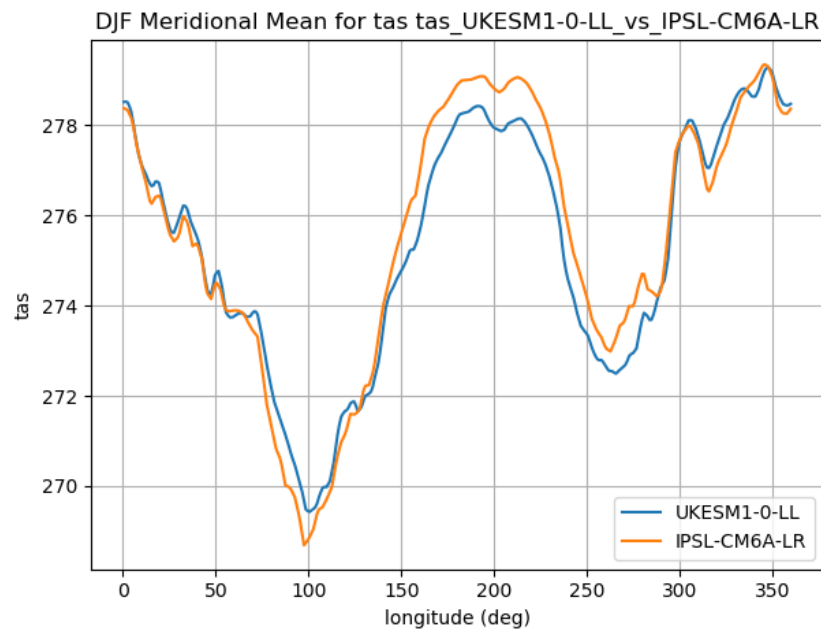


Fig. 59: Meridional seasonal mean for winter (DJF) comparison between CMIP6 UKESM1 and IPSL models.

14.30 Radiation Budget

14.30.1 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

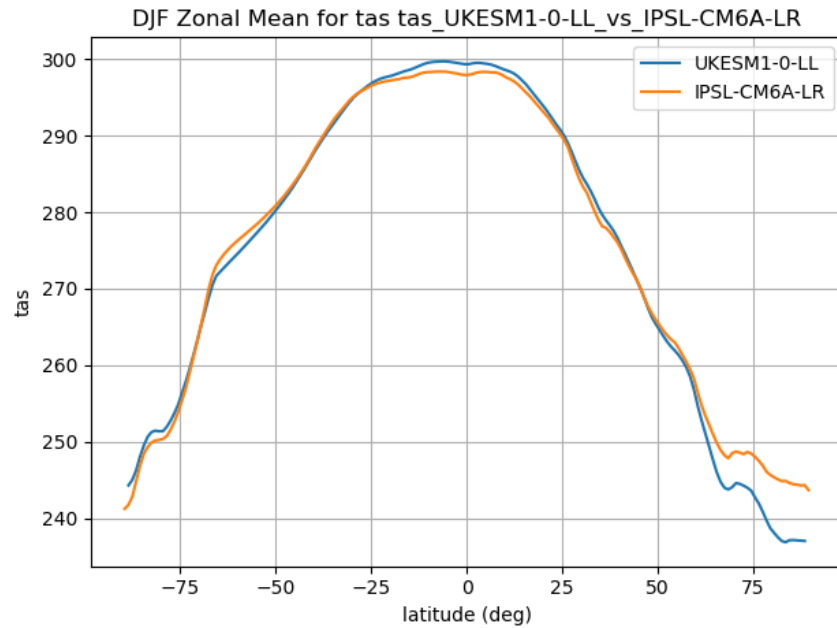


Fig. 60: Zonal seasonal mean for winter (DJF) comparison between CMIP6 UKESM1 and IPSL models.

with the hydro-
logical cycle
and climate
change.

This diagnos-
tic 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 pre-
dictions 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).

14.30.2 Available recipes and diagnostics

Recipes are stored in `esm-valtool/recipes/`

- `recipe_radiation_budget.yml`
Diagnostics are stored in `esm-valtool/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.

14.30.3 User settings in recipe

None

14.30.4 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)

14.30.5 Observations and re- format 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, rluts,
rsut, rsuts -
obs4MIPs)

- De-
mory
ob-
ser-
va-
tions
can
be
found
in
es-
m-
val-
tool/diag_scripts/radiation_budget/Demory_et_al_2014_obs_Energy_Budget.yml
and are from
Figure 2 in
Demory et al.
(2014).

- Stephens
ob-
ser-
va-
tions
can
be
found
in
es-
m-

val-
tool/diag_scripts/radiation_budget/Stephens_et_al_2012_obs_Energy_Budget.yml
from figure 1b
in Stephens et
al. (2012).

14.30.6 References

- Demory,
ME., Vi-
dale, P.L.,
Roberts,
M.J. et al.
The role
of horizon-
tal reso-
lution in
simulating
drivers of
the global
hydrolog-
ical cy-
cle. Clim
Dyn 42,
2201–2225
(2014).
<https://doi.org/10.1007/s00382-013-1924-4>
- Stephens,
G., Li,
J., Wild,
M. et al.
An up-
date on
Earth's en-
ergy bal-
ance in
light of
the lat-
est global
obser-
vations.
Nature
Geosci 5,
691–696
(2012).
<https://doi.org/10.1038/ngeo1580>

14.30.7 Example plots

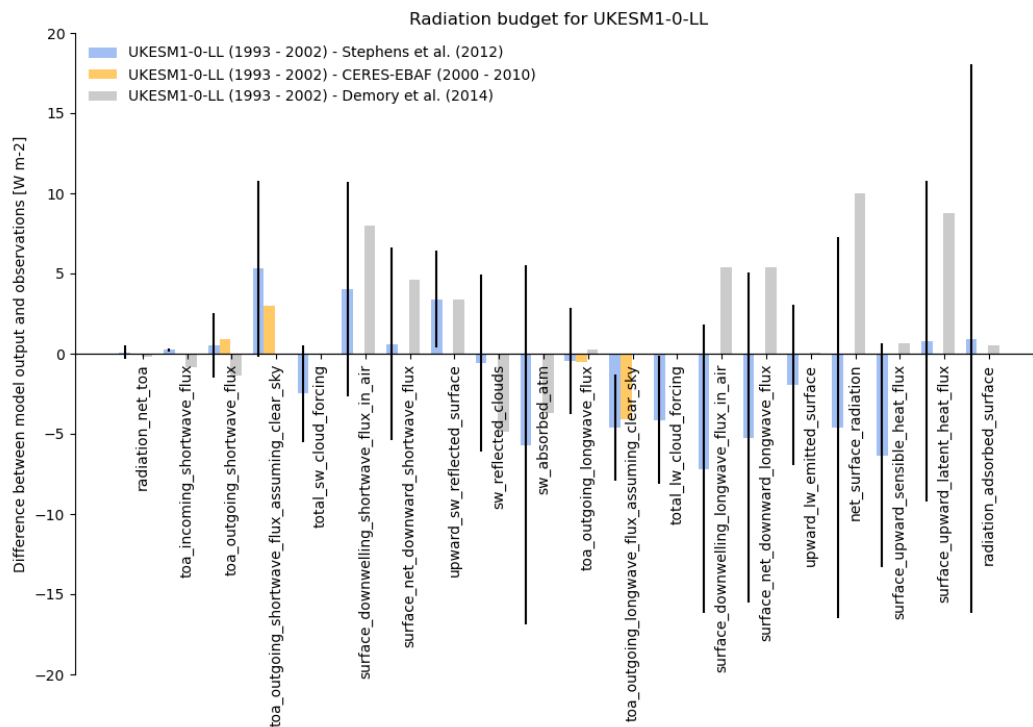


Fig. 61: Radiation budget for UKESM1-0-LL

CLIMATE METRICS

15.1 Performance metrics for essential climate parameters

15.1.1 Overview

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

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

15.1.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_perfmetrics_CMIP5.yml`
- `recipe_perfmetrics_CMIP5_cds.yml`
- `recipe_perfmetrics_land_CMIP5.yml`

Diagnostics are stored in `diag_scripts/perfmetrics/`

- `main.ncl`: calculates and (optionally) plots annual/seasonal cycles, zonal means, lat-lon fields and time-lat-lon fields. The calculated fields can also be plotted as difference w.r.t. a given reference dataset. `main.ncl` also calculates RMSD, bias and taylor metrics. Input data have to be regridded to a common grid in the preprocessor. Each plot type is created by a separated routine, as detailed below.
- `cycle.ncl`: creates an annual/seasonal cycle plot.
- `zonal.ncl`: creates a zonal (lat-pressure) plot.
- `latlon.ncl`: creates a lat-lon plot.
- `cycle_latlon.ncl`: precalculates the metrics for a time-lat-lon field, with different options for normalization.
- `collect.ncl`: collects and plots the metrics previously calculated by `cycle_latlon.ncl`.

15.1.3 User settings in recipe

1. Script main.ncl

Required settings (scripts)

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

Optional settings (scripts)

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

Required settings (variables)

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

Optional settings (variables)

- `alternative_dataset`: a second dataset to compare with.

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

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

15.1.4 Variables

1. `recipe_perfmetrics_CMIP5.yml`

- `clt` (atmos, monthly mean, longitude latitude time)
- `hus` (atmos, monthly mean, longitude latitude lev time)
- `od550aer`, `od870aer`, `od550abs`, `od550lt1aer` (aero, monthly mean, longitude latitude time)
- `pr` (atmos, monthly mean, longitude latitude time)
- `rlut`, `rlutcs`, `rsut`, `rsutcs` (atmos, monthly mean, longitude latitude time)
- `sm` (land, monthly mean, longitude latitude time)
- `ta` (atmos, monthly mean, longitude latitude lev time)
- `tas` (atmos, monthly mean, longitude latitude time)
- `toz` (atmos, monthly mean, longitude latitude time)
- `ts` (atmos, monthly mean, longitude latitude time)
- `ua` (atmos, monthly mean, longitude latitude lev time)
- `va` (atmos, monthly mean, longitude latitude lev time)
- `zg` (atmos, monthly mean, longitude latitude lev time)

2. `recipe_perfmetrics_land_CMIP5.yml`

- `sm` (land, monthly mean, longitude latitude time)
- `nbp` (land, monthly mean, longitude latitude time)
- `gpp` (land, monthly mean, longitude latitude time)
- `lai` (land, monthly mean, longitude latitude time)

- fgco2 (ocean, monthly mean, longitude latitude time)
- et (land, monthly mean, longitude latitude time)
- rlus, rlds, rsus, rdsd (atmos, monthly mean, longitude latitude time)

15.1.5 Observations and reformat scripts

The following list shows the currently used observational data sets for this recipe with their variable names and the reference to their respective reformat scripts in parentheses. Please note that obs4MIPs data can be used directly without any reformatting. 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.

#. 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/cmorizers/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 (tas, ta, ua, va, zg - [esmvaltool/cmorizers/data/formatters/datasets/ncep.ncl](#))
- NIWA-BS (toz - [esmvaltool/cmorizers/data/formatters/datasets/niwa_bs.ncl](#))
- PATMOS-x (clt - [esmvaltool/cmorizers/data/formatters/datasets/patmos_x.ncl](#))

1. recipe_perfmetrics_land_CMIP5.yml

- CERES-EBAF (rlus, rlds, rsus, rdsd - 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](#))

15.1.6 References

- Gleckler, P. J., K. E. Taylor, and C. Doutriaux, Performance metrics for climate models, *J. Geophys. Res.*, 113, D06104, doi: 10.1029/2007JD008972 (2008).
- Righi, M., Eyring, V., Klinger, C., Frank, F., Gottschaldt, K.-D., Jöckel, P., and Cionni, I.: Quantitative evaluation of oone and selected climate parameters in a set of EMAC simulations, *Geosci. Model Dev.*, 8, 733, doi: 10.5194/gmd-8-733-2015 (2015).

15.1.7 Example plots

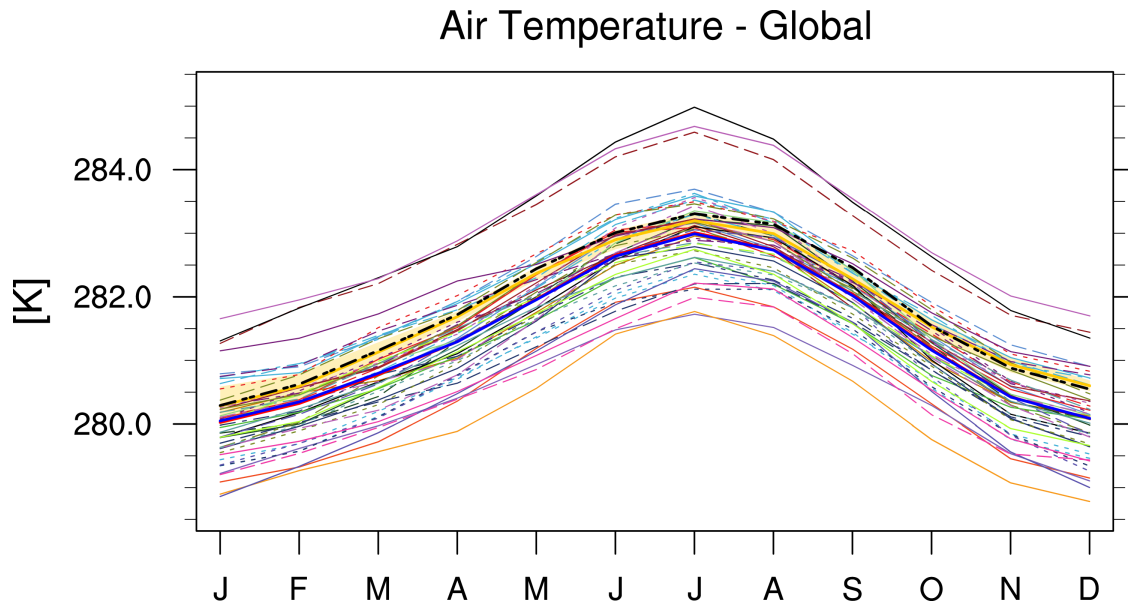


Fig. 1: Annual cycle of globally averaged temperature at 850 hPa (time period 1980-2005) for different CMIP5 models (historical simulation) (thin colored lines) in comparison to ERA-Interim (thick yellow line) and NCEP (thick black dashed line) reanalysis data.

15.2 Single Model Performance Index (SMPI)

15.2.1 Overview

This diagnostic calculates the Single Model Performance Index (SMPI) following Reichler and Kim (2008). The SMPI (called “ I^2 ”) is based on the comparison of several different climate variables (atmospheric, surface and oceanic) between climate model simulations and observations or reanalyses, and it focuses on the validation of the time-mean state of climate. For I^2 to be determined, the differences between the climatological mean of each model variable and observations at each of the available data grid points are calculated, and scaled to the interannual variance from the validating observations. This interannual variability is determined by performing a bootstrapping method (random selection with replacement) for the creation of a large synthetic ensemble of observational climatologies. The results are then scaled to the average error from a reference ensemble of models, and in a final step the mean over all climate variables and one model is calculated. The plot shows the I^2 values for each model (orange circles) and the multi-model mean (black circle), with the diameter of each circle representing the range of I^2 values encompassed by the 5th and 95th percentiles of the bootstrap ensemble. The I^2 values vary around one, with values greater than one for underperforming models, and values less than one for more accurate models.

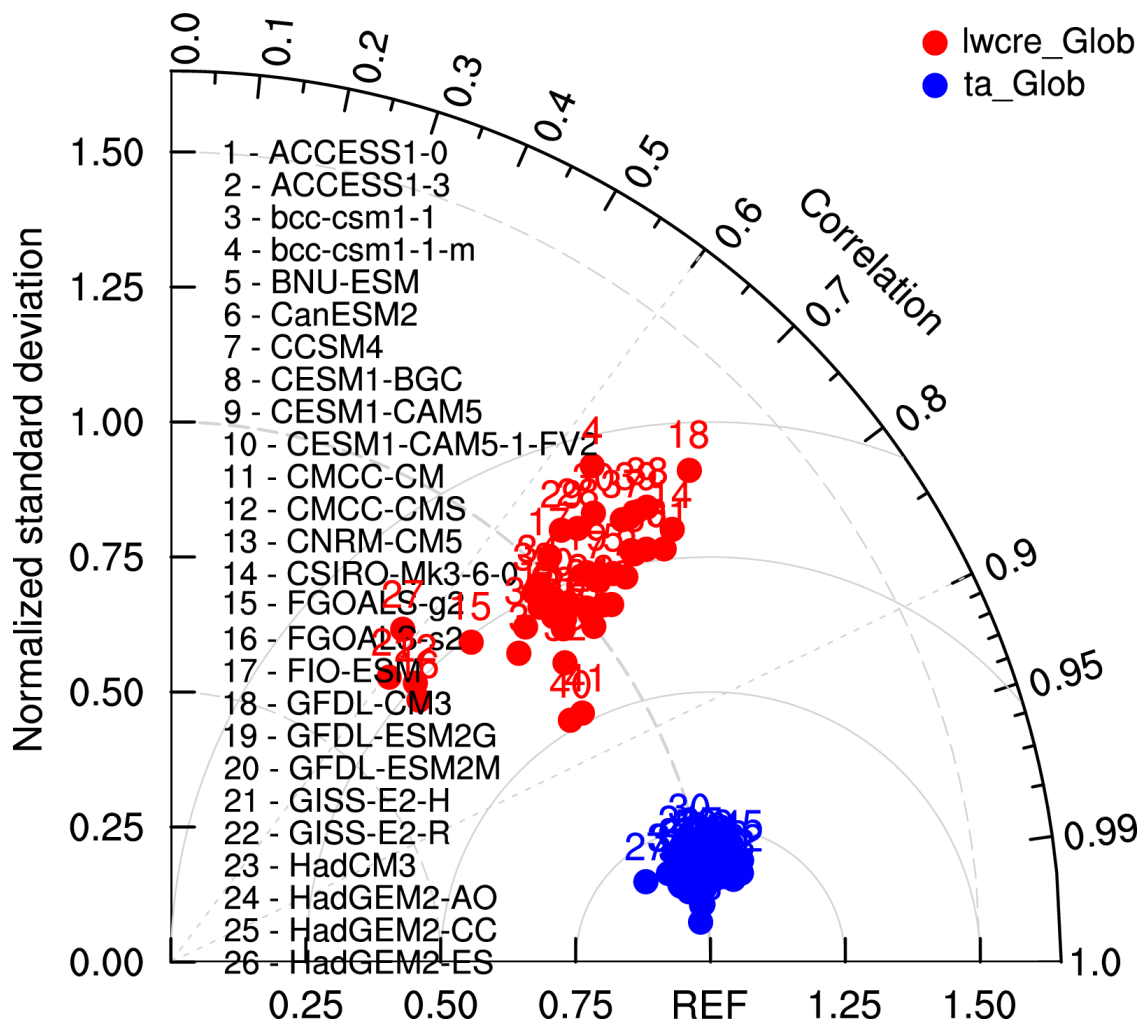


Fig. 2: Taylor diagram of globally averaged temperature at 850 hPa (ta) and longwave cloud radiative effect (lwcre) for different CMIP5 models (historical simulation, 1980-2005). Reference data (REF) are ERA-Interim for temperature (1980-2005) and CERES-EBAF (2001-2012) for longwave cloud radiative effect.

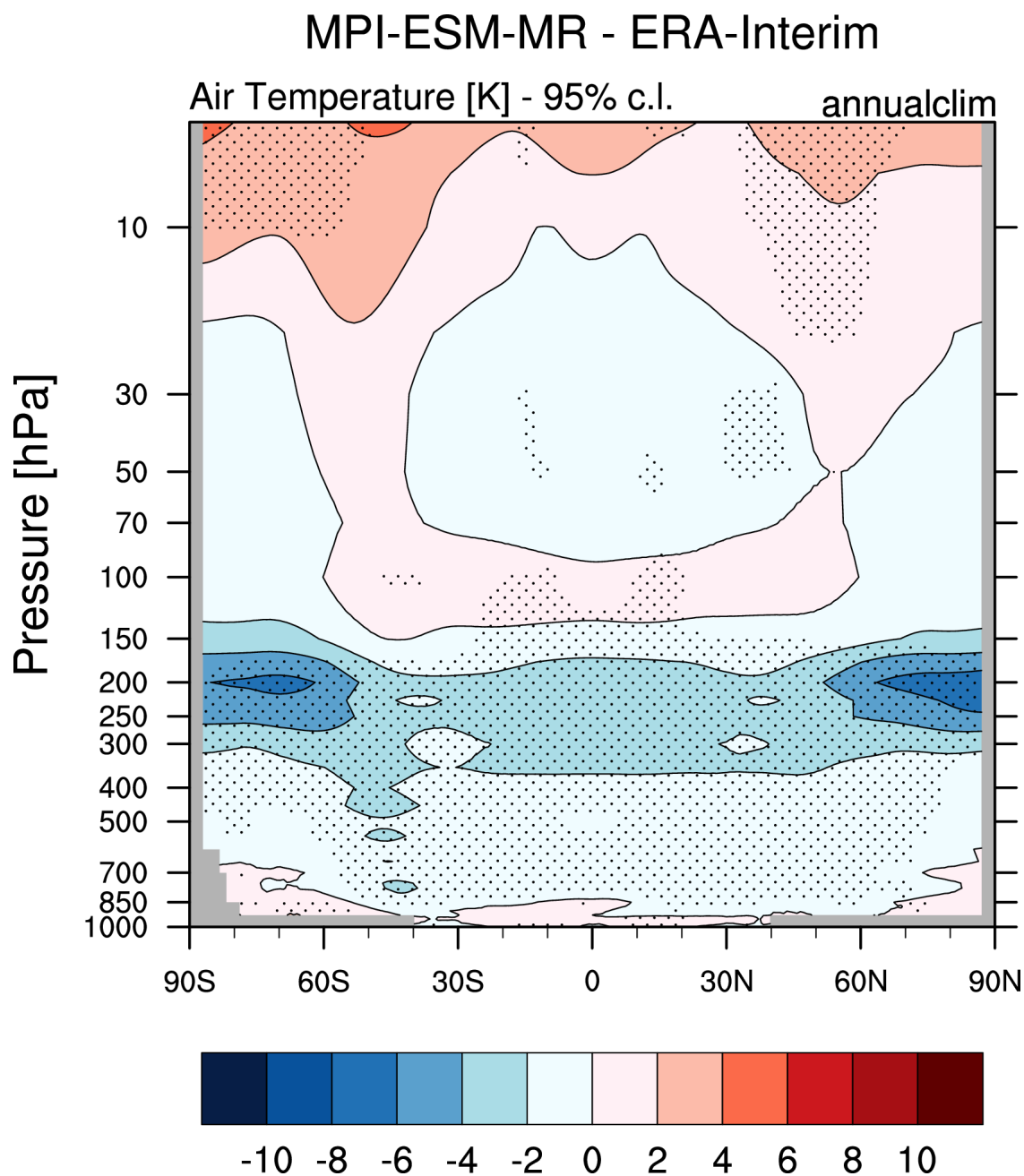


Fig. 3: Difference in annual mean of zonally averaged temperature (time period 1980-2005) between the CMIP5 model MPI-ESM-MR (historical simulation) and ERA-Interim. Stippled areas indicate differences that are statistically significant at a 95% confidence level.

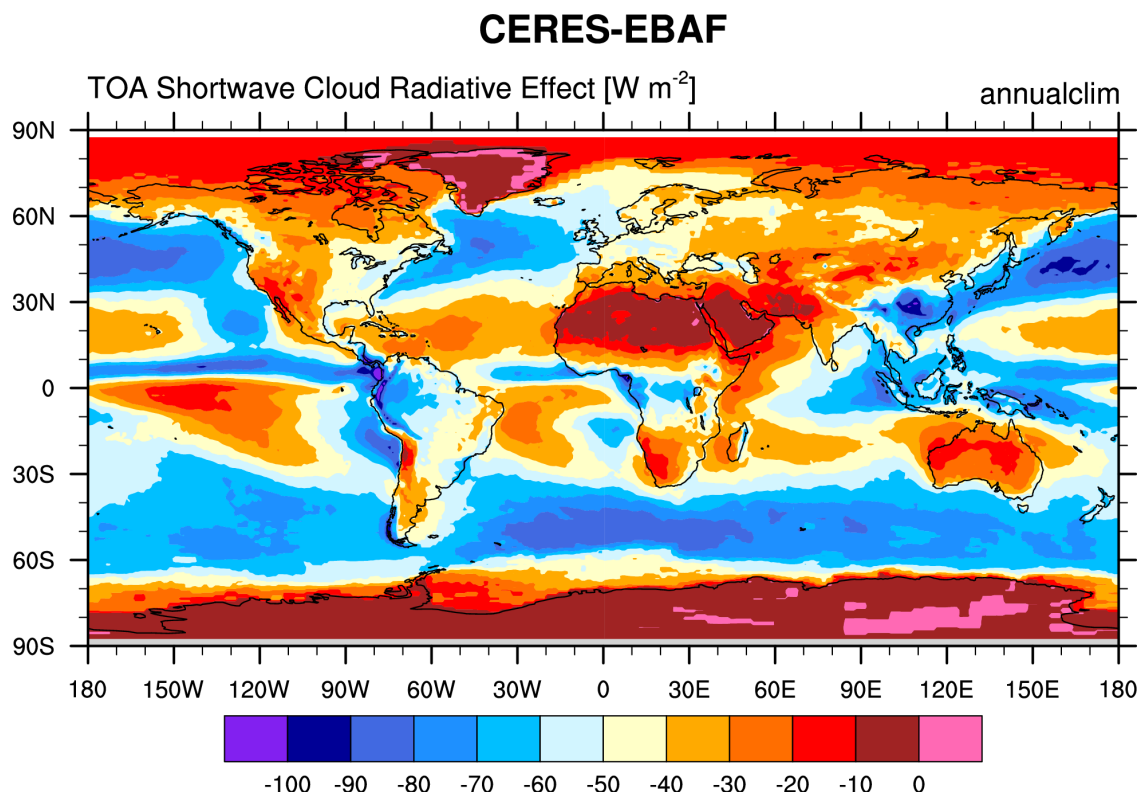


Fig. 4: Annual mean (2001-2012) of the shortwave cloud radiative effect from CERES-EBAF.

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

15.2.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_smpi.yml`
- `recipe_smpi_4cds.yml`

Diagnostics are stored in `diag_scripts/perfmetrics/`

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

15.2.3 User settings

1. perfmetrics/main.ncl

Required settings for script

- `plot_type`: only “cycle_latlon (time, lat, lon)” and “cycle_zonal (time, plev, lat)” available for SMPI; usage is defined in the recipe and is dependent on the used variable (2D variable: `cycle_latlon`, 3D variable: `cycle_zonal`)
- `time_avg`: type of time average (only “yearly” allowed for SMPI, any other settings are not supported for this diagnostic)
- `region`: selected region (only “global” allowed for SMPI, any other settings are not supported for this diagnostic)
- `normalization`: metric normalization (“CMIP5” for analysis of CMIP5 simulations; to be adjusted accordingly for a different CMIP phase)
- `calc_grading`: calculates grading metrics (has to be set to “true” in the recipe)
- `metric`: chosen grading metric(s) (if `calc_grading` is True; has to be set to “SMPI”)
- `smpi_n_bootstrap`: number of bootstrapping members used to determine uncertainties on model-reference differences (typical number of bootstrapping members: 100)

Required settings for variables

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

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

1. collect.ncl

Required settings for script

- `metric`: selected metric (has to be “SMPI”)

15.2.4 Variables

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

15.2.5 Observations and reformat scripts

The following list shows the currently used observational data sets for this recipe with their variable names and the reference to their respective reformat scripts in parentheses. Please note that obs4MIPs data can be used directly without any 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-SG (pr - obs4MIPs)

15.2.6 References

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

15.2.7 Example plots

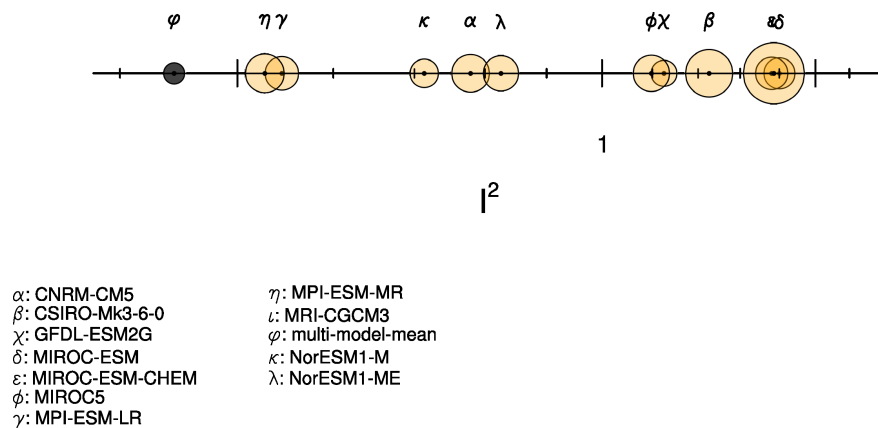


Fig. 6: Performance index I^2 for individual models (circles). Circle sizes indicate the length of the 95% confidence intervals. The black circle indicates the I^2 of the multi-model mean (similar to Reichler and Kim (2008), Figure 1).

FUTURE PROJECTIONS

16.1 Climate model Weighting by Independence and Performance (ClimWIP)

16.1.1 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>

16.1.2 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 `auxiliary_data_dir` (with is specified in the `config-user.yml`).

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>

16.1.3 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_brunner2019_med.yml`: Slightly modified results for one region from [Brunner et al. \(2019\)](#) (to change regions see below)
- `recipe_climwip_brunner2020esd.yml`: Slightly modified results for [Brunner et al. \(2020\)](#)

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.

16.1.4 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. 5](#) below) but not automatically picked - the user can decide to use it anyway by setting it in the recipe (not recommended).
- `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`

16.1.5 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 calculate 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.

16.1.6 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.

16.1.7 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, rlms (atmos, monthly mean, longitude latitude time)
- more variables can be added if available for all datasets.

16.1.8 Observations and reformat scripts

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

16.1.9 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

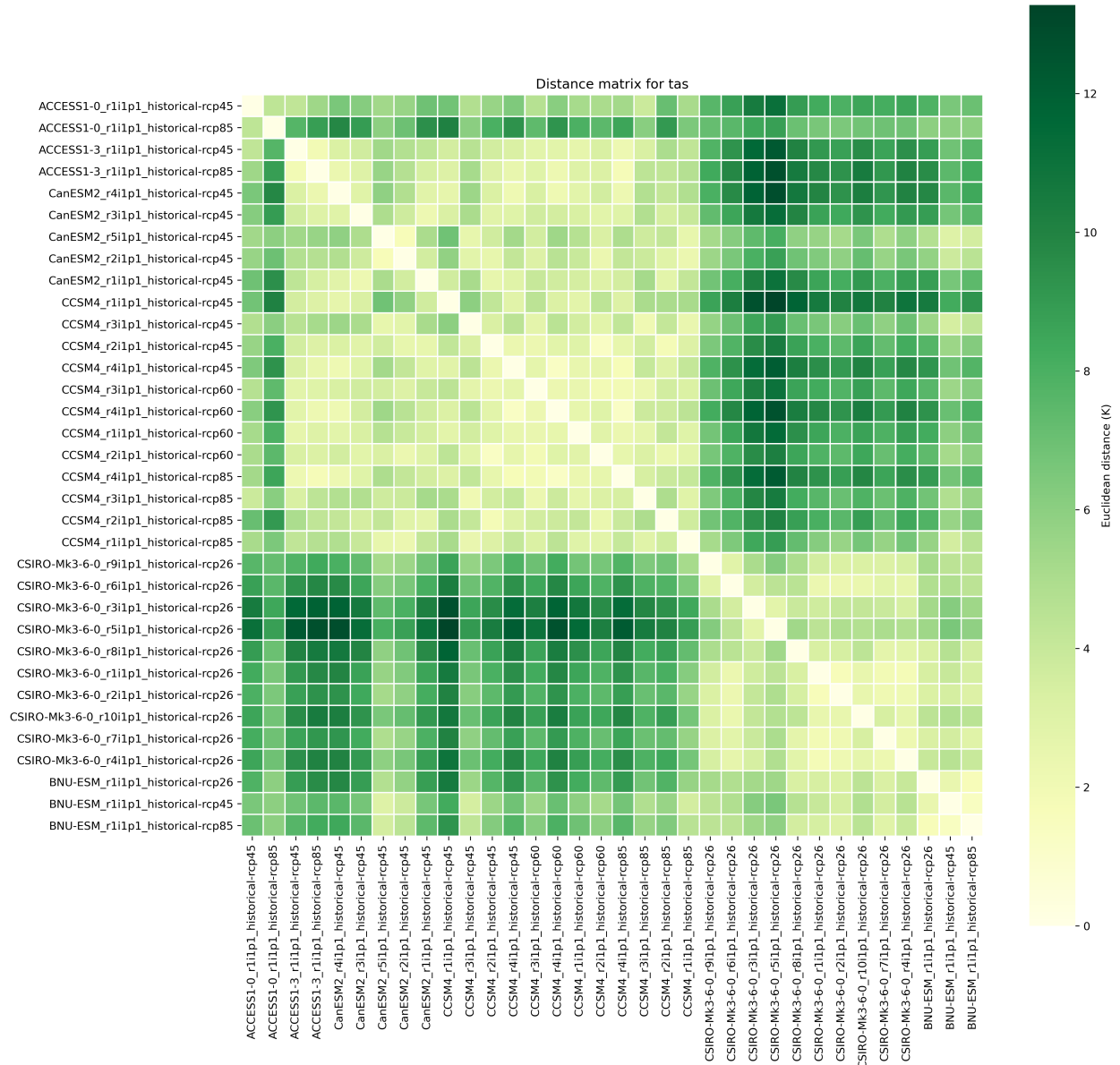


Fig. 1: Distance matrix for temperature, providing the independence metric.

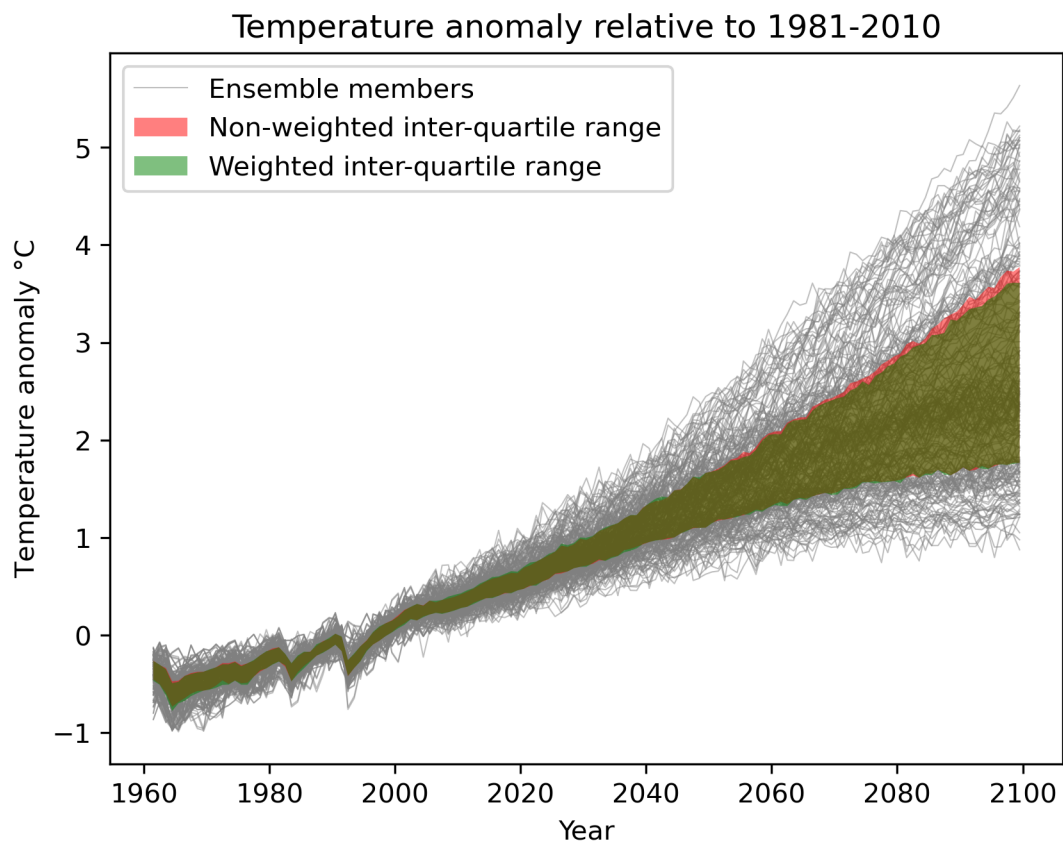


Fig. 4: Interquartile range of temperature anomalies relative to 1981-2010, weighted versus non-weighted.

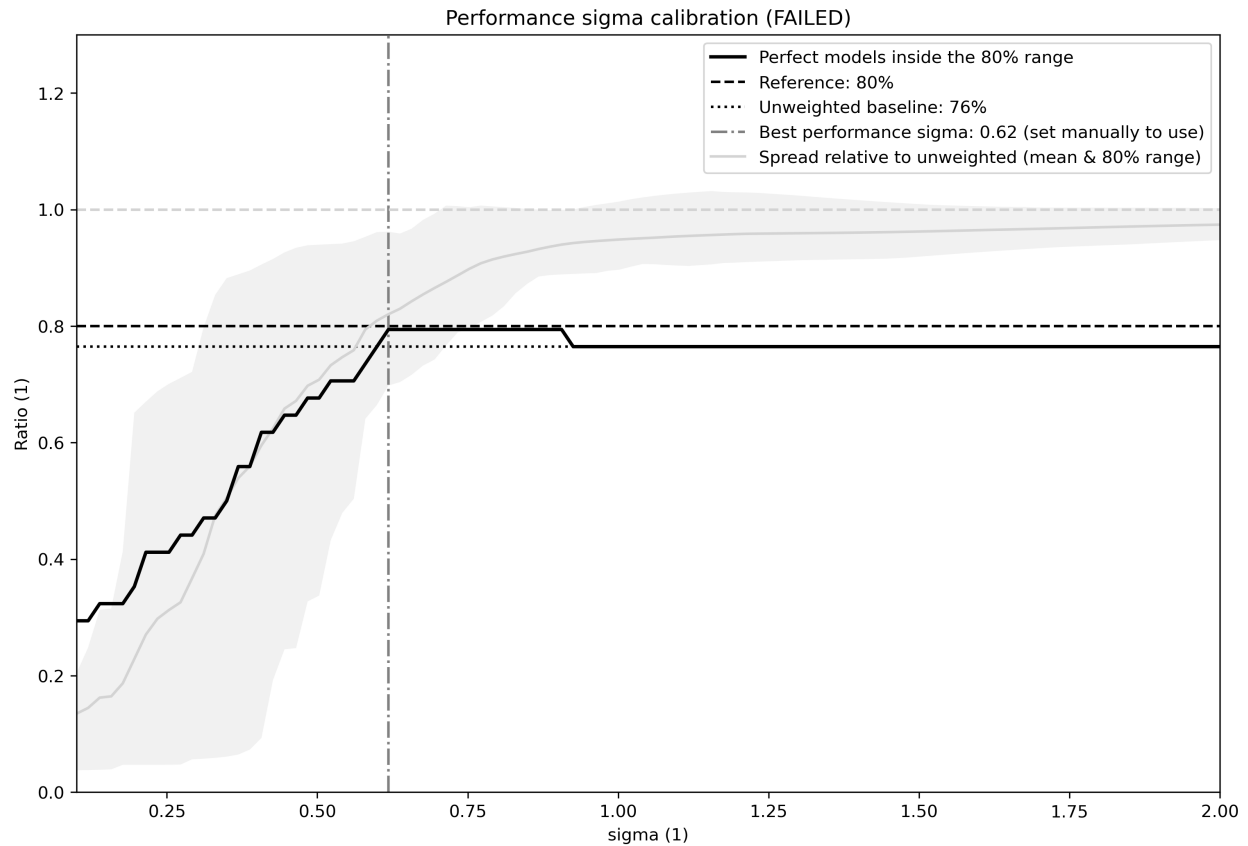


Fig. 5: Performance sigma calibration: The thick black line gives the reliability (c.f., weather forecast verification) which should reach at least 80%. The thick grey line gives the mean change in spread between the unweighted and weighted 80% ranges as an indication of the weighting strength (if it reaches 1, the weighting has no effect on uncertainty). The smallest sigma (i.e., strongest weighting) which is not overconfident (reliability $\geq 80\%$) 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).

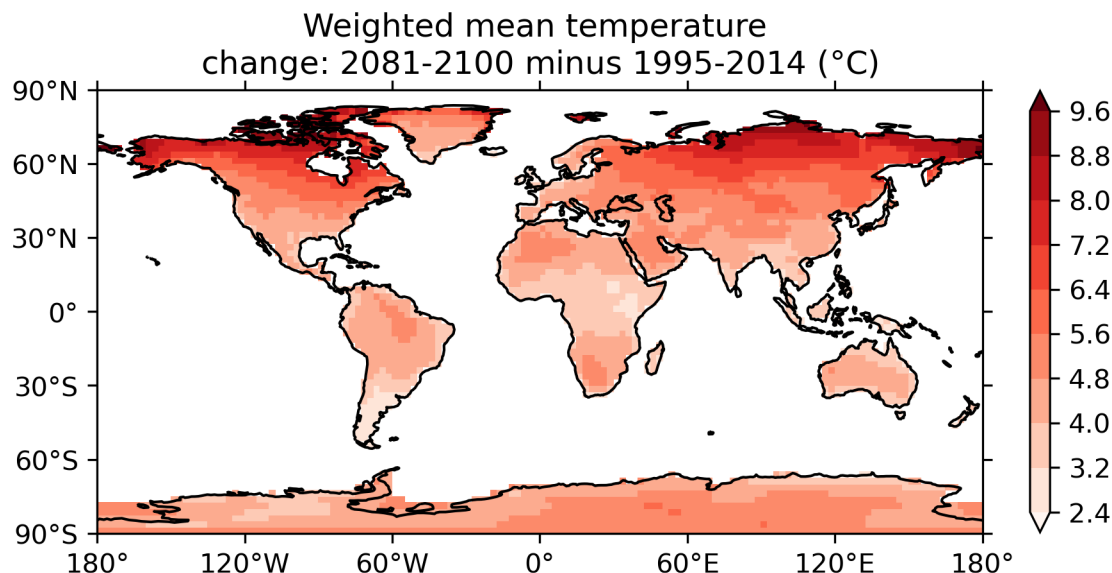


Fig. 6: Map of weighted mean temperature change 2081-2100 relative to 1995-2014

16.1.10 Example plots

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

16.2.1 Overview

Following [Li et al. \(2017\)](#) the change between present-day and future Indian Summer Monsoon (ISM) precipitation is constrained using the precipitation over the tropical western Pacific compared to a fixed, observed amount of 6 mm d^{-1} from Global Precipitation Climatology Project (GPCP) ([Adler et al., 2003](#)) for 1980-2009. For CMIP6, historical data for 1980-2009 should be used. For CMIP5 historical data from 1980-2005 should be used, due to the length of the data sets. At the moment it is not possible to use a combined ['historical', 'rcp'] data set, because the diagnostic requires that a historical data set is given.

16.2.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_li17natcc.yml`

Diagnostics are stored in `diag_scripts/`

- `emergent_constraints/lif1f2.py`

16.2.3 User settings in recipe

The recipe can be run with different CMIP5 and CMIP6 models. For each model, two experiments must be given: one historical run, possibly between 1980-2009 and one other model experiment. The user can choose the other model experiment, but it needs to be the same for all given models. The start and end year for the second data set can be chosen 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.

16.2.4 Variables

- *pr* (atmos, monthly, longitude, latitude, time)
- *ua* (atmos, monthly, longitude, latitude, plev, time)
- *va* (atmos, monthly, longitude, latitude, plev, time)
- *ts* (atmos, monthly, longitude, latitude, time)

16.2.5 Observations and reformat scripts

None

16.2.6 References

- Li, G., Xie, S. P., He, C., and Chen, Z. S.: Western Pacific emergent constraint lowers projected increase in Indian summer monsoon rainfall, *Nat Clim Change*, 7, 708-+, 2017

16.2.7 Example plots

16.3 Constraining uncertainty in projected gross primary production (GPP) with machine learning

16.3.1 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.

16.3.2 Available recipes and diagnostics

Recipes are stored in `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 `diag_scripts/`

- `mlr/evaluate_residuals.py`

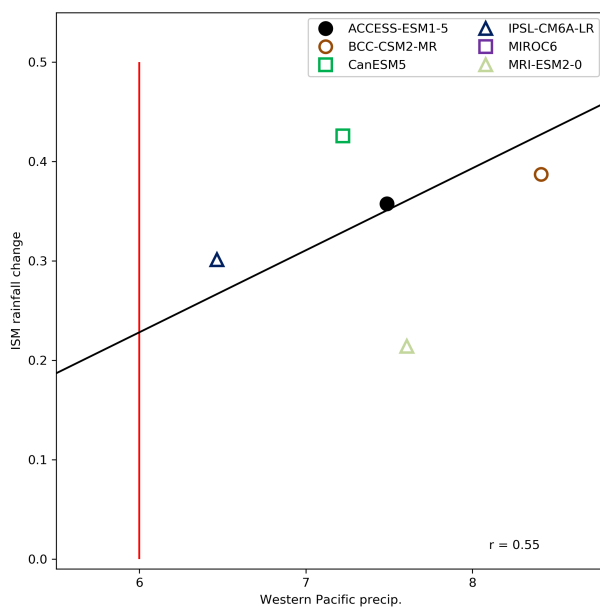


Fig. 7: Scatter plot of the simulated tropical western Pacific precipitation (mm d^{-1}) versus projected average ISM (Indian Summer Monsoon) rainfall changes under the ssp585 scenario. The red line denotes the observed present-day western Pacific precipitation and the inter-model correlation (r) is shown. (CMIP6).

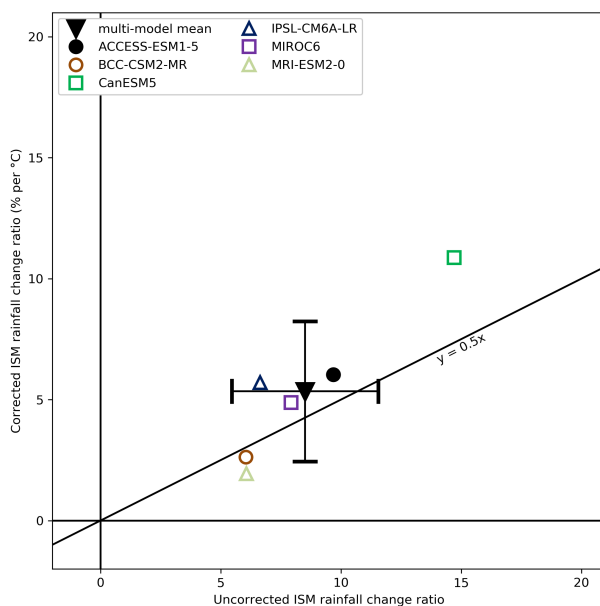


Fig. 8: Scatter plot of the uncorrected versus corrected average ISM (Indian Summer Monsoon) rainfall change ratios (% per degree Celsius of global SST warming). The error bars for the Multi-model mean indicate the standard deviation spread among models and the 2:1 line ($y = 0.5x$) is used to illustrate the Multi-model mean reduction in projected rainfall increase. (CMIP6).

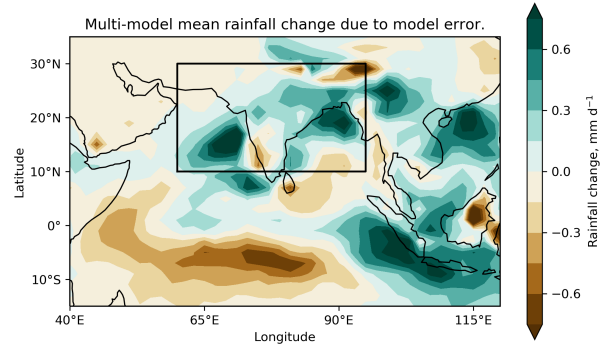


Fig. 9: Multi-model mean rainfall change due to model error. Box displays the area used to define the average ISM (Indian Summer Monsoon) rainfall. Precipitation changes are normalized by the corresponding global mean SST increase for each model. (CMIP6).

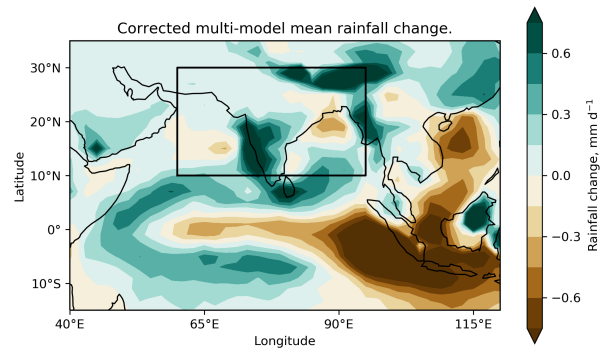


Fig. 10: Corrected multi-model mean rainfall change. Box displays the area used to define the average ISM (Indian Summer Monsoon) rainfall. Precipitation changes are normalized by the corresponding global mean SST increase for each model. (CMIP6).

- *mlr/main.py*
- *mlr/mmm.py*
- *mlr/plot.py*
- *mlr/postprocess.py*
- *mlr/preprocess.py*
- *mlr/rescale_with_emergent_constraint.py*

General information (including an example and more details) on machine learning regression (MLR) diagnostics is given [here](#). The API documentation is available [here](#).

16.3.3 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)

16.3.4 Observations and reformat scripts

- CRU (*pr*, *tas*)
- ERA-Interim (*rsds*)
- LAI3g (*lai*)
- MTE (*gpp*, *gppStderr*)
- Scripps-CO2-KUM (*co2s*)

16.3.5 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>.

16.3.6 Example plots

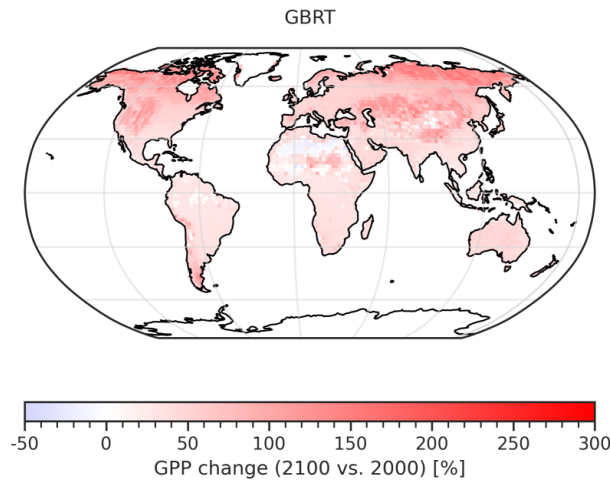


Fig. 11: GBRT-based prediction of the fractional GPP change over the 21st century ($= \text{GPP}(2091-2100) / \text{GPP}(1991-2000)$).

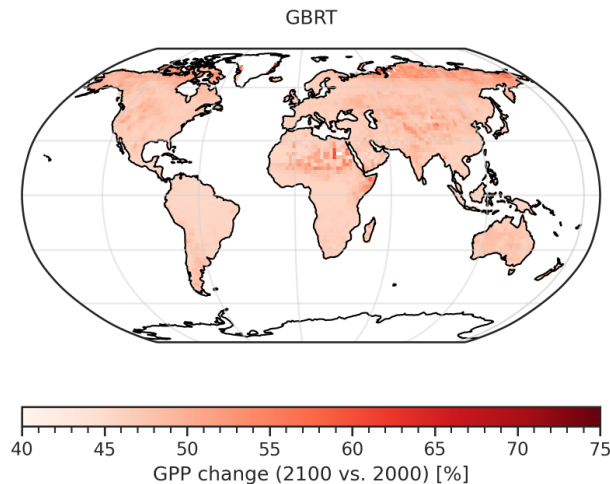


Fig. 12: Corresponding error of the GBRT-based prediction of the fractional GPP change over the 21st century (considering errors in the MLR model and errors in the predictors).

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

16.4.1 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.

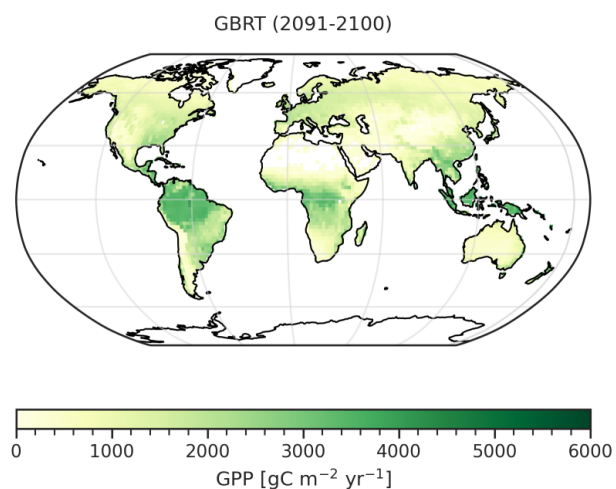


Fig. 13: GBRT-based prediction of the absolute GPP at the end of the 21st century (2091-2100).

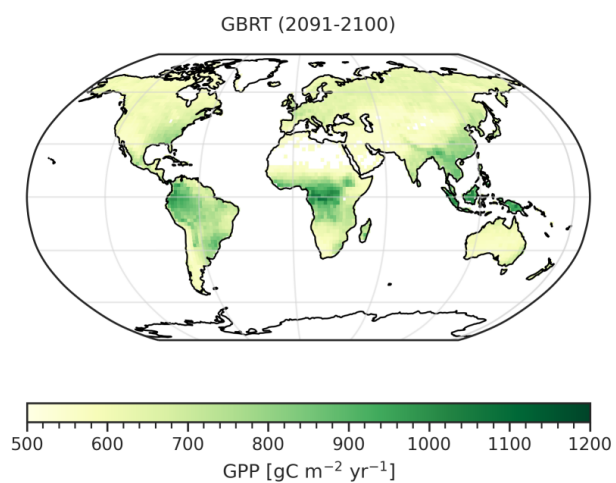


Fig. 14: Corresponding error of the GBRT-based prediction of the absolute GPP at the end of the 21st century (considering errors in the MLR model and errors in the predictors).

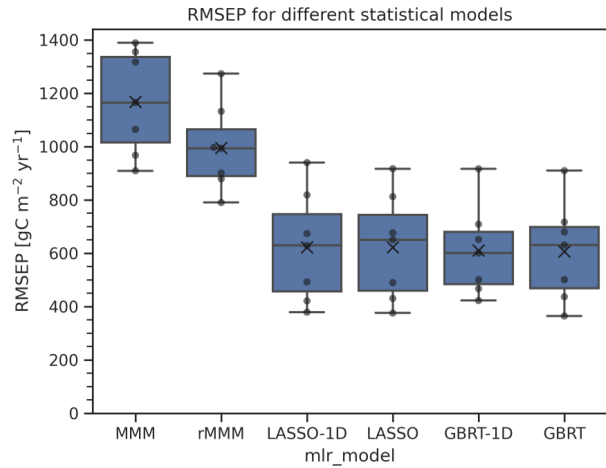


Fig. 15: Boxplot of the root mean square error of prediction (RMSEP) distributions for six different statistical models used to predict future absolute GPP (2091-2100) using a leave-one-model-out cross-validation approach. The distribution for each statistical model contains seven points (black dots, 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

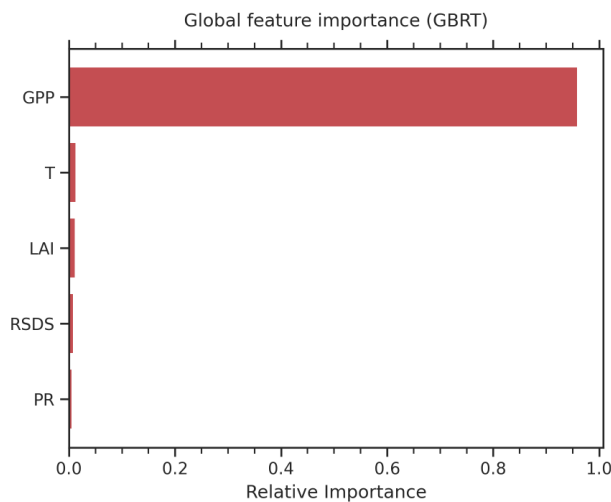


Fig. 16: Global feature importance of the GBRT model for prediction of the absolute GPP at the end of the 21st century (2091-2100).

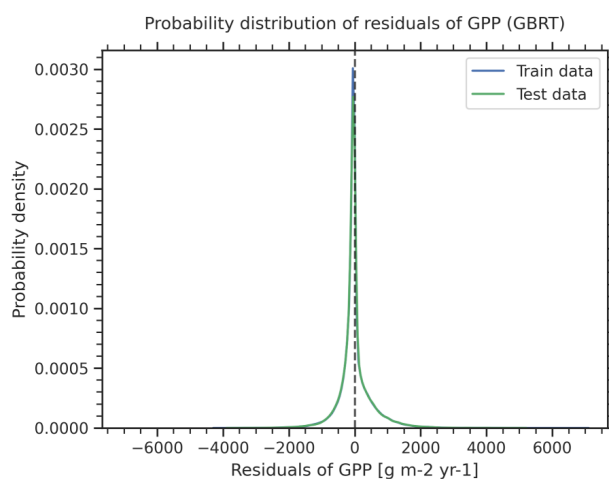


Fig. 17: Distribution of the residuals of the GBRT model for the prediction of absolute GPP at the end of the 21st century (2091-2100) for the training data (blue) and test data excluded from training (green).

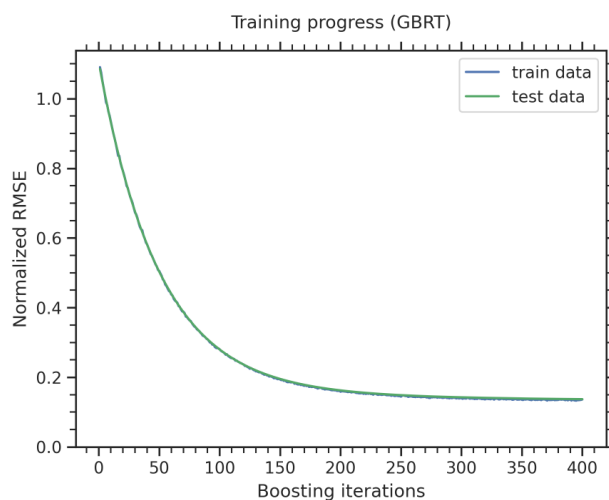


Fig. 18: Training progress of the GBRT model for the prediction of absolute GPP at the end of the 21st century (2091-2100) evaluated as normalized root mean square error on the training data (blue) and test data excluded from training (green).

16.4.2 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

16.4.3 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](#).

16.4.4 Variables

- *rlut* (atmos, monthly, longitude, latitude, time)
- *rsdt* (atmos, monthly, longitude, latitude, time)
- *rsut* (atmos, monthly, longitude, latitude, time)
- *tas* (atmos, monthly, longitude, latitude, time)

16.4.5 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.

16.4.6 Example plots

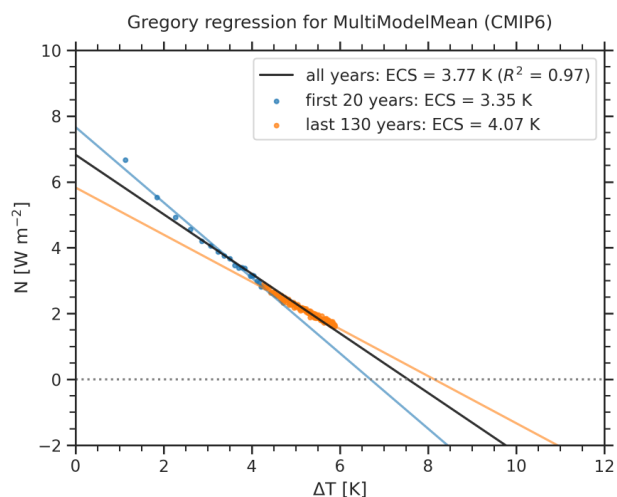


Fig. 19: ECS calculated for the CMIP6 models using the Gregory method over different time scales. Using the entire 150-year $4\times\text{CO}_2$ experiment (black line), there is an ECS value of 3.8 K; using only the first 20 years (blue dots and blue line), there is an ECS of 3.4 K; and using the last 130 years, there is an ECS of 4.1 K (orange dots and orange line).

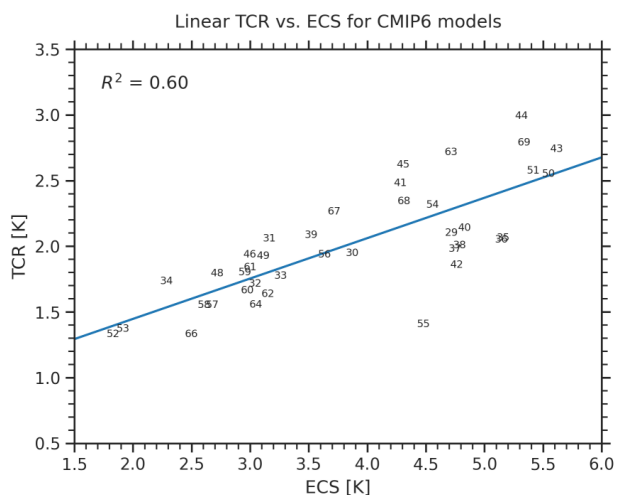


Fig. 20: TCR as a function of ECS for the CMIP6 models (black line is a linear fit). The R^2 values are given in the upper left parts of each panel. The numbers denote individual CMIP6 models.

16.5 Emergent constraints for equilibrium climate sensitivity

16.5.1 Overview

Calculates equilibrium climate sensitivity (ECS) versus

- 1) S index, D index and lower tropospheric mixing index (LTMI); similar to fig. 5 from Sherwood et al. (2014)
- 2) southern ITCZ index and tropical mid-tropospheric humidity asymmetry index; similar to fig. 2 and 4 from Tian (2015)
- 3) covariance of shortwave cloud reflection (Brient and Schneider, 2016)
- 4) climatological Hadley cell extent (Lipat et al., 2017)
- 5) temperature variability metric; similar to fig. 2 from Cox et al. (2018)
- 6) total cloud fraction difference between tropics and mid-latitudes; similar to fig. 3 from Volodin (2008)
- 7) response of marine boundary layer cloud (MBLC) fraction changes to sea surface temperature (SST); similar to fig. 3 of Zhai et al. (2015)
- 8) Cloud shallowness index (Brient et al., 2016)
- 9) Error in vertically-resolved tropospheric zonal average relative humidity (Su et al., 2014)

The results are displayed as scatterplots.

Note: The recipe `recipe_ecs_scatter.yml` requires pre-calculation of the equilibrium climate sensitivities (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 cdli-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`)
-

16.5.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_ecs_scatter.yml`
- `recipe_ecs_constraints.yml`

Diagnostics are stored in `diag_scripts`

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

16.5.3 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)

- calcm: 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: Additional 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](#).

16.5.4 Variables

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

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

16.5.5 Observations and reformat scripts

Note:

- (1) Obs4mips data can be used directly without any preprocessing.
 - (2) See headers of reformat scripts for non-obs4MIPs data for download instructions.
-

- AIRS (obs4MIPs): hus, husStderr
- AIRS-2-0 (obs4MIPs): hur
- CERES-EBAF (obs4MIPs): rsdt, rsut, rsutcs
- ERA-Interim (OBS6): hur, ta, va, wap
- GPCP-SG (obs4MIPs): pr
- HadCRUT4 (OBS): tasa
- HadISST (OBS): ts
- MLS-AURA (OBS6): hur
- TRMM-L3 (obs4MIPs): pr, prStderr

16.5.6 References

- Brient, F., and T. Schneider, J. Climate, 29, 5821-5835, doi:10.1175/JCLIM-D-15-0897.1, 2016.
- Brient et al., Clim. Dyn., 47, doi:10.1007/s00382-015-2846-0, 2016.
- Cox et al., Nature, 553, doi:10.1038/nature25450, 2018.
- Gregory et al., Geophys. Res. Lett., 31, doi:10.1029/2003GL018747, 2004.
- Lipat et al., Geophys. Res. Lett., 44, 5739-5748, doi:10.1002/2017GL73151, 2017.
- Sherwood et al., nature, 505, 37-42, doi:10.1038/nature12829, 2014.
- Su, et al., J. Geophys. Res. Atmos., 119, doi:10.1002/2014JD021642, 2014.
- Tian, Geophys. Res. Lett., 42, 4133-4141, doi:10.1002/2015GL064119, 2015.

- Volodin, Izvestiya, Atmospheric and Oceanic Physics, 44, 288-299, doi:10.1134/S0001433808030043, 2008.
- Zhai, et al., Geophys. Res. Lett., 42, doi:10.1002/2015GL065911, 2015.

16.5.7 Example plots

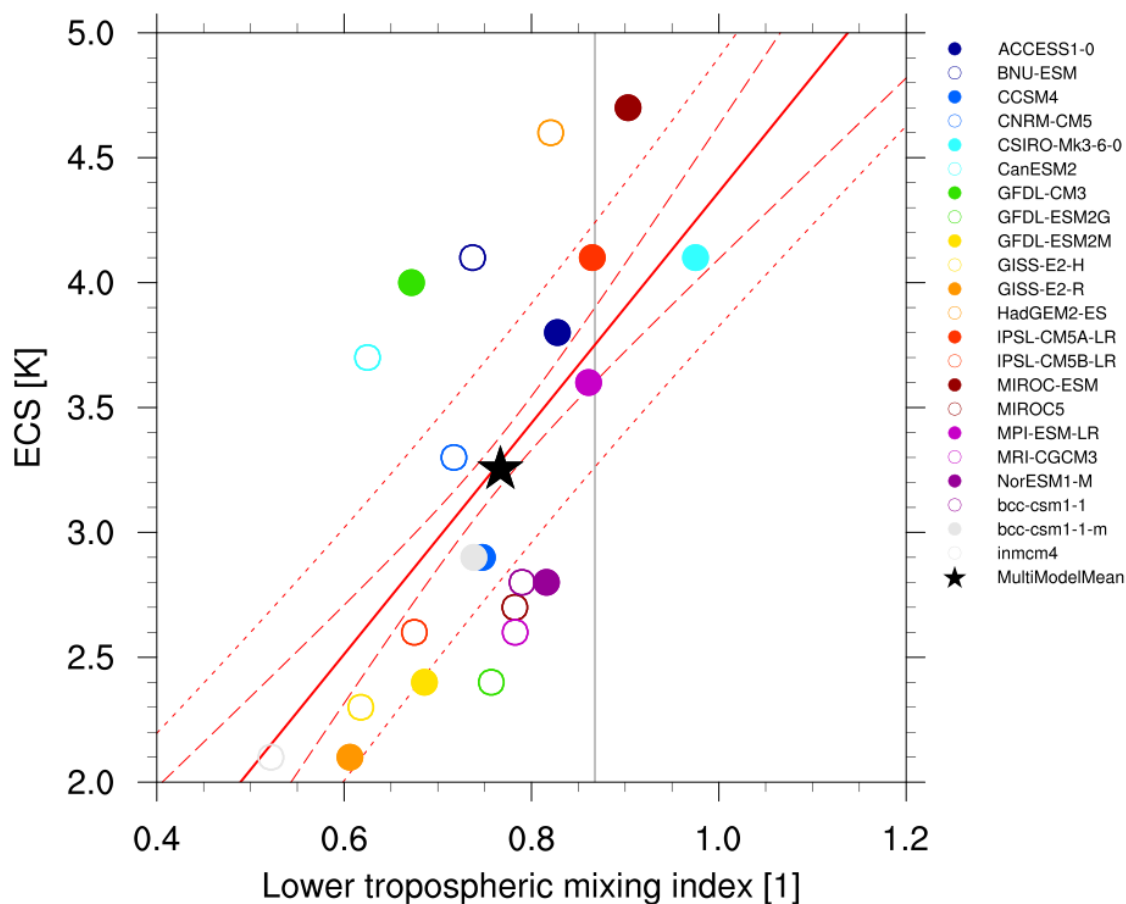


Fig. 21: Lower tropospheric mixing index (LTMI; Sherwood et al., 2014) vs. equilibrium climate sensitivity from CMIP5 models.

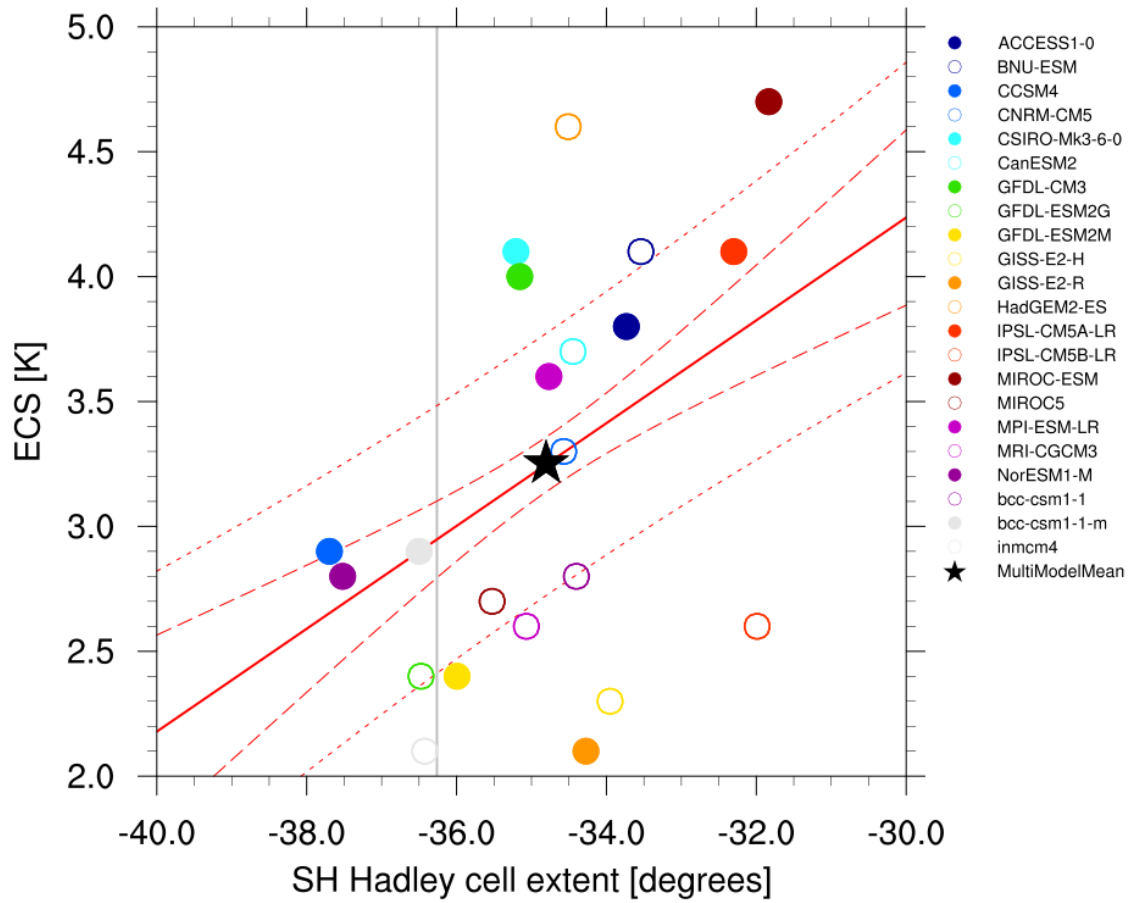


Fig. 22: Climatological Hadley cell extent (Lipat et al., 2017) vs. equilibrium climate sensitivity from CMIP5 models.

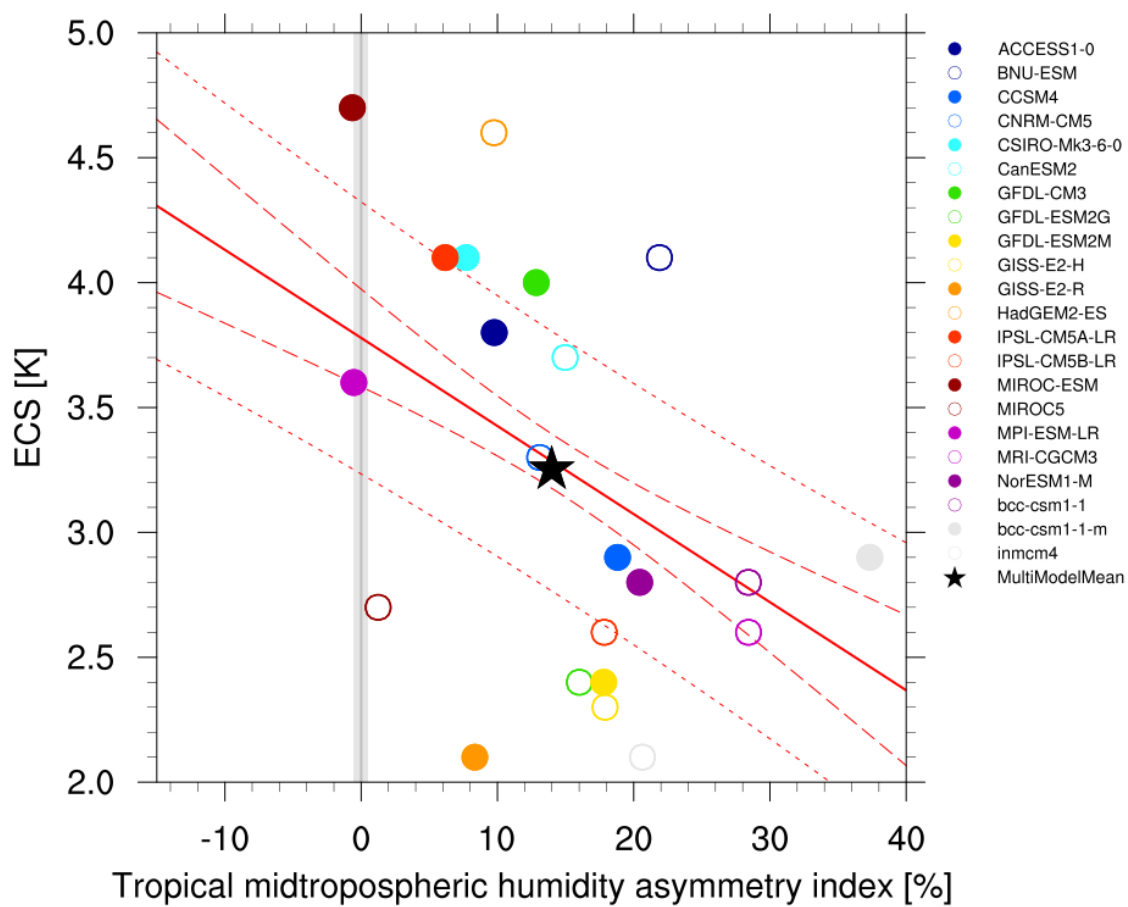


Fig. 23: Tropical mid-tropospheric humidity asymmetry index (Tian, 2015) vs. equilibrium climate sensitivity from CMIP5 models.

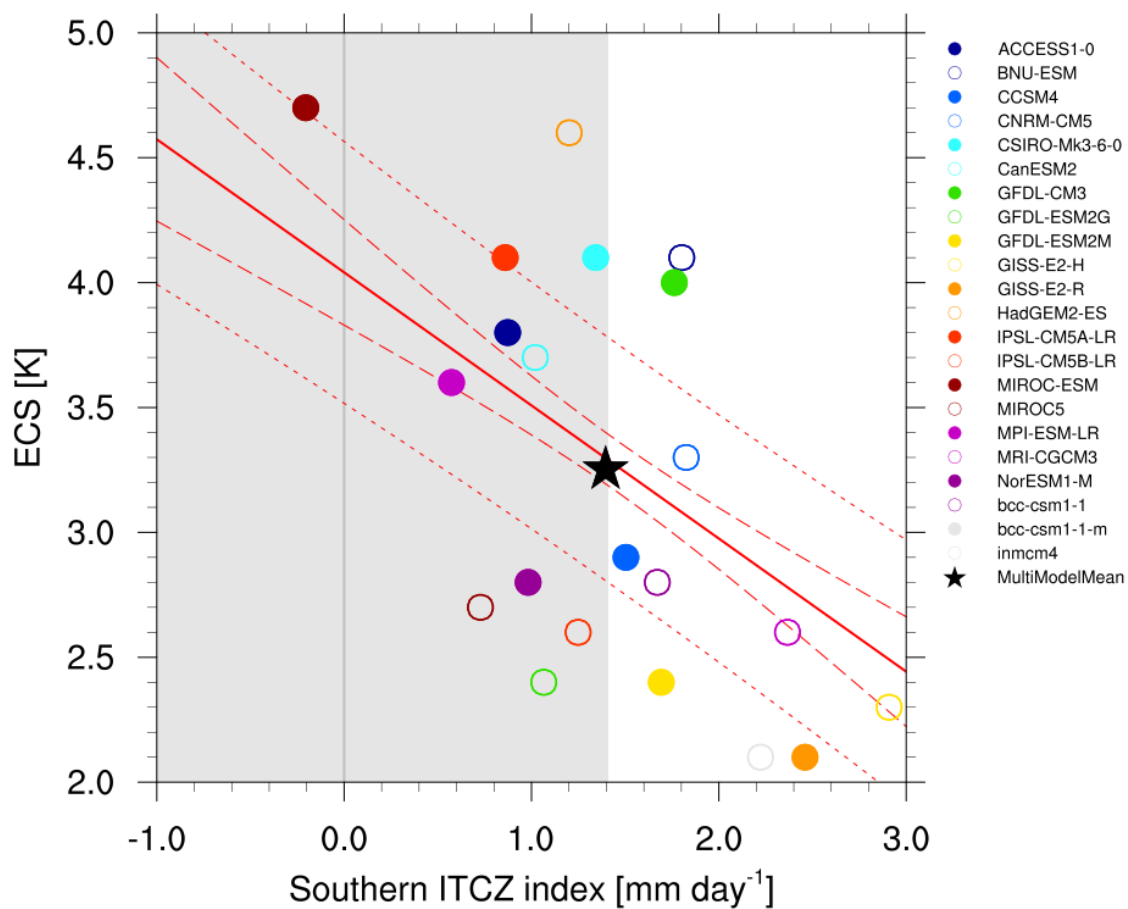


Fig. 24: Southern ITCZ index (Tian, 2015) vs. equilibrium climate sensitivity from CMIP5 models.

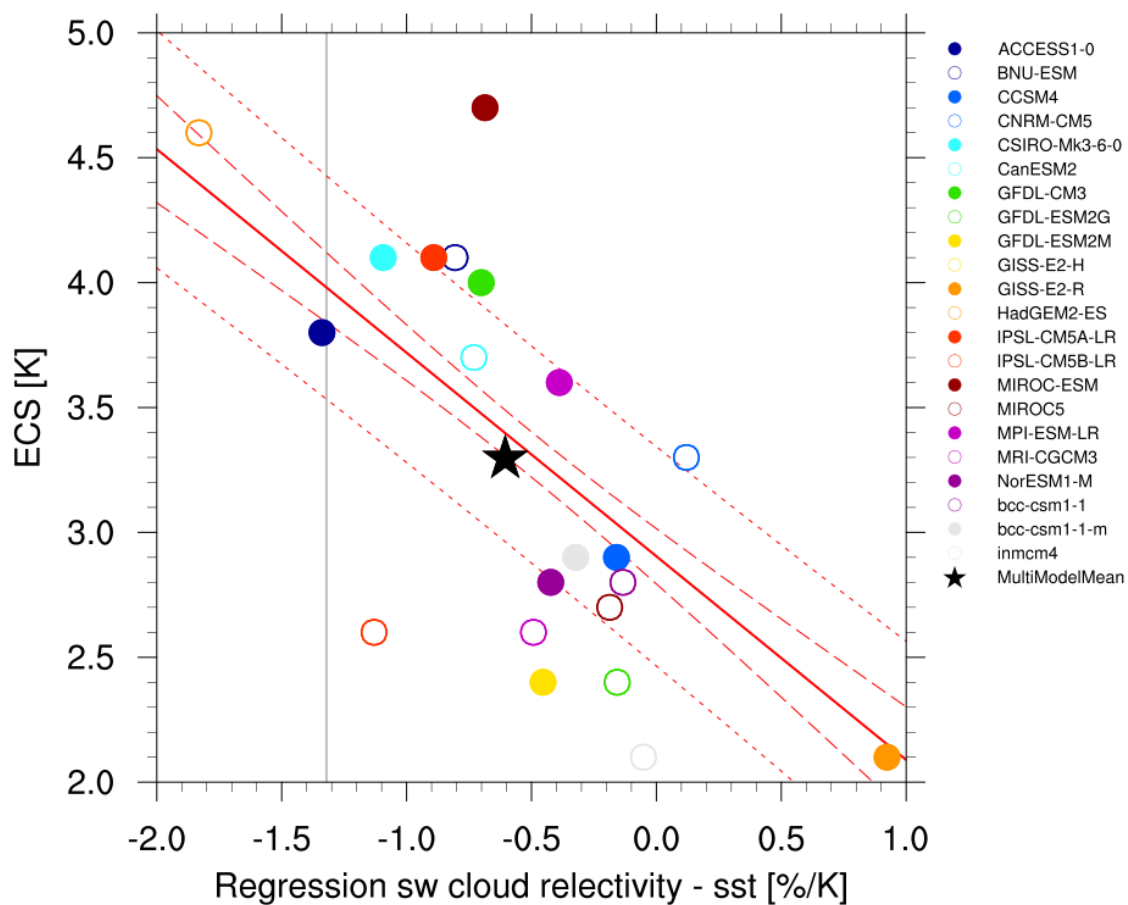


Fig. 25: Covariance of shortwave cloud reflection (Brient and Schneider, 2016) vs. equilibrium climate sensitivity from CMIP5 models.

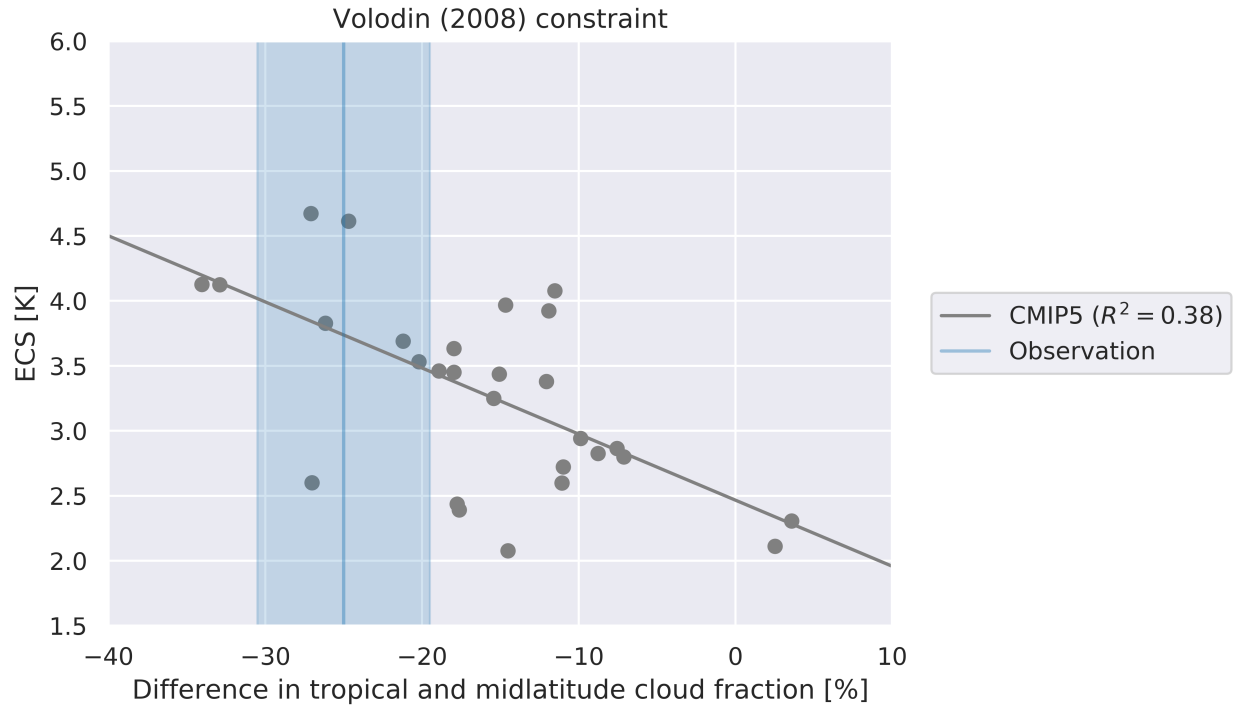


Fig. 26: Difference in total cloud fraction between tropics (28°S - 28°N) and Southern midlatitudes (56°S - 36°S) (Volodin, 2008) vs. equilibrium climate sensitivity from CMIP5 models.

16.6 Emergent constraints on carbon cycle feedbacks

16.6.1 Overview

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

16.6.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_wenzel14jgr.yml

Diagnostics are stored in diag_scripts/

- carbon_tsline.ncl: time line plots of annual means for spatial averages
- carbon_gammaHist.ncl: scatter plot of annual mean anomalies of two different variables; diagnosing and saving GammaIAV
- carbon_constraint.ncl: scatter plot of GammaLT vs. GammaIAV + line plot of probability density functions, diagnosing GammaLT

16.6.3 User settings

Note: Make sure to run this recipe setting `max_parallel_tasks: 1` in the `config_user.yml` file or using the CLI flag `--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 large 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 eruptions (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

16.6.4 Variables

- tas (atmos, monthly mean, longitude latitude time)
- nbp (land, monthly mean, longitude latitude time)
- fgco2 (ocean, monthly mean, longitude latitude time)

16.6.5 Observations and reformat scripts

- GCP2018: Global Carbon Budget including land (nbp) and ocean (fgco2) carbon fluxes
- NCEP: National Centers for Environmental Prediction reanalysis data for near surface temperature

16.6.6 References

- Cox, P. M., D. B. Pearson, B. B. Booth, P. Friedlingstein, C. C. Huntingford, C. D. B. Jones, and C. M. Luke, 2013, Sensitivity of tropical carbon to climate change constrained by carbon dioxide variability, *Nature*, 494(7437), 341-344. doi: 10.1038/nature11882
- Wenzel, S., P. M. Cox, V. Eyring, and P. Friedlingstein, 2014, Emergent Constraints on Climate Carbon Cycle Feedbacks in the CMIP5 Earth System Models, *JGR Biogeoscience*, 119(5), doi: 2013JG002591.

16.6.7 Example plots

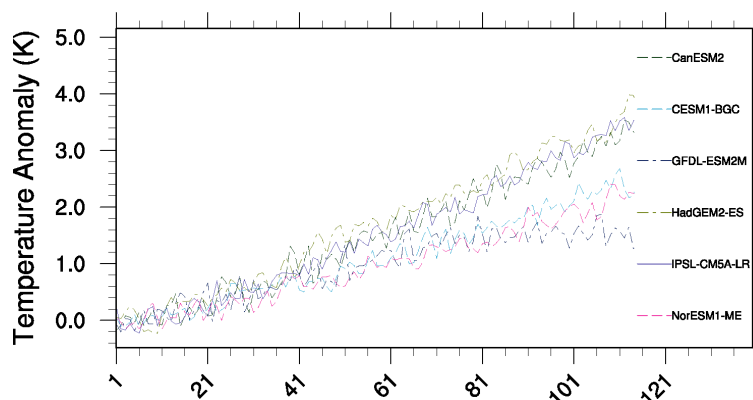


Fig. 27: Time series of tropical (30S to 30N) mean near surface temperature (tas) change between year 30 and year 110 for the CMIP5 models simulated with prescribed CO₂ (1%/yr CO₂ increase) coupled simulation (1pctCO₂).

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

16.7.1 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.

16.7.2 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`

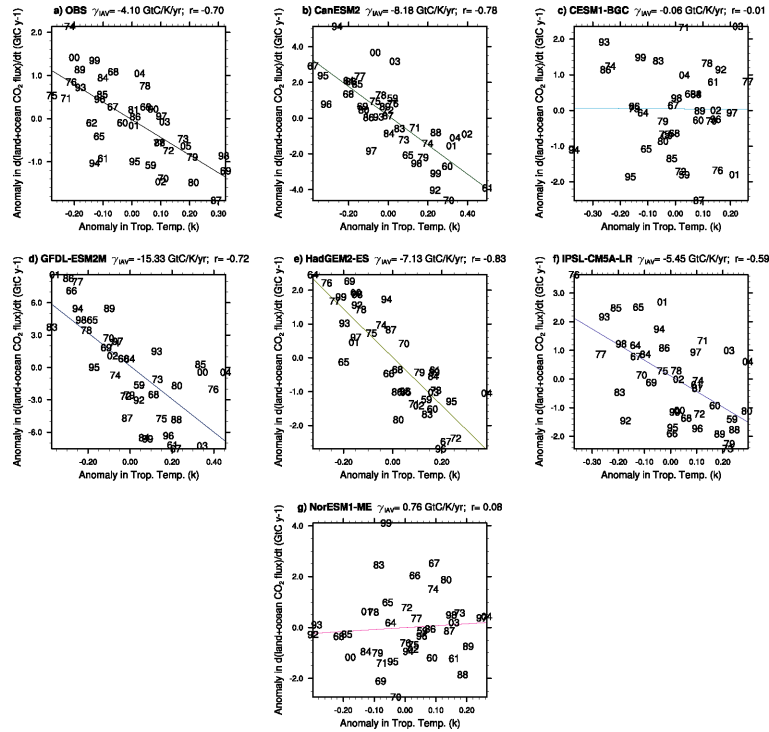


Fig. 28: Correlations between the interannual variability of global co2flux (nbpc+fgco2) and tropical temperature for the individual CMIP5 models using esmHistorical simulations, and for observations.

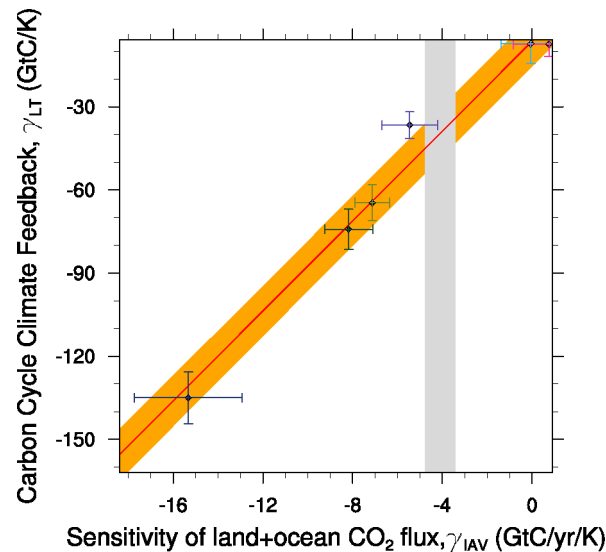


Fig. 29: Carbon cycle-climate feedback of tropical land carbon vs. the sensitivity of co2flux to interannual temperature variability in the tropics (30S to 30N). The red line shows the linear best fit of the regression together with the prediction error (orange shading) and the gray shading shows the observed range.

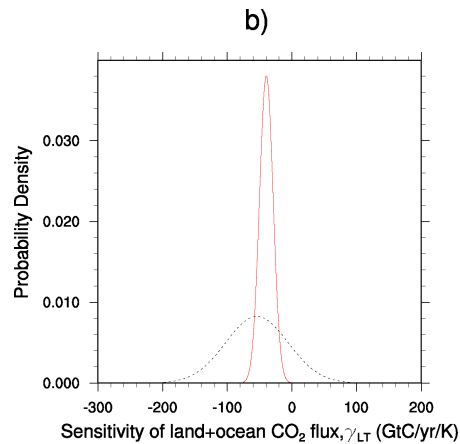


Fig. 30: Probability Density Functions for the pure CMIP5 ensemble (black dashed) and after applying the observed constraint to the models (red solid)

- [emergent_constraints/multiple_constraints.py](#)

More details on the emergent constraint module are given in the API documentation which is available [here](#).

16.7.3 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)

16.7.4 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*)

16.7.5 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.

16.7.6 Example plots

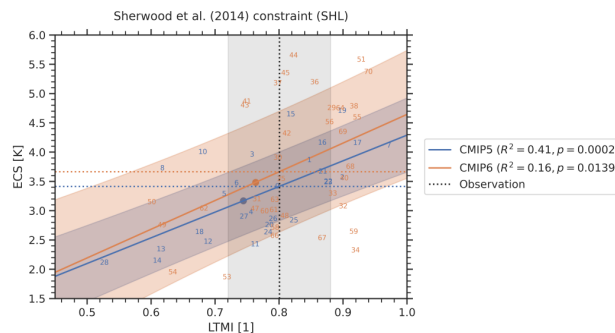


Fig. 31: Emergent relationship (solid blue and orange lines) of the Sherwood et al. (2014) emergent constraint, which is based on the lower tropospheric mixing index (LTMI). The numbers correspond to individual CMIP models. The shaded area around the regression line corresponds to the standard prediction 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.

16.8 Emergent constraint on equilibrium climate sensitivity from global temperature variability

16.8.1 Overview

This recipe reproduces the emergent constraint proposed by Cox et al. (2018) for the equilibrium climate sensitivity (ECS) using global temperature variability. The latter is defined by a metric which can be calculated from the global

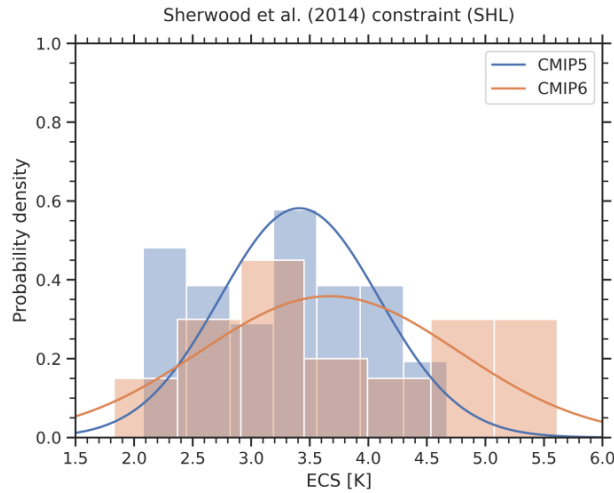


Fig. 32: Probability densities for the constrained ECS (solid lines) and the unconstrained model ensembles (histograms) of the emergent relationship shown in the figure above.

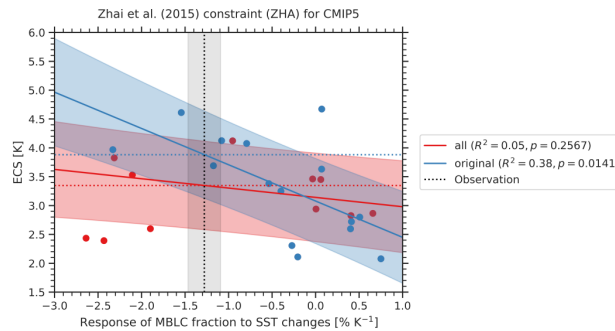


Fig. 33: Emergent relationship of the [Zhai et al. \(2015\)](#) emergent constraint for different subsets of CMIP5 models. Blue circles show the 15 CMIP5 models used in the original publication (except for CESM1-CAM5); the solid blue line and blue shaded area show the emergent relationships evaluated 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.

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).

16.8.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_cox18nature.yml`

Diagnostics are stored in `diag_scripts/`

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

16.8.3 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.

16.8.4 Variables

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

16.8.5 Observations and reformat scripts

- HadCRUT4 (*tasa*)

16.8.6 References

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

16.8.7 Example plots

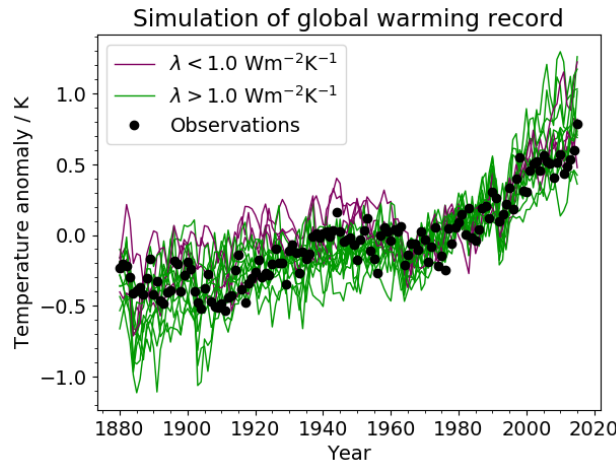


Fig. 34: Simulated change in global temperature from CMIP5 models (coloured lines), compared to the global temperature anomaly from the HadCRUT4 dataset (black dots). The anomalies are relative to a baseline period of 1961–1990. The model lines are colour-coded, with lower-sensitivity models ($> 1 \text{ Wm}^{-2}\text{K}^{-1}$) shown by green lines and higher-sensitivity models ($< 1 \text{ Wm}^{-2}\text{K}^{-1}$) shown by magenta lines.

16.9 Emergent constraint on snow-albedo effect

16.9.1 Overview

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

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

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

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

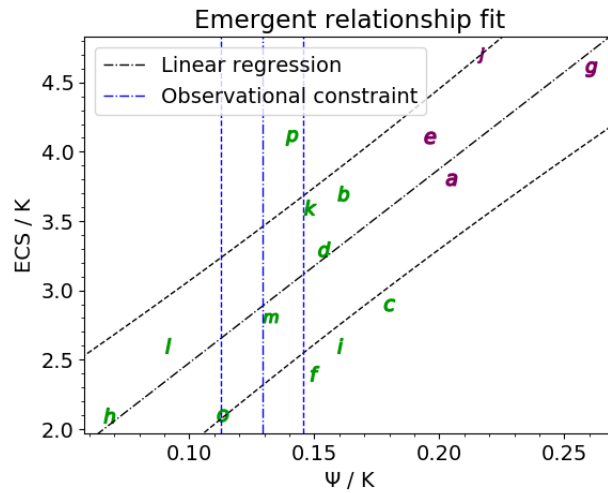


Fig. 35: Emergent relationship between ECS and the metric. The black dot-dashed line shows the best-fit linear regression across the model ensemble, with the prediction error for the fit given by the black dashed lines. The vertical blue lines show the observational constraint from the HadCRUT4 observations: the mean (dot-dashed line) and the mean plus and minus one standard deviation (dashed lines).

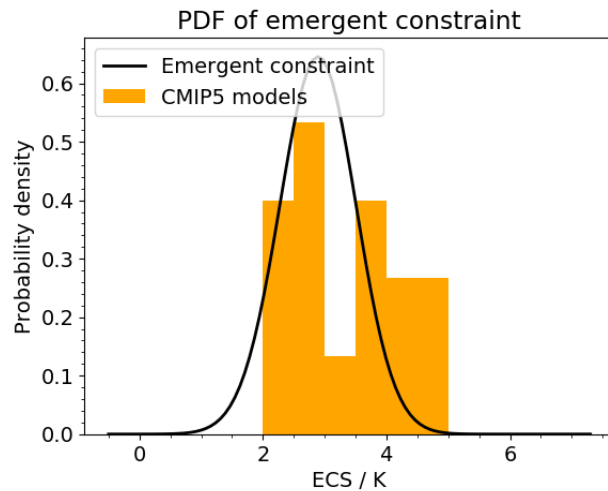


Fig. 36: The PDF for ECS. The orange histograms (both panels) show the prior distributions that arise from equal weighting of the CMIP5 models in 0.5 K bins.

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.

16.9.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_snowalbedo.yml

Diagnostics are stored in diag_scripts/emergent_constraints/

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

16.9.3 User settings in recipe

1. Script snowalbedo.ncl

Required settings for script

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

Optional settings for script

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

Required settings for variables

- ref_model: name of reference data set

Optional settings for variables

none

Required settings (scripts)

none

Optional settings (scripts)

16.9.4 Variables

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

16.9.5 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)

16.9.6 References

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

16.9.7 Example plots

16.10 Equilibrium climate sensitivity

16.10.1 Overview

Equilibrium climate sensitivity is defined as the change in global mean temperature as a result of a doubling of the atmospheric CO₂ 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.

16.10.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_ecs.yml

Diagnostics are stored in diag_scripts/

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

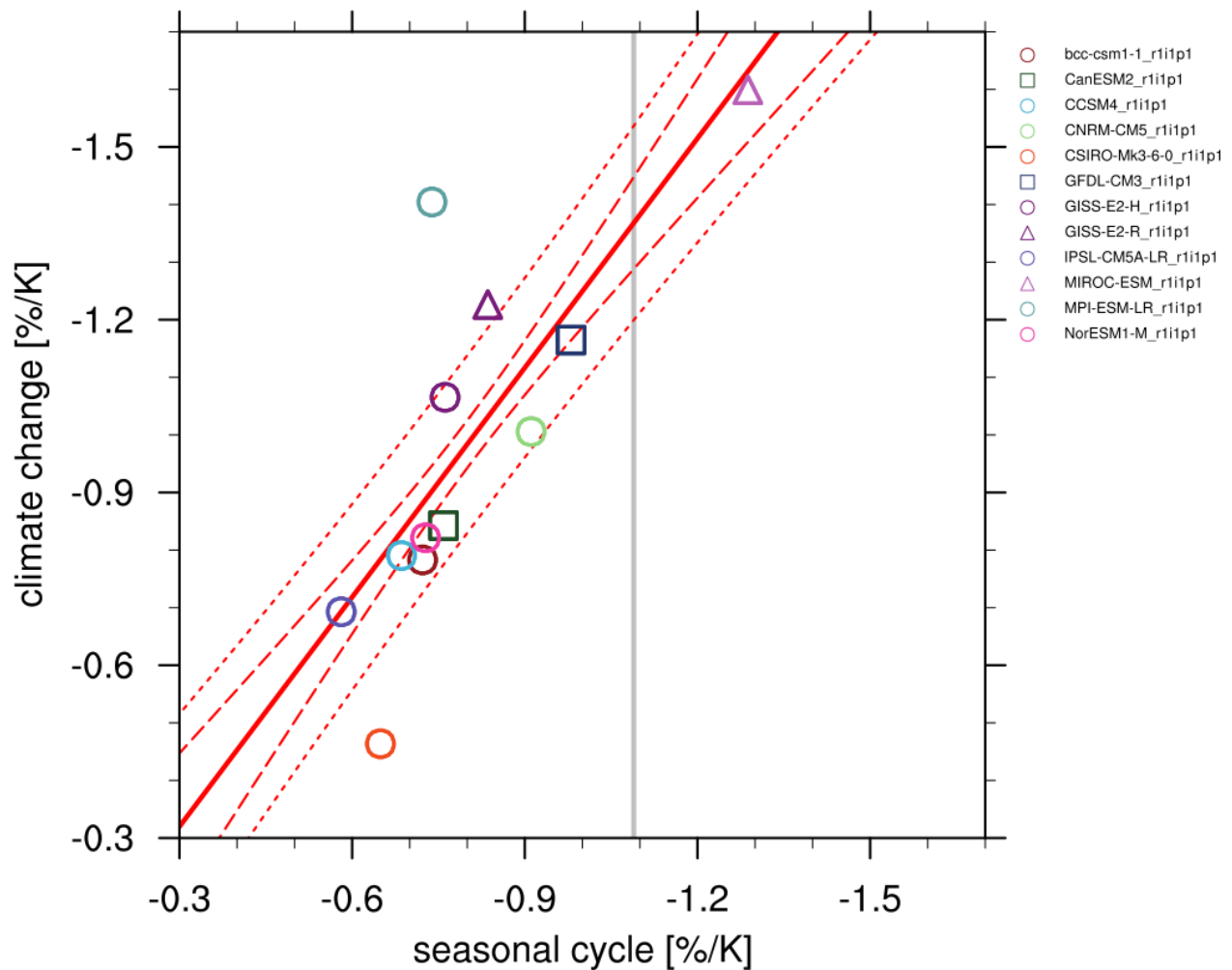


Fig. 37: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle in transient climate change experiments (CMIP5 historical experiments: 1901-2000, RCP4.5 experiments: 2101-2200). Similar to IPCC AR5 Chapter 9 (Flato et al., 2013), Figure 9.45a.

16.10.3 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()` (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()` (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()` (affects all plots).
 - `y_range`, *list of float*, optional: Range for the Y axis of the plot.

16.10.4 Variables

- *rlut* (atmos, monthly, longitude, latitude, time)
- *rsdt* (atmos, monthly, longitude, latitude, time)
- *rsut* (atmos, monthly, longitude, latitude, time)
- *tas* (atmos, monthly, longitude, latitude, time)

16.10.5 Observations and reformat scripts

None

16.10.6 References

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

16.10.7 Example plots

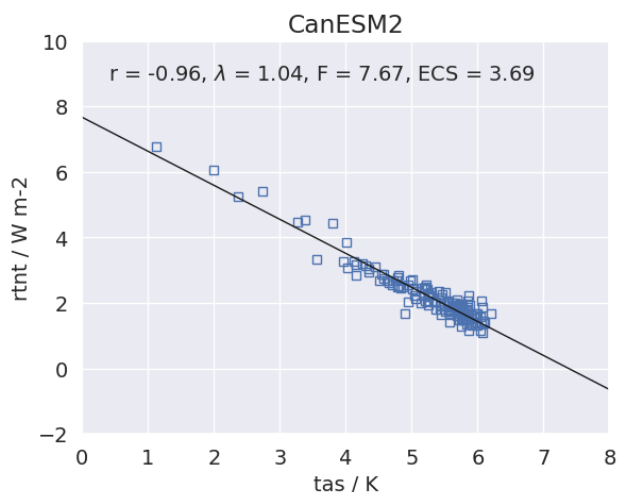


Fig. 38: Scatterplot between TOA radiance and global mean surface temperature anomaly for 150 years of the abrupt 4x CO₂ experiment including linear regression to calculate ECS for CanESM2 (CMIP5).

16.11 KNMI Climate Scenarios 2014

16.11.1 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](#).

16.11.2 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 in 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. Note that you can specify [multiple data paths](#) in the user configuration file.

16.11.3 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.

16.11.4 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:

```
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.

16.11.5 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)

16.11.6 References

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

16.11.7 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

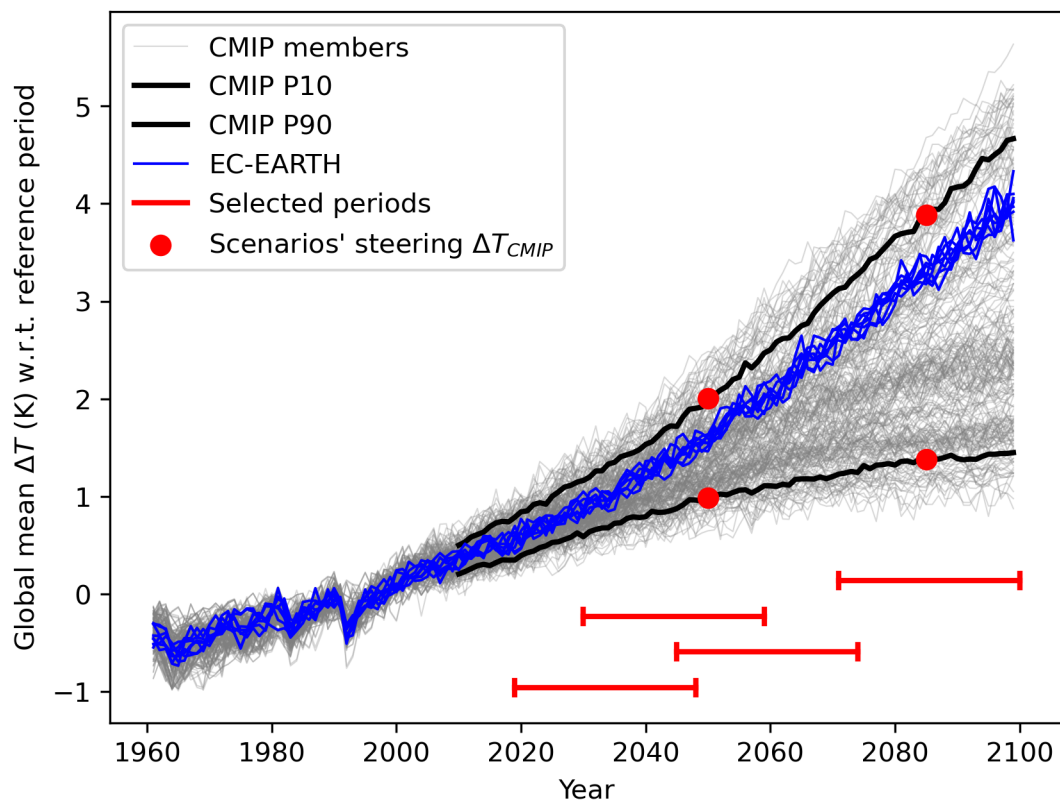
which is printed to the log file and also saved as a csv-file `scenarios.csv`. Additionally, a figure is created showing the CMIP spread in global temperature change, AND highlighting the selected steering parameters and resampling periods:

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 6	Segment 7
↪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			

(continues on next page)

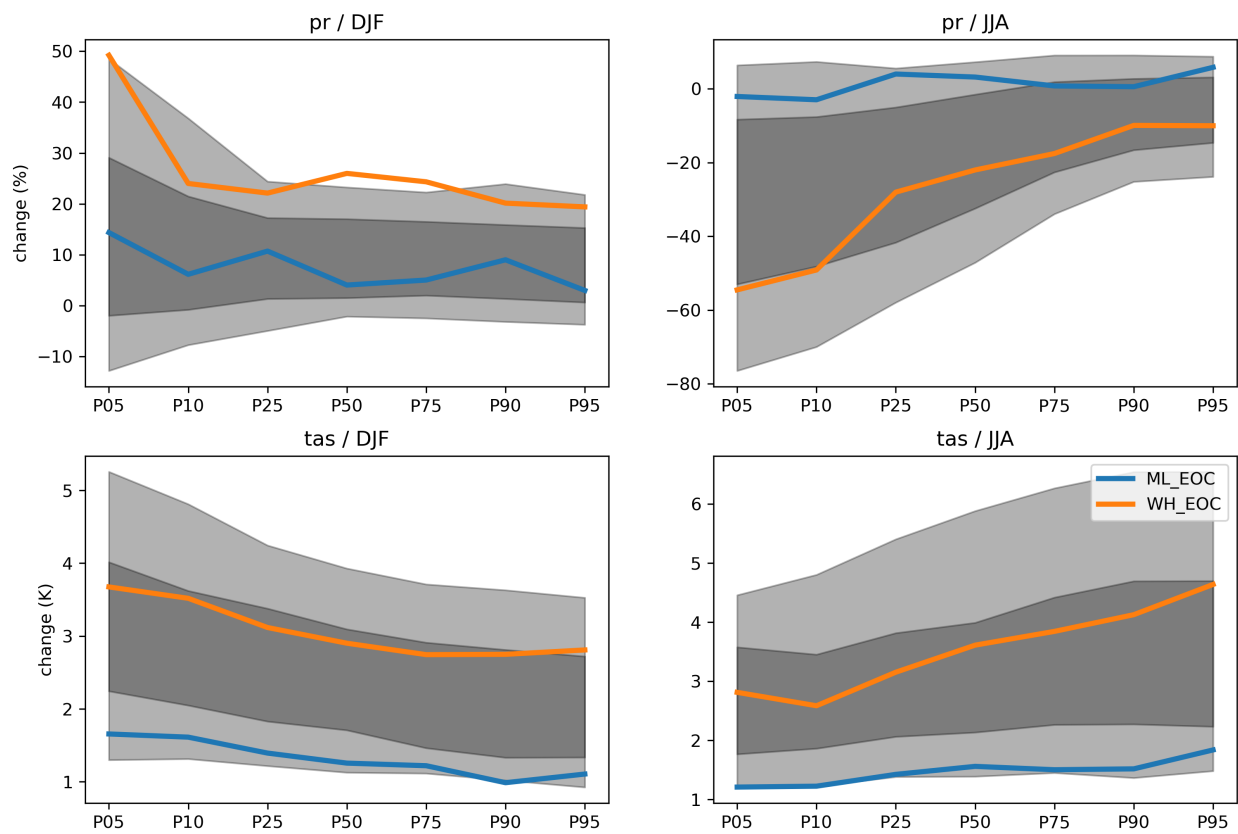


(continued from previous page)

Combination 1	0	3	0	4	3	2	└
→4	1	6	1	3	0		
Combination 2	2	4	3	7	4	2	└
→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.:

Year: 2085



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

16.12.1 Overview

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

The following plots are reproduced:

- Absolute correlation between the target variable and the diagnostics.
- Scatterplot between the target variable and the MDER-calculated linear combination of diagnostics.
- Boxplot of RMSE for the unweighted multi-model mean and the (MDER) weighted multi-model mean of the target variable in a pseudo-reality setup.
- Time series of the target variable for all models, observations and MDER predictions.
- Errorbar plots for all diagnostics.
- Scatterplots between the target variable and all diagnostics.

16.12.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_wenzel16jclim.yml`

Diagnostics are stored in `diag_scripts/`

- `austral_jet/asr.ncl`
- `austral_jet/main.ncl`
- `mder/absolute_correlation.ncl`
- `mder/regression_stepwise.ncl`
- `mder/select_for_mder.ncl`

16.12.3 User settings in recipe

1. Preprocessor

- `extract_region`: Region extraction.
- `extract_levels`: Pressure level extraction.
- `area_statistics`: Spatial average calculations.

2. Script `austral_jet/asr.ncl`

- `season, str`: Season.
- `average_ens, bool`, optional (default: `False`): Average over all given ensemble members of a climate model.
- `wdiag, array of str`, optional: Names of the diagnostic for MDER output. Necessary when MDER output is desired.
- `wdiag_title, array of str`, optional: Names of the diagnostic in plots.

3. Script austral_jet/main.ncl

- `styleset`, *str*: Style set used for plotting the multi-model plots.
- `season`, *str*: Season.
- `average_ens`, *bool*, optional (default: `False`): Average over all given ensemble members of a climate model.
- `rsondes`, *array of str*, optional: Additional observations used in the plot but not for MDER output.
- `rsondes_file`, *array of str*, optional: Paths to the additional observations Necessary when `rsondes` is given.
- `rsondes_yr_min`, *int*, optional: Minimum year for additional observations. Necessary when `rsondes` is given.
- `rsondes_yr_max`, *int*, optional: Maximum year for additional observations. Necessary when `rsondes` is given.
- `wdiag`, *array of str*, optional: Names of the diagnostic for MDER output. Necessary when MDER output is desired.
- `wdiag_title`, *array of str*, optional: Names of the diagnostic in plots.
- `derive_var`, *str*, optional: Derive variables using NCL functions. Must be one of "tpp", "mmstf".
- `derive_latrange`, *array of float*, optional: Latitude range for variable derivation. Necessary if `derive_var` is given.
- `derive_lev`, *float*, optional: Pressure level (given in *Pa*) for variable derivation. Necessary if `derive_var` is given.

4. Script mder/absolute_correlation.ncl

- `p_time`, *array of int*: Start years for future projections.
- `p_step`, *int*: Time range for future projections (in years).
- `scal_time`, *array of int*: Time range for base period (in years) for anomaly calculations used when `calc_type` = "trend".
- `time_oper`, *str*: Operation used in NCL `time_operation` function.
- `time_opt`, *str*: Option used in NCL `time_operation` function.
- `calc_type`, *str*: Calculation type for the target variable. Must be one of "trend", "pos", "int".
- `domain`, *str*: Domain tag for provenance tracking.
- `average_ens`, *bool*, optional (default: `False`): Average over all given ensemble members of a climate model.
- `region`, *str*, optional: Region used for area aggregation. Necessary if input of target variable is multidimensional.
- `area_oper`, *str*, optional: Operation used in NCL `area_operation` function. Necessary if multidimensional is given.
- `plot_units`, *str*, optional (attribute for `variable_info`): Units for the target variable used in the plots.

5. Script mder/regression_stepwise.ncl

- `p_time`, *array of int*: Start years for future projections.
- `p_step`, *int*: Time range for future projections (in years).

- `scal_time`, *array of int*: Time range for base period (in years) for anomaly calculations used when `calc_type = "trend"`.
- `time_oper`, *str*: Operation used in NCL `time_operation` function.
- `time_opt`, *str*: Option used in NCL `time_operation` function.
- `calc_type`, *str*: Calculation type for the target variable. Must be one of "trend", "pos", "int".
- `domain`, *str*: Domain tag for provenance tracking.
- `average_ens`, *bool*, optional (default: False): Average over all given ensemble members of a climate model.
- `smooth`, *bool*, optional (default: False): Smooth time period with 1-2-1 filter.
- `iter`, *int*, optional: Number of iterations for smoothing. Necessary when `smooth` is given.
- `cross_validation_mode`, *bool*, optional (default: False): Perform cross-validation.
- `region`, *str*, optional: Region used for area aggregation. Necessary if input of target variable is multidimensional.
- `area_oper`, *str*, optional: Operation used in NCL `area_operation` function. Necessary if multidimensional is given.
- `plot_units`, *str*, optional (attribute for `variable_info`): Units for the target variable used in the plots.

6. Script `mdcr/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".

16.12.4 Variables

- *ta* (atmos, monthly, longitude, latitude, pressure level, time)
- *uajet* (atmos, monthly, time)
- *va* (atmos, monthly, longitude, latitude, pressure level, time)
- *ps* (atmos, monthly, longitude, latitude, time)
- *asr* (atmos, monthly, longitude, latitude, time)

16.12.5 Observations and reformat scripts

- ERA-Interim (*ta*, *uajet*, *va*, *ps*)
- CERES-EBAF (*asr*)

16.12.6 References

- Wenzel, S., V. Eyring, E.P. Gerber, and A.Y. Karpechko: Constraining Future Summer Austral Jet Stream Positions in the CMIP5 Ensemble by Process-Oriented Multiple Diagnostic Regression. *J. Climate*, 29, 673–687, doi:10.1175/JCLI-D-15-0412.1, 2016.

16.12.7 Example plots

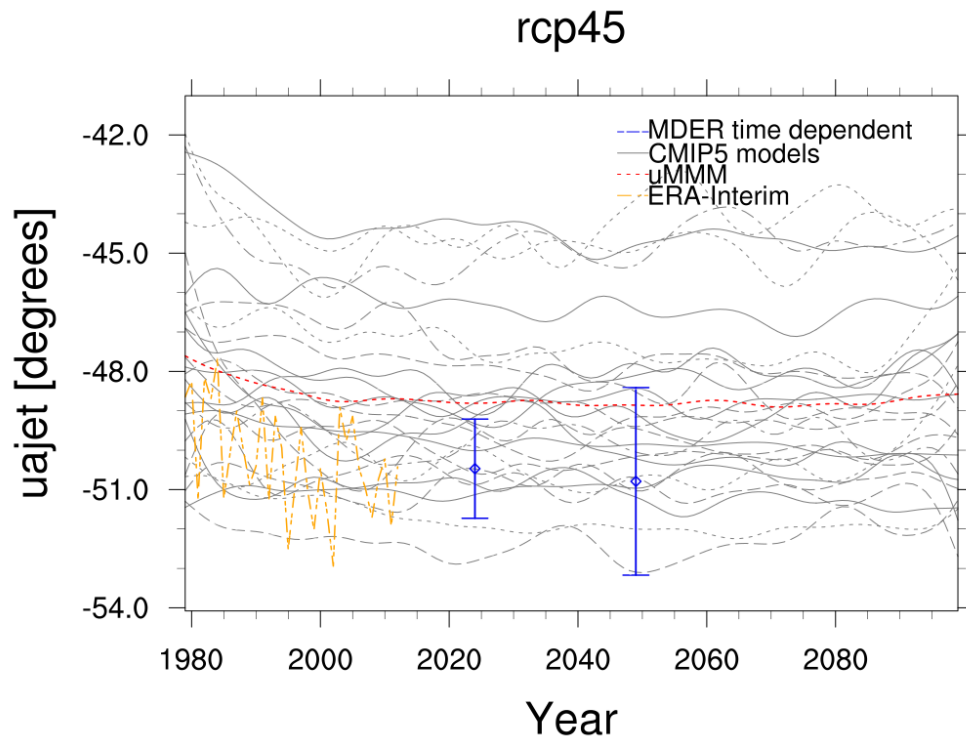


Fig. 39: Time series of the target variable (future austral jet position in the RCP 4.5 scenario) for the CMIP5 ensemble, observations, unweighted multi-model mean projections and (MDER) weighted multi-model mean projections.

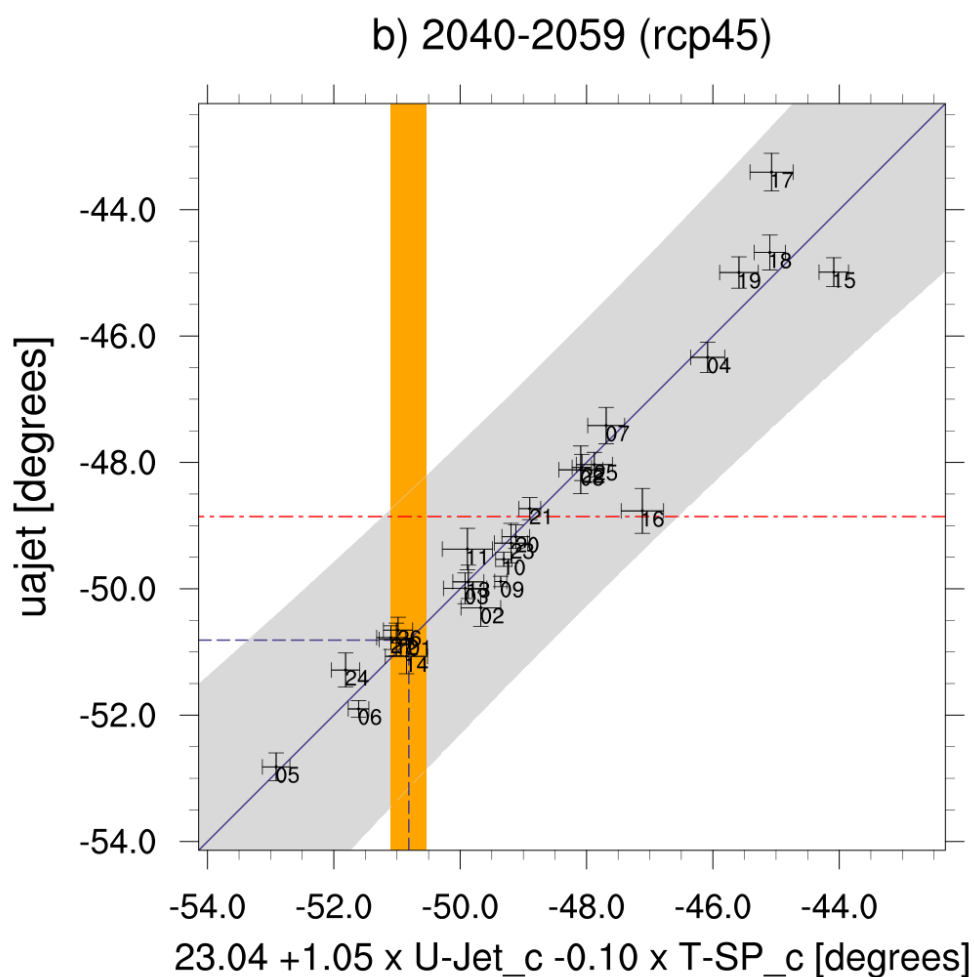


Fig. 40: Scatterplot of the target variable (future austral jet position in the RCP 4.5 scenario) vs. the MDER-determined linear combination of diagnostics for the CMIP5 ensemble.

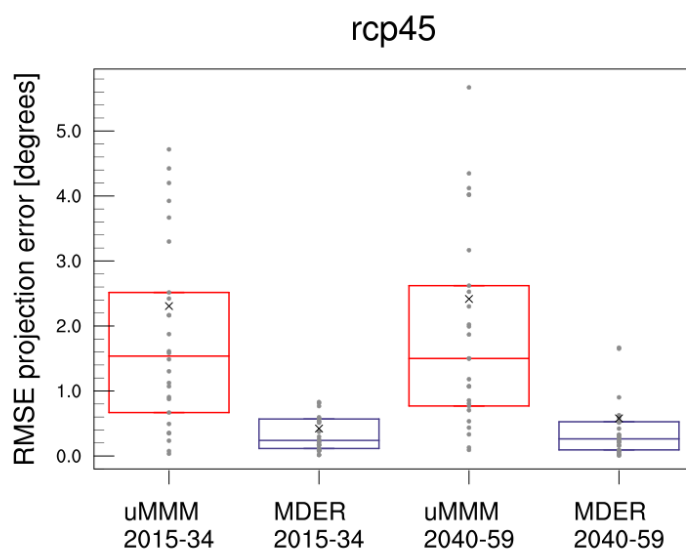


Fig. 41: Boxplot for the RMSE of the target variable for the unweighted and (MDER) weighted multi-model mean projections in a pseudo-reality setup.

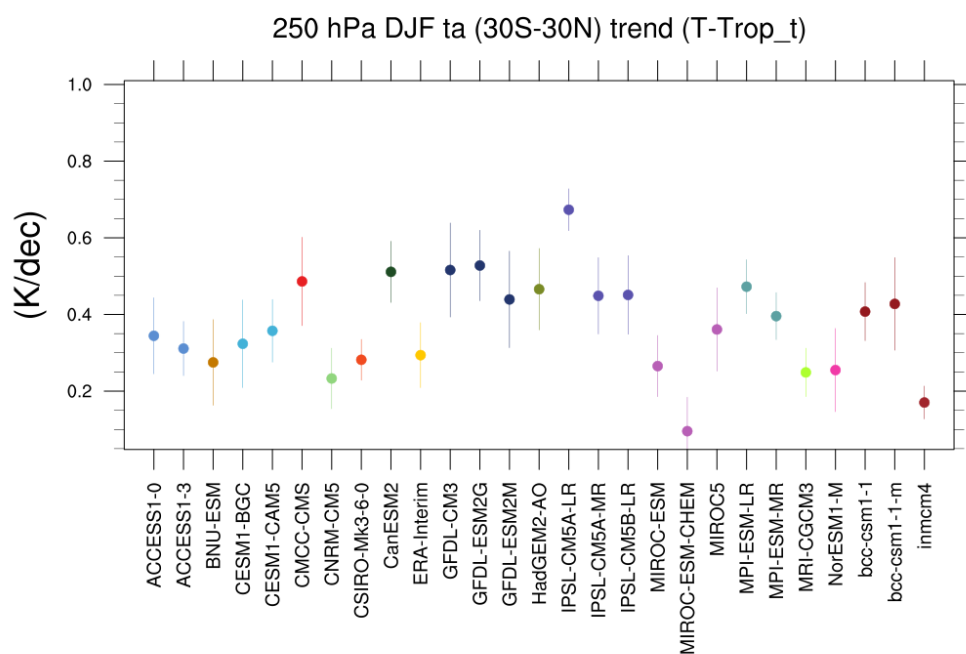


Fig. 42: Trends in tropical DJF temperature at 250hPa for different CMIP5 models and observations.

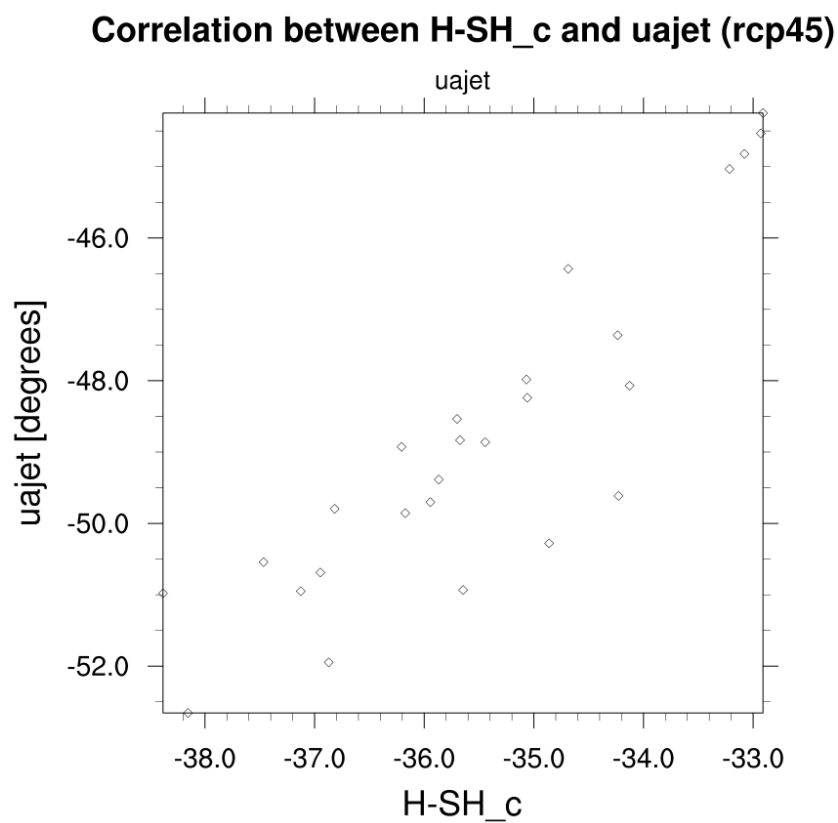


Fig. 43: Scatterplot of the target variable (future austral jet position in the RCP 4.5 scenario) vs. a single diagnostic, the historical location of the Southern hemisphere Hadley cell boundary for the CMIP5 ensemble.

16.13 Projected land photosynthesis constrained by changes in the seasonal cycle of atmospheric CO₂

16.13.1 Overview

Selected figures from [Wenzel et al. \(2016\)](#) are reproduced with `recipe_wenzel16nat.yml`. Gross primary productivity (gpp) and atmospheric CO₂ 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 CO₂ seasonal cycle amplitude due to rising atmospheric CO₂ levels. As a key figure of this recipe, the diagnosed values from the models beta vs. the change in CO₂ amplitude are compared in a scatter plot constituting an emergent constraint.

16.13.2 Available recipe and diagnostics

Recipes are stored in `recipes/`

- `recipe_wenzel16nat.yml`

Diagnostics are stored in `diag_scripts/carbon_ec/`

- `carbon_beta`: (1) scatter plot of annual gpp vs. annual CO₂ and (2) barchart of $\text{gpp}(2\text{xCO}_2)/\text{gpp}(1\text{xCO}_2)$; calculates beta for emergent constraint (`carbon_co2_cycle.ncl`)
- `carbon_co2_cycle.ncl`: (1) scatter plot of CO₂ amplitude vs. annual CO₂, (2) barchart of sensitivity of CO₂ amplitude to CO₂, (3) emergent constraint: $\text{gpp}(2\text{xCO}_2)/\text{gpp}(1\text{xCO}_2)$ vs. sensitivity of CO₂ amplitude to CO₂, (4) probability density function of constrained and unconstrained sensitivity of CO₂ amplitude to CO₂

16.13.3 User settings

Note: Make sure to run this recipe setting `output_file_type: pdf` in the `config_user.yml` file or using the CLI flag `--output_file_type=pdf`.

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`)
- `stylesheet`: 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

16.13.4 Variables

- `co2s` (atmos, monthly mean, plev longitude latitude time)
- `gpp` (land, monthly mean, longitude latitude time)

16.13.5 Observations and reformat scripts

- ESRL: Earth System Research Laboratory, ground-based CO₂ measurements

16.13.6 References

- Wenzel, S., Cox, P., Eyring, V. et al., 2016, Projected land photosynthesis constrained by changes in the seasonal cycle of atmospheric CO₂. Nature 538, 499501, doi: doi.org/10.1038/nature19772

16.13.7 Example plots

16.14 Transient Climate Response

16.14.1 Overview

The transient climate response (TCR) is defined as the global and annual mean surface air temperature anomaly in the *IpctCO2* scenario (1% CO₂ increase per year) for a 20 year period centered at the time of CO₂ 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 *IpctCO2* experiment prior to the TCR calculation (see [Gregory and Forster, 2008](#)).

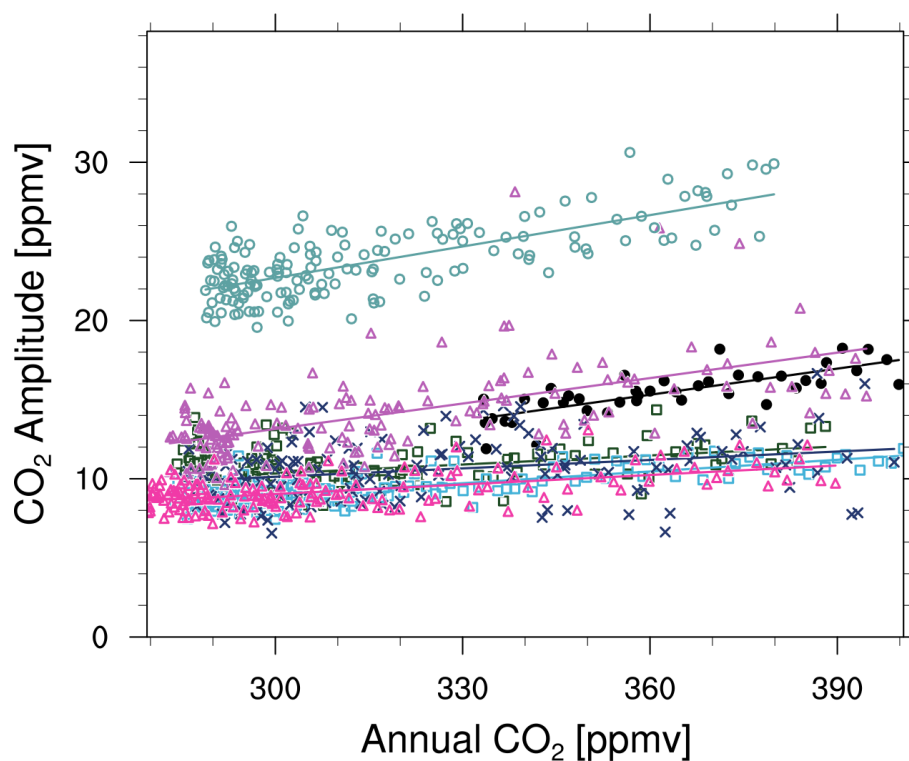


Fig. 44: Comparison of CO₂ seasonal amplitudes for CMIP5 historical simulations and observations showing annual mean atmospheric CO₂ versus the amplitudes of the CO₂ seasonal cycle at Pt. Barrow, Alaska (produced with carbon_co2_cycle.ncl, similar to Fig. 1a from Wenzel et al. (2016)).

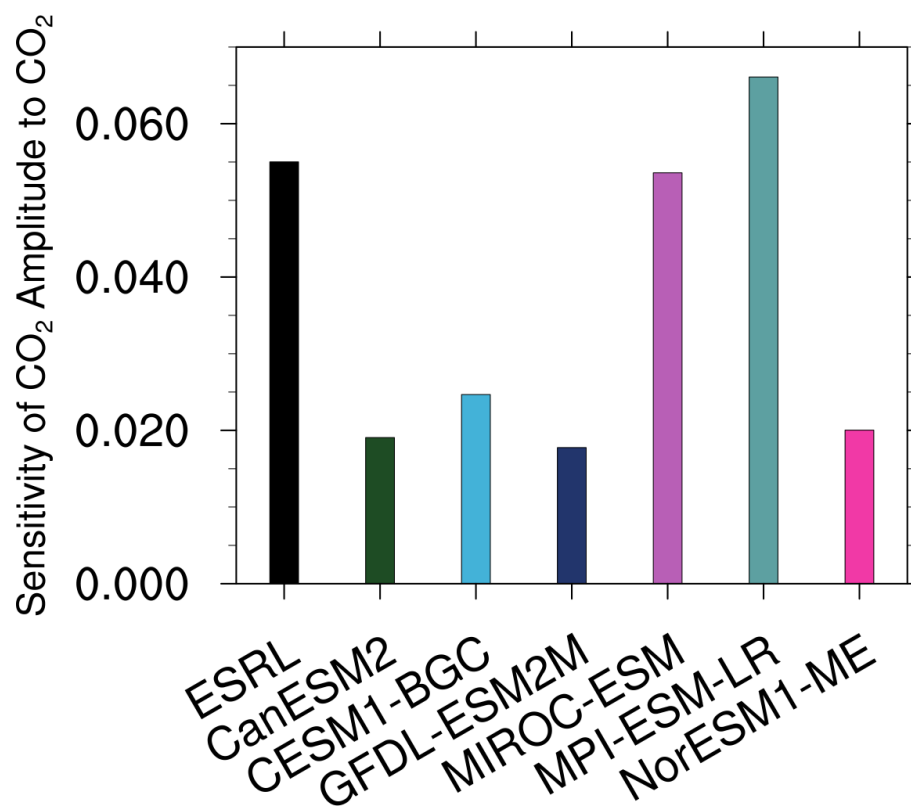


Fig. 45: Barchart showing the gradient of the linear correlations for the comparison of CO₂ seasonal amplitudes for CMIP5 historical for at Pt. Barrow, Alaska (produced with carbon_co2_cycle.ncl, similar to Fig. 1b from Wenzel et al. (2016)).

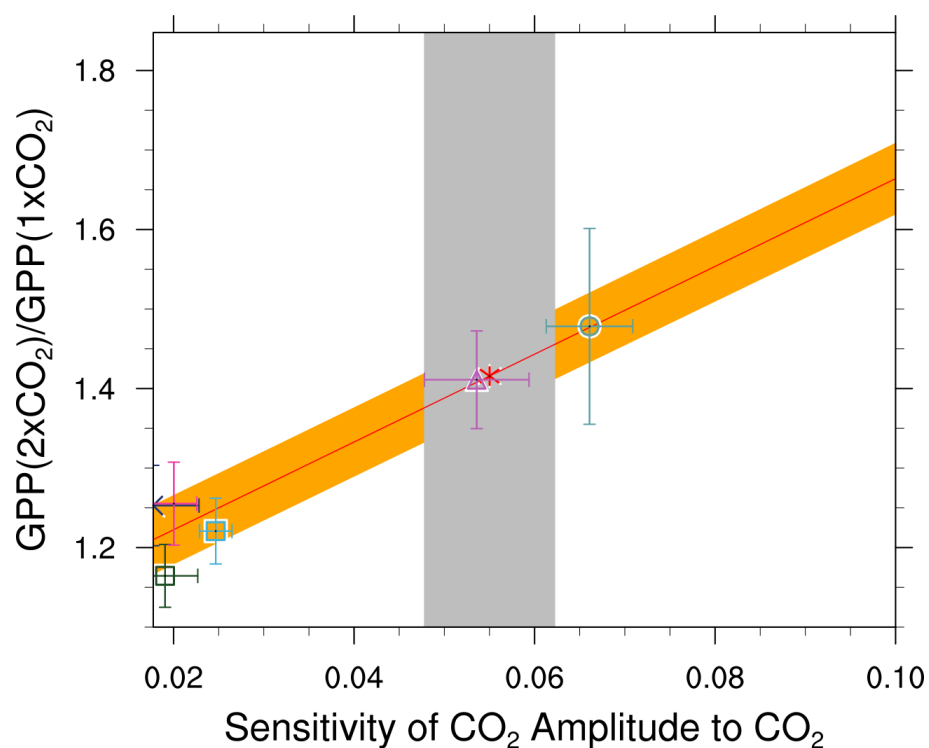


Fig. 46: Emergent constraint on the relative increase of large-scale GPP for a doubling of CO₂, showing the correlations between the sensitivity of the CO₂ amplitude to annual mean CO₂ increases at Pt. Barrow (x-axis) and the high-latitude (60N - 90N) CO₂ fertilization on GPP at 2xCO₂. The red line shows 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)).

16.14.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_tcr.yml`

Diagnostics are stored in `diag_scripts/`

- `climate_metrics/tcr.py`
- `climate_metrics/create_barplot.py`
- `climate_metrics/create_scatterplot.py`

16.14.3 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.
 - `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()` (affects all plots).
- Script `climate_metrics/create_barplot.py`

See [here](#).
- Script `climate_metrics/create_scatterplot.py`

See [here](#).

16.14.4 Variables

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

16.14.5 Observations and reformat scripts

None

16.14.6 References

- Gregory, J. M., and P. M. Forster. “Transient climate response estimated from radiative forcing and observed temperature change.” *Journal of Geophysical Research: Atmospheres* 113.D23 (2008).

16.14.7 Example plots

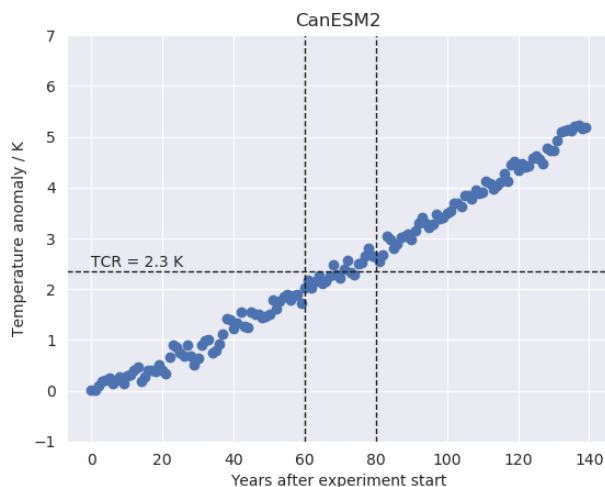


Fig. 47: Time series of the global mean surface air temperature anomaly (relative to the linear fit of the pre-industrial control run) of CanESM2 (CMIP5) for the 1% CO₂ increase per year experiment. The horizontal dashed line indicates the transient climate response (TCR) defined as the 20 year average temperature anomaly centered at the time of CO₂ doubling (vertical dashed lines).

16.15 Climate model projections from the ScenarioMIP of CMIP6

16.15.1 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>

16.15.2 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.

16.15.3 User settings in recipe

1. Script `calc_timeseries_across_realization_stddev_runave.ncl`

Required settings for script

- `scenarios`: list with scenarios included in figure
- `years`: 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 stippling 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
- syyears: list with start years of time periods to include in the table
- eyyears: 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: minimum 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: minimum 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: minimum 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: minimum 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: minimum value on y-axis, default calculated from data
- ymax: maximum value on y-axis
- colormap: alternative colormap, path to rgb file or ncl name

16.15.4 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.

16.15.5 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., Khari, S., Koshiro, T., Ma, L., Olivie, 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>

16.15.6 Example plots

16.16 Climate Change Hotspot

16.16.1 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.

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 SWm^{-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.

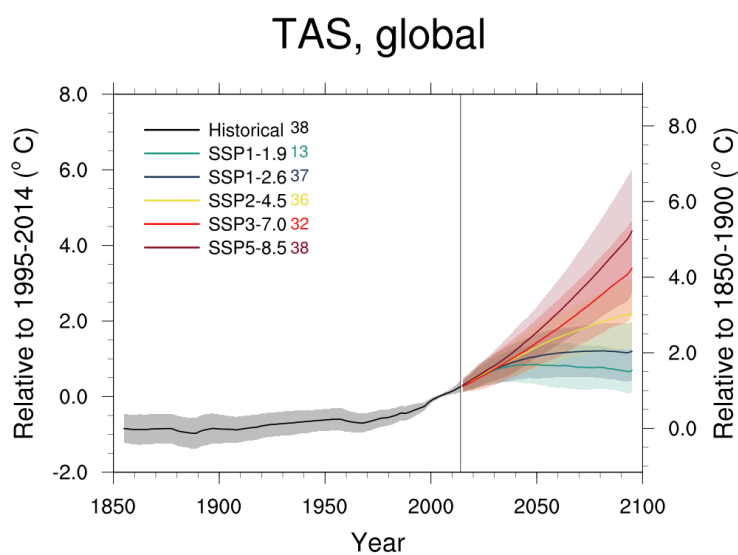


Fig. 48: Global average temperature time series (11-year running averages) of changes from current baseline (1995–2014, left axis) and pre-industrial baseline (1850–1900, right axis, obtained by adding a 0.84 °C offset) for SSP1-1.9, SSP1-2.6, SSP2-4.5, SSP3-7.0 and SSP5-8.5.

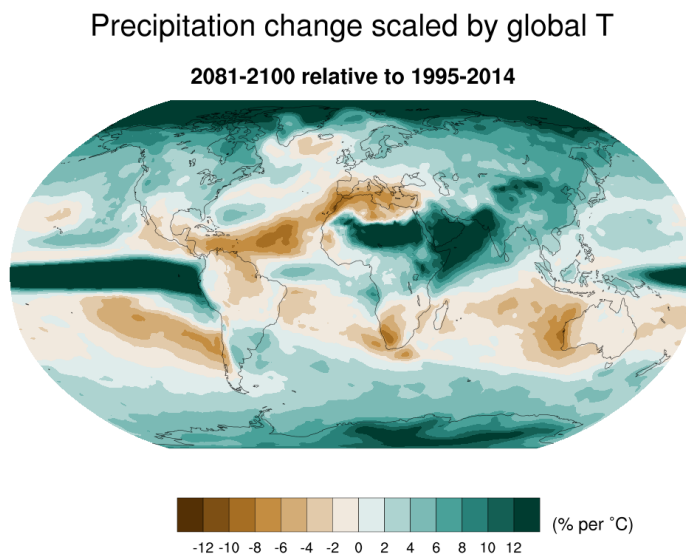


Fig. 49: Patterns of temperature (a) and percent precipitation change (b) normalized by global average temperature change (averaged across CMIP6 models and all Tier 1 plus SSP1-1.9 scenarios).

16.16.2 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. \(2022\)](#).].

16.16.3 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.
- **`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].

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. 50: 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.

- `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.

16.16.4 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
    preprocessor: ensemble_members
    additional_datasets: *cmip6_85
scripts:
```

(continues on next page)

(continued from previous page)

```
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.

16.16.5 Variables

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

16.16.6 References

- Cos et al. 2022, Earth Syst. Dynam., 13, 321–340

16.16.7 Example plots

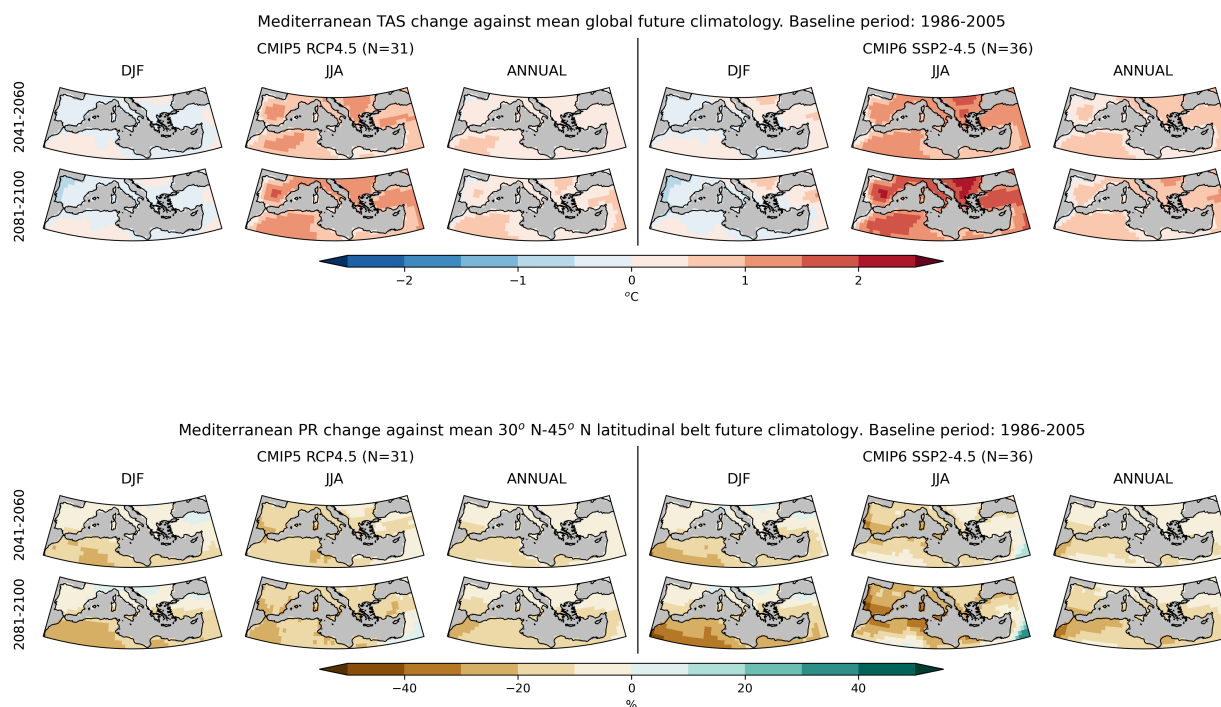


Fig. 51: Mediterranean region temperature (upper rows) and precipitation (lower rows) change differences against the mean global temperature change and the mean 30–45° N latitudinal belt precipitation change respectively. The changes for the periods 2041–2060 (first and third row) and 2081–2100 (second 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.

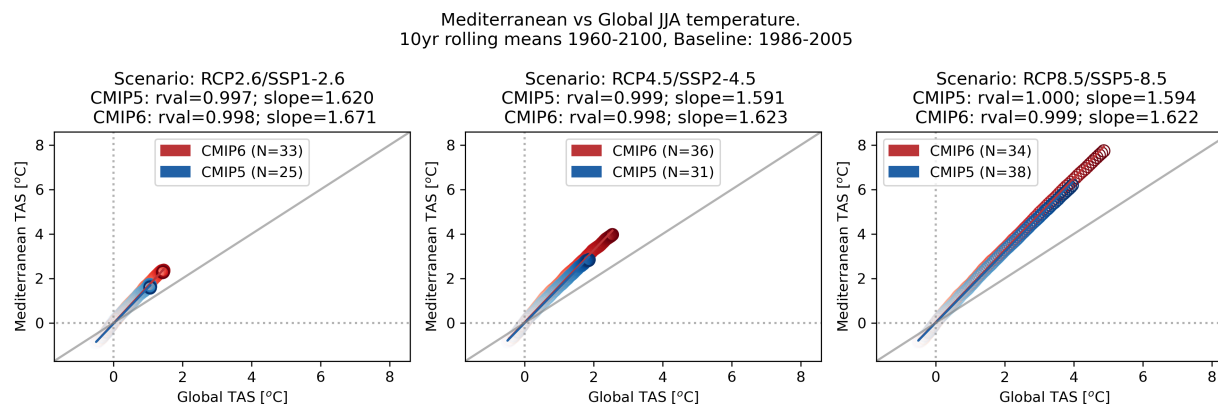


Fig. 52: Mediterranean region warming against global warming for the summer 2.6, 4.5 and 8.5 Wm^{-2} RCP and SSP scenarios for the CMIP5 and CMIP6 ensemble means. Each dot represents a 10-year mean change beginning from the period 1960-1969 (light colouring) until 2091-2100 (opaque coloring). The changes are computed with 1986-2005 as the baseline. An ordinary least squares linear regression is computed and the slope and r^2 values are shown. N indicates the number of models included in the ensemble mean.

17.1 IPCC AR6 Chapter 3 (selected figures)

17.1.1 Overview

This recipe collects selected diagnostics used in IPCC AR6 WGI Chapter 3: Human influence on the climate system. 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”.

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 3.13: Precipitation - Model Bias
- Figure 3.15: Precipitation Anomaly

17.1.2 Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/ipccwg1ar6ch3/`

- `recipe_ipccwg1ar6ch3_atmosphere.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. 3.13:

- ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl: See [here](#).
- ipcc_ar6/model_bias.ncl

Fig. 3.15:

- ipcc_ar6/precip_anom.ncl

17.1.3 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/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

5. 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

6. 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

17.1.4 Variables

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

17.1.5 Observations and reformat scripts

- BerkeleyEarth (tasa - esmvaltool/cmorizers/data/formatters/datasets/berkeleyearth.py)
- CRU (pr - esmvaltool/cmorizers/data/formatters/datasets/cru.py)
- ERA5 (tas - ERA5 data can be used via the native6 project)
- GHCN (pr - esmvaltool/cmorizers/data/formatters/datasets/ghcn.ncl)
- GPCP-SG (pr - obs4MIPs)
- HadCRUT5 (tasa - esmvaltool/cmorizers/data/formatters/datasets/hadcrut5.py)
- Kadow2020 (tasa - esmvaltool/cmorizers/data/formatters/datasets/kadow2020.py)
- NOAAGlobalTemp (tasa - esmvaltool/cmorizers/data/formatters/datasets/noaaglobaltemp.py)

17.1.6 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 University Press. In Press.

17.1.7 Example plots

17.2 IPCC AR5 Chapter 9 (selected figures)

17.2.1 Overview

The goal of this recipe is to collect diagnostics to reproduce Chapter 9 of AR5, so that the plots can be readily reproduced and compared to previous CMIP versions. In this way we can next time start with what was available in the previous round and can focus on developing more innovative methods of analysis rather than constantly having to “re-invent the wheel”.

The plots are produced collecting the diagnostics from individual recipes. The following figures from Flato et al. (2013) can currently be reproduced:

- Figure 9.2 a,b,c: Annual-mean surface air temperature for the period 1980–2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from ERA-Interim (Dee et al., 2011), c) mean absolute model error with respect to the climatology from ERA-Interim.
- Figure 9.3: Seasonality (December–January–February minus June–July–August) of surface (2 m) air temperature (°C) for the period 1980–2005. (a) Multi-model mean for the historical experiment. (b) Multi-model mean of absolute seasonality. (c) Difference between the multi-model mean and the ERA-Interim reanalysis seasonality. (d) Difference between the multi-model mean and the ERA-Interim absolute seasonality.
- Figure 9.4: Annual-mean precipitation rate (mm day⁻¹) for the period 1980–2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from the Global Precipitation Climatology Project (Adler et al., 2003), c) multi-model mean absolute error with respect to observations, and d) multi-model mean error relative to the multi-model mean precipitation itself.
- Figure 9.5: Climatological (1985–2005) annual-mean cloud radiative effects in Wm⁻² for the CMIP5 models against CERES EBAF (2001–2011) in Wm⁻². Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model-mean biases against CERES EBAF 2.6 are shown on the left, whereas the right panels show zonal averages from CERES EBAF 2.6 (black), the individual CMIP5 models (thin gray lines), and the multi-model mean (thick red line).
- Figure 9.6: Centred pattern correlations between models and observations for the annual mean climatology over the period 1980–1999. Results are shown for individual CMIP3 (black) and CMIP5 (blue) models as thin dashes, along with the corresponding ensemble average (thick dash) and median (open circle). The four variables shown are surface air temperature (TAS), top of the atmosphere (TOA) outgoing longwave radiation (RLUT), precipitation (PR) and TOA shortwave cloud radiative effect (SW CRE). The correlations between the reference and alternate observations are also shown (solid green circles).
- Figure 9.8: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1961–1990 time-mean of each individual time series. The reference period 1961–1990 is indicated by yellow shading; vertical dashed grey lines represent times of major volcanic eruptions. Single simulations for CMIP5 models (thin lines); multi-model mean (thick red line); different observations (thick black lines). Dataset pre-processing like described in Jones et al., 2013.
- Figure 9.14: Sea surface temperature plots for zonal mean error, equatorial (5 deg north to 5 deg south) mean error, and multi model mean for zonal error and equatorial mean.
- Figure 9.24: Time series of (a) Arctic and (b) Antarctic sea ice extent; trend distributions of (c) September Arctic and (d) February Antarctic sea ice extent.
- Figure 9.26: Ensemble-mean global ocean carbon uptake (a) and global land carbon uptake (b) in the CMIP5 ESMs for the historical period 1900–2005. For comparison, the observation-based estimates provided by the Global Carbon Project (GCP) are also shown (thick black line). The confidence limits on the ensemble mean are derived by assuming that the CMIP5 models are drawn from a t-distribution. The grey areas show the range of

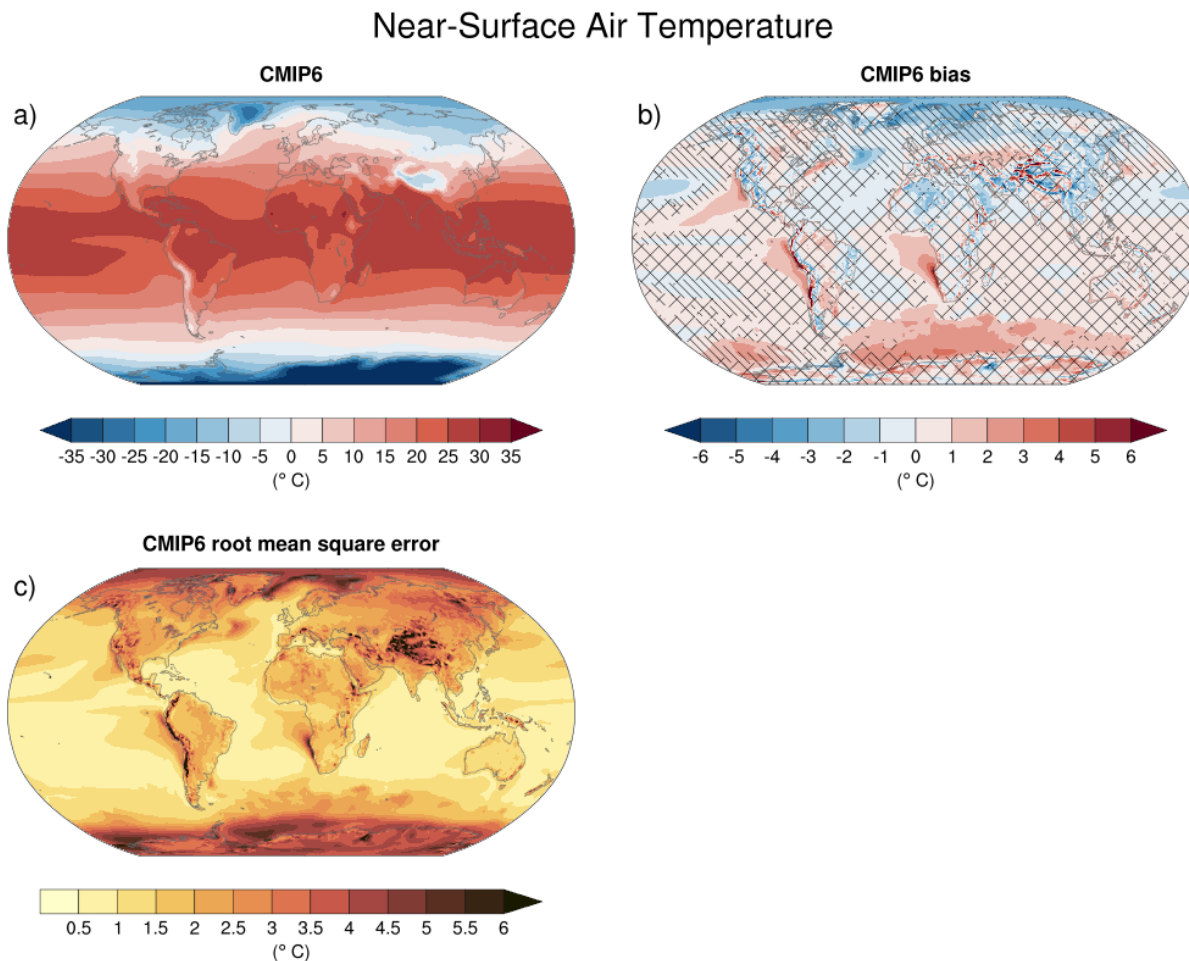


Fig. 1: Figure 3.3: Annual mean near-surface (2 m) air temperature (°C) for the period 1995–2014. (a) Multi-model (ensemble) mean constructed with one realization of the CMIP6 historical experiment from each model. (b) Multi-model mean bias, defined as the difference between the CMIP6 multi-model mean 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.

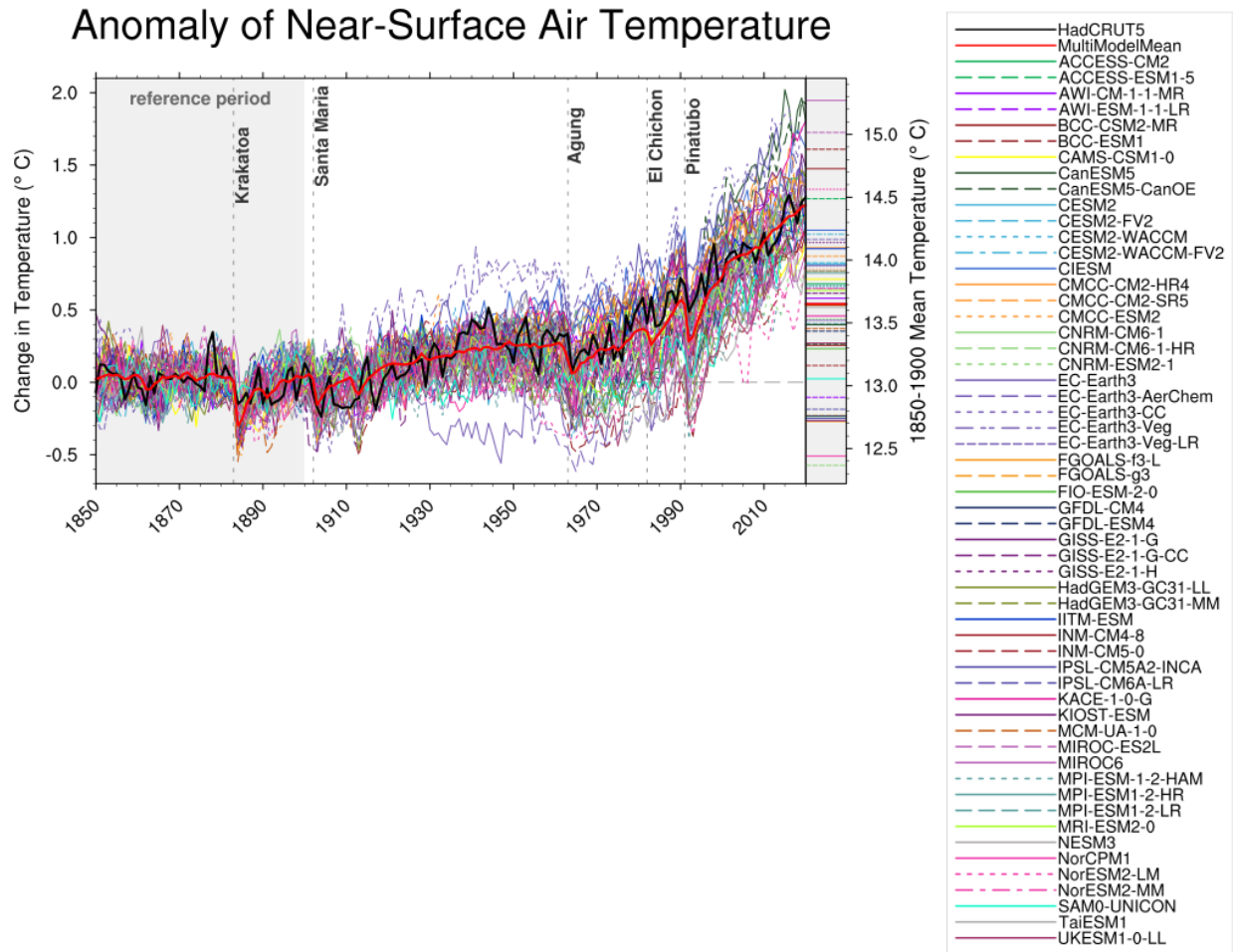


Fig. 2: Figure 3.4a: Observed and simulated time series of the anomalies in annual and global mean surface air temperature (GSAT). All anomalies are differences from the 1850–1900 time-mean of each individual time series. The reference period 1850–1900 is indicated by grey shading. (a) Single simulations 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.

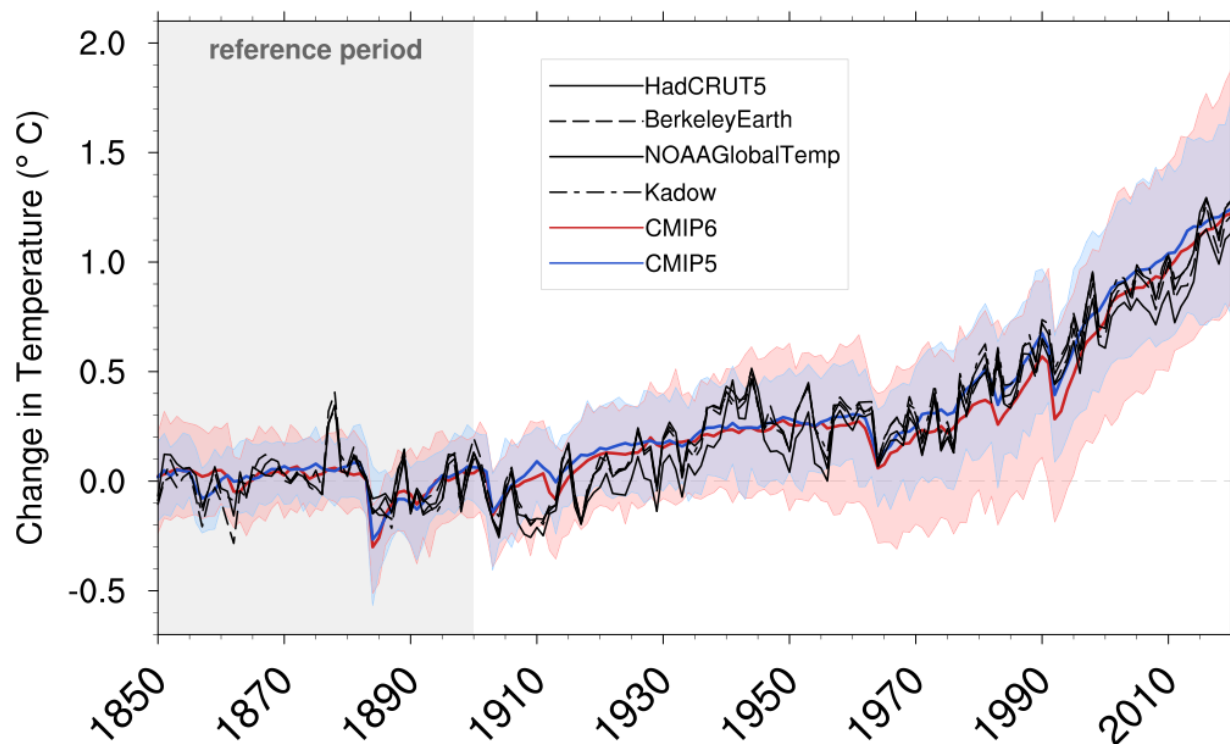


Fig. 3: Figure 3.4b: Observed and simulated time series of the anomalies in annual and global mean surface air temperature (GSAT). All anomalies are differences from the 1850–1900 time-mean of each individual time series. The reference period 1850–1900 is indicated by grey shading. (b) Multi-model means 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 NOAA GlobalTemp 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).

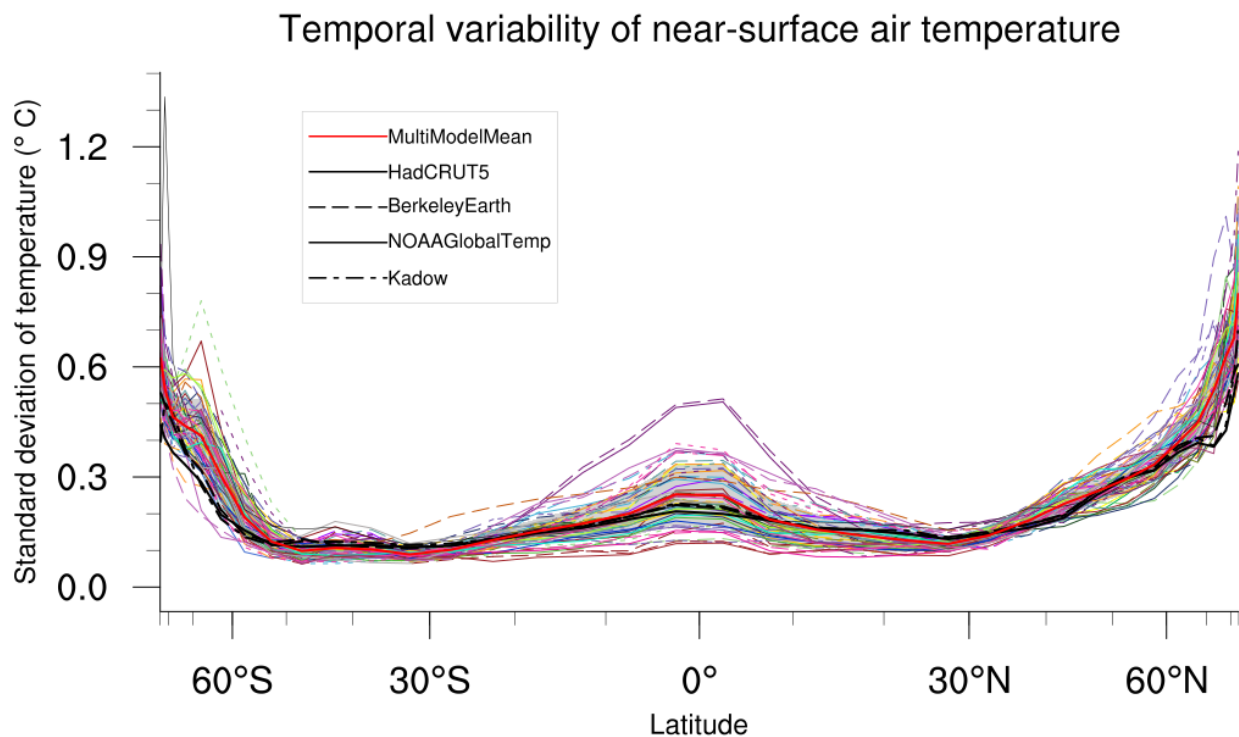


Fig. 4: Figure 3.5: The standard deviation of annually averaged zonal-mean near-surface air temperature. This is shown for four detrended observed temperature datasets (HadCRUT5, Berkeley Earth, NOAAGlobalTemp and Kadow et al. (2020), for the years 1995-2014) and 59 CMIP6 pre-industrial control simulations (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.

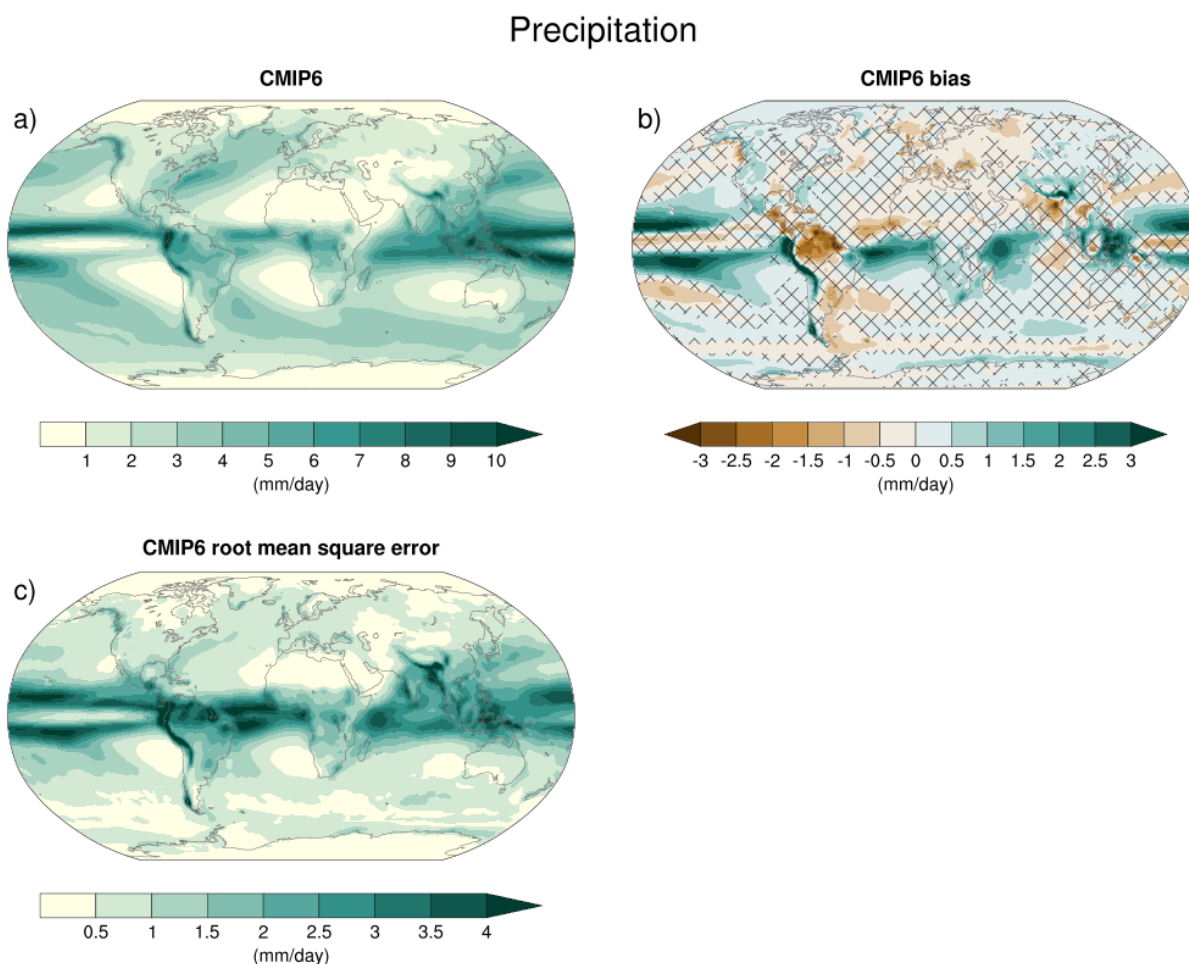


Fig. 5: Figure 3.13: Annual-mean precipitation rate (mm day⁻¹) for the period 1995–2014. (a) Multi-model (ensemble) mean constructed with one realization of the CMIP6 historical experiment from each model. (b) Multi-model mean bias, defined as the difference between the CMIP6 multi-model mean and precipitation 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.

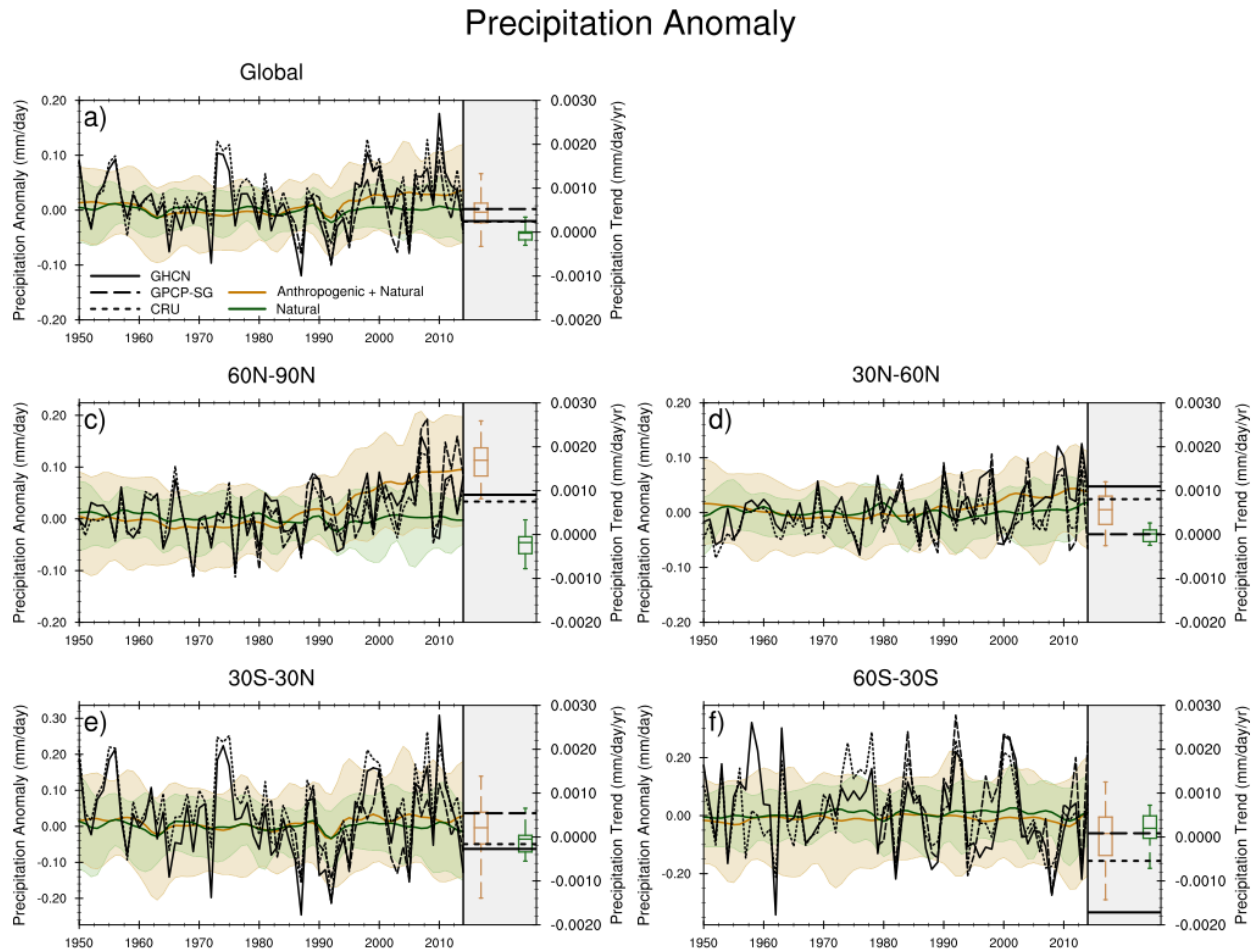


Fig. 6: Figure 3.15: Observed and simulated time series of anomalies in zonal average annual mean precipitation. (a), (c–f) Evolution of global and zonal average annual mean precipitation (mm day^{-1}) over areas of land where there are observations, expressed relative to the base period of 1961–1990, 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.

annual mean fluxes simulated across the model ensemble. This figure includes results from all CMIP5 models that reported land CO₂ fluxes, ocean CO₂ fluxes, or both (Anav et al., 2013).

- Figure 9.27: Simulation of global mean (a) atmosphere–ocean CO₂ fluxes (“fgCO₂”) and (b) net atmosphere–land CO₂ fluxes (“NBP”), by ESMs for the period 1986–2005. For comparison, the observation-based estimates provided by Global Carbon Project (GCP) and the Japanese Meteorological Agency (JMA) atmospheric inversion are also shown. The error bars for the ESMs and observations represent interannual variability in the fluxes, calculated as the standard deviation of the annual means over the period 1986–2005.
- Figure 9.42a: Equilibrium climate sensitivity (ECS) against the global mean surface air temperature, both for the period 1961–1990 and for the pre-industrial control runs.
- Figure 9.42b: Transient climate response (TCR) against equilibrium climate sensitivity (ECS).
- Figure 9.45a: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $d(\alpha_s)/d(T_s)$ values in the seasonal cycle in transient climate change experiments (Hall and Qu, 2006).

17.2.2 Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/`

- `recipe_flato13ipcc.yml`

Diagnostics are stored in `esmvaltool/diag_scripts/`

- `carbon_cycle/main.ncl`: See [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_isccp`: global maps of multi-model mean minus observations + zonal averages of individual models, multi-model mean and observations (Fig. 9.5)
- `ipcc_ar5/ch09_fig09_3.ncl`: multi-model mean seasonality of near-surface temperature (Fig. 9.3)
- `ipcc_ar5/ch09_fig09_6.ncl`: calculating pattern correlations of annual mean climatologies for one variable (Fig 9.6 preprocessing)
- `ipcc_ar5/ch09_fig09_6_collect.ncl`: collecting pattern correlation for each variable and plotting correlation plot (Fig 9.6)
- `ipcc_ar5/tsline.ncl`: time series of the global mean (anomaly) (Fig. 9.8)
- `ipcc_ar5/ch09_fig09_14.py`: Zonally averaged and equatorial SST (Fig. 9.14)
- `seaice/seaice_tsline.ncl`: Time series of sea ice extent (Fig. 9.24a/b)
- `seaice/seaice_trends.ncl`: Trend distributions of sea ice extent (Fig 9.24c/d)
- `ipcc_ar5/ch09_fig09_42a.py`: ECS vs. surface air temperature (Fig. 9.42a)
- `ipcc_ar5/ch09_fig09_42b.py`: TCR vs. ECS (Fig. 9.42b)
- `emergent_constraints/snowalbedo.ncl`: snow-albedo effect (Fig. 9.45a)

17.2.3 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

4. 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 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

5. 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

6. Script ipcc_ar5/tsline.ncl

Required settings for script

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

Optional settings for script

- time_avg: type of time average (currently only “yearly” and “monthly” are available).
- ts_anomaly: calculates anomalies with respect to the defined period; for each 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

7. 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

8. Script ipcc_ar5/ch09_fig09_6.ncl

Required settings for variables

- reference_dataset: name of reference observation

Optional settings for variables

- alternative_dataset: name of alternative observations

9. Script ipcc_ar5/ch09_fig09_6_collect.ncl

Required settings for script

none

Optional settings for script

- diag_order: List of diagnostic names in the order variables should appear on x-axis

10. Script seaice/seaice_trends.ncl

Required settings (scripts)

- month: selected month (1, 2, ..., 12) or annual mean ("A")
- region: region to be analyzed ("Arctic" or "Antarctic")

Optional settings (scripts)

- fill_pole_hole: fill observational hole at North pole, Default: False

Optional settings (variables)

- ref_model: array of references plotted as vertical lines

11. Script seaice/seaice_tsline.ncl

Required settings (scripts)

- region: Arctic, Antarctic
- month: annual mean (A), or month number (3 = March, for Antarctic; 9 = September for Arctic)

Optional settings (scripts)

- styleset: for plot_type cycle only (cmip5, cmip6, default)
- multi_model_mean: plot multi-model mean and standard deviation (default: False)
- EMs_in_lg: create a legend label for individual ensemble members (default: False)
- fill_pole_hole: fill polar hole (typically in satellite data) with sic = 1 (default: False)

12. Script ipcc_ar5/ch09_fig09_42a.py

Required settings for script

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()` (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()` (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

17.2.4 Variables

- areacello (fx, longitude latitude)
- fgco2 (ocean, monthly mean, longitude latitude time)
- nbp (ocean, monthly mean, longitude latitude time)
- pr (atmos, monthly mean, longitude latitude time)
- rlut, rlutcs (atmos, monthly mean, longitude latitude time)
- rsdt (atmos, monthly mean, longitude latitude time)
- rsuscs, rsdscs (atmos, monthly mean, longitude latitude time)
- rsut, rsutcs (atmos, monthly mean, longitude latitude time)
- sic (ocean-ice, monthly mean, longitude latitude time)
- tas (atmos, monthly mean, longitude latitude time)
- tos (ocean, monthly mean, longitude, latitude, time)

17.2.5 Observations and reformat scripts

Note: (1) obs4MIPs data can be used directly without any preprocessing; (2) see headers of reformat scripts for non-obs4MIPs data for download instructions.

- CERES-EBAF (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)

17.2.6 References

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

17.2.7 Example plots

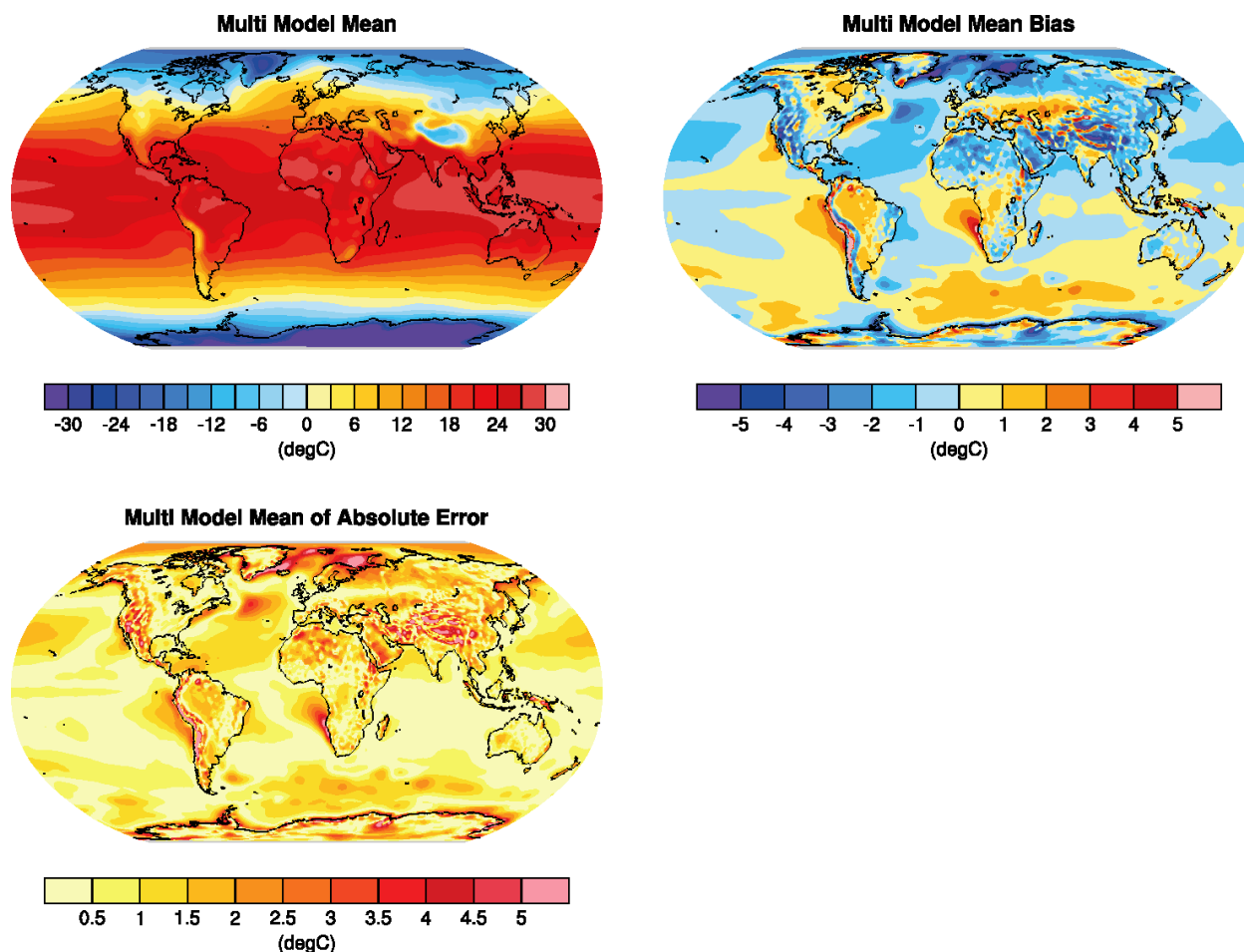


Fig. 7: Figure 9.2 a,b,c: Annual-mean surface air temperature for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from ERA-Interim (Dee et al., 2011), c) mean absolute model error with respect to the climatology from ERA-Interim.

17.3 IPCC AR5 Chapter 12 (selected figures)

17.3.1 Overview

The goal is to create a standard recipe for creating selected Figures from IPCC AR5 Chapter 12 on “Long-term Climate Change: Projections, Commitments and Irreversibility”. These include figures showing the change in a variable between historical and future periods, e.g. maps (2D variables), zonal means (3D variables), timeseries showing the change in certain variables from historical to future periods for multiple scenarios, and maps visualizing change in variables normalized by global mean temperature change (pattern scaling) as in Collins et al., 2013.

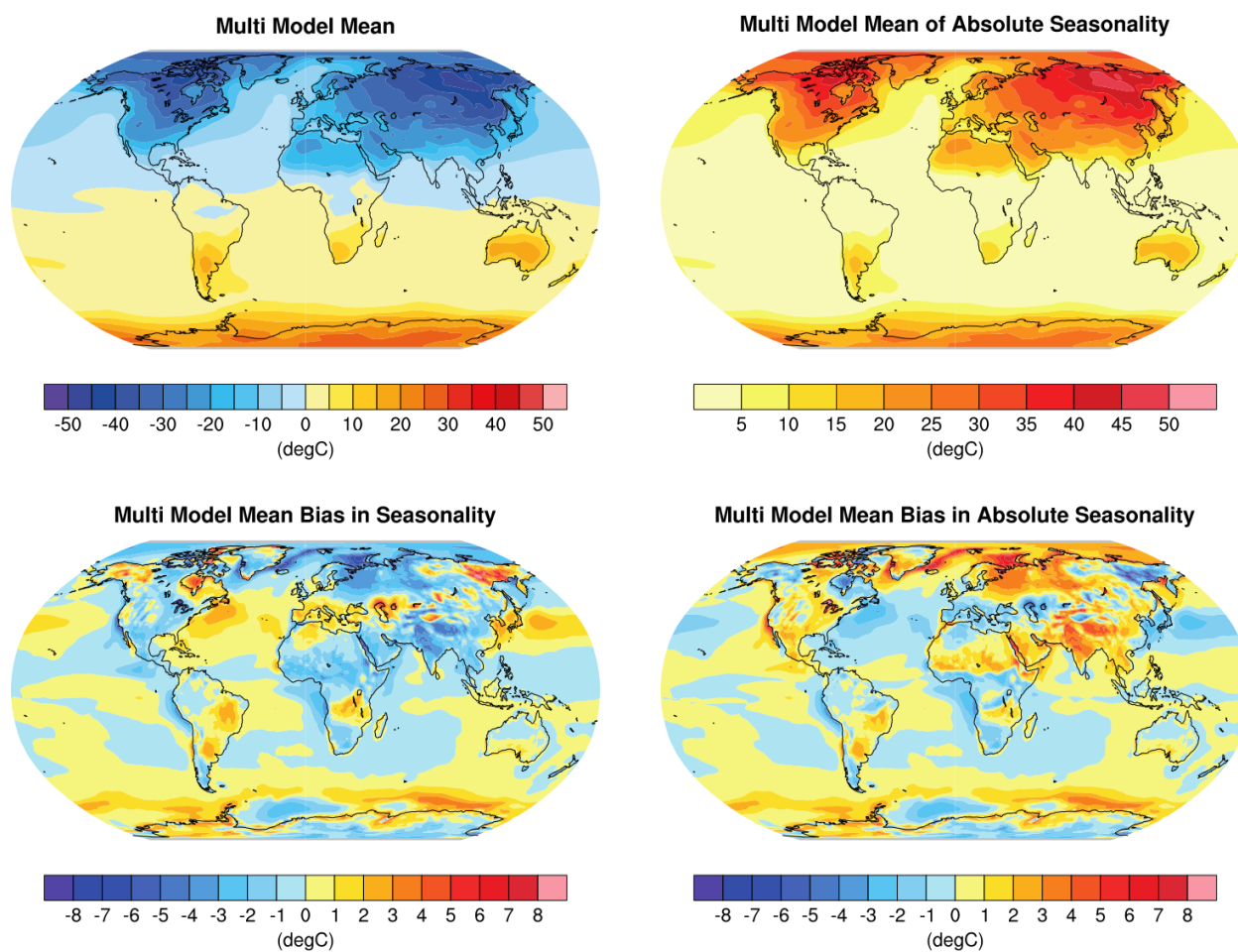


Fig. 8: Figure 9.3: Multi model values for seasonality of near-surface temperature, from top left to bottom right: mean, mean of absolute seasonality, mean bias in seasonality, mean bias in absolute seasonality. Reference dataset: ERA-Interim.

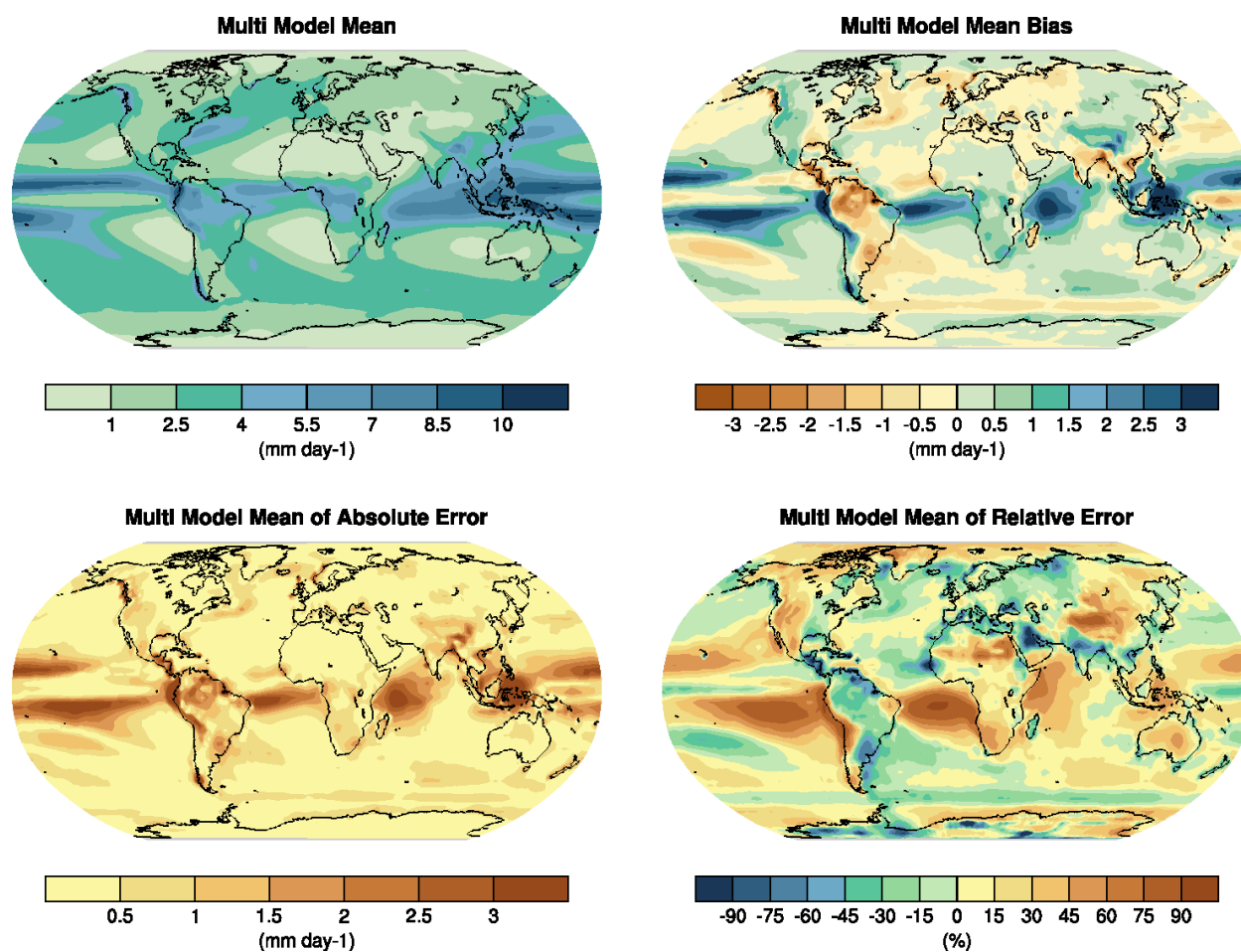


Fig. 9: Figure 9.4: Annual-mean precipitation rate (mm day⁻¹) for the period 1980-2005. a) multi-model mean, b) bias as the difference between the CMIP5 multi-model mean and the climatology from the Global Precipitation Climatology Project (Adler et al., 2003), c) multi-model mean absolute error with respect to observations, and d) multi-model mean error relative to the multi-model mean precipitation itself.

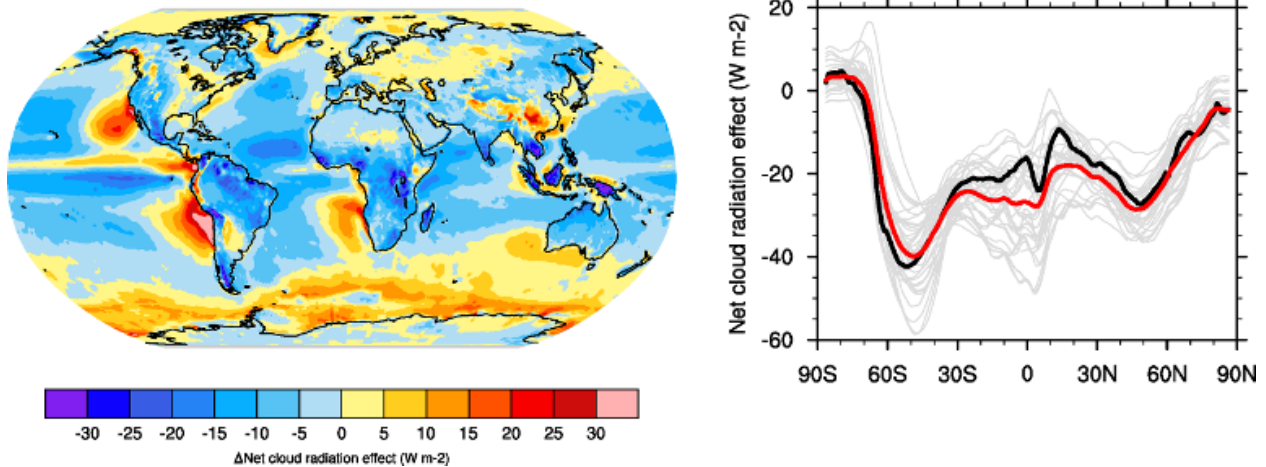


Fig. 10: Figure 9.5: Climatological (1985-2005) annual-mean cloud radiative effects in Wm^{-2} for the CMIP5 models against CERES EBAF (2001-2011) in Wm^{-2} . Top row shows the shortwave effect; middle row the longwave effect, and bottom row the net effect. Multi-model-mean biases against CERES EBAF 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).

17.3.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_collins13ipcc.yml`

Diagnostics are stored in `diag_scripts/`

- `ipcc_ar5/ch12_map_diff_each_model_fig12-9.ncl`: calculates the difference between future and historical runs for one scenario for each given model individually on their native grid and plots all of them in one Figure. As in Figure 12.9 in AR5.
- `ipcc_ar5/ch12_ts_line_mean_spread.ncl`: calculates time series for one variable, change in future relative to base period in historical, multi-model mean as well as spread around it (as standard deviation).
- `ipcc_ar5/ch12_plot_ts_line_mean_spread.ncl`: plots the timeseries multi-model mean and spread calculated above. As in Figure 12.5 in AR5.
- `ipcc_ar5/ch12_calc_IAV_for_stippandhatch.ncl`: calculates the interannual variability over piControl runs, either over the whole time period or in chunks over some years.
- `ipcc_ar5/ch12_calc_map_diff_mmm_stippandhatch.ncl`: calculates the difference between future and historical periods for each given model and then calculates multi-model mean as well as significance. Significant is where the multi-model mean change is greater than two standard deviations of the internal variability and where at least 90% of the models agree on the sign of change. Not significant is where the multi-model mean change is less than one standard deviation of internal variability.
- `ipcc_ar5/ch12_plot_map_diff_mmm_stipp.ncl`: plots multi-model mean maps calculated above including stippling where significant and hatching where not significant. As in Figure 12.11 in AR5.
- `ipcc_ar5/ch12_calc_zonal_cont_diff_mmm_stippandhatch.ncl`: calculates zonal means and the difference between future and historical periods for each given model and then calculates multi-model mean as well as significance as above.
- `ipcc_ar5/ch12_plot_zonal_diff_mmm_stipp.ncl`: plots the multi-model mean zonal plots calculated above including stippling where significant and hatching where not significant. As in Figure 12.12 in AR5.

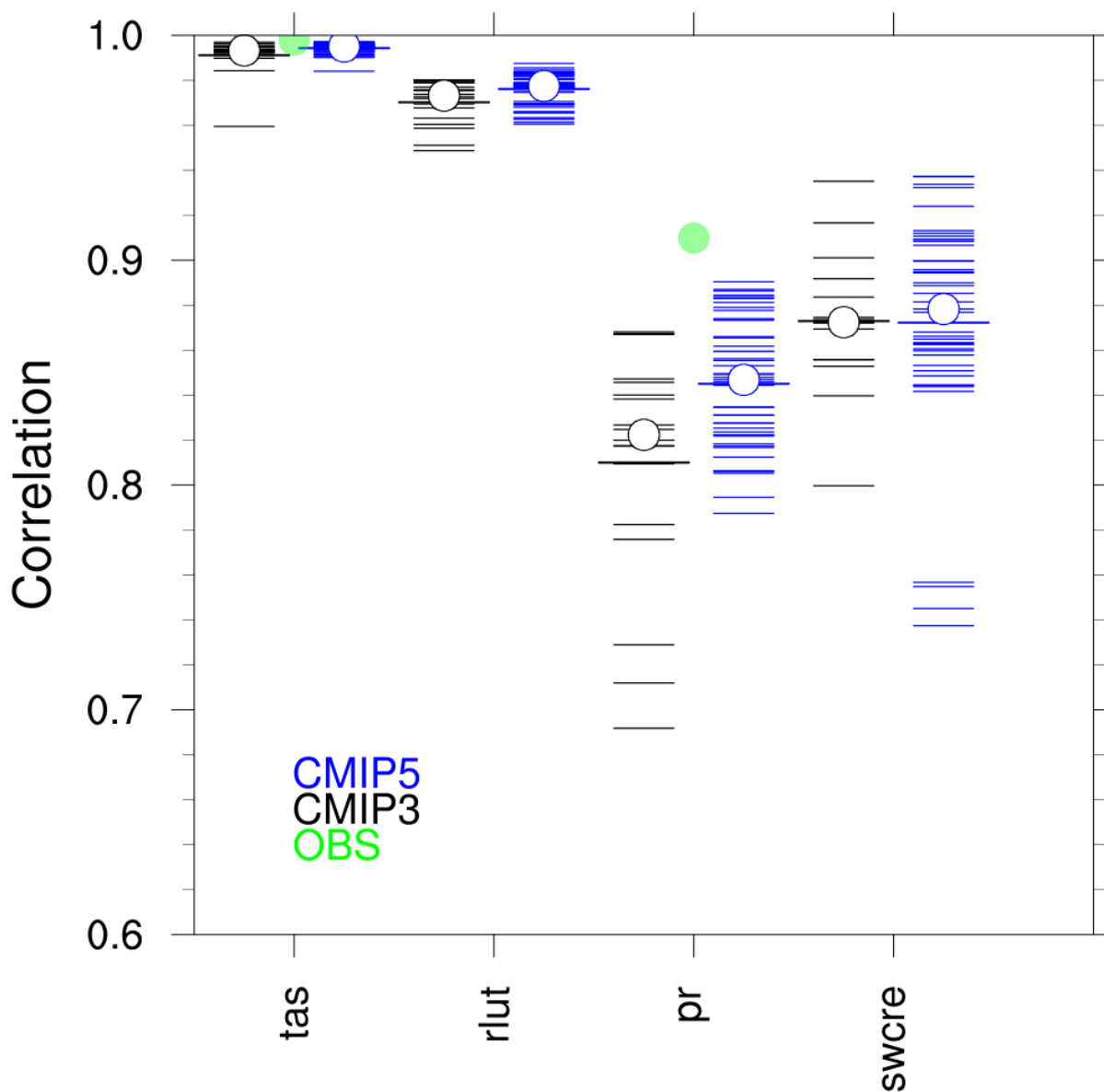


Fig. 11: Figure 9.6: Centred pattern correlations between models and observations for the annual mean climatology over the period 1980–1999. Results are shown for individual CMIP3 (black) and CMIP5 (blue) models as thin dashes, along with the corresponding ensemble average (thick dash) and median (open circle). The four variables shown are surface air temperature (TAS), top of the atmosphere (TOA) outgoing longwave radiation (RLUT), precipitation (PR) and TOA shortwave cloud radiative effect (SW CRE). The correlations between the reference and alternate observations are also shown (solid green circles).

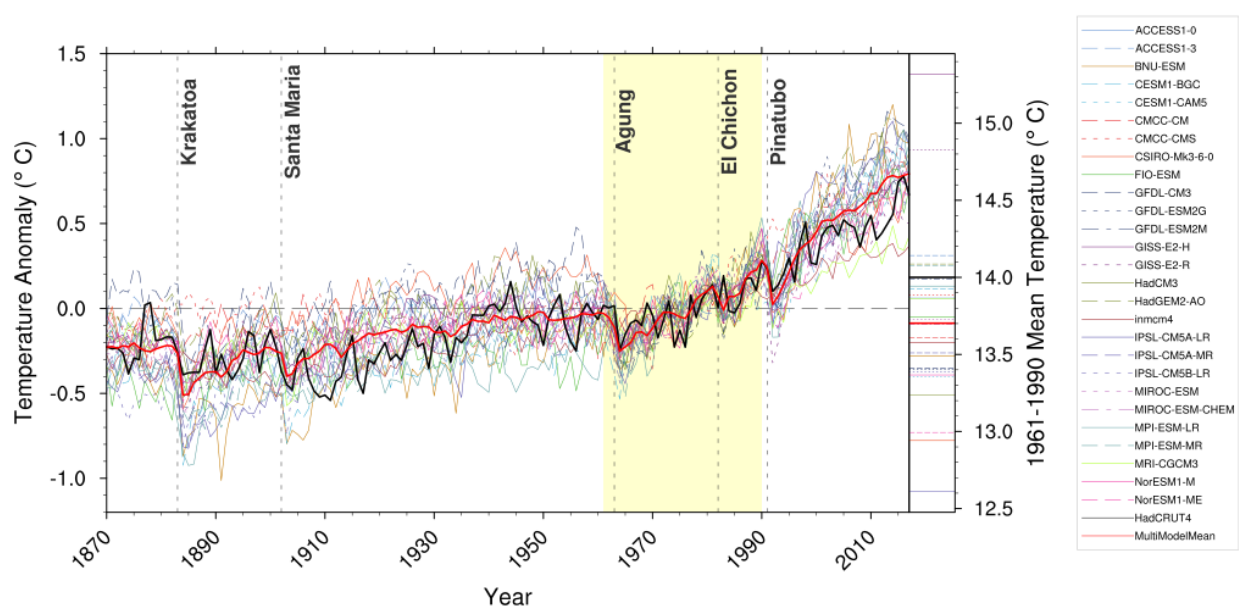


Fig. 12: Figure 9.8: Observed and simulated time series of the anomalies in annual and global mean surface temperature. All anomalies are differences from the 1961-1990 time-mean of each individual time series. The reference period 1961-1990 is indicated by yellow shading; vertical dashed grey lines 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.

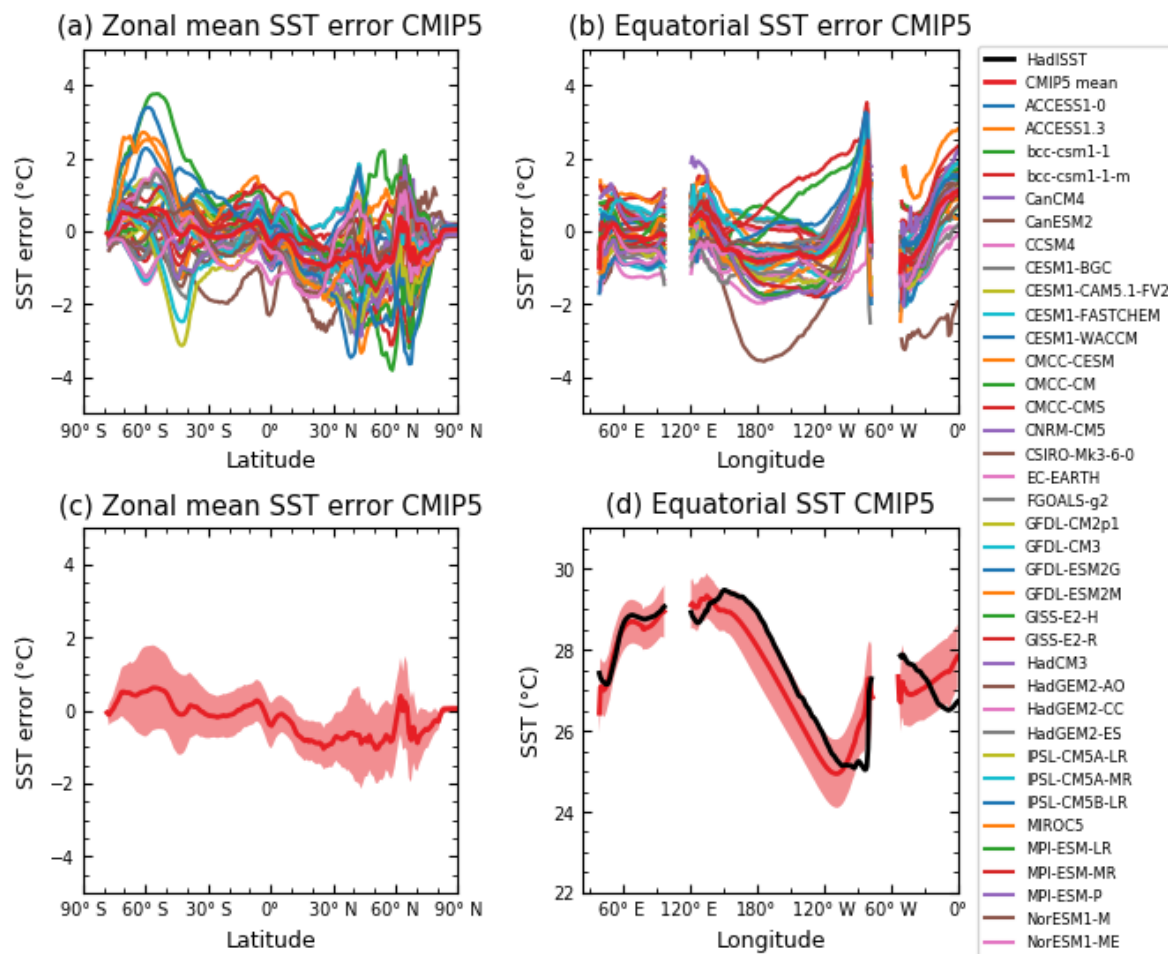


Fig. 13: Figure 9.14: (a) Zonally averaged sea surface temperature (SST) error in CMIP5 models. (b) Equatorial SST error in CMIP5 models. (c) Zonally averaged multi-model mean SST error for CMIP5 together with inter-model standard deviation (shading). (d) Equatorial multi-model mean SST in CMIP5 together 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).

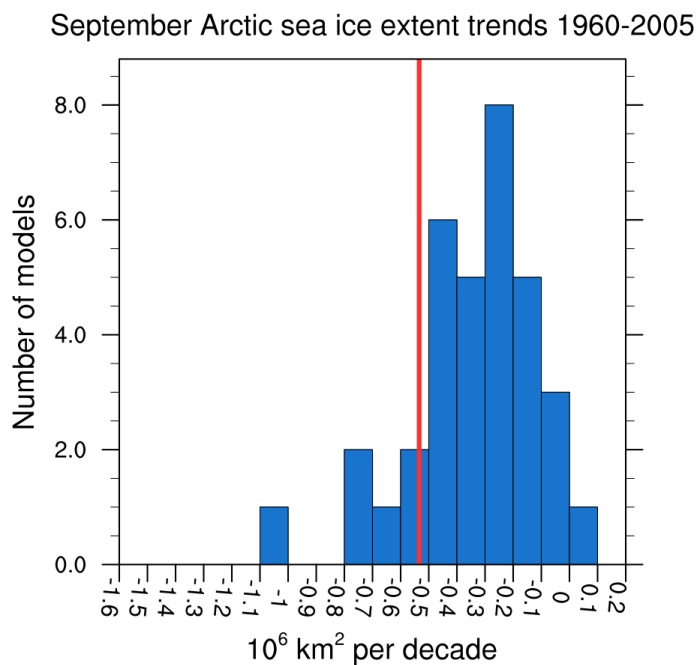


Fig. 14: Figure 9.24c: Sea ice extent trend distribution for the Arctic in September.

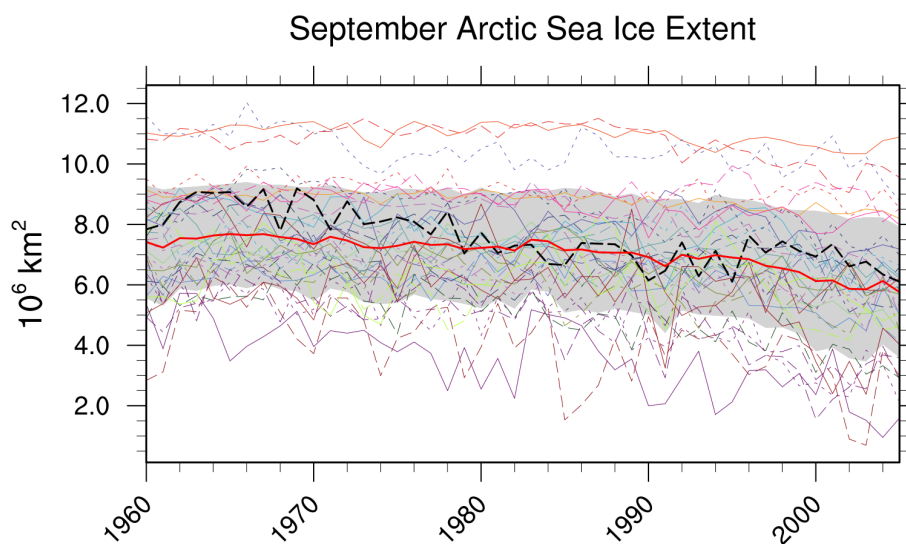


Fig. 15: Figure 9.24a: Time series of total sea ice area and extent (accumulated) for the Arctic in September including multi-model mean and standard deviation.

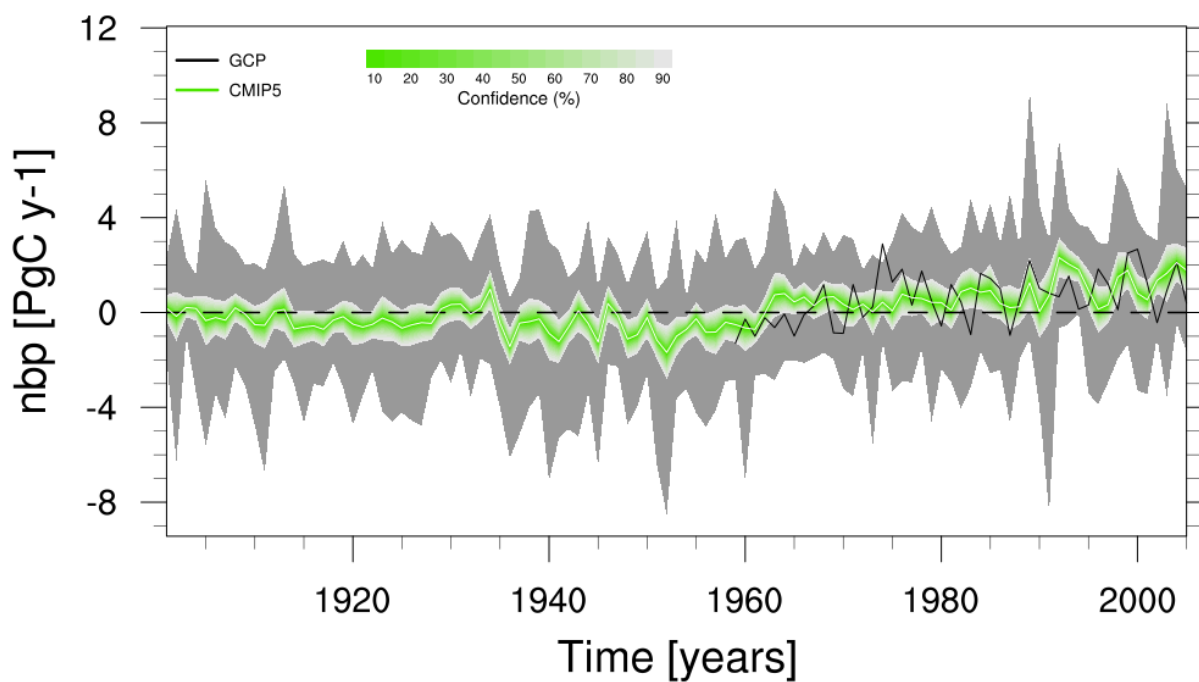


Fig. 16: Figure 9.26 (bottom): Ensemble-mean global land carbon uptake in the CMIP5 ESMs for the historical period 1900–2005. For comparison, the observation-based estimates provided by the Global Carbon Project (GCP) are also shown (black line). The confidence limits on the ensemble mean are derived 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.

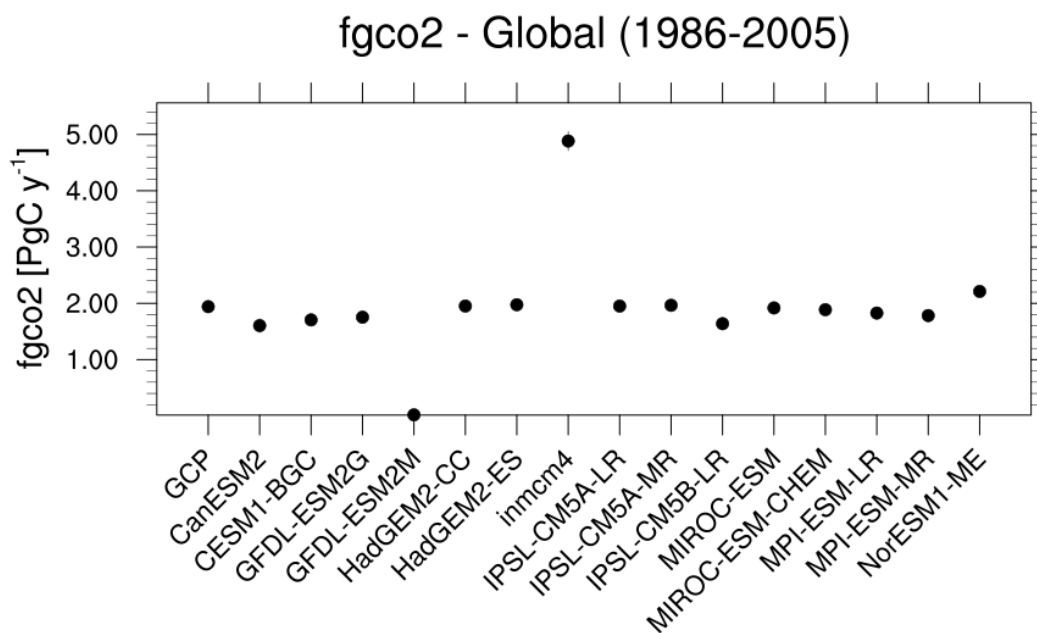


Fig. 17: Figure 9.27 (top): Simulation of global mean atmosphere–ocean CO₂ fluxes (“fgCO₂”) by ESMs for the period 1986–2005. For comparison, the observation-based estimates provided by Global Carbon Project (GCP) are also shown. The error bars for the ESMs and observations represent interannual variability in the fluxes, calculated as the standard deviation of the annual means over the period 1986–2005.

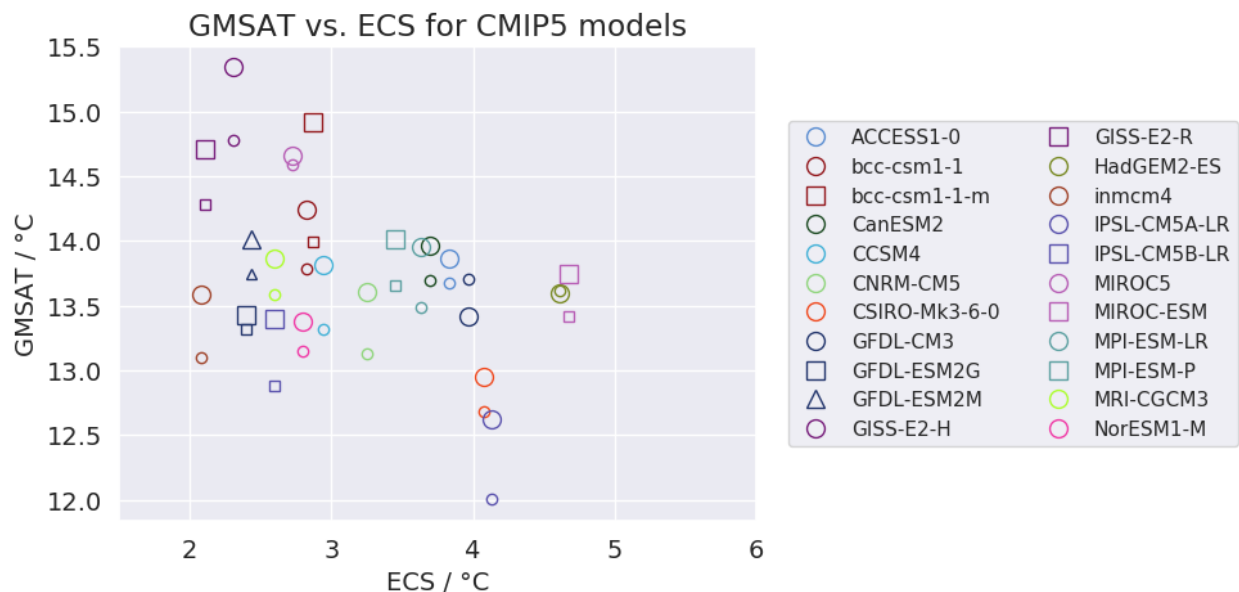


Fig. 18: Figure 9.42a: Equilibrium climate sensitivity (ECS) against the global mean surface air temperature of CMIP5 models, both for the period 1961-1990 (larger symbols) and for the pre-industrial control runs (smaller symbols).

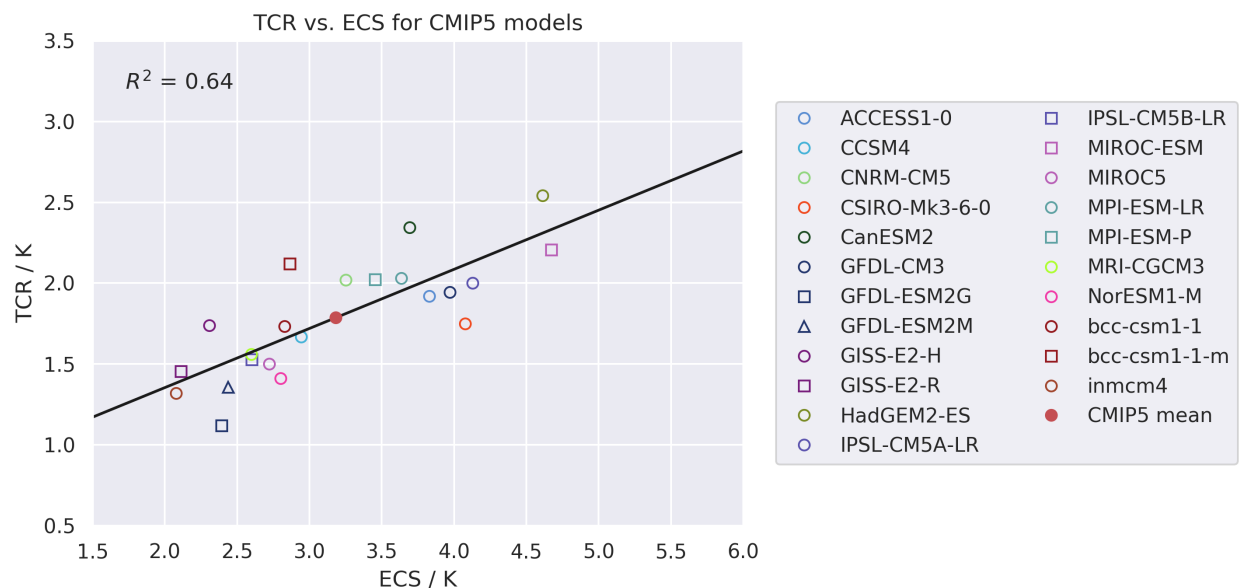


Fig. 19: Figure 9.42b: Transient climate response (TCR) against equilibrium climate sensitivity (ECS) for CMIP5 models.

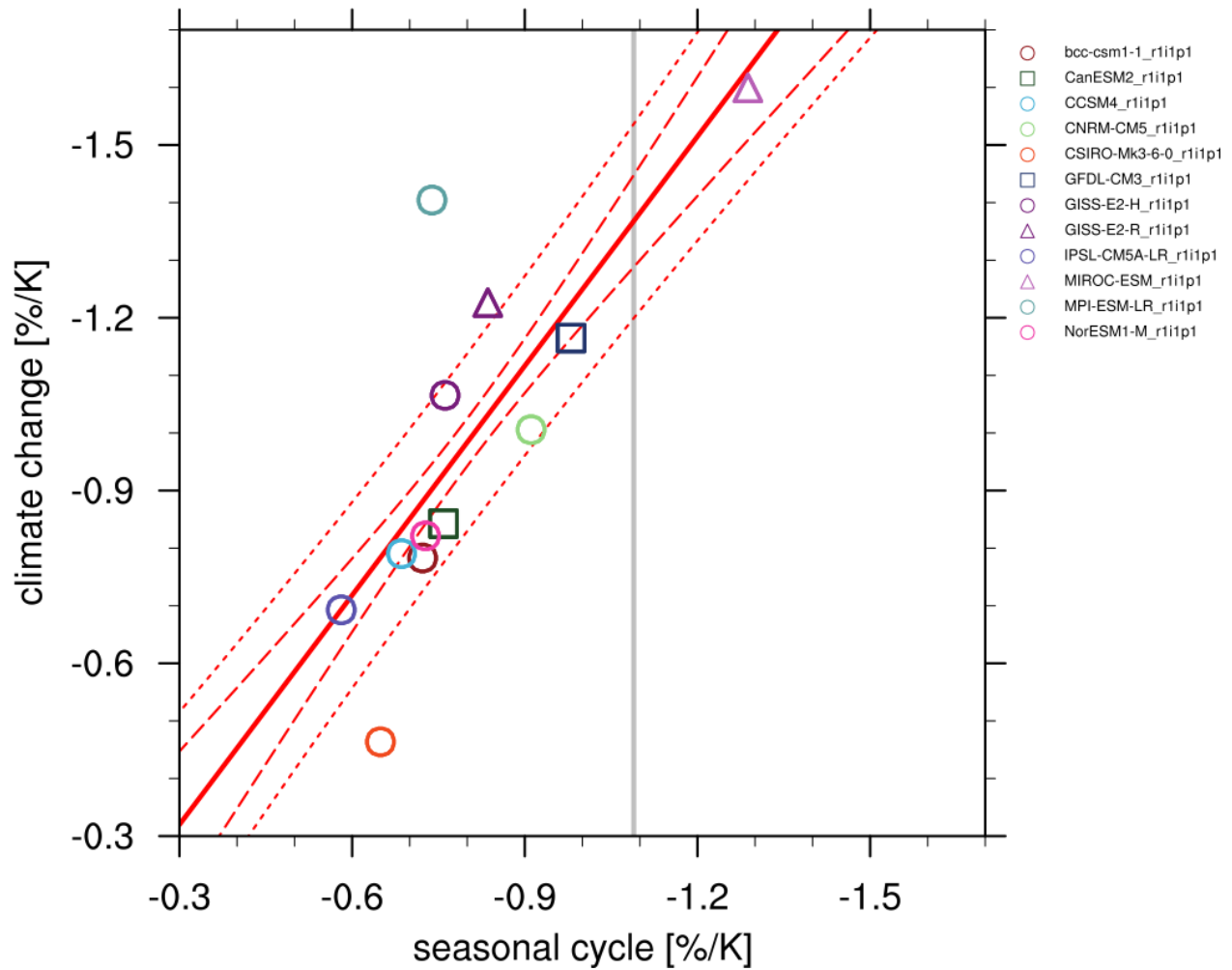


Fig. 20: Figure 9.45a: Scatterplot of springtime snow-albedo effect values in climate change vs. springtime $\Delta\alpha_s/\Delta T_s$ values in the seasonal cycle in transient climate change experiments (CMIP5 historical experiments: 1901-2000, RCP4.5 experiments: 2101-2200).

- `ipcc_ar5/ch12_calc_map_diff_scaleT_mmm_stipp.ncl`: calculates the change in variable between future and historical period normalized by global 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`.

17.3.3 User settings

1. Script `ipcc_ar5/ch12_map_diff_each_model_fig12-9.ncl`

Required settings (script)

- `time_avg`: time averaging (“annualclim”, “seasonalclim”)
- `experiment`: IPCC Scenario, used to pair historical and rcp runs from same model

Optional settings (script)

- `projection`: map projection, any valid ncl projection, default = Robinson
- `max_vert`: maximum number of plots in vertical
- `max_hori`: maximum number of plots in horizontal
- `title`: plot title
- `colormap`: alternative colormap, path to rgb file or ncl name
- `diff_levs`: list with contour levels for plots
- `span`: span whole colormap? (True, False, default = False)

Required settings (variables)

- `project`: CMIP5 (or CMIP6)
- `mip`: variable mip, generally Amon or Omon

2. Script `ipcc_ar5/ch12_ts_line_mean_spread.ncl`

Required settings (script)

- `scenarios`: list with scenarios included in figure
- `syears`: list with start years in time periods (e.g. start of historical period and reps)
- `eyears`: list with end years in time periods (end year of historical runs and reps)
- `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: minimum 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 levels 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
- syeas: 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”)
- rcg_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

17.3.4 Variables

Note: These are the variables tested and used in IPCC AR5. However, the code is flexible and in theory other variables of the same kind can be used.

- areacello (fx, longitude latitude)
- clt (atmos, monthly mean, longitude latitude time)
- evspsbl (atmos, monthly mean, longitude latitude time)
- hurs (atmos, monthly mean, longitude latitude time)
- mrro (land, monthly mean, longitude latitude time)
- mrsos (land, monthly mean, longitude latitude time)
- pr (atmos, monthly mean, longitude latitude time)
- psl (atmos, monthly mean, longitude latitude time)
- rlut, rsut, rtmt (atmos, monthly mean, longitude latitude time)
- sic (ocean-ice, monthly mean, longitude latitude time)
- snw (land, monthly mean, longitude latitude time)
- sos (ocean, monthly mean, longitude latitude time)
- ta (atmos, monthly mean, longitude latitude lev time)
- tas (atmos, monthly mean, longitude latitude time)
- thetao (ocean, monthly mean, longitude latitude lev time)
- ua (atmos, monthly mean, longitude latitude lev time)

17.3.5 Observations and reformat scripts

- HadISST (sic - esmvaltool/cmorizers/data/formatters/datasets/hadisst.ncl)

17.3.6 Reference

- Collins, M., R. Knutti, J. Arblaster, J.-L. Dufresne, T. Fichefet, P. Friedlingstein, X. Gao, W.J. Gutowski, T. Johns, G. Krinner, M. Shongwe, C. Tebaldi, A.J. Weaver and M. Wehner, 2013: Long-term Climate Change: Projections, Commitments and Irreversibility. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (eds.)]. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.

17.3.7 Example plots

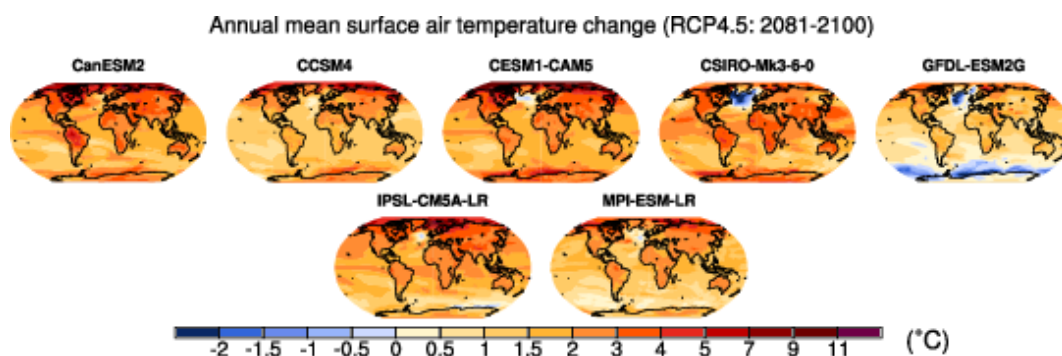


Fig. 21: Surface air temperature change in 2081–2100 displayed as anomalies with respect to 1986–2005 for RCP4.5 from individual CMIP5 models.

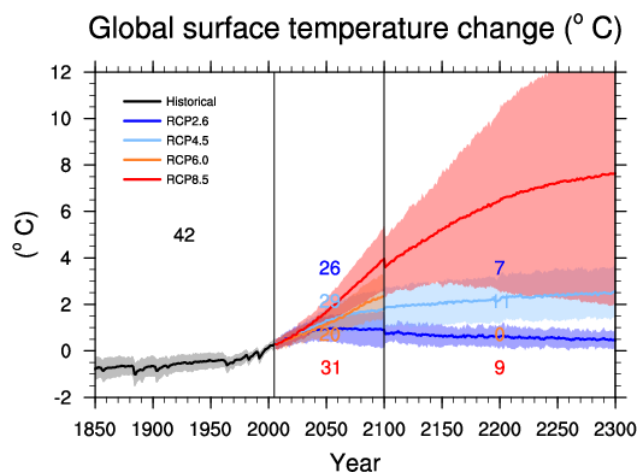


Fig. 22: Time series of global annual mean surface air temperature anomaly (relative to 1986–2005) from CMIP5 concentration-driven experiments.

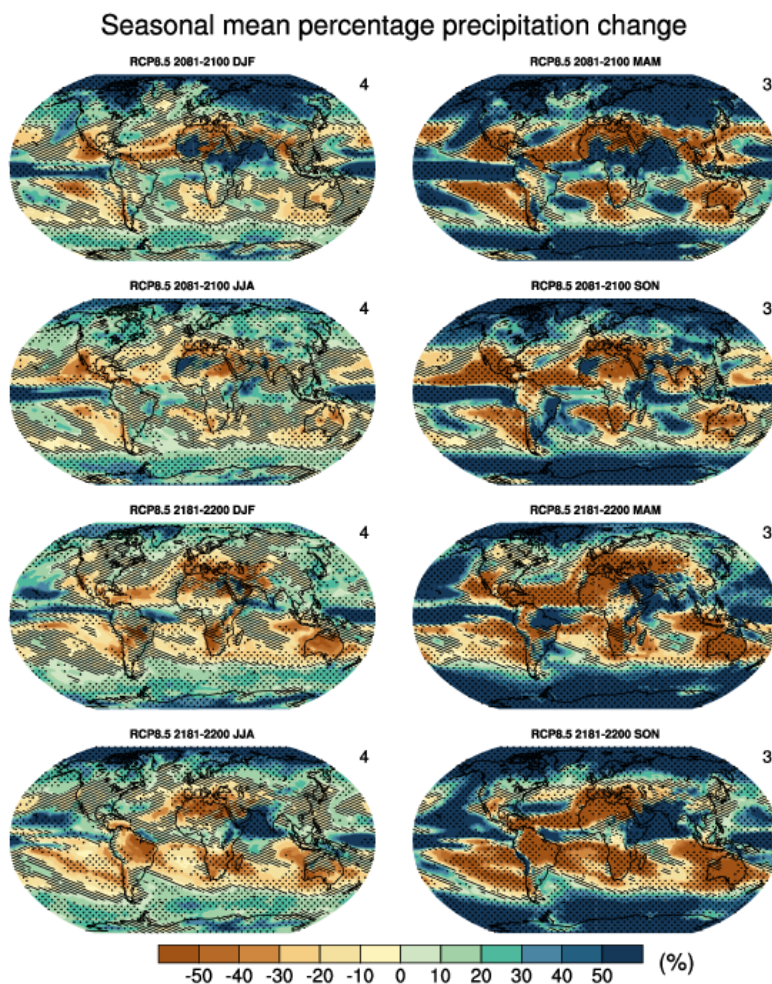


Fig. 23: Multi-model CMIP5 average percentage change in seasonal mean precipitation relative to the reference period 1986–2005 averaged over the periods 2081–2100 and 2181–2200 under the RCP8.5 forcing scenario. Hatching indicates regions where the multi-model mean change is less than one standard deviation 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

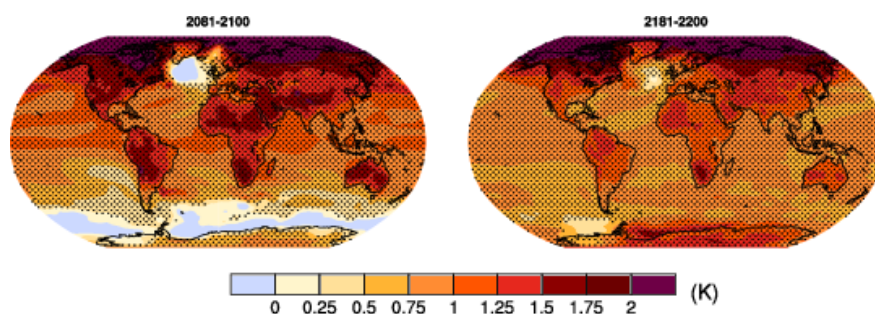


Fig. 24: Temperature change patterns scaled to 1°C of global mean surface temperature change.

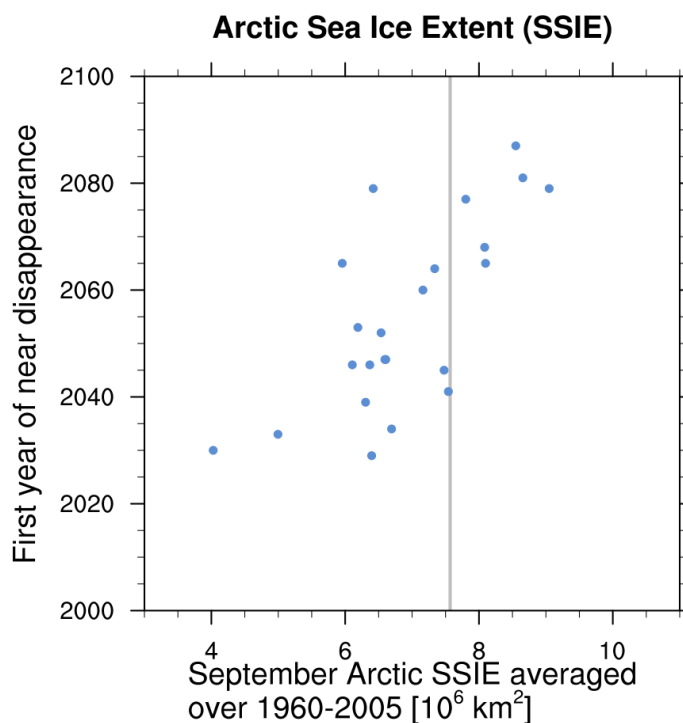


Fig. 25: Scatter plot of mean historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a).

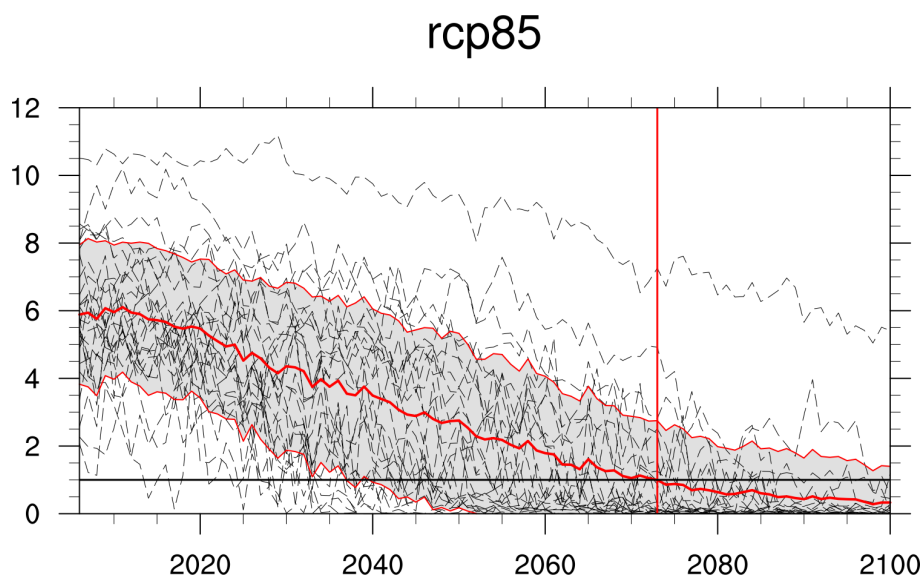


Fig. 26: Time series of September Arctic sea ice extent for individual CMIP5 models, multi-model mean and multi-model standard deviation, year of disappearance (similar to IPCC AR5 Chapter 12, Fig. 12.31e).

18.1 Landcover - Albedo

18.1.1 Overview

The diagnostic determines the coefficients of multiple linear regressions fitted between the albedo values and the tree, shrub, short vegetation (crops and grasses) fractions of each grid cell within spatially moving windows encompassing 5x5 model grid cells. Solving these regressions provides the albedo values for trees, shrubs and short vegetation (crops and grasses) from which the albedo changes associated with transitions between these three landcover types are derived. The diagnostic distinguishes between snow-free and snow-covered grid cells.

18.1.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_albedolandcover.yml

Diagnostics are stored in diag_scripts/landcover/

- albedolandcover.py

18.1.3 User settings

Several parameters can be set in the recipe

18.1.4 Variables

- rsus (atmos, monthly mean, time latitude longitude)
- rsds (atmos, monthly mean, time latitude longitude)
- snc (landice, monthly mean, time latitude longitude)
- grassFrac (land, monthly mean, time latitude longitude)
- treeFrac (land, monthly mean, time latitude longitude)
- shrubFrac (land, monthly mean, time latitude longitude)
- cropFrac (land, monthly mean, time latitude longitude)
- pastureFrac (land, monthly mean, time latitude longitude)

18.1.5 Observations and reformat scripts

A reformatting script for observational data is available here:

- `esmvaltool/cmorizers/data/formatters/datasets/duveiller2018.py`

18.1.6 References

- Duveiller, G., Hooker, J. and Cescatti, A., 2018a. A dataset mapping the potential biophysical effects of vegetation cover change. *Scientific Data*, 5: 180014.
- Duveiller, G., Hooker, J. and Cescatti, A., 2018b. The mark of vegetation change on Earth's surface energy balance. *Nature communications*, 9(1): 679.

18.1.7 Example plots

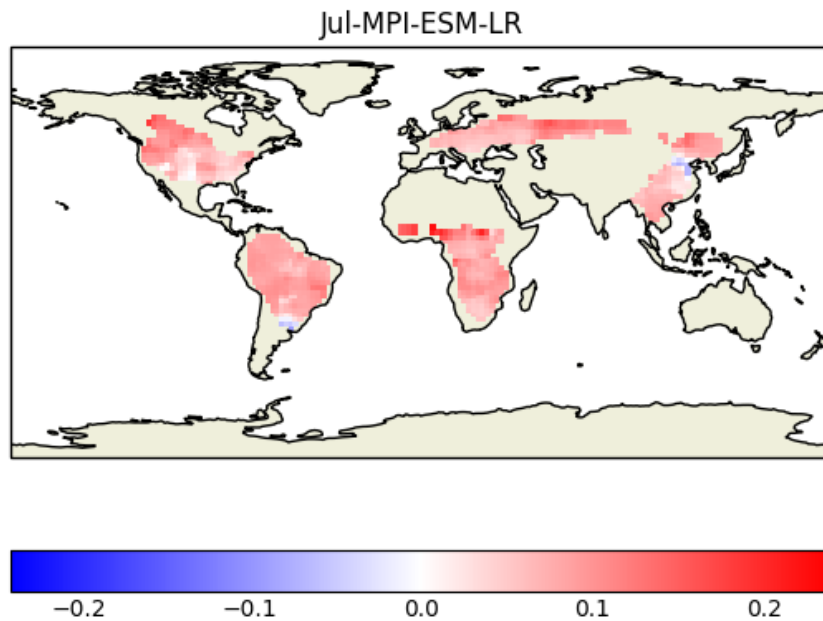


Fig. 1: Example of albedo change from tree to crop and grass for the CMIP5 model MPI-ESM-LR derived for the month of July and averaged over the years 2000 to 2004.

18.2 Turnover time of carbon over land ecosystems

18.2.1 Overview

This recipe evaluates the turnover time of carbon over land ecosystems (τ_{ctotal}) based on the analysis of [Carvalhais et al. \(2014\)](#). In summary, it provides an overview on:

- Comparisons of global distributions of τ_{ctotal} from all models against observation and other models
- Variation of τ_{ctotal} across latitude (zonal distributions)
- Variation of association of τ_{ctotal} and climate across latitude (zonal correlations)
- metrics of global τ_{ctotal} and correlations

18.2.2 Calculation of turnover time

First, the total carbon content of land ecosystems is calculated as,

$$ctotal = cSoil + cVeg$$

where $cSoil$ and $cVeg$ are the carbon contents in soil and vegetation. **Note that this is not fully consistent with `Carvalhais et al. (2014)`_, in which `ctotal` includes all carbon storages that respire to the atmosphere. Due to inconsistency across models, it resulted in having different carbon storage components in calculation of ctotal for different models.**

The turnover time of carbon is then calculated as,

$$\tau_{ctotal} = \frac{ctotal}{gpp}$$

where $ctotal$ and gpp are temporal means of total carbon content and gross primary productivity, respectively. **The equation is valid for steady state, and is only applicable when both ctotal and gpp are long-term averages.** Therefore, the recipe should always include the mean operator of `climate_statistics` in preprocessor.

18.2.3 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_carvalhais14nat.yml`

Diagnostics are stored in `diag_scripts/`

- `land_carbon_cycle/diag_global_turnover.py`
- `land_carbon_cycle/diag_zonal_turnover.py`
- `land_carbon_cycle/diag_zonal_correlation.py`

18.2.4 User settings in recipe

Observation-related details

The settings needed for loading the observational dataset in all diagnostics are provided in the recipe through *obs_info* within *obs_details* section.

- **obs_data_subdir**: subdirectory of **auxiliary_data_dir** (set in config-user file) where observation data are stored {e.g., `data_ESMValTool_Carvalhais2014`}.
- **source_label**: source data label {'Carvalhais2014'}.
- **variant_label**: variant of the observation {'BE'} for best estimate.
- **grid_label**: label denoting the spatial grid specification {'gn'}.
- **frequency**: temporal frequency of the observation data {'fx'}

The observation data file used in the recipe should be changed through the fields above, as these are used to generate observation file name and locations. For details, see [Observations](#) section.

Preprocessor

- **climate_statistics**: {mean} - calculate the mean over full time period.
- **regrid**: {nearest} - nearest neighbor regridding to the selected observation resolution.
- **mask_landsea**: {sea} - mask out all the data points from sea.
- **multi_model_statistics**: {median} - calculate and include the multimodel median.

Script `land_carbon_cycle/diag_global_turnover.py`

- Required settings:
 - **obs_variable**: {str} list of the variable(s) to be read from the observation files
- Optional settings:
 - **ax_fs**: {float, 7.1} - fontsize in the figure.
 - **fill_value**: {float, nan} - fill value to be used in analysis and plotting.
 - **x0**: {float, 0.02} - X - coordinate of the left edge of the figure.
 - **y0**: {float, 1.0} Y - coordinate of the upper edge of the figure.
 - **wp**: {float, 1 / number of models} - width of each map.
 - **hp**: {float, = wp} - height of each map.
 - **xsp**: {float, 0} - spacing 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.

18.2.5 Required Variables

- `tas` (atmos, monthly, longitude, latitude, time)
- `pr` (atmos, monthly, longitude, latitude, time)
- `gpp` (land, monthly, longitude, latitude, time)
- `cVeg` (land, monthly, longitude, latitude, time)
- `cSoil` (land, monthly, longitude, latitude, time)

18.2.6 Observations

The observations needed in the diagnostics are publicly available for download from the [Data Portal of the Max Planck Institute for Biogeochemistry](#) after registration.

Due to inherent dependence of the diagnostic on uncertainty estimates in observation, the data needed for each diagnostic script are processed at different spatial resolutions (as in Carvalhais et al., 2014), and provided in 11 different resolutions (see Table 1). Note that the uncertainties were estimated at the resolution of the selected models, and, thus, only the pre-processed observed data can be used with the recipe. It is not possible to use regridding functionalities of ESMValTool to regrid the observational data to other spatial resolutions, as the uncertainty estimates cannot be regridded.

Table 1. A summary of the observation datasets at different resolutions.

Reference	target_grid	grid_label*
Observation	0.5x0.5	gn
NorESM1-M	2.5x1.875	gr
bcc-csm1-1	2.812x2.813	gr1
CCSM4	1.25x0.937	gr2
CanESM2	2.812x2.813	gr3
GFDL-ESM2G	2.5x2.0	gr4
HadGEM2-ES	1.875x1.241	gr5
inmcm4	2.0x1.5	gr6
IPSL-CM5A-MR	2.5x1.259	gr7
MIROC-ESM	2.812x2.813	gr8
MPI-ESM-LR	1.875x1.875	gr9

* The `grid_label` is suffixed with `z` for data in zonal/latitude coordinates: the zonal turnover and zonal correlation.

To change the spatial resolution of the evaluation, change {grid_label} in `obs_details` and the corresponding {target_grid} in `regrid_preprocessor` of the recipe.

At each spatial resolution, four data files are provided:

- `tau_ctotal_fx_Carvalhais2014_BE_gn.nc` - global data of `tau_ctotal`
- `tau_ctotal_fx_Carvalhais2014_BE_gnz.nc` - zonal data of `tau_ctotal`

- `r_tau_ctotal_tas_fx_Carvalhais2014_BE_gnz.nc` - zonal correlation of `tau_ctotal` and `tas`, controlled for `pr`
- `r_tau_ctotal_pr_fx_Carvalhais2014_BE_gnz.nc` - zonal correlation of `tau_ctotal` and `pr`, controlled for `tas`.

The data is produced in obs4MIPs standards, and provided in netCDF4 format. The filenames use the convention:

`{variable}_{frequency}_{source_label}_{variant_label}_{grid_label}.nc`

- `{variable}`: variable name, set in every diagnostic script as `obs_variable`
- `{frequency}`: temporal frequency of data, set from `obs_details`
- `{source_label}`: observational source, set from `obs_details`
- `{variant_label}`: observation variant, set from `obs_details`
- `{grid_label}`: temporal frequency of data, set from `obs_details`

Refer to the [Obs4MIPs Data Specifications](#) for details of the definitions above.

All data variables have additional variables (`{variable}_5` and `{variable}_95`) in the same file. These variables are necessary for a successful execution of the diagnostics.

18.2.7 References

- Carvalhais, N., et al. (2014), Global covariation of carbon turnover times with climate in terrestrial ecosystems, *Nature*, 514(7521), 213-217, doi: 10.1038/nature13731.

18.2.8 Example plots

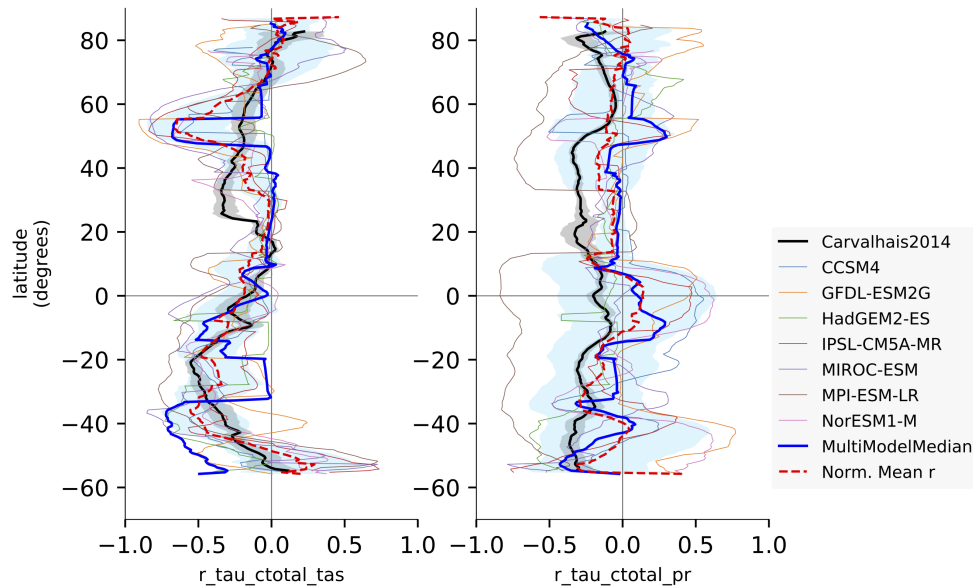


Fig. 2: Comparison of latitudinal (zonal) variations of pearson correlation between turnover time and climate: turnover time and precipitation, controlled for temperature (left) and vice-versa (right). Reproduces figures 2c and 2d in [Carvalhais et al. \(2014\)](#).

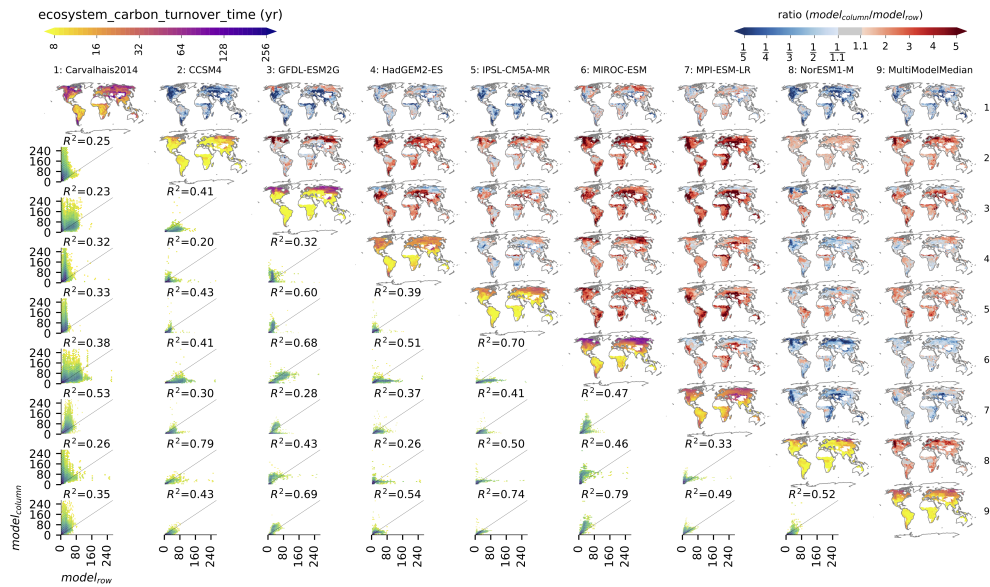


Fig. 3: Comparison of observation-based and modelled ecosystem carbon turnover time. Along the diagonal, τ_{total} are plotted, above the bias, and below density plots. The inset text in density plots indicate the correlation.

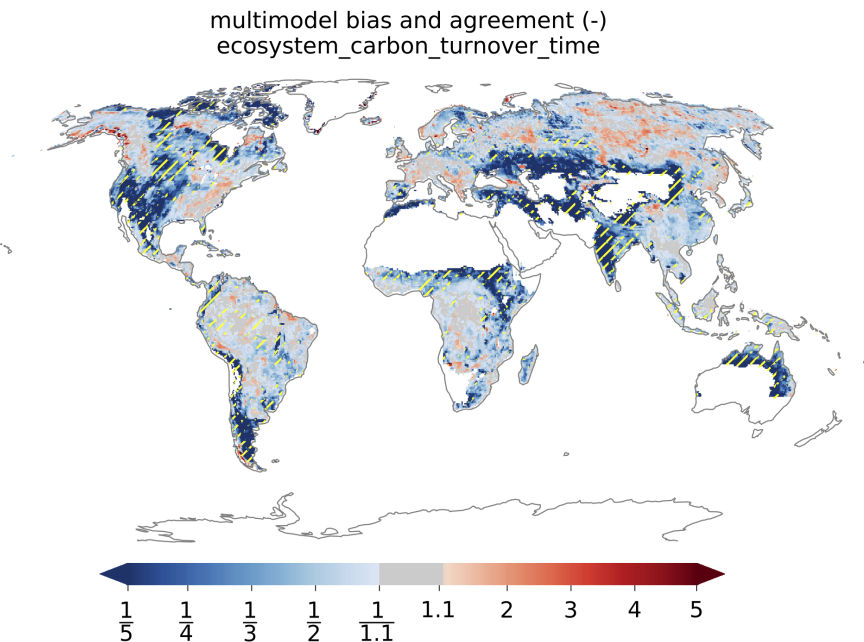


Fig. 4: Global distributions of multimodel bias and model agreement. Multimodel bias is calculated as the ratio of multimodel median turnover time and that from observation. Stippling indicates the regions where only less than one quarter of the models fall within the range of observational uncertainties (5th and 95th percentiles). Reproduces figure 3 in Carvalhais et al. (2014).

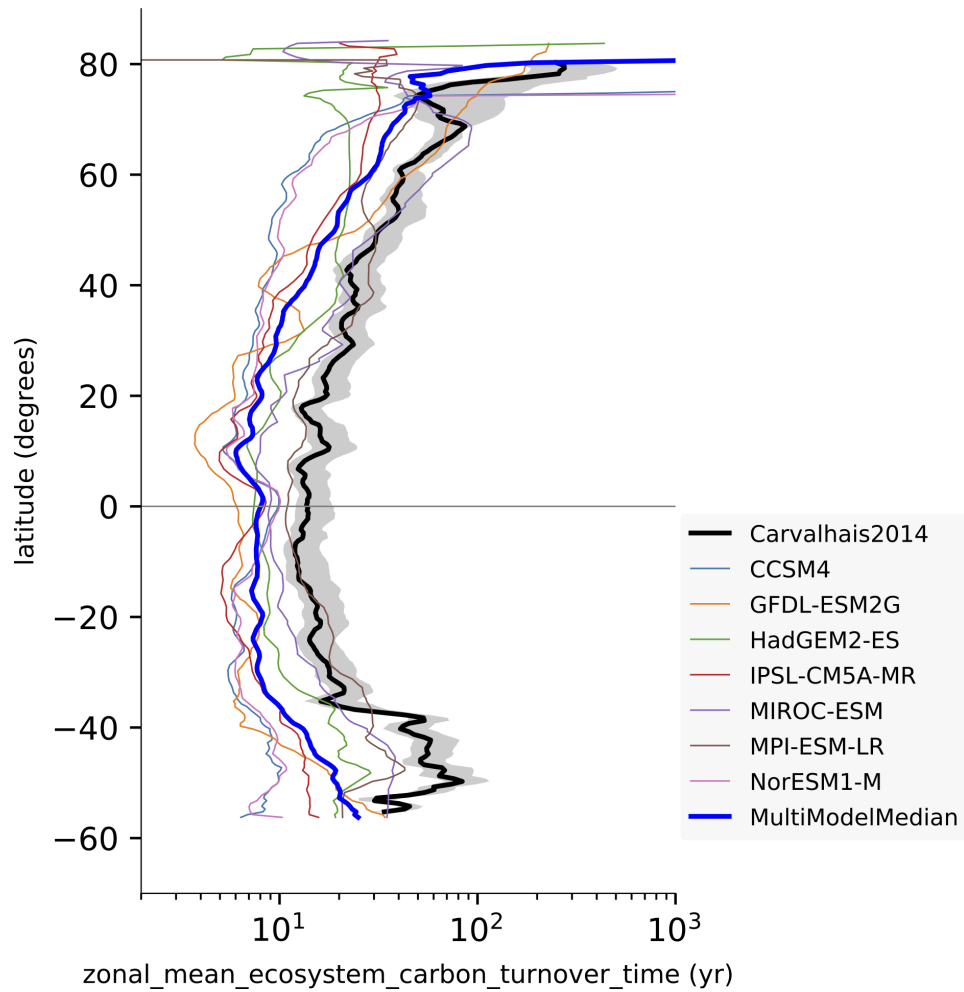


Fig. 5: Comparison of latitudinal (zonal) variations of observation-based and modelled ecosystem carbon turnover time. The zonal turnover time is calculated as the ratio of zonal *ctotal* and *gpp*. Reproduces figures 2a and 2b in Carvalhois et al. (2014).

18.3 Hydrological models - data pre-processing

18.3.1 Overview

We provide a collection of scripts that pre-processes environmental data for use in several hydrological models:

PCR-GLOBWB

PCR-GLOBWB (PCRaster Global Water Balance) is a large-scale hydrological model intended for global to regional studies and developed at the Department of Physical Geography, Utrecht University (Netherlands). The recipe pre-processes ERA-Interim reanalyses data for use in the PCR-GLOBWB.

MARRMoT

MARRMoT (Modular Assessment of Rainfall-Runoff Models Toolbox) is a rainfall-runoff model comparison framework that allows objective comparison between different conceptual hydrological model structures <https://github.com/wknoben/MARRMoT>. The recipe pre-processes ERA-Interim and ERA5 reanalyses data for use in the MARRMoT.

MARRMoT requires potential evapotranspiration (evspsblpot). The variable evspsblpot is not available in ERA-Interim. Thus, we use the `debruin` function (De Bruin et al. 2016) to obtain evspsblpot using both ERA-Interim and ERA5. This function needs the variables `tas`, `psl`, `rsds`, and `rsdt` as input.

wflow_sbm and wflow_topoflex

Forcing data for the `wflow_sbm` and `wflow_topoflex` hydrological models can be prepared using `recipe_wflow.yml`. If PET is not available from the source data (e.g. ERA-Interim), then it can be derived from `psl`, `rsds` and `rsdt` using De Bruin's 2016 formula (De Bruin et al. 2016). For daily ERA5 data, the time points of these variables are shifted 30 minutes with respect to one another. This is because in ERA5, accumulated variables are recorded over the past hour, and in the process of cmorization, we shift the time coordinates to the middle of the interval over which is accumulated. However, computing daily statistics then averages the times, which results in 12:00 UTC for accumulated variables and 11:30 UTC for instantaneous variables. Therefore, in this diagnostic, the time coordinates of the daily instantaneous variables are shifted 30 minutes forward in time.

LISFLOOD

LISFLOOD is a spatially distributed water resources model, developed by the Joint Research Centre (JRC) of the European Commission since 1997. We provide a recipe to produce meteorological forcing data for the Python 3 version of LISFLOOD.

LISFLOOD has a separate preprocessor LISVAP that derives some additional variables. We don't replace LISVAP. Rather, we provide input files that can readily be passed to LISVAP and then to LISFLOOD.

HYPE

The hydrological catchment model HYPE simulates water flow and substances on their way from precipitation through soil, river and lakes to the river outlet. HYPE is developed at the Swedish Meteorological and Hydrological Institute. The recipe pre-processes ERA-Interim and ERA5 data for use in HYPE.

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.

18.3.2 Available recipes and diagnostics

Recipes are stored in esmvaltool/recipes/hydrology

- recipe_pcrglobwb.yml
- recipe_marrmot.yml
- recipe_wflow.yml
- recipe_lisflood.yml
- recipe_hype.yml
- recipe_globwat.yml

Diagnostics are stored in esmvaltool/diag_scripts/hydrology

- pcrglobwb.py
- marrmot.py
- wflow.py
- lisflood.py
- hype.py
- globwat.py

18.3.3 User settings in recipe

All hydrological recipes require a shapefile as an input to produce forcing data. This shapefile determines the shape of the basin for which the data will be cut out and processed. All recipes are tested with [the shapefiles](#) that are used for the eWaterCycle project. In principle any shapefile can be used, for example, the freely available basin shapefiles from the [HydroSHEDS project](#).

1. recipe_pcrglobwb.yml

Required preprocessor settings:

- start_year: 1979

- end_year: 1979

2. recipe_marrmot.yml

There is one diagnostic `diagnostic_daily` for using daily data.

Required preprocessor settings:

The settings below should not be changed.

extract_shape:

- shapefile: Meuse.shp (MARRMoT is a hydrological Lumped model that needs catchment-aggregated forcing data. The catchment is provided as a shapefile, the path can be relative to `auxiliary_data_dir` as defined in `config-user.yml`).
- method: contains
- crop: true

Required diagnostic script settings:

- basin: Name of the catchment

3. recipe_wflow.yml

Optional preprocessor settings:

- extract_region: the region specified here should match the catchment

Required diagnostic script settings:

- basin: name of the catchment
- dem_file: netcdf file containing a digital elevation model with elevation in meters and coordinates latitude and longitude. 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)

18.3.4 Variables

1. `recipe_pcrglobwb.yml`

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

2. `recipe_marrmot.yml`

- pr (atmos, daily or hourly mean, longitude, latitude, time)
- psl (atmos, daily or hourly mean, longitude, latitude, time)
- rsds (atmos, daily or hourly mean, longitude, latitude, time)
- rsdt (atmos, daily or hourly mean, longitude, latitude, time)

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

3. recipe_wflow.yml

- orog (fx, longitude, latitude)
- pr (atmos, daily or hourly mean, longitude, latitude, time)
- tas (atmos, daily or hourly mean, longitude, latitude, time)

Either potential evapotranspiration can be provided:

- evspsblpot(atmos, daily or hourly mean, longitude, latitude, time)

or it can be derived from tas, psl, rsds, and rsdt using the De Bruin formula, in that case the following variables need to be provided:

- psl (atmos, daily or hourly mean, longitude, latitude, time)
- rsds (atmos, daily or hourly mean, longitude, latitude, time)
- rsdt (atmos, daily or hourly mean, longitude, latitude, time)

4. recipe_lisflood.yml

- pr (atmos, daily, longitude, latitude, time)
- tas (atmos, daily, longitude, latitude, time)
- tasmax (atmos, daily, longitude, latitude, time)
- tasmin (atmos, daily, longitude, latitude, time)
- tdps (atmos, daily, longitude, latitude, time)
- uas (atmos, daily, longitude, latitude, time)
- vas (atmos, daily, longitude, latitude, time)
- rsds (atmos, daily, longitude, latitude, time)

5. recipe_hype.yml

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

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)

18.3.5 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)

18.3.6 Output

1. recipe_pcrglobwb.yml

2. recipe_marrmot.yml

The forcing data, the start and end times of the forcing data, the latitude and longitude of the catchment are saved in a .mat file as a data structure readable by MATLAB or Octave.

3. recipe_wflow.yml

The forcing data, stored in a single NetCDF file.

4. recipe_lisflood.yml

The forcing data, stored in separate files per variable.

5. recipe_globwat.yml

The forcing data, stored in separate files per timestep and variable.

18.3.7 References

- Sutanudjaja, E. H., van Beek, R., Wanders, N., Wada, Y., Bosmans, J. H. C., Drost, N., van der Ent, R. J., de Graaf, I. E. M., Hoch, J. M., de Jong, K., Karssenberg, D., López López, P., Peßenteiner, S., Schmitz, O., Straatsma, M. W., Vannamettee, 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>)

18.4 Hydro forcing comparison

18.4.1 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 aggregated data over a defined period
3. Plot the monthly / daily climatology statistics over a defined period

18.4.2 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

18.4.3 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`.

18.4.4 Variables

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

18.4.5 Observations

All data can be used directly without any preprocessing.

- ERA-Interim
- ERA5
- MSWEP

18.4.6 Example plots

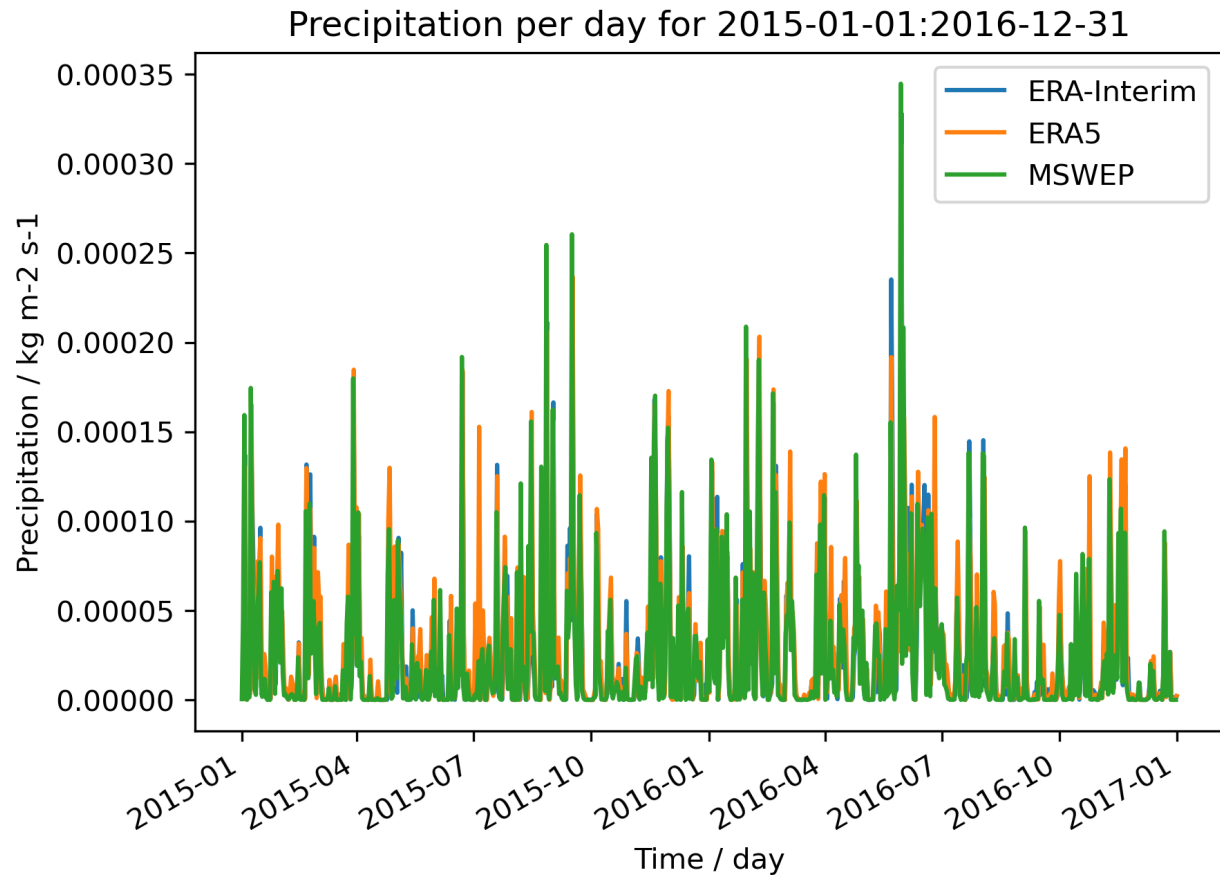


Fig. 6: Precipitation per day for 2015-01-01:2016-12-31.

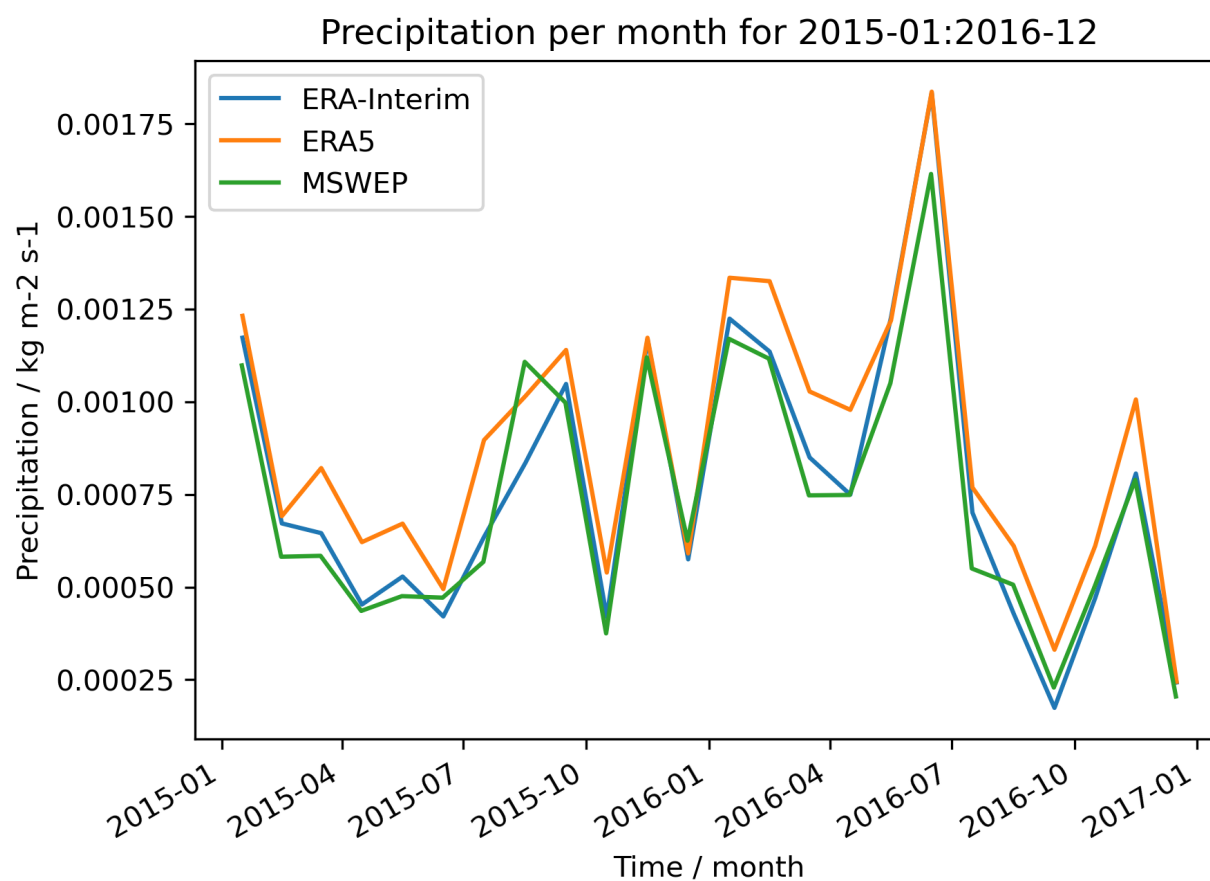


Fig. 7: Precipitation per month for 2015-01:2016-12.

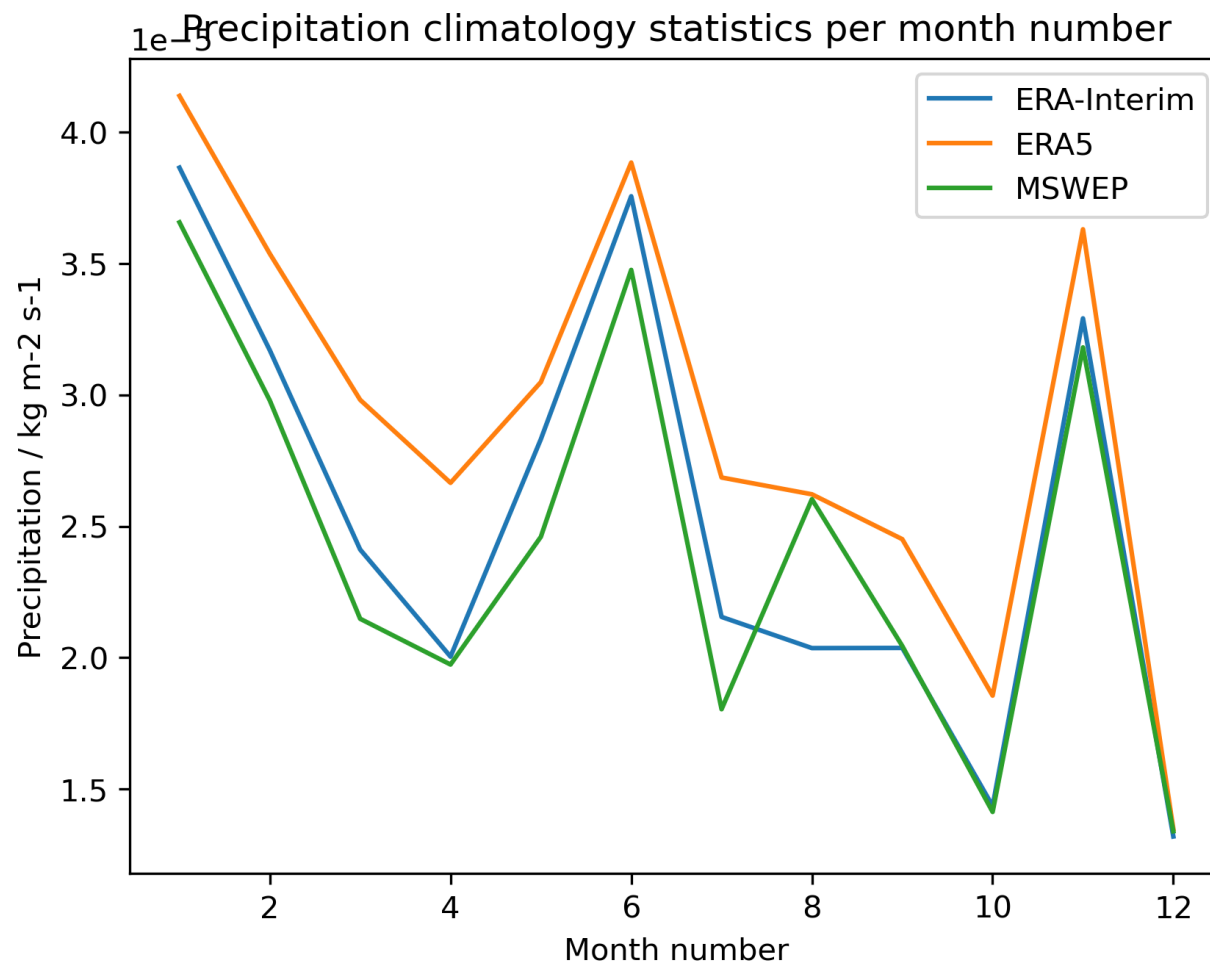


Fig. 8: Precipitation climatology statistics per month number.

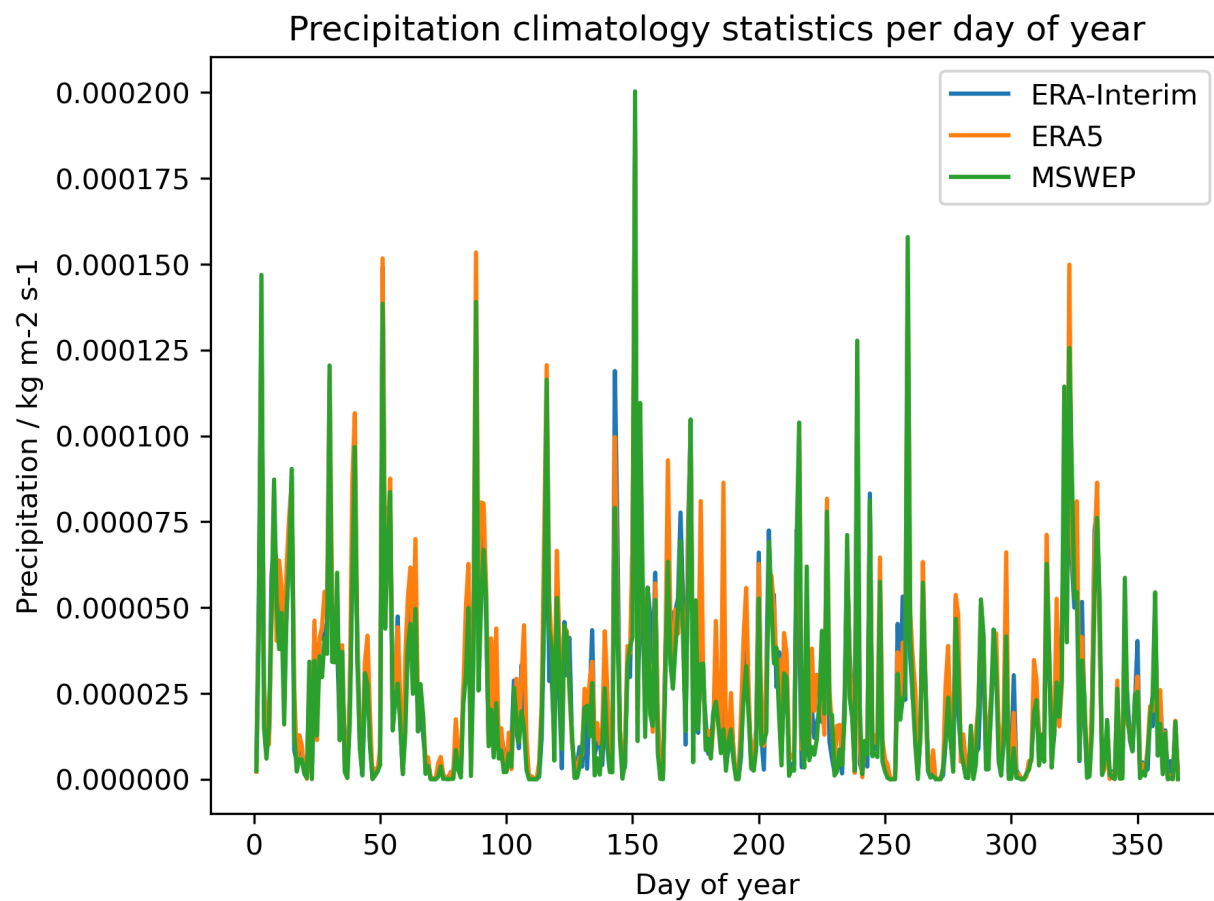


Fig. 9: Precipitation climatology statistics per day of year.

18.5 Landcover diagnostics

18.5.1 Overview

The diagnostic computes the accumulated and fractional extent of major land cover classes, namely bare soil, crops, grasses, shrubs and trees. The numbers are compiled for the whole land surface as well as separated into Tropics, northern Extratropics and southern Extratropics. The cover fractions are compared to ESA-CCI land cover data.

18.5.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_landcover.yml`

Diagnostics are stored in `diag_scripts/landcover/`

- `landcover.py`: bar plots showing the accumulated area and mean fractional coverage for five land cover classes for all experiments as well as their bias compared to observations.

18.5.3 User settings

script `landcover.py`

Required settings for script

- `reference_dataset`: land cover extent dataset for comparison. The script was developed using ESACCI-LANDCOVER observations.

Optional settings for script

- `comparison`: [variable, model] Choose whether one plot per land cover class is generated comparing the different experiments (default) or one plot per model comparing the different land cover classes.
- `colorscheme`: Plotstyle used for the bar plots. A list of available style is found at https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html. Seaborn is used as default.

18.5.4 Variables

- `baresoilFrac` (land, monthly mean, time latitude longitude)
- `grassFrac` (land, monthly mean, time latitude longitude)
- `treeFrac` (land, monthly mean, time latitude longitude)
- `shrubFrac` (land, monthly mean, time latitude longitude)
- `cropFrac` (land, monthly mean, time latitude longitude)

18.5.5 Observations and reformat scripts

ESA-CCI land cover data (Defourny et al., 2015) needs to be downloaded manually by the user and converted to netCDF files containing the grid cell fractions for the five major land cover types. The data and a conversion tool are available at <https://maps.elie.ucl.ac.be/CCI/viewer/> upon registration. After obtaining the data and the user tool, the remapping to 0.5 degree can be done with:

```
./bin/aggregate-map.sh
-PgridName=GEOGRAPHIC_LAT_LON
-PnumRows=360
-PoutputLCCSClasses=true
-PnumMajorityClasses=0
ESACCI-LC-L4-LCCS-Map-300m-P1Y-2015-v2.0.7b.nc
```

Next, the data needs to be aggregated into the five major classes (PFT) similar to the study of Georgievski & Hagemann (2018) and converted from grid cell fraction into percentage.

PFT	ESA-CCI Landcover Classes
baresoil-Frac	Bare_Soil
cropFrac	Managed_Grass
grassFrac	Natural_Grass
shrubFrac	Shrub_Broadleaf_Deciduous + Shrub_Broadleaf_Evergreen + Shrub_Needleleaf_Evergreen
treeFrac	Tree_Broadleaf_Deciduous + Tree_Broadleaf_Evergreen + Tree_Needleleaf_Deciduous + Tree_Needleleaf_Evergreen

Finally, it might be necessary to adapt the grid structure to the experiments files, e.g converting the -180 → 180 degree grid to 0 → 360 degree and inverting the order of latitudes. Note, that all experiments will be regridded onto the grid of the land cover observations, thus it is recommended to convert to the coarser 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.

18.5.6 References

- Defourny et al. (2015): ESA Land Cover Climate Change Initiative (ESA LC_cci) data: ESACCI-LC-L4-LCCS-Map-300m-P5Y-[2000,2005,2010]-v1.6.1 via Centre for Environmental Data Analysis
- Georgievski, G. & Hagemann, S. Characterizing uncertainties in the ESA-CCI land cover map of the epoch 2010 and their impacts on MPI-ESM climate simulations, Theor Appl Climatol (2018). <https://doi.org/10.1007/s00704-018-2675-2>

18.5.7 Example plots

18.6 Land and ocean components of the global carbon cycle

18.6.1 Overview

This recipe reproduces most of the figures of Anav et al. (2013):

- Timeseries plot for different regions
- Seasonal cycle plot for different regions

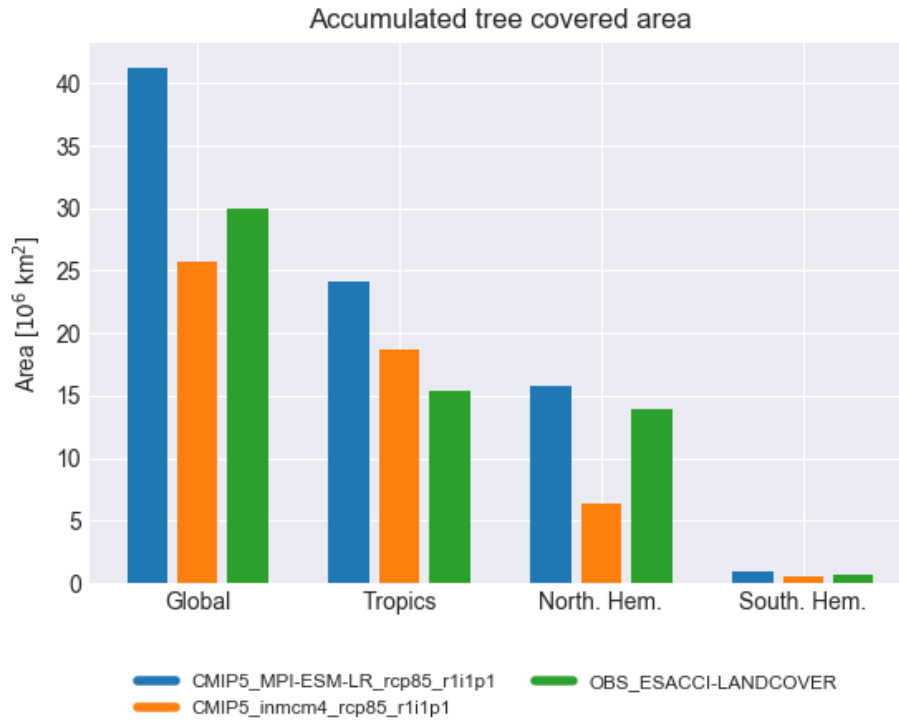


Fig. 10: Accumulated tree covered area for different regions and experiments.

- Errorbar plot for different regions showing mean and standard deviation
- Scatterplot for different regions showing mean vs. interannual variability
- 3D-scatterplot for different regions showing mean vs. linear trend and the model variability index (MVI) as a third dimension (color coded)
- Scatterplot for different regions comparing two variable against each other (*cSoil* vs. *cVeg*)

In addition, performance metrics are calculated for all variables using the performance metric diagnostics (see details in [Performance metrics for essential climate parameters](#)).

18.6.2 MVI calculation

The Model variability index (MVI) on a single grid point (calculated in `carbon_cycle/mvi.nc1` is defined as

$$MVI = \left(\frac{s^M}{s^O} - \frac{s^O}{s^M} \right)^2$$

where s^M and s^O are the standard deviations of the annual time series on a single grid point of a climate model M and the reference observation O . In order to get a global or regional result, this index is simple averaged over the respective domain.

In its given form, this equation is prone to small standard deviations close to zero. For example, values of $s^M = 10^{-5}\mu$ and $s^O = 10^{-7}\mu$ (where μ is the mean of s^O over all grid cells) results in a MVI of the order of 10^4 for this single grid cell even though the two standard deviations are close to zero and negligible compared to other grid cells. Due to the use of the arithmetic mean, a single high value is able to distort the overall MVI.

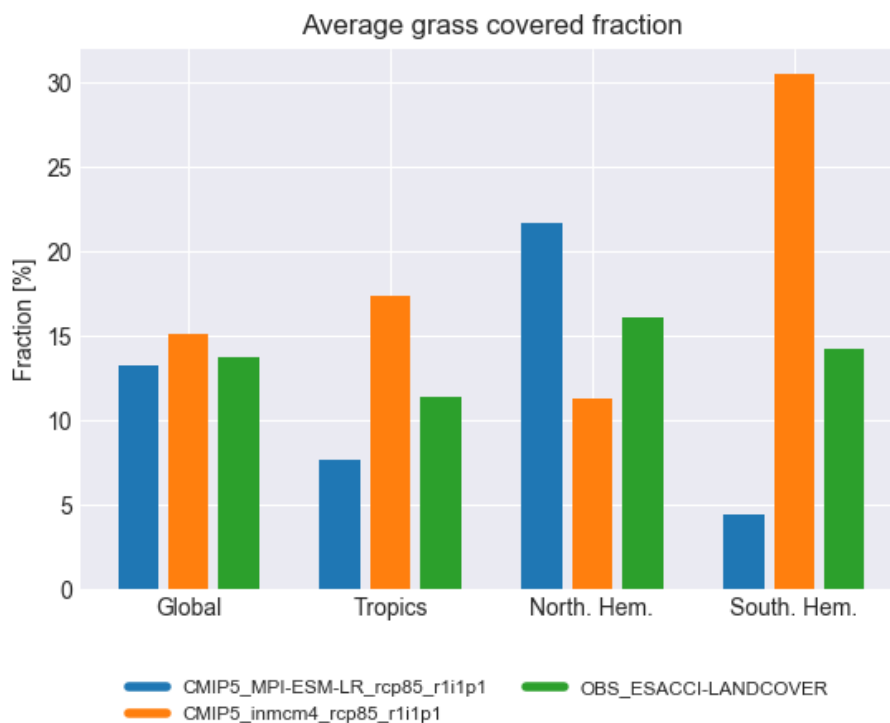


Fig. 11: Average grass cover fraction for different regions and experiments

In the original publication, the maximum MVI is in the order of 10 (for the variable *gpp*). However, a naive application of the MVI definition yields values over 10^9 for some models. Unfortunately, [Anav et al. \(2013\)](#) do not provide an explanation on how to deal with this problem. Nevertheless, this script provides two configuration options to avoid high MVI values, but they are not related to the original paper or any other peer-reviewed study and should be used with great caution (see *User settings in recipe*).

18.6.3 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_anav13jclim.yml`

Diagnostics are stored in `diag_scripts/`

- `carbon_cycle/main.ncl`
- `carbon_cycle/mvi.ncl`
- `carbon_cycle/two_variables.ncl`
- `perfmetrics/main.ncl`
- `perfmetrics/collect.ncl`

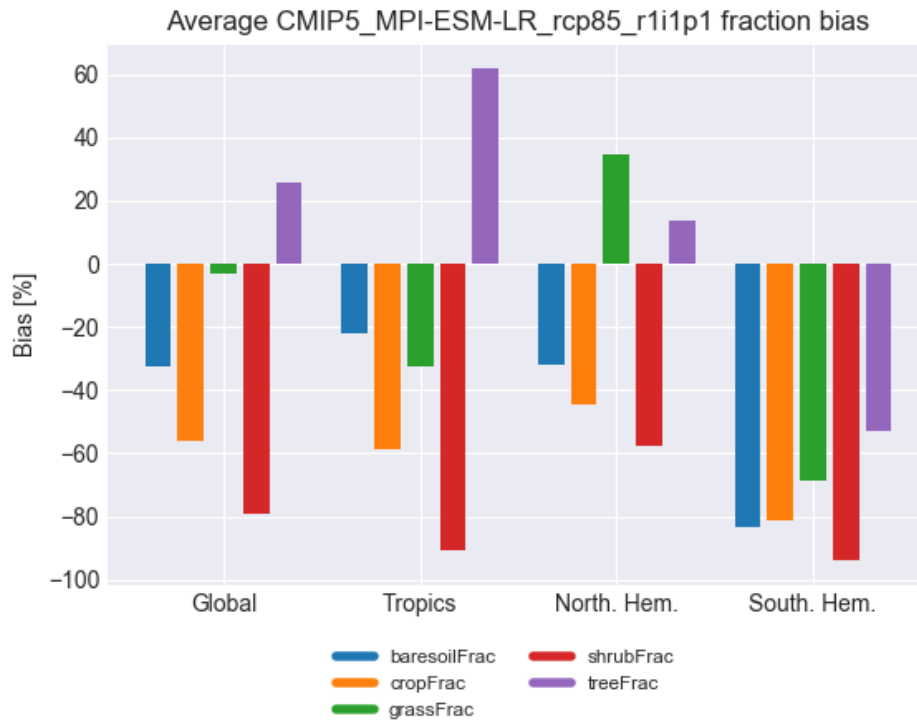


Fig. 12: Biases in five major land cover fractions for different regions and one experiment.

18.6.4 User settings in recipe

1. Preprocessor

- `mask_fillvalues`: Mask common missing values on different datasets.
- `mask_landsea`: Mask land/ocean.
- `regrid`: Regridding.
- `weighting_landsea_fraction`: Land/ocean fraction weighting.

2. Script carbon_cycle/main.ncl

- `region, str`: Region to be averaged.
- `legend_outside, bool`: Plot legend in a separate file (does not affect errorbar plot and evolution plot)
- `seasonal_cycle_plot, bool`: Draw seasonal cycle plot.
- `errorbar_plot, bool`: Draw errorbar plot.
- `mean_IAV_plot, bool`: Draw Mean (x-axis), IAV (y-axis) plot.
- `evolution_plot, bool`: Draw time evolution of a variable comparing a reference dataset to multi-dataset mean; requires `ref_dataset` in recipe.
- `sort, bool`, optional (default: False): Sort dataset in alphabetical order.
- `anav_month, bool`, optional (default: False): Conversion of y-axis to PgC/month instead of /year.

- `evolution_plot_ref_dataset`, *str*, optional: Reference dataset for `evolution_plot`. Required when `evolution_plot` is `True`.
- `evolution_plot_anomaly`, *str*, optional (default: `False`): Plot anomalies in evolution plot.
- `evolution_plot_ignore`, *list*, optional: Datasets to ignore in evolution plot.
- `evolution_plot_volcanoes`, *bool*, optional (default: `False`): Turns on/off lines of volcano eruptions in evolution plot.
- `evolution_plot_color`, *int*, optional (default: `0`): Hue of the contours in the evolution plot.
- `ensemble_name`, *string*, optional: Name of ensemble for use in evolution plot legend

3. Script `carbon_cycle/mvi.ncl`

- `region`, *str*: Region to be averaged.
- `reference_dataset`, *str*: Reference dataset for the MVI calculation specified for each variable separately.
- `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](#).

18.6.5 Variables

- *tas* (atmos, monthly, longitude, latitude, time)
- *pr* (atmos, monthly, longitude, latitude, time)
- *nbp* (land, monthly, longitude, latitude, time)
- *gpp* (land, monthly, longitude, latitude, time)
- *lai* (land, monthly, longitude, latitude, time)
- *cveg* (land, monthly, longitude, latitude, time)
- *csoil* (land, monthly, longitude, latitude, time)
- *tos* (ocean, monthly, longitude, latitude, time)
- *fgco2* (ocean, monthly, longitude, latitude, time)

18.6.6 Observations and reformat scripts

- CRU (*tas, pr*)
- JMA-TRANSCOM (*nbp, fgco2*)
- MTE (*gpp*)
- LAI3g (*lai*)
- NDP (*cveg*)
- HWSO (*csoil*)
- HadISST (*tos*)

18.6.7 References

- Anav, A. et al.: Evaluating the land and ocean components of the global carbon cycle in the CMIP5 Earth System Models, *J. Climate*, 26, 6901-6843, doi: 10.1175/JCLI-D-12-00417.1, 2013.

18.6.8 Example plots

18.7 Runoff, Precipitation, Evapotranspiration

18.7.1 Overview

This diagnostic calculates biases of long-term climatological annual means of total runoff R , precipitation P and evapotranspiration E for 12 large-scale catchments on different continents and climates. For total runoff, catchment averaged model values are compared to climatological GRDC station observations of river runoff (Duemenil Gates et al., 2000). Due to the incompleteness of these station data, a year-to-year correspondence of data cannot be achieved in a generalized way, so that only climatological data are considered, such it has been done in Hagemann, et al. (2013). For precipitation, catchment-averaged WFDEI precipitation data (Weedon et al., 2014) from 1979-2010 is used as reference. For evapotranspiration, observations are estimated using the difference of the above mentioned precipitation reference minus the climatological GRDC river runoff.

The catchments are Amazon, Congo, Danube, Ganges-Brahmaputra, Lena, Mackenzie, Mississippi, Murray, Niger, Nile, Parana and Yangtze-Kiang. Variable names are expected to follow CMOR standard, e.g. precipitation as *pr*, total runoff as *mrro* and evapotranspiration as *evspsbl* with all fluxes given in $\text{kg m}^{-2} \text{s}^{-1}$. Evapotranspiration furthermore has to be defined positive upwards.

The diagnostic produces text files with absolute and relative bias to the observations, as well as the respective absolute values. Furthermore it creates a bar plot for relative and absolute bias, calculates and plots biases in runoff coefficient (R/P) and evapotranspiration coefficient (E/P) and saves everything as one pdf file per model or one png file per model and analysis.

The bias of the runoff coefficient is calculated via: $C_R = \frac{R_{\text{model}}}{P_{\text{model}}} - \frac{R_{\text{GRDC}}}{P_{\text{WFDEI}}}$ and similar for the evapotranspiration coefficient. In a very first approximation, evapotranspiration and runoff are determined only by precipitation. In other words $R = P - E$. Hence, the runoff coefficient (and similar the evapotranspiration coefficient) tells you how important runoff (or evapotranspiration) is in this region. By plotting the bias of the runoff coefficient against the evapotranspiration coefficient we can immediately see whether there is a shift from runoff to evapotranspiration. On the other hand, by plotting the bias of the runoff coefficient against the relative bias of precipitation we can see whether an error in runoff is due to an error in precipitation.

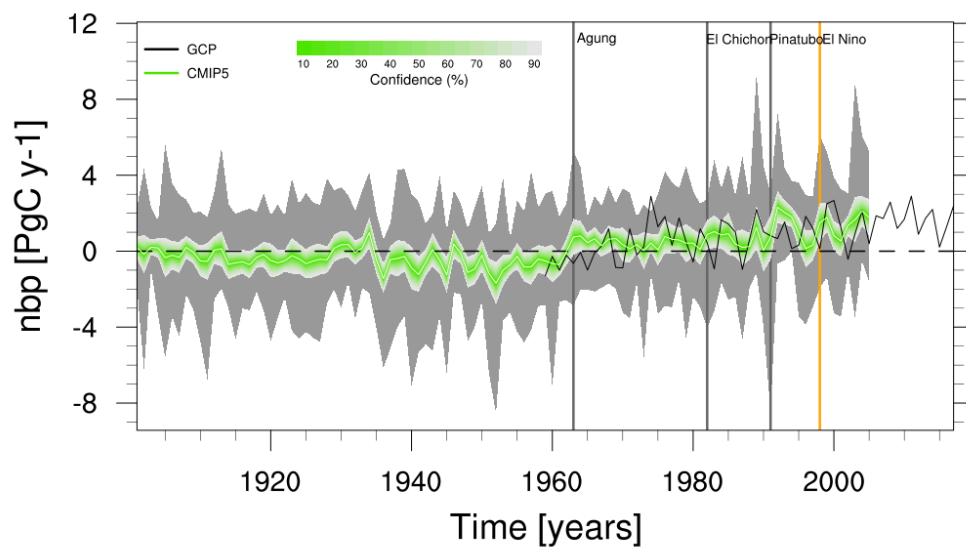


Fig. 13: Time series of global net biome productivity (NBP) over the period 1901-2005. Similar to Anav et al. (2013), Figure 5.

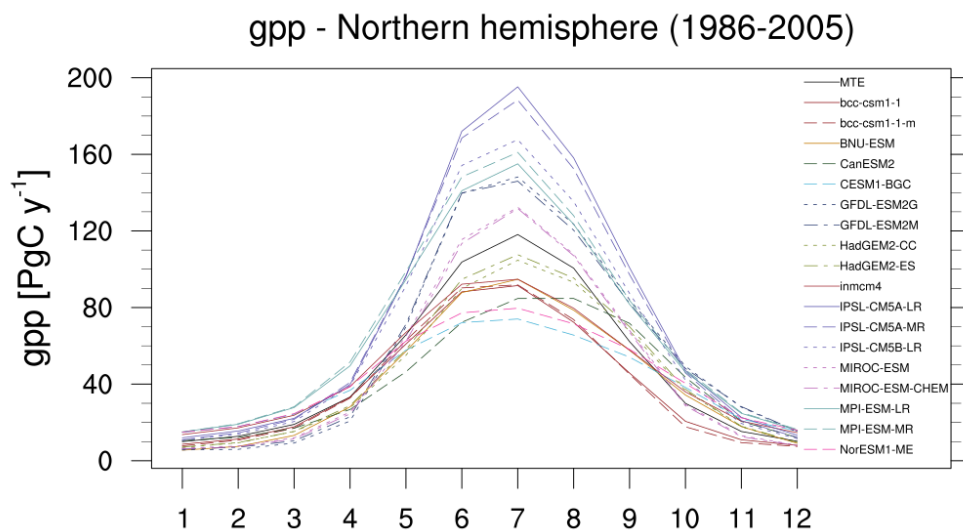


Fig. 14: Seasonal cycle plot for northern hemisphere gross primary production (GPP) over the period 1986-2005. Similar to Anav et al. (2013), Figure 9.

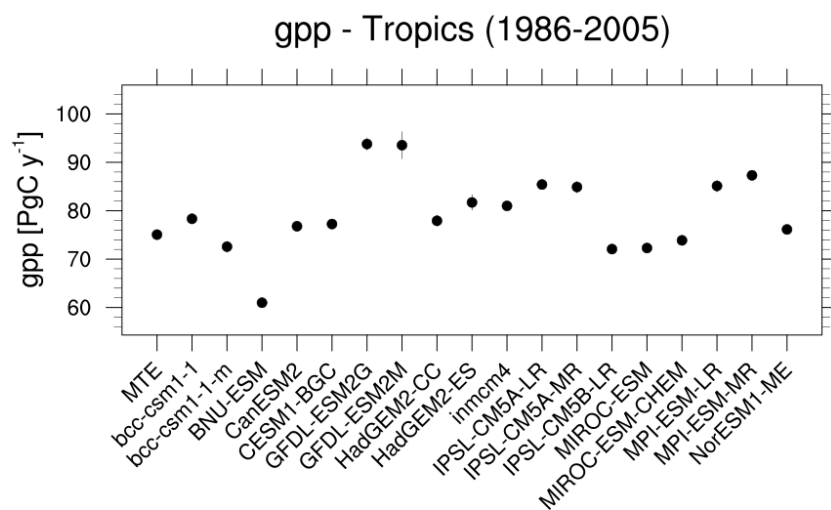


Fig. 15: Errorbar plot for tropical gross primary production (GPP) over the period 1986-2005.

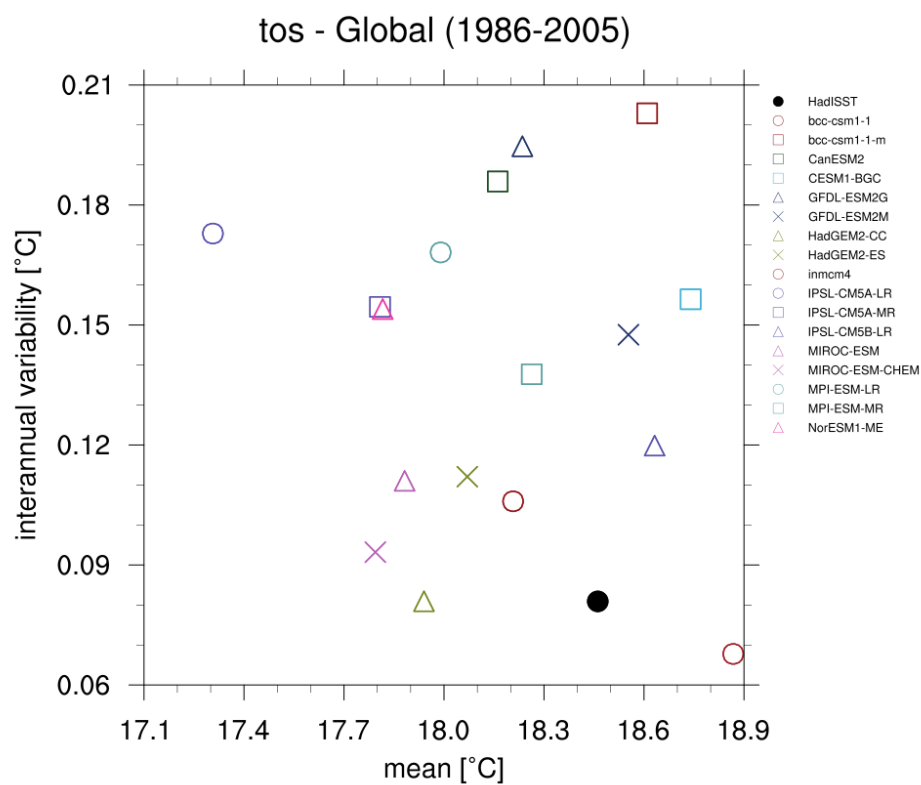


Fig. 16: Scatterplot for interannual variability and mean of global sea surface temperature (TOS) over the period 1986-2005.

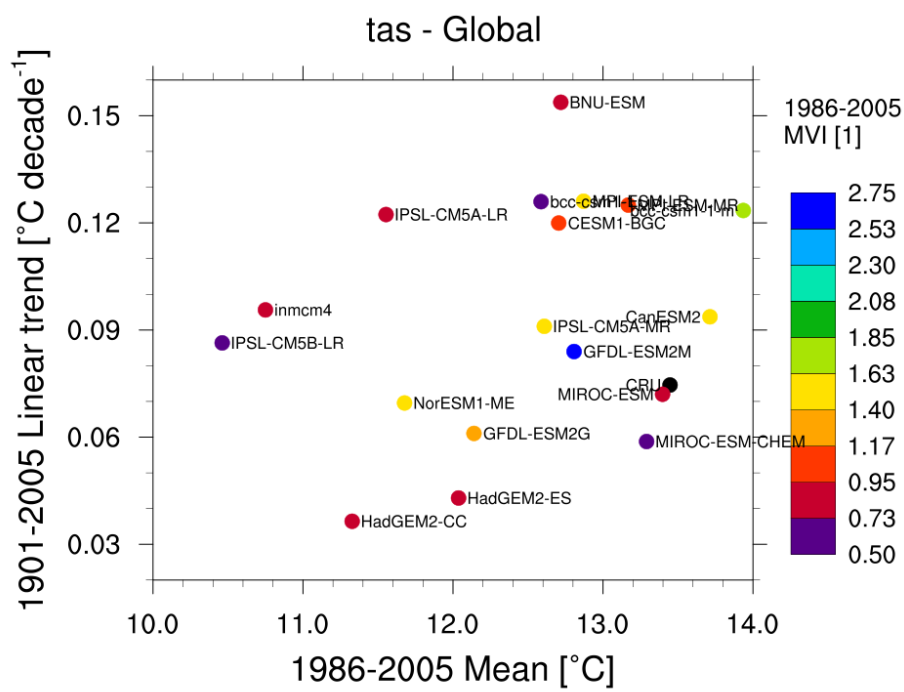


Fig. 17: Scatterplot for multiyear average of 2m surface temperature (TAS) in x axis, its linear trend in y axis, and MVI. Similar to Anav et al. (2013) Figure 1 (bottom).

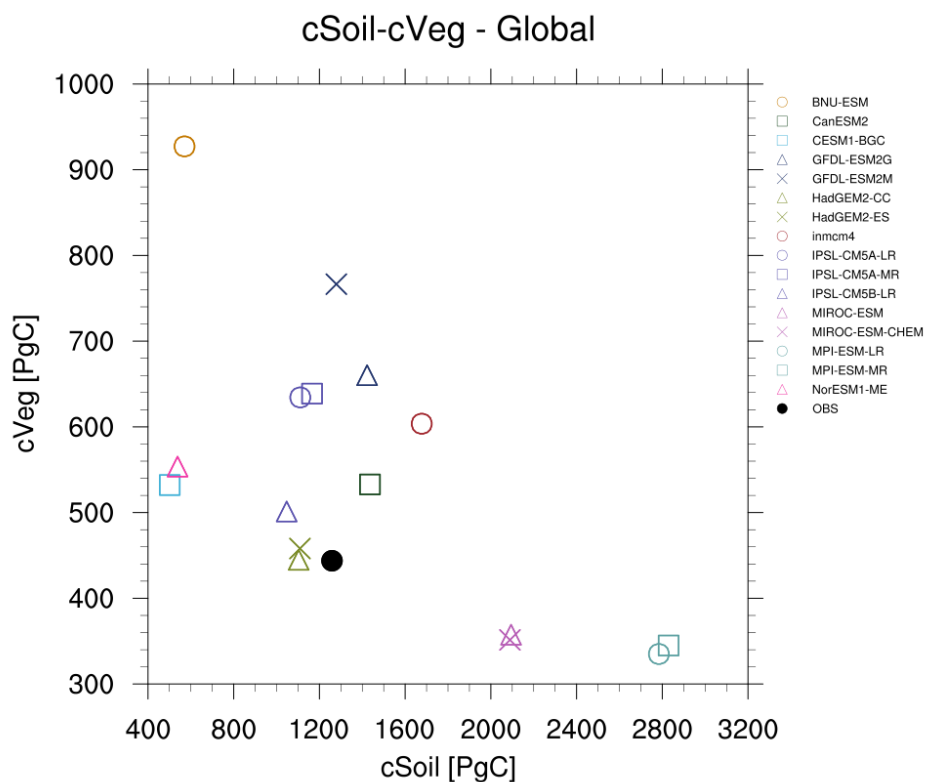


Fig. 18: Scatterplot for vegetation carbon content (cVeg) and soil carbon content (cSoil) over the period 1986-2005. Similar to Anav et al. (2013), Figure 12.

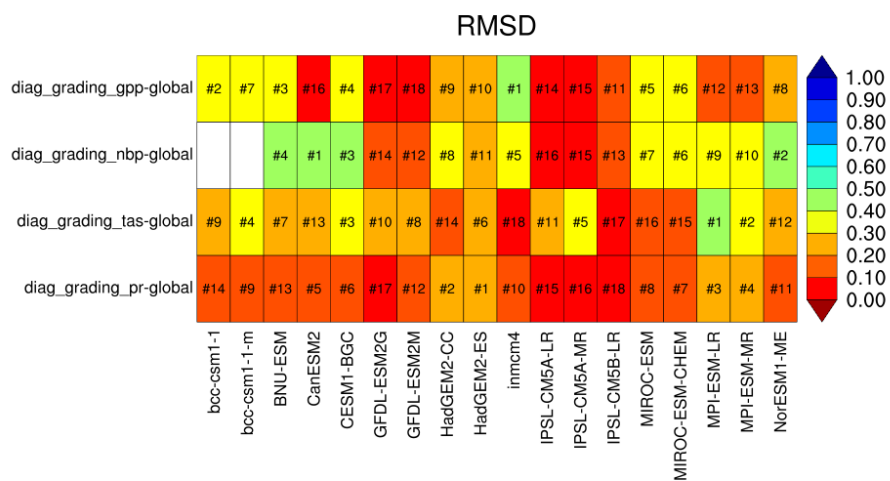


Fig. 19: Performance metrics plot for carbon-cycle-relevant diagnostics.

18.7.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_runoff_et.yml`

Diagnostics are stored in `diag_scripts/runoff_et/`

- `catchment_analysis.py`: bar and scatter plots for catchment averages of runoff, evapotranspiration and precipitation

18.7.3 User settings in recipe

1. Script `catchment_analysis.py`

Required settings (scripts)

- `catchmentmask`: netCDF file indicating the grid cell for a specific catchment. Modus of distribution not yet clarified. 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.

18.7.4 Variables

- `evspsbl` (atmos, monthly mean, time latitude longitude)
- `pr` (atmos, monthly mean, time latitude longitude)
- `mrro` (land, monthly mean, time latitude longitude)

18.7.5 Observations and reformat scripts

Default reference data based on GRDC and WFDEI are included in the diagnostic script as catchment averages. They can be replaced with any gridded dataset by defining a `reference_dataset`. The necessary catchment mask is available at

All other datasets are remapped onto the catchment mask grid as part of the diagnostics.

18.7.6 References

- Duemenil Gates, L., S. Hagemann and C. Golz, Observed historical discharge data from major rivers for climate model validation. Max Planck Institute for Meteorology Report 307, Hamburg, Germany, 2000.
- Hagemann, S., A. Loew, A. Andersson, Combined evaluation of MPI-ESM land surface water and energy fluxes J. Adv. Model. Earth Syst., 5, doi:10.1029/2012MS000173, 2013.
- Weedon, G. P., G. Balsamo, N. Bellouin, S. Gomes, M. J. Best, and P. Viterbo, The WFDEI meteorological forcing data set: WATCH Forcing Data methodology applied to ERA-Interim reanalysis data, Water Resour. Res., 50, 7505–7514, doi: 10.1002/2014WR015638, 2014

18.7.7 Example plots

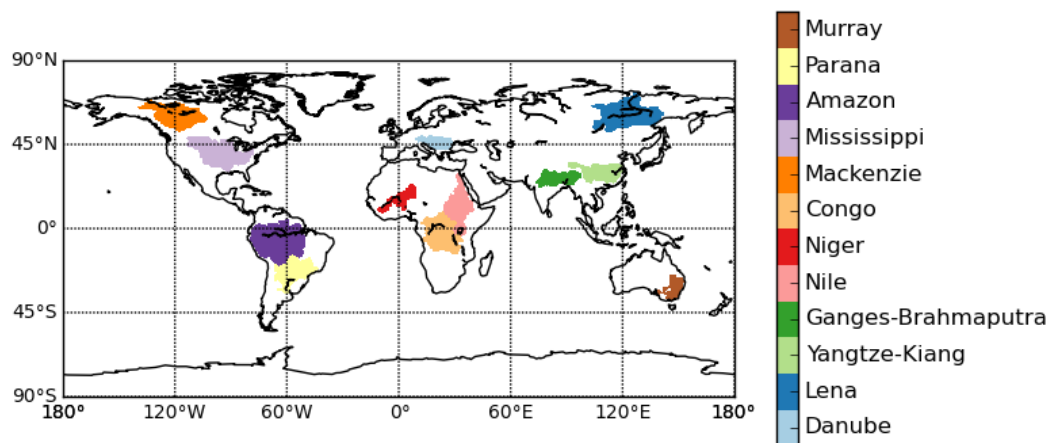


Fig. 20: Catchment definitions used in the diagnostics.

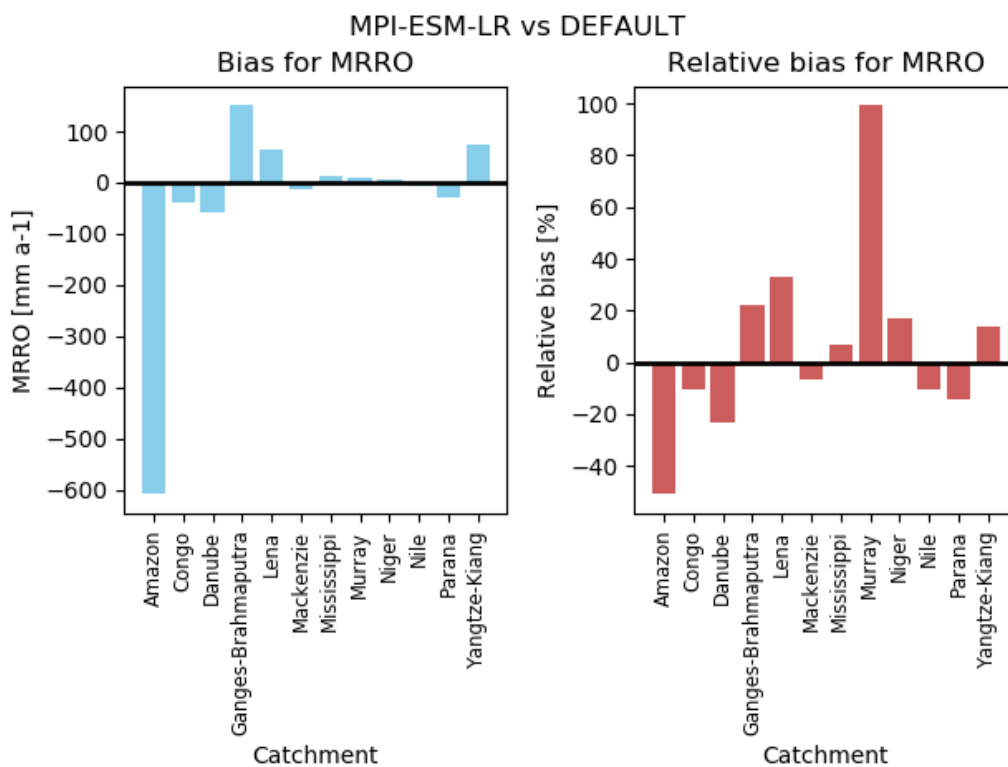


Fig. 21: Barplot indicating the absolute and relative bias in annual runoff between MPI-ESM-LR (1970-2000) and long term GRDC data for specific catchments.

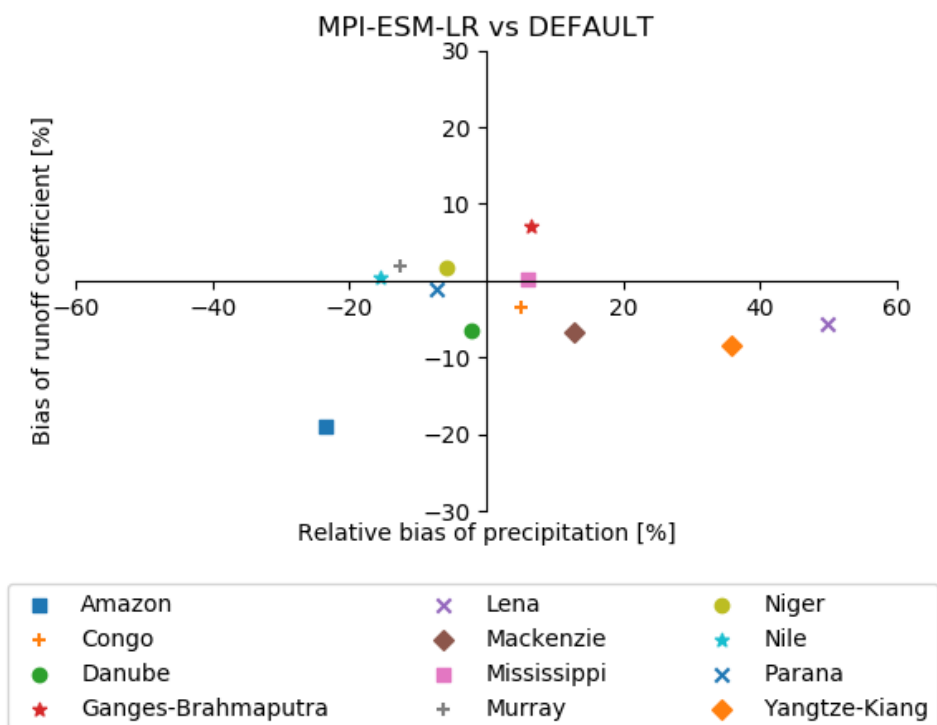


Fig. 22: Biases in runoff coefficient (runoff/precipitation) and precipitation for major catchments of the globe. The MPI-ESM-LR historical simulation (1970-2000) is used as an example.

19.1 Recipe for evaluating Arctic Ocean

19.1.1 Overview

The Arctic ocean is one of the areas of the Earth where the effects of climate change are especially visible today. Two most prominent processes are Arctic amplification [e.g. Serreze and Barry, 2011] and decrease of the sea ice area and thickness. Both receive good coverage in the literature and are already well-studied. Much less attention is paid to the interior of the Arctic Ocean itself. In order to increase our confidence in projections of the Arctic climate future proper representation of the Arctic Ocean hydrography is necessary.

The main focus of this diagnostics is evaluation of ocean components of climate models in the Arctic Ocean, however most of the diagnostics are implemented in a way that can be easily expanded to other parts of the World Ocean. Most of the diagnostics aim at model comparison to climatological data (PHC3), so we target historical CMIP simulations. However scenario runs also can be analysed to have an impression of how Arctic Ocean hydrography will change 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.

19.1.2 Available recipes

Recipe is stored in recipes/

- `recipe_arctic_ocean.yml` : contains all settings necessary to run diagnostics and metrics.

Currently the workflow does not allow to easily separate diagnostics from each other, since some of the diagnostics rely on the results of other diagnostics. The recipe currently does not use preprocessor options, so input files are CMORised monthly mean 3D ocean variables 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 plotted.
hofm_plot: True
# Define colormap (as a list, same size as list with variables)
# Only cmaps from matplotlib and cmocean are supported.
# Additional cmap - 'custom_salinity1'.
hofm_cmap: ['Spectral_r', 'custom_salinity1']
# Data limits for plots,
# List of the same size as the list of the variables
# each entry is [vmin, vmax, number of levels, rounding limit]
hofm_limits: [[-2, 2.3, 41, 1], [30.5, 35.1, 47, 2]]
# Number of columns in the plot
hofm_ncol: 3
```

Vertical profiles

The vertical structure of temperature and salinity (T and S) in the ocean model is a key diagnostic that is used for ocean model evaluation. Realistic T and S distributions means that model properly represent dynamic and thermodynamic processes in the ocean. Different ocean basins have different hydrological regimes so it is important to perform analysis of vertical TS distribution for different basins separately. The basic diagnostic in this sense is mean vertical profiles of temperature and salinity over some basin averaged for relatively long period of time. In addition to individual vertical profiles for every model, we also show the mean over all participating models and similar profile from climatological data (PHC3).

Several settings for vertical profiles (region, variables, maximum depths) will be determined by the Hovmoeller diagram settings. The reason is that vertical profiles are calculated from Hovmoeller diagram data. Mean vertical profile is calculated by linearly interpolating data on standard WOA/PHC depths.

Related settings in the recipe:

```
# Define regions, as a list.
# 'EB' - Eurasian Basin of the Arctic Ocean
# 'AB' - Amerasian Basin of the Arctic Ocean
```

(continues on next page)

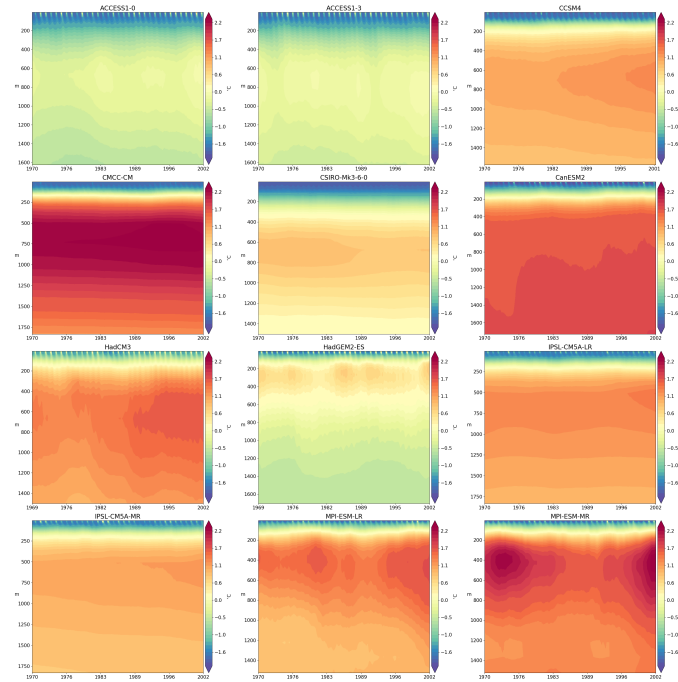


Fig. 1: Hovmöller diagram of monthly spatially averaged potential temperature in the Eurasian Basin of the Arctic Ocean for selected CMIP5 climate models (1970-2005).

(continued from previous page)

```
# '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
```

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 either 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)

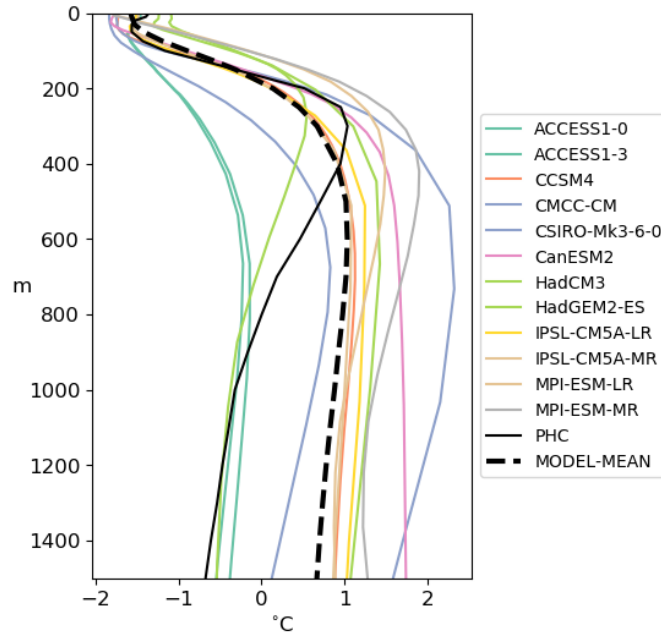


Fig. 2: Mean (1970-2005) vertical potential temperature distribution in the Eurasian basin for participating CMIP5 coupled ocean models, PHC3 climatology (dotted red line) and multi-model mean (dotted black line).

(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
```

Spatial distribution maps of biases

For temperature and salinity, we have implemented spatial maps of model biases from the observed climatology. For the model biases, values from the original model levels are linearly interpolated to the climatology and then spatially interpolated from the model grid to the regular PHC (climatology) grid. Resulting fields show model performance in simulating spatial distribution of temperature and salinity.

Related settings in the recipe:

```
plot2d_bias_depths: [10, 100]
# Variables to plot spatial distribution of the bias for.
plot2d_bias_vars: ['thetao', 'so']
# Color map names for every variable
plot2d_bias_cmap: ['balance', 'balance']
```

(continues on next page)

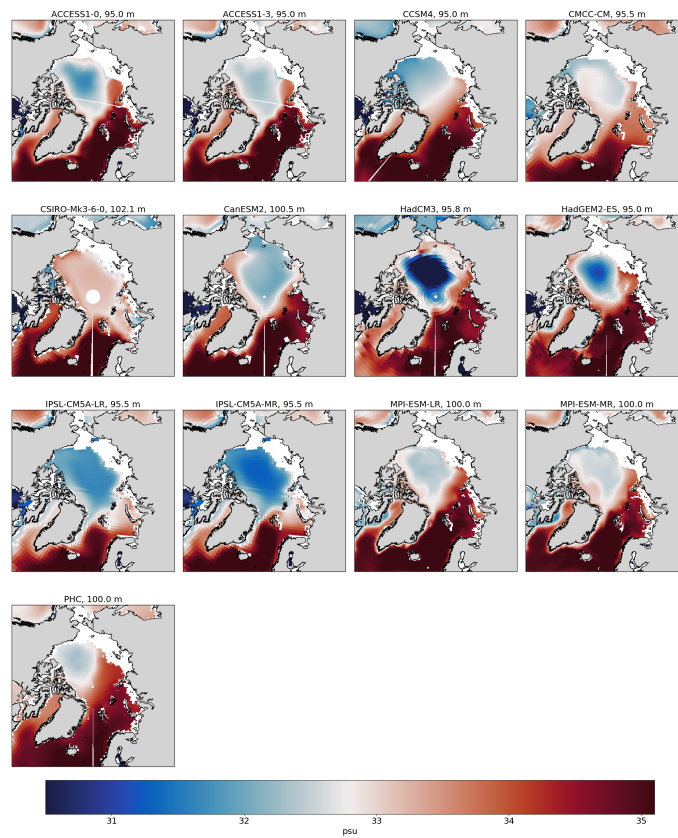


Fig. 3: Mean (1970-2005) salinity distribution at 100 meters.

(continued from previous page)

```

# 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

```

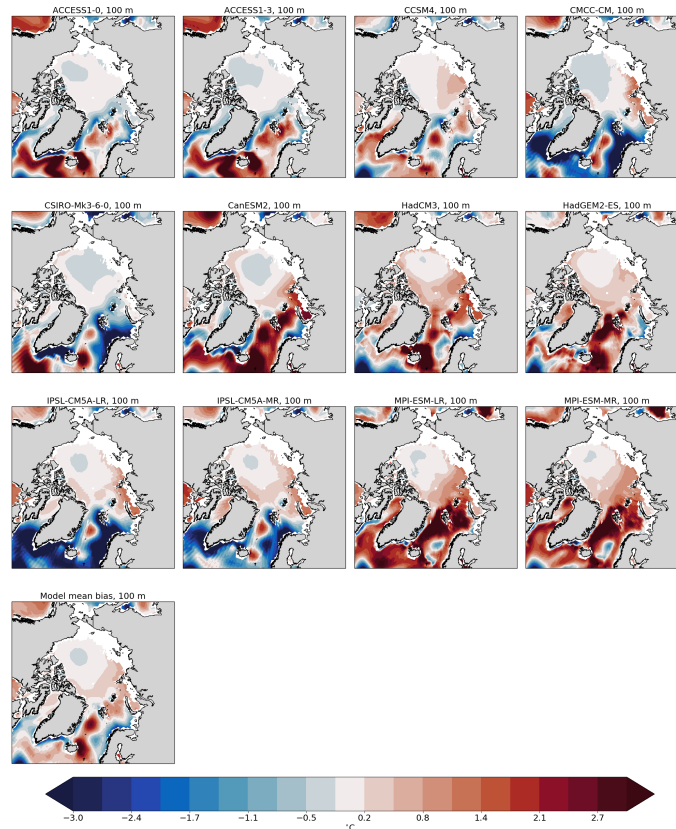


Fig. 4: Mean (1970-2005) salinity bias at 100m relative to PHC3 climatology

Transects

Vertical transects through arbitrary sections are important for analysis of vertical distribution of ocean water properties and especially useful when exchange between different ocean basins is evaluated. We have implemented diagnostics that allow for the definition of an arbitrary ocean section by providing set of points on the ocean surface. For each point, a vertical profile on the original model levels is interpolated. All profiles are then connected to form a transect. The great-circle distance between the points is calculated and used as along-track distance.

One of the main use cases is to create vertical sections across ocean passages, for example Fram Strait.

Plots transect maps for pre-defined set of transects (defined in *regions.py*, see below). The *transect_depth* defines maximum depth of the transect. Transects are calculated from data averaged over the whole time period.

Related settings in the recipe:


```

# Select regions (transects) to plot
# Available options are:
# AWpath - transect along the path of the Atlantic Water
# Fram - Fram strait
transects_regions: ["AWpath", "Fram"]
# Variables to plot on transects
transects_vars: ['thetao', 'so']
# Color maps for every variable
transects_cmap: ['Spectral_r', 'custom_salinity1']
# Data limits for plots,
# List of the same size as the list of the variables
# each entry is [vmin, vmax, number of levels, rounding limit]
transects_limits: [[-2, 4, 20, 1], [30.5, 35.1, 47, 2]]
# Maximum depth to plot the data
transects_depth: 1500
# number of columns
transects_ncol: 3

```

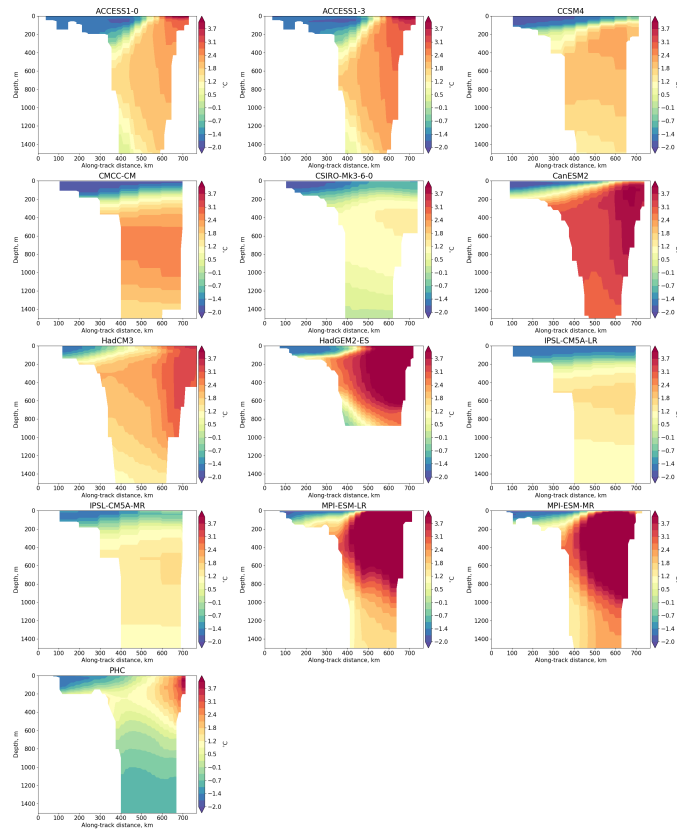


Fig. 5: Mean (1970-2005) potential temperature across the Fram strait.

Atlantic Water core depth and temperature

Atlantic water is a key water mass of the Arctic Ocean and its proper representation is one of the main challenges in Arctic Ocean modelling. We have created two metrics by which models can be easily compared in terms of Atlantic water simulation. The temperature of the Atlantic Water core is calculated for every model as the maximum potential temperature between 200 and 1000 meters depth in the Eurasian Basin. The depth of the Atlantic Water core is calculated as the model level depth where the maximum temperature is found in Eurasian Basin (Atlantic water core temperature).

The AW core depth and temperature will be calculated from data generated for Hovmoeller diagrams for *EB* region, so it should be selected in the Hovmoeller diagrams settings as one of the *hofm_regions*.

In order to evaluate the spatial distribution of Atlantic water in different climate models we also provide diagnostics with maps of the spatial temperature distribution at model's Atlantic Water depth.

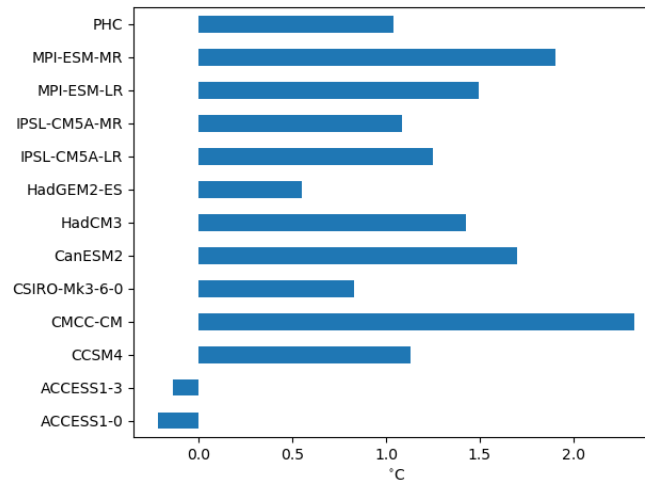


Fig. 6: Mean (1970-2005) Atlantic Water core temperature. PHC33 is an observed climatology.

TS-diagrams

T-S diagrams combine temperature and salinity, which allows the analysis of water masses and their potential for mixing. The lines of constant density for specific ranges of temperature and salinity are shown on the background of the T-S diagram. The dots on the diagram are individual grid points from specified region at all model levels within user specified depth range.

Related settings in the recipe:

```
tsdiag_regions: ["AB" , 'EB']
# Maximum depth to consider data for TS diagrams
tsdiag_depth: 1500
# Number of columns
tsdiag_ncol: 3
```

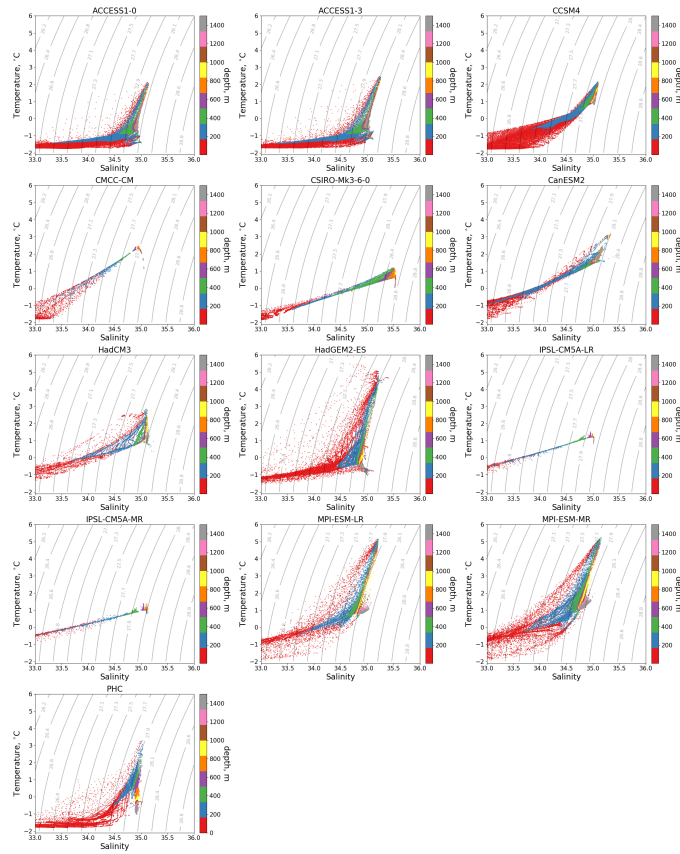


Fig. 7: Mean (1970-2005) T-S diagrams for Eurasian Basin of the Arctic Ocean.

19.1.3 Available diagnostics

The following python modules are included in the diagnostics package:

- `arctic_ocean.py` : Reads settings from the recipe and call functions to do analysis and plots.
- `getdata.py` : Deals with data preparation.
- `interpolation.py` : Include horizontal and vertical interpolation functions specific for ocean models.
- `plotting.py` : Ocean specific plotting functions
- `regions.py` : Contains code to select specific regions, and definition of the regions themselves.
- `utils.py` : Helpful utilites.

Diagnostics are stored in `diag_scripts/arctic_ocean/`

Variables

- `thetao` (ocean, monthly, longitude, latitude, time)
- `so` (ocean, monthly, longitude, latitude, time)

Observations and reformat scripts

- PHC3 climatology

References

- Ilicak, 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

19.2 Climate Variability Diagnostics Package (CVDP)

19.2.1 Overview

The Climate Variability Diagnostics Package (CVDP) developed by NCAR's Climate Analysis Section is an analysis tool that documents the major modes of climate variability in models and observations, including ENSO, Pacific Decadal Oscillation, Atlantic Multi-decadal Oscillation, Northern and Southern Annular Modes, North Atlantic Oscillation, Pacific North and South American teleconnection patterns. For details please refer to the [1] and [2].

19.2.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_cvdp.yml

Diagnostics are stored in diag_scripts/cvdp/

- cvdp_wrapper.py

19.2.3 User settings in recipe

The recipe can be run with several data sets including different model ensembles, multi-model mean statistics are currently not supported.

19.2.4 Variables

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

19.2.5 Observations and reformat scripts

None.

19.2.6 References

[1] http://www.cesm.ucar.edu/working_groups/CVC/cvdp/

[2] <https://github.com/NCAR/CVDP-ncl>

19.2.7 Example plots

19.3 Nino indices, North Atlantic Oscillation (NAO), Souther Oscillation Index (SOI)

19.3.1 Overview

The goal of this diagnostic is to compute indices based on area averages.

In recipe_combined_indices.yml, after defining the period (historical or future projection), the variable is selected. The predefined areas are:

- Nino 3
- Nino 3.4
- Nino 4
- North Atlantic Oscillation (NAO)

NAM PR Regressions (Annual)

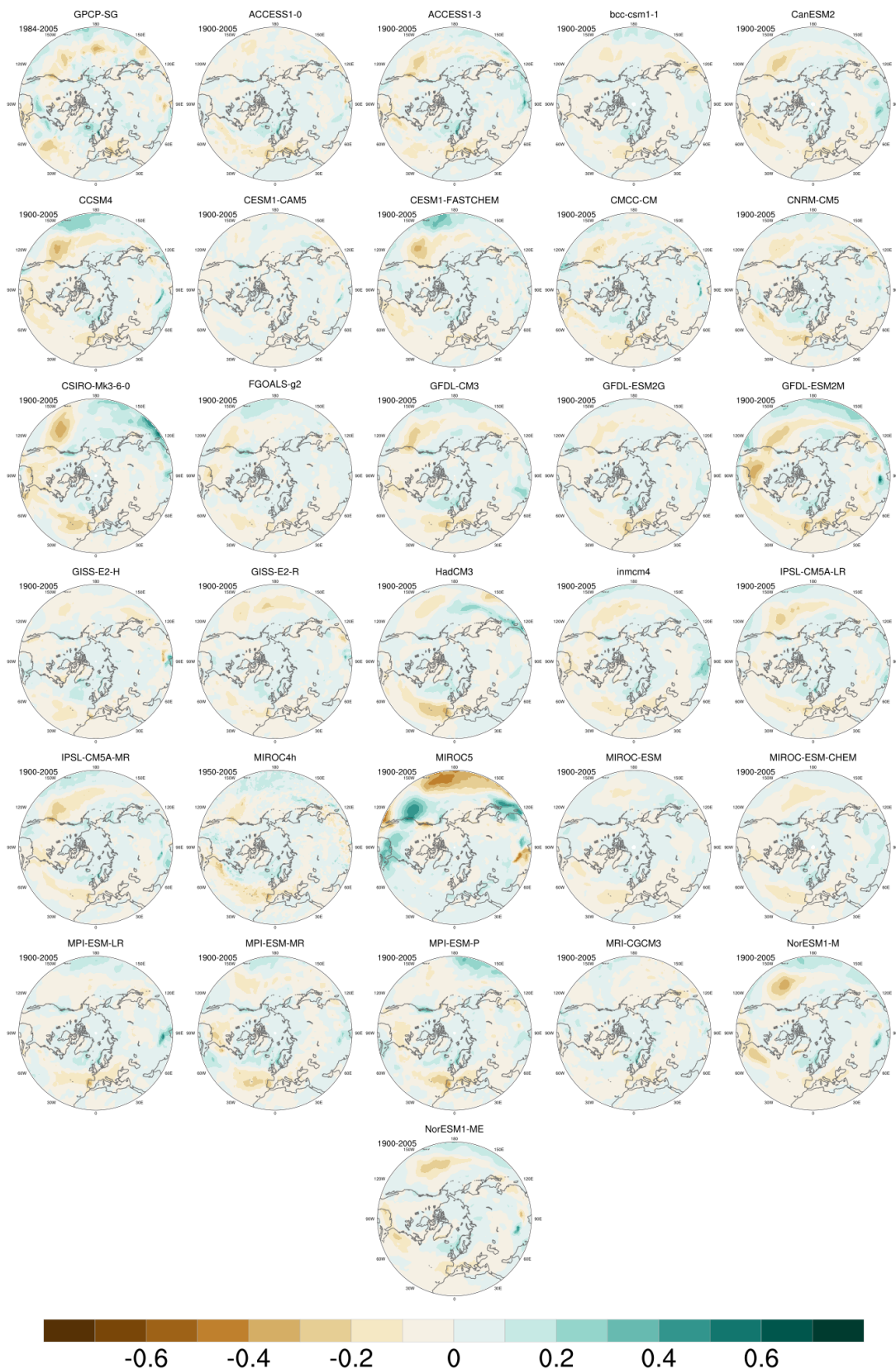


Fig. 8: Regression of the precipitation anomalies (PR) onto the Northern Annular Mode (NAM) index for the time period 1900-2005 for 30 CMIP5 models and observations (GPCP (pr) / IFS-Cy31r2 (psl); time period 1984-2005).

- Southern Oscillation Index (SOI)

19.3.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_combined_indices.yml

Diagnostics are stored in diag_scripts/magic_bsc/

- combined_indices.R : calculates the area-weighted means and multi-model means, with or without weights

19.3.3 User settings

User setting files are stored in recipes/

1. recipe_combined_indices.yml

Required settings for script

- region: one of the following strings Nino3, Nino3.4, Nino4, NAO, SOI
- running_mean: an integer specifying the length of the window (in months) to be used for computing the running mean.
- moninf: an integer can be given to determine the first month of the seasonal mean to be computed (from 1 to 12, corresponding to January to December respectively).
- monsup: an integer specifying the last month to be computed (from 1 to 12, corresponding to January to December respectively).
- standardized: 'true' or 'false' to specify whether to compute the standarization of the variable.

Required settings for preprocessor (only for 3D variables)

extract_levels:

- levels: [50000] # e.g. for 500 hPa level
- scheme: nearest

19.3.4 Variables

- all variables (atmos/ocean, monthly, longitude, latitude, time)

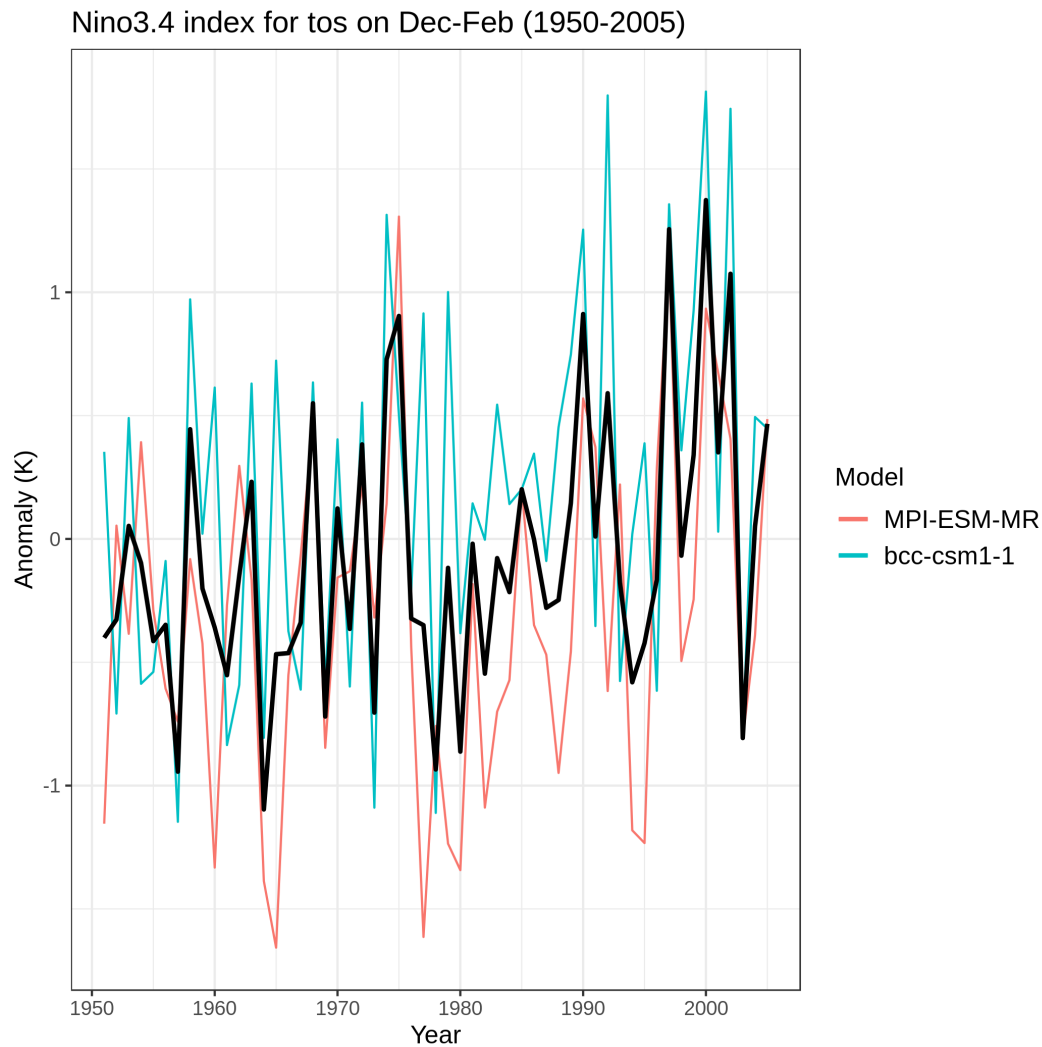
19.3.5 Observations and reformat scripts

None

19.3.6 References

- Trenberth, Kevin & National Center for Atmospheric Research Staff (Eds). Last modified 11 Jan 2019. "The Climate Data Guide: Nino SST Indices (Nino 1+2, 3, 3.4, 4; ONI and TNI)." Retrieved from <https://climatedataguide.ucar.edu/climate-data/nino-sst-indices-nino-12-3-34-4-oni-and-tni>.

19.3.7 Example plots



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.

19.4 Ocean chlorophyll in ESMs compared to ESA-CCI observations.

19.4.1 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.

19.4.2 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

19.4.3 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,})

19.4.4 Variables

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

19.4.5 Observations and reformat scripts

- ESACCI-OC (chl)

Reformat script: reformat_scripts/obs/reformat_obs_esacci_oc.py

19.4.6 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>

19.4.7 Example plots

Mass Concentration of Total Phytoplankton Expressed as Chlorophyll in Sea Water [mg m⁻³]

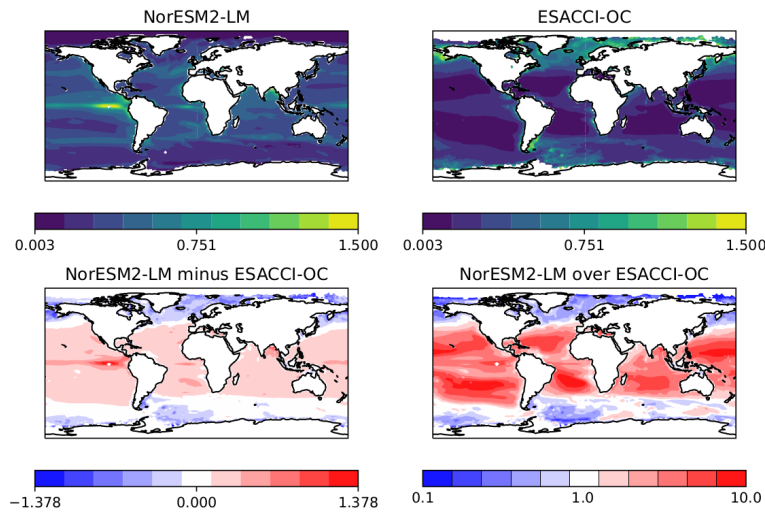


Fig. 9: Surface chlorophyll from ESACCI-OC ocean colour data version 5.0 and the CMIP6 model NorESM2-LM. This model overestimates chlorophyll compared to the observations.

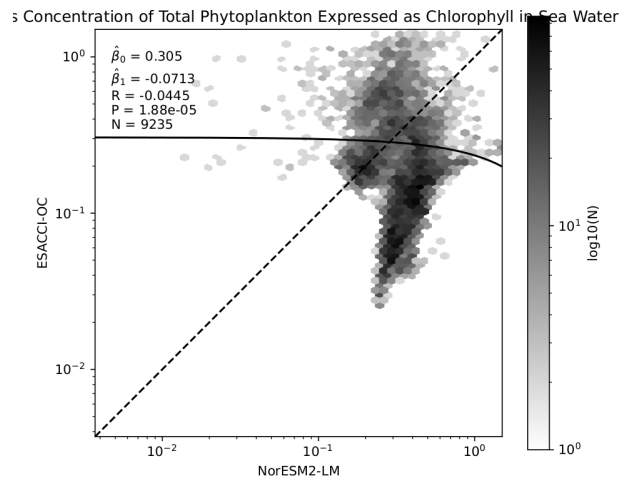


Fig. 10: Scatter plot of surface chlorophyll from ESACCI-OC ocean colour data version 5.0 and the CMIP6 model NorESM2-LM.

19.5 Ocean diagnostics

19.5.1 Overview

These recipes are used for evaluating the marine component of models of the earth system. Using these recipes, it should be possible to evaluate both the physical models and biogeochemistry models. All these recipes use the ocean diagnostics package.

The ocean diagnostics package contains several diagnostics which produce figures and statistical information of models of the ocean. The datasets have been pre-processed by ESMValTool, based on recipes in the recipes directory. Most of

the diagnostics produce two or less types of figure, and several diagnostics are called by multiple recipes.

Each diagnostic script expects a metadata file, automatically generated by ESMValTool, and one or more pre-processed dataset. These are passed to the diagnostic by ESMValTool in the settings.yml and metadata.yml files.

The ocean diagnostics toolkit can not figure out how to plot data by itself. The current version requires the recipe to produce the correct pre-processed data for each diagnostic script. ie: to produce a time series plot, the preprocessor must produce a time-dimensional dataset.

While these tools were built to evaluate the ocean component models, they also can be used to produce figures for other domains. However, there are some ocean specific elements, such as the z-direction being positive and reversed, and some of the map plots have the continents coloured in by default.

As elsewhere, both the model and observational datasets need to be compliant with the CMOR data.

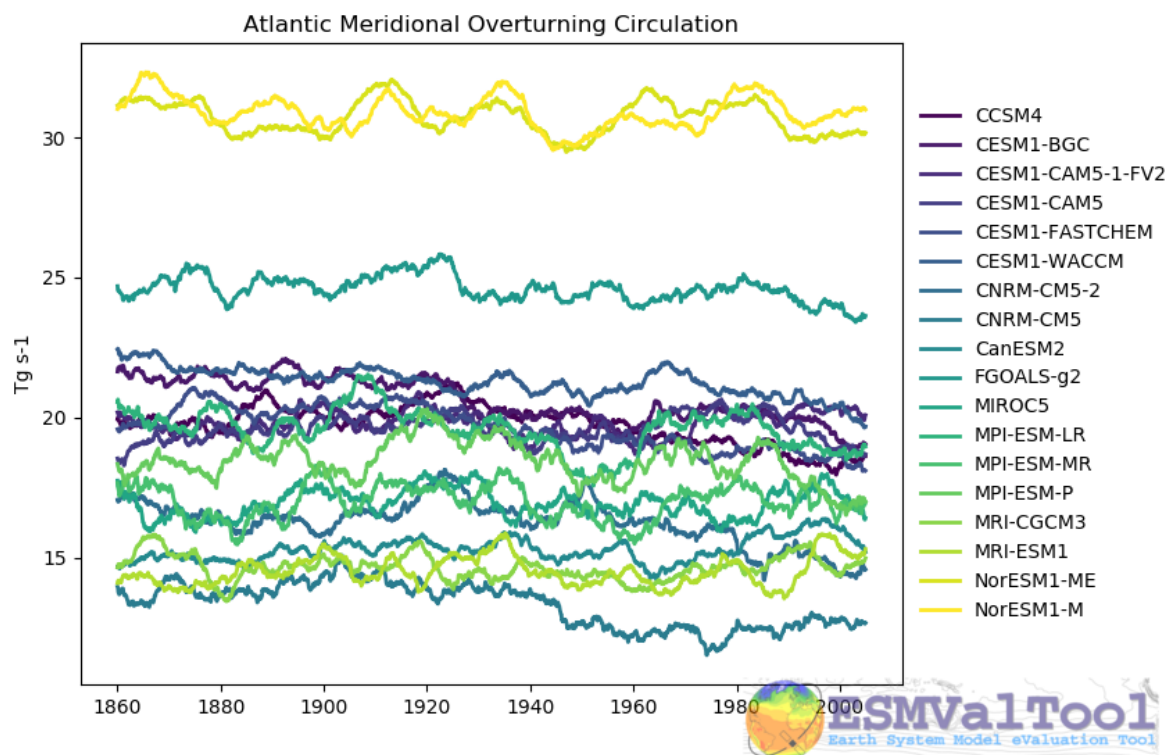
19.5.2 Available recipes

- `recipe_ocean_amoc.yml`
- `recipe_ocean_example.yml`
- `recipe_ocean_scalar_fields.yml`
- `recipe_ocean_bgc.yml`
- `recipe_ocean_quadmap.yml`
- *`recipe_ocean_ice_extent.yml`*
- `recipe_ocean_multimap.yml`

`recipe_ocean_amoc.yml`

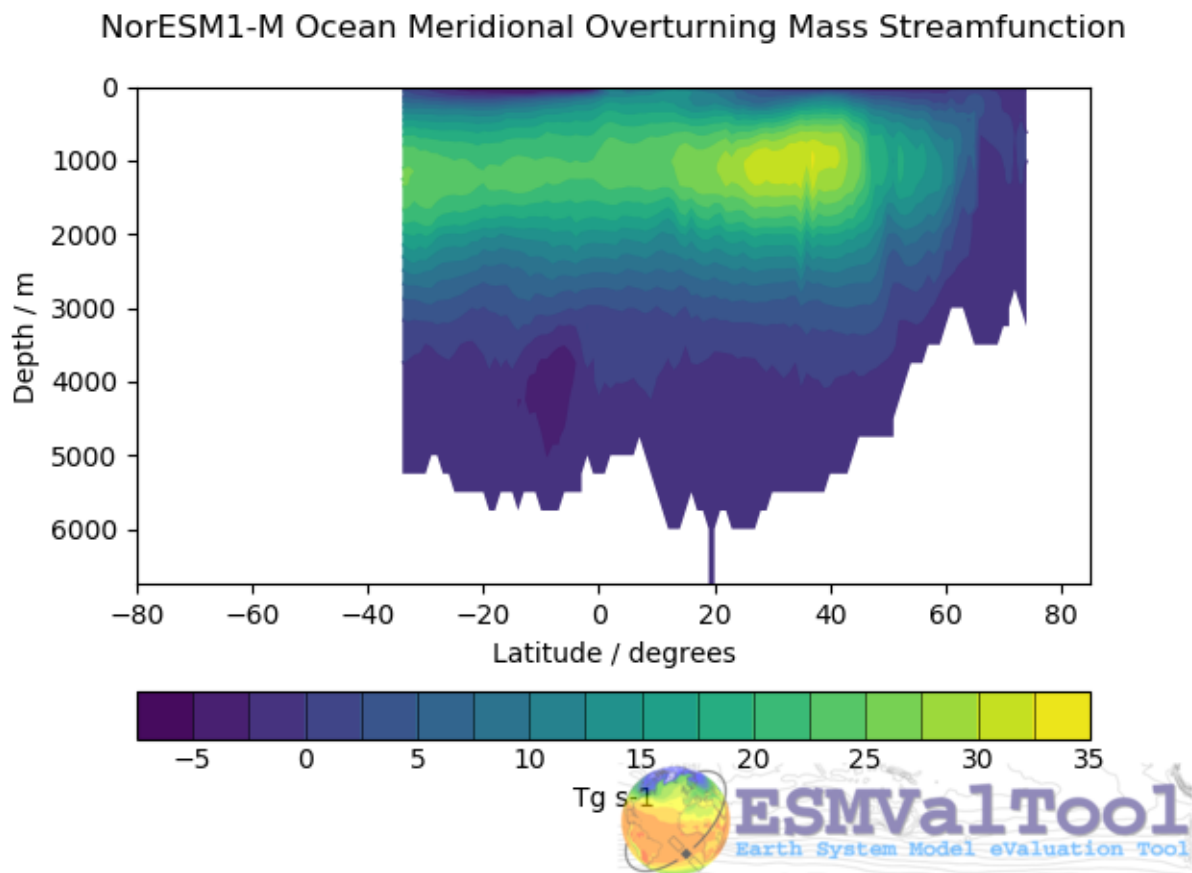
The `recipe_ocean_amoc.yml` is an recipe that produces figures describing the Atlantic Meridional Overturning Circulation (AMOC) and the drake passage current.

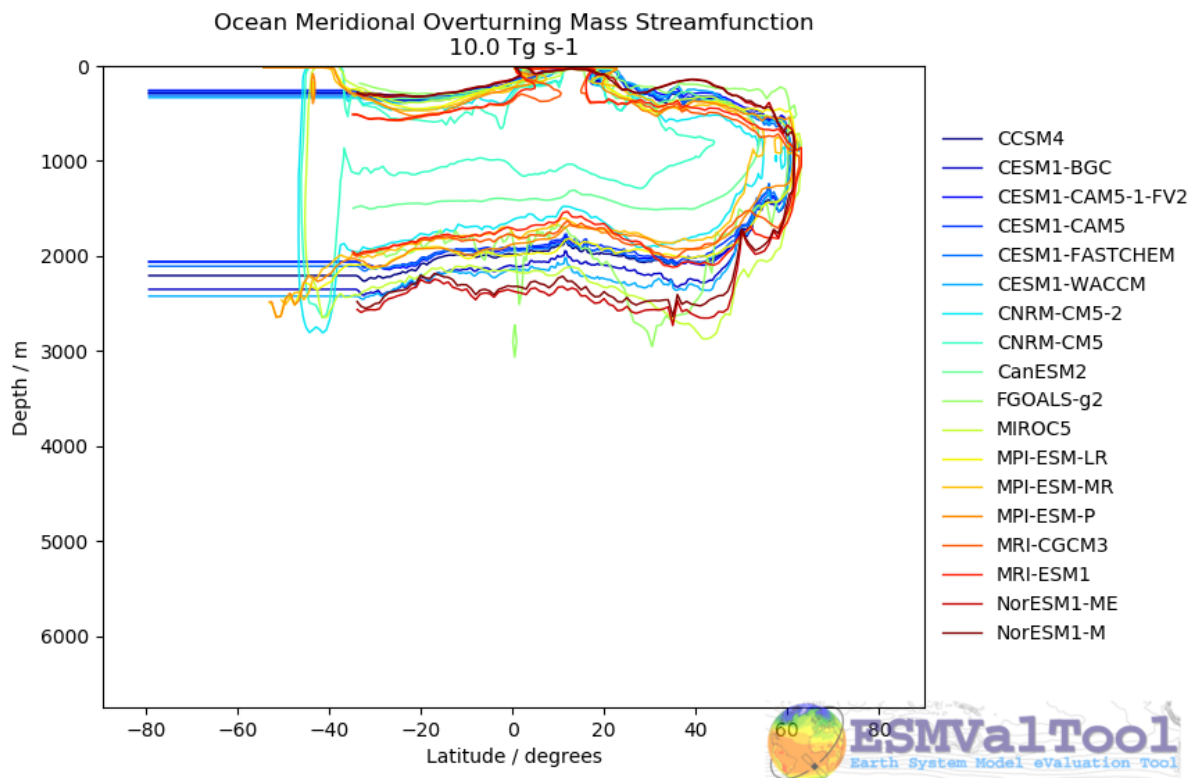
The recipes produces time series of the AMOC at 26 north and the drake passage current.



This figure shows the multi model comparison of the AMOC from several CMIP5 historical simulations, with a 6 year moving average (3 years either side of the central value). A similar figure is produced for each individual model, and for the Drake Passage current.

This recipe also produces a contour transect and a coloured transect plot showing the Atlantic stream function for each individual model, and a multi-model contour is also produced:





recipe_ocean_example.yml

The `recipe_ocean_example.yml` is an example recipe which shows several examples of how to manipulate marine model data using the ocean diagnostics tools.

While several of the diagnostics here have specific uses in evaluating models, it is meant to be a catch-all recipe demonstrating many different ways to evaluate models.

All example calculations are performed using the ocean temperature in a three dimensional field (thetao), or at the surface (tos). This recipe demonstrates the use of a range of preprocessors in a marine context, and also shows many of the standard model-only diagnostics (no observational component is included.)

This recipe includes examples of how to manipulate both 2D and 3D fields to produce:

- Time series:
 - Global surface area weighted mean time series
 - Volume weighted average time series within a specific depth range
 - Area weighted average time series at a specific depth
 - Area weighted average time series at a specific depth in a specific region.
 - Global volume weighted average time series
 - Regional volume weighted average time series
- Maps:
 - Global surface map (from 2D and 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.

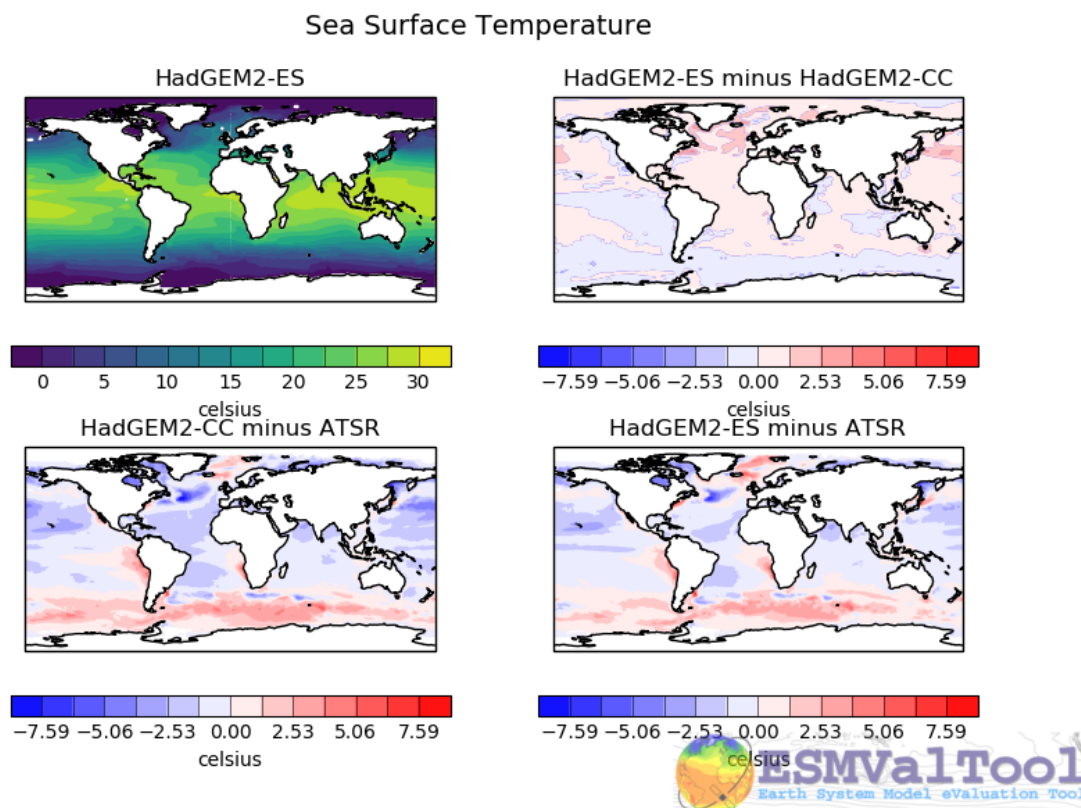
recipe_ocean_quadmap.yml

The `recipe_ocean_quadmap.yml` is an example recipe showing the `diagnostic_maps_quad.py` diagnostic. This diagnostic produces an image showing four maps. Each of these four maps show latitude vs longitude and the cube value is used as the colour scale. The four plots are:

model1	model 1 minus model2
model2 minus obs	model1 minus obs

These figures are also known as Model vs Model vs Obs plots.

The figure produced by this recipe compares two versions of the HadGEM2 model against ATSR sea surface temperature:



This kind of figure can be very useful when developing a model, as it allows model developers to quickly see the impact of recent changes to the model.

recipe_ocean_ice_extent.yml

The `recipe_ocean_ice_extent.yml` recipe produces several metrics describing the behaviour of sea ice in a model, or in multiple models.

This recipe has four preprocessors, a combinatorial combination of

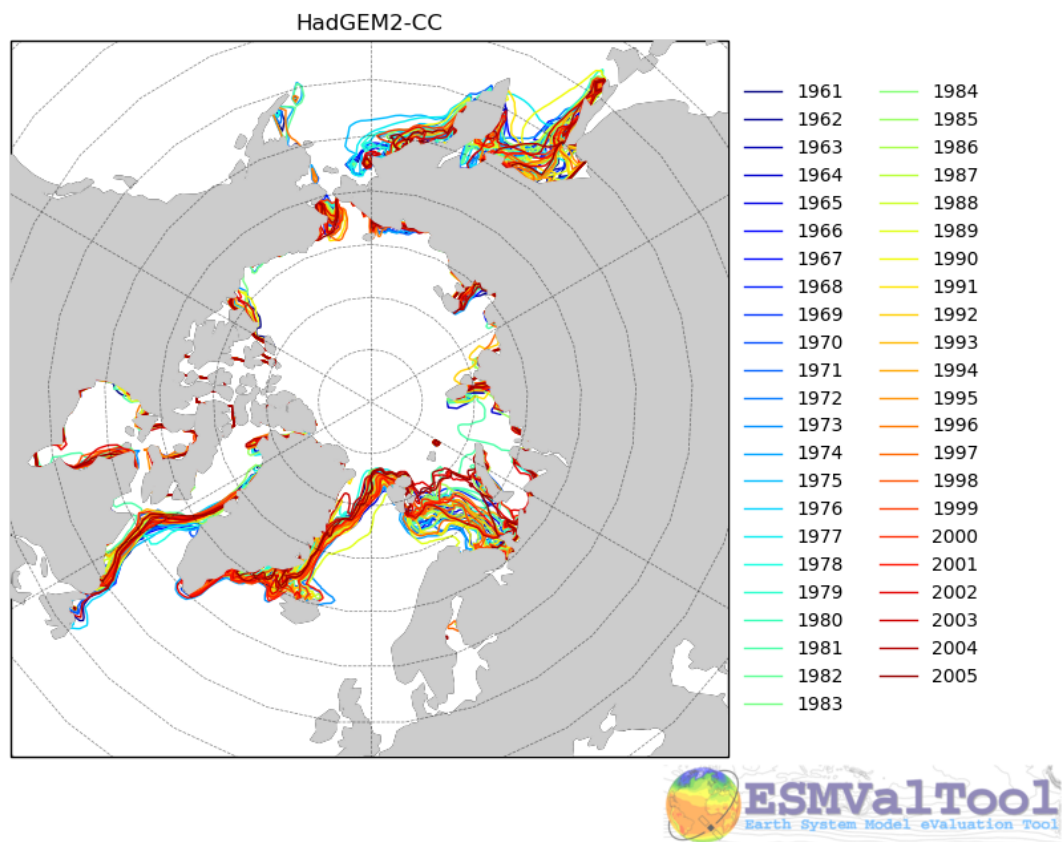
- Regions: Northern or Southern Hemisphere
- Seasons: December-January-February or June-July-August

Once these seasonal hemispherical fractional ice cover is processed, the resulting cube is passed 'as is' to the `diagnostic_seaice.py` diagnostic.

This diagnostic produces the plots:

- Polar Stereographic projection Extent plots of individual models years.
- Polar Stereographic projection maps of the ice cover and ice extent for individual models.
- A time series of Polar Stereographic projection Extent plots - see below.
- Time series plots of the total ice area and the total ice extent.

The following image shows an example of the sea ice extent plot, showing the Summer Northern hemisphere ice extent for the HadGEM2-CC model, in the historical scenario.



The sea ice diagnostic is unlike the other diagnostics in the ocean diagnostics toolkit. The other tools are build to be generic plotting tools which work with any field (ie `diagnostic_timeseries.py` works fine for Temperature, Chlorophyll, or any other field. On the other hand, the sea ice diagnostic is the only tool that performs a field specific evaluation.

The `diagnostic_seaice.py` diagnostic is more fully described below.

recipe_ocean_multimap.yml

The `recipe_ocean_multimap.yml` is an example recipe showing the `diagnostic_maps_multimodel.py` diagnostic. This diagnostic produces an image showing Model vs Observations maps or only Model fields when observational data are not provided. Each map shows latitude vs longitude fields and user defined values are used to set the colour scale. Plot layout can be modified by modifying the `layout_rowcol` argument.

The figure produced by this recipe compares the ocean surface CO2 fluxes for 16 different CMIP5 model against Landschuetzer2016 observations.

The `diagnostic_maps_multimodel.py` diagnostic is documented below.

19.5.3 Available diagnostics

Diagnostics are stored in the `diag_scripts` directory: `ocean`.

The following python modules are included in the ocean diagnostics package. Each module is described in more detail both below and inside the module.

- `diagnostic_maps.py`
- `diagnostic_maps_quad.py`
- `diagnostic_model_vs_obs.py`
- `diagnostic_profiles.py`
- `diagnostic_seaice.py`
- `diagnostic_timeseries.py`
- `diagnostic_tools.py`
- `diagnostic_transects.py`
- `diagnostic_maps_multimodel.py`

diagnostic_maps.py

The `diagnostic_maps.py` produces a spatial map from a NetCDF. It requires the input netCDF to have the following dimensions. Either:

- A two dimensional file: latitude, longitude.
- A three dimensional file: depth, latitude, longitude.

In the case of a 3D netCDF file, this diagnostic produces a map for EVERY layer. For this reason, we recommend extracting a small number of specific layers in the preprocessor, using the `extract_layer` preprocessor.

This script can not process NetCDFs with multiple time steps. Please use the `climate_statistics` preprocessor to collapse the time dimension.

This diagnostic also includes the optional arguments, `threshold` and `thresholds`.

- `threshold`: a single float.
- `thresholds`: a list of floats.

Only one of these arguments should be provided at a time. These two arguments produce a second kind of diagnostic map plot: a contour map showing the spatial distribution of the threshold value, for each dataset. Alternatively, if the `thresholds` argument is used instead of `threshold`, the single-dataset contour map shows the contours of all the values in the `thresholds` list.

If multiple datasets are provided, in addition to the single dataset contour, a multi-dataset contour map is also produced for each value in the thresholds list.

Some appropriate preprocessors for this diagnostic would be:

For a Global 2D field:

```
prep_map_1:
  climate_statistics:
```

For a regional 2D field:

```
prep_map_2:
  extract_region:
    start_longitude: -80.
    end_longitude: 30.
    start_latitude: -80.
    end_latitude: 80.
  climate_statistics:
    operator: mean
```

For a Global 3D field at the surface and 10m depth:

```
prep_map_3:
  custom_order: true
  extract_levels:
    levels: [0., 10.]
    scheme: linear_horizontal_extrapolate_vertical
  climate_statistics:
    operator: mean
```

For a multi-model comparison mean of 2D global fields including contour thresholds.

```
prep_map_4:
  custom_order: true
  climate_statistics:
    operator: mean
  regrid:
    target_grid: 1x1
    scheme: linear
```

And this also requires the threshold key in the diagnostic:

```
diagnostic_map:
  variables:
    tos: # Temperature ocean surface
    preprocessor: prep_map_4
    field: T02M
  scripts:
    Ocean_regrid_map:
      script: ocean/diagnostic_maps.py
      thresholds: [5, 10, 15, 20]
```

diagnostic_maps_quad.py

The `diagnostic_maps_quad.py` diagnostic produces an image showing four maps. Each of these four maps show latitude vs longitude and the cube value is used as the colour scale. The four plots are:

model1	model 1 minus model2
model2 minus obs	model1 minus obs

These figures are also known as Model vs Model vs Obs plots.

This diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cubes received by this diagnostic (via the `settings.yml` and `metadata.yml` files) have no time component, a small number of depth layers, and a latitude and longitude coordinates.

An appropriate preprocessor for a 2D field would be:

```
prep_quad_map:
climate_statistics:
  operator: mean
```

and an example of an appropriate diagnostic section of the recipe would be:

```
diag_map_1:
  variables:
    tos: # Temperature ocean surface
    preprocessor: prep_quad_map
    field: T02Ms
    mip: Omon
  additional_datasets:
#     filename: tos_ATSR_L3_ARC-v1.1.1_199701-201112.nc
#     download from: https://datashare.is.ed.ac.uk/handle/10283/536
    - {dataset: ATSR, project: obs4MIPs, level: L3, version: ARC-v1.1.1,
      ↪ start_year: 2001, end_year: 2003, tier: 3}
  scripts:
    Global_Ocean_map:
      script: ocean/diagnostic_maps_quad.py
      control_model: {dataset: HadGEM2-CC, project: CMIP5, mip: Omon, exp:
      ↪ historical, ensemble: r1i1p1}
      exper_model: {dataset: HadGEM2-ES, project: CMIP5, mip: Omon, exp:
      ↪ historical, ensemble: r1i1p1}
      observational_dataset: {dataset: ATSR, project: obs4MIPs,}
```

Note that the details about the control model, the experiment models and the observational dataset are all provided in the script section of the recipe.

diagnostic_model_vs_obs.py

The `diagnostic_model_vs_obs.py` diagnostic makes model vs observations maps and scatter plots. The map plots shows four latitude vs longitude maps:

Model	Observations
Model minus Observations	Model over Observations

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the `settings.yml` and `metadata.yml` files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

This diagnostic also includes the optional arguments, `maps_range` and `diff_range` to manually define plot ranges. Both arguments are a list of two floats to set plot range 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 been 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:
```

(continues on next page)

(continued from previous page)

```
extract_levels:
  levels: [100., ]
  scheme: linear_extrap
climate_statistics:
  operator: mean
  regrid:
    target_grid: 1x1
    scheme: linear
```

diagnostic_profiles.py

The `diagnostic_profiles.py` diagnostic produces images of the profile over time from a cube. These plots show cube value (ie temperature) on the x-axis, and depth/height on the y axis. The colour scale is the annual mean of the cube data. Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has a time component, and depth component, but no latitude or longitude coordinates.

An appropriate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_profile:
    extract_volume:
      long1: 0.
      long2: 20.
      lat1: -30.
      lat2: 30.
      z_min: 0.
      z_max: 3000.
    area_statistics:
      operator: mean
```

diagnostic_timeseries.py

The `diagnostic_timeseries.py` diagnostic produces images of the time development of a metric from a cube. These plots show time on the x-axis and cube value (ie temperature) on the y-axis.

Two types of plots are produced: individual model timeseries plots and multi model time series plots. The individual plots show the results from a single cube, even if this cube is a multi-model mean made by the *multimodel* preprocessor.

The multi model time series plots show several models on the same axes, where each model is represented by a different line colour. The line colours are determined by the number of models, their alphabetical order and the *jet* colour scale. Observational datasets and multimodel means are shown as black lines.

This diagnostic assumes that the preprocessors do the bulk of the work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) is time-dimensional cube. This means that the pre-processed netcdf has a time component, no depth component, and no latitude or longitude coordinates.

Some appropriate preprocessors would be :

For a global area-weighted average 2D field:

```
area_statistics:
  operator: mean
```

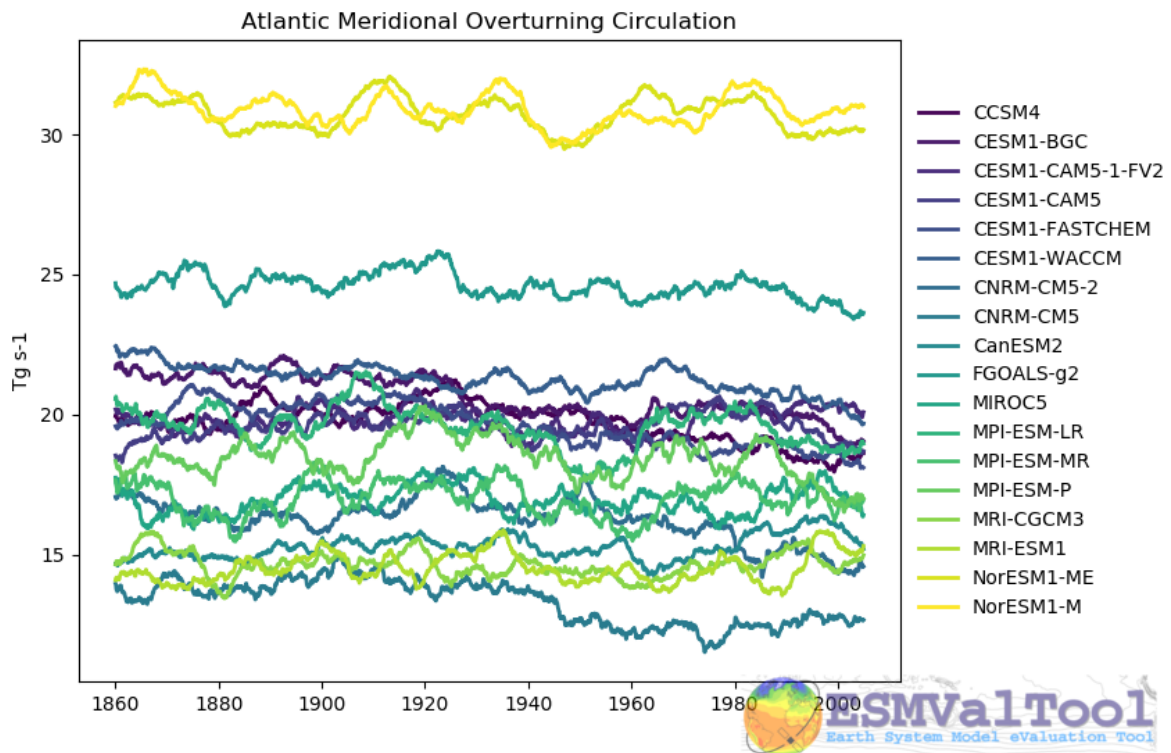
For a global volume-weighted average 3D field:

```
volume_statistics:
  operator: mean
```

For a global area-weighted surface of a 3D field:

```
extract_levels:
  levels: [0., ]
  scheme: linear_horizontal_extrapolate_vertical
area_statistics:
  operator: mean
```

An example of the multi-model time series plots can seen here:



diagnostic_transects.py

The `diagnostic_transects.py` diagnostic produces images of a transect, typically along a constant latitude or longitude.

These plots show 2D plots with either latitude or longitude along the x-axis, depth along the y-axis and the cube value is used as the colour scale.

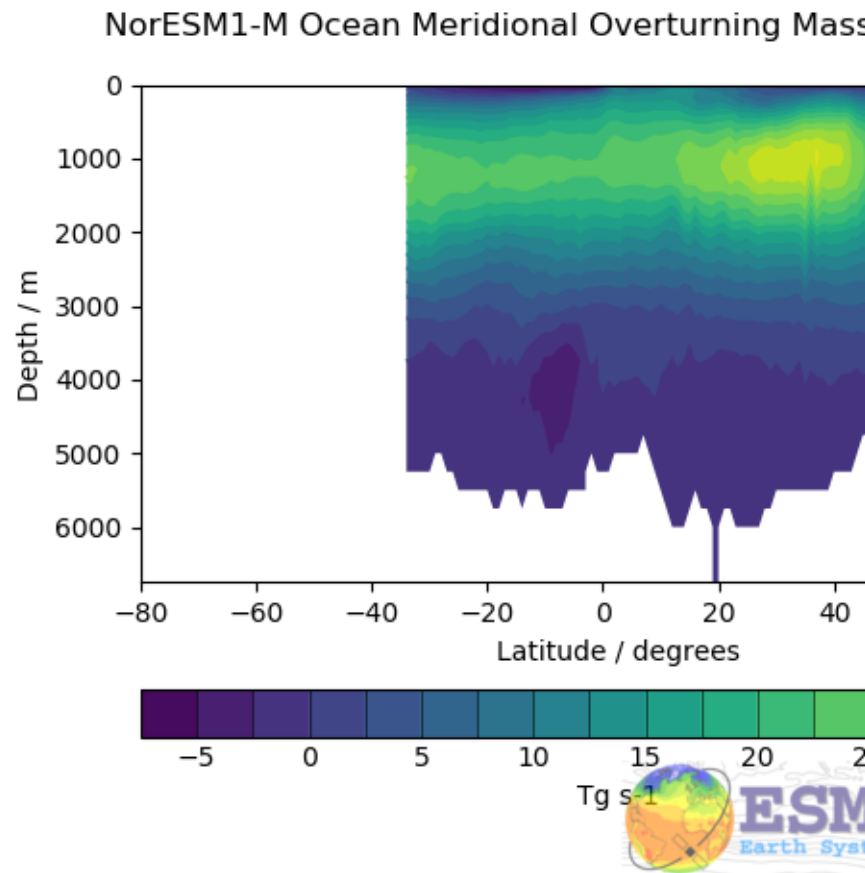
This diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the `settings.yml` and `metadata.yml` files) has no time component, and one of the latitude or longitude coordinates has been reduced to a single value.

An appropriate preprocessor for a 3D+time field would be:

```

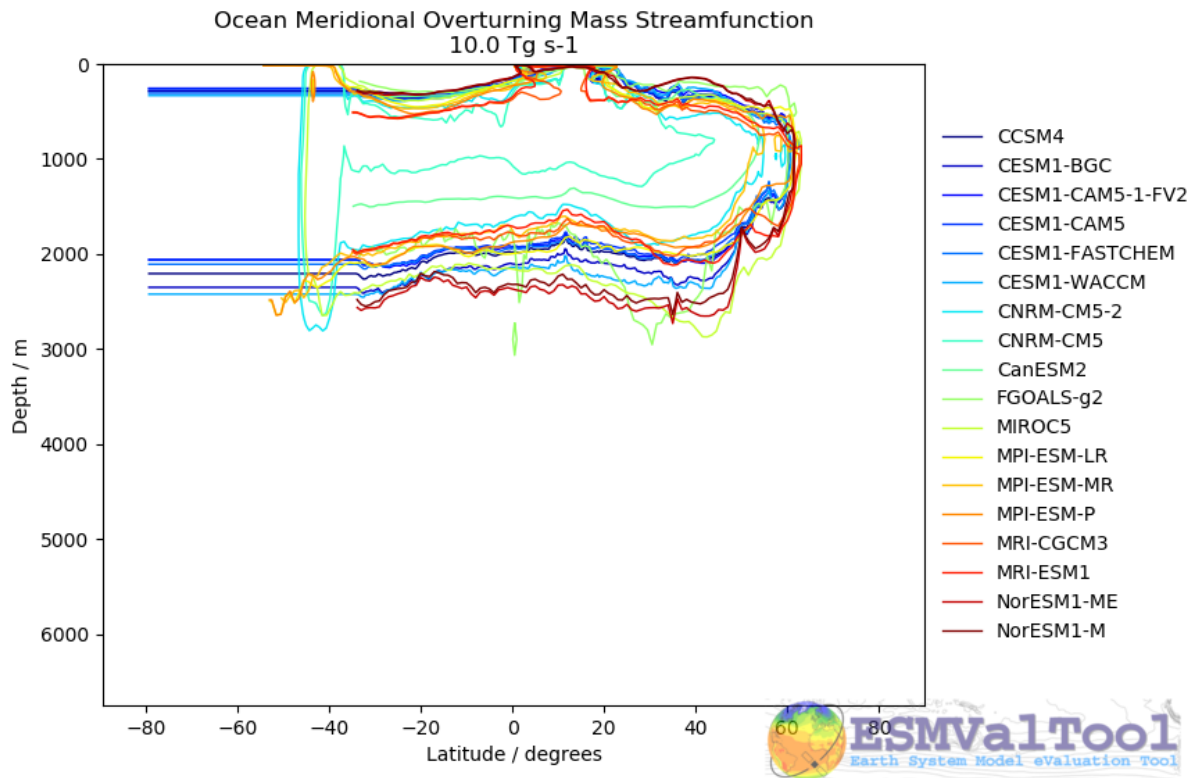
climate_statistics:
  operator: mean
extract_slice:
  latitude: [-50., 50.]
  longitude: 332.

```



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
```

(continues on next page)

(continued from previous page)

```
extract_region:
  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 extent.

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 user config.yml.
- *get_image_path*: creates a path for an image output.
- *make_cube_layer_dict*: makes a dictionary for several layers of a cube.

We just show a simple description here, each individual function is more fully documented in the `diagnostic_tools.py` module.

19.5.4 A note on the auxiliary data directory

Some of these diagnostic scripts may not function on machines with no access to the internet, as cartopy may try to download the shape files. The solution to this issue is to put the relevant cartopy shapefiles in a directory which is visible to esmvaltool, then link that path to ESMValTool via the `auxiliary_data_dir` variable in your `config-user.yml` file.

The cartopy masking files can be downloaded from: <https://www.naturalearthdata.com/downloads/>

In these recipes, cartopy uses the 1:10, physical coastlines and land files:

```
110m_coastline.dbf
110m_coastline.shp
110m_coastline.shx
110m_land.dbf
110m_land.shp
110m_land.shx
```

19.5.5 Associated Observational datasets

The following observations datasets are used by these recipes:

World Ocean ATLAS

These data can be downloaded from: <https://www.nodc.noaa.gov/OC5/woa13/woa13data.html> (last access 10/25/2018) Select the “All fields data links (1° grid)” netCDF file, which contain all fields.

The following WOA datasets are used by the ocean diagnostics:

- Temperature
- Salinity
- Nitrate
- Phosphate
- Silicate
- Dissolved Oxygen

These files need to be reformatted using the `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.

19.6 Sea Surface Salinity Evaluation

19.6.1 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.

19.6.2 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 chosen also, even though plots may look odd if too few or too many are selected.

19.6.3 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

19.6.4 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.

19.6.5 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

19.6.6 Variables

- sos (ocean, monthly, time latitude longitude)

19.6.7 Observations and reformat scripts

- ESACCI-SEA-SURFACE-SALINITY (sos)

19.6.8 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>

19.6.9 Example plots

19.7 Ocean metrics

19.7.1 Overview

The Southern Ocean is central to the global climate and the global carbon cycle, and to the climate's response to increasing levels of atmospheric greenhouse gases. Global coupled climate models and earth system models, however, vary widely in their simulations of the Southern Ocean and its role in, and response to, the ongoing anthropogenic trend. Observationally-based metrics are critical for discerning processes and mechanisms, and for validating and comparing climate and earth system models. New observations and understanding have allowed for progress in the creation of observationally-based data/model metrics for the Southern Ocean.

The metrics presented in this recipe provide a means to assess multiple simulations relative to the best available observations and observational products. Climate models that perform better according to these metrics also better simulate the uptake of heat and carbon by the Southern Ocean. Russell et al. 2018 assessed only a few of the available CMIP5 simulations, but most of the available CMIP5 and CMIP6 climate models can be analyzed with these recipes.

The goal is to create a recipe for recreation of metrics in Russell, J.L., et al., 2018, J. Geophys. Res. – Oceans, 123, 3120-3143, doi: 10.1002/2017JC013461.

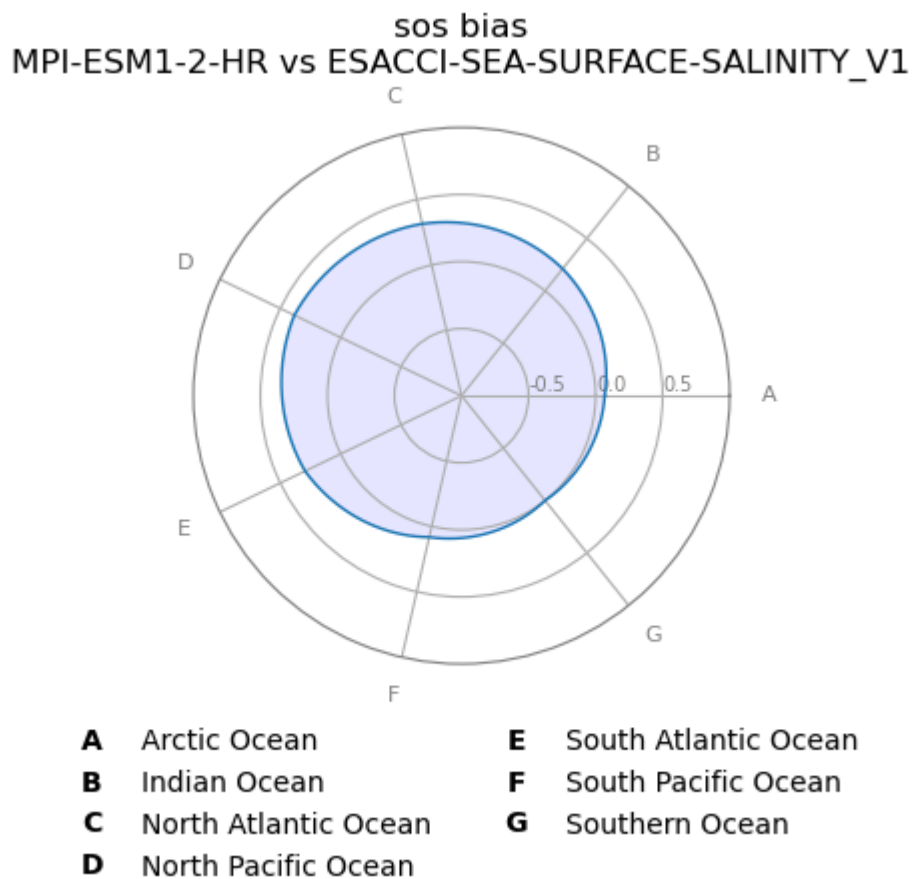


Fig. 11: Radar plot showing the mean state biases (simulation minus observations) for the regional averages of sea surface salinity in the selected ocean basins and seas.

sos std_ratio
MPI-ESM1-2-HR vs ESACCI-SEA-SURFACE-SALINITY_V1

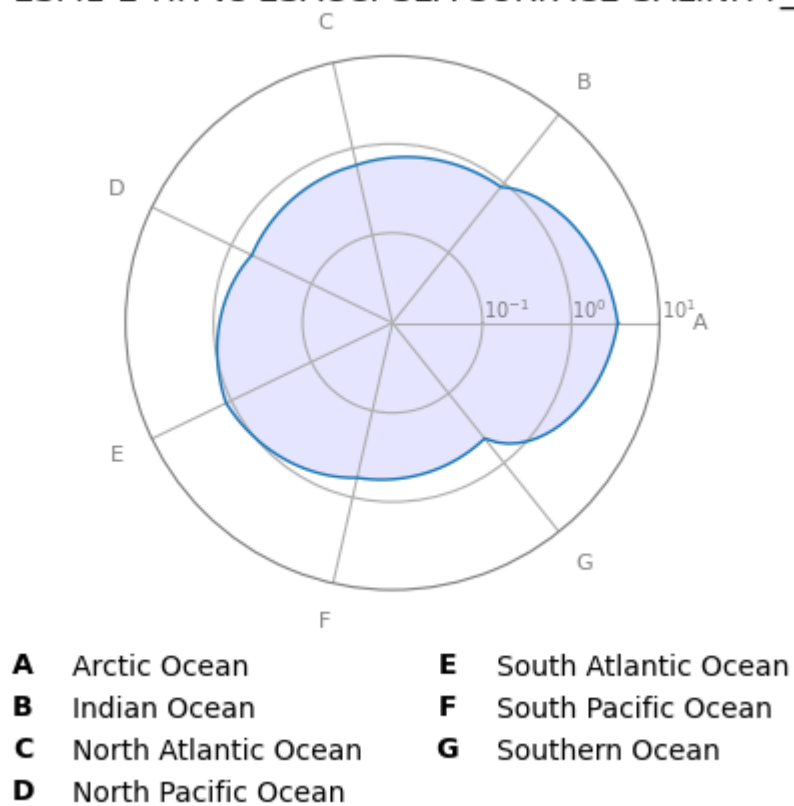


Fig. 12: Radar plot showing the ratio between the simulated and observed standard deviations of the regional averages of sea surface salinity in the selected ocean basins and seas.

19.7.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_russell18jgr.yml

Diagnostics are stored in diag_scripts/russell18jgr/

- russell18jgr-polar.ncl (figures 1, 7, 8): calculates and plots annual-mean variables (tauu, sic, fgco2, pH) as polar contour map.
- russell18jgr-fig2.ncl: calculates and plots The zonal and annual means of the zonal wind stress (N/m²).
- 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 * m²).
- 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.

19.7.3 User settings in recipe

1. Script russell18jgr-polar.ncl

Required settings (scripts)

- styleset : CMIP5(recommended), default, etc.
- ncd : 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` : `BIWhRe` (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

19.7.4 Variables

- tauu (atmos, monthly mean, longitude latitude time)
- tauuo, hfds, fgco2 (ocean, monthly mean, longitude latitude time)
- thetao, so, vo (ocean, monthly mean, longitude latitude lev time)
- pH (ocnBgchem, monthly mean, longitude latitude time)
- uo (ocean, monthly mean, longitude latitude lev time)
- sic (seaIce, monthly mean, longitude latitude time)

19.7.5 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`)

19.7.6 References

- Russell, J.L., et al., 2018, J. Geophys. Res. – Oceans, 123, 3120-3143. <https://doi.org/10.1002/2017JC013461>
- Talley, L.D., 2003. Shallow, intermediate and deep overturning components of the global heat budget. Journal of Physical Oceanography 33, 530–560

19.7.7 Example plots

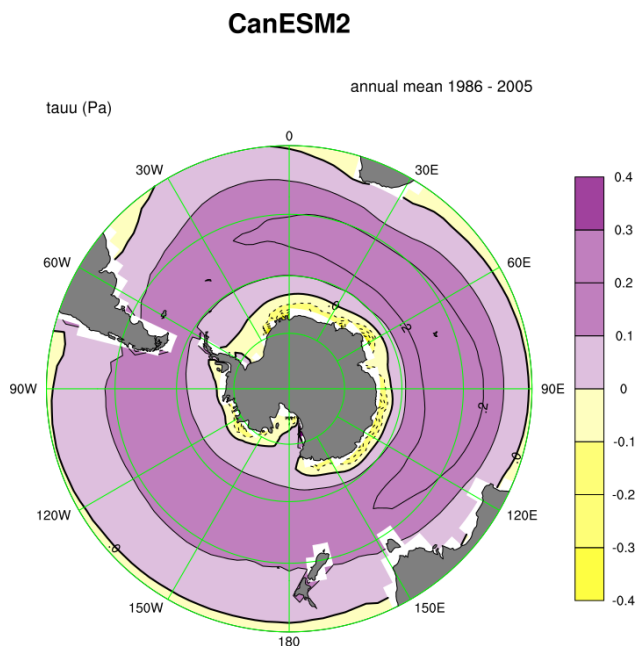


Fig. 13: Figure 1: Annual-mean zonal wind stress (τ_{uu} - N/m^2) with eastward wind stress as positive plotted as a polar contour map.

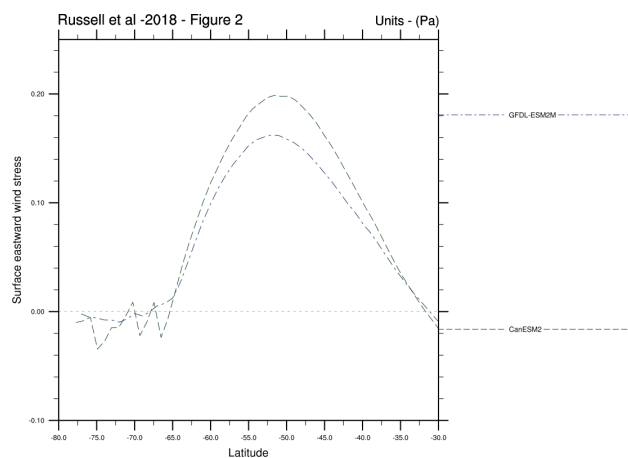


Fig. 14: Figure 2: The zonal and annual means of the zonal wind stress (N/m^2) plotted in a line plot.

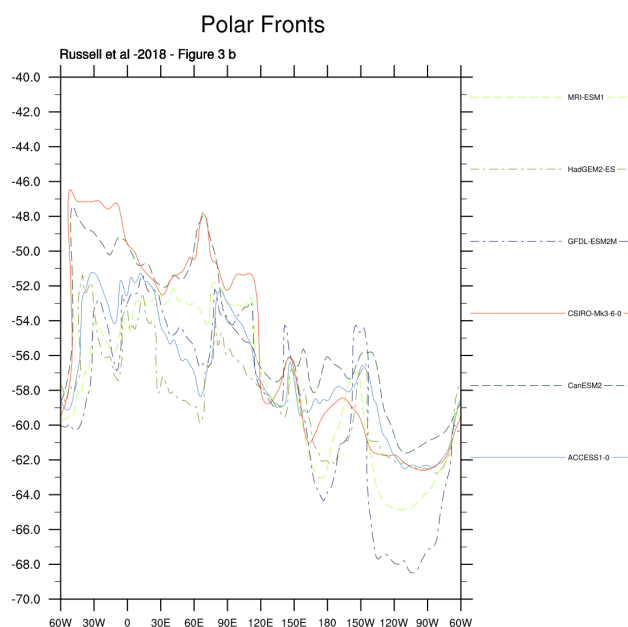


Fig. 15: Figure 3a: The latitudinal position of Subantarctic Front using definitions from Orsi et al (1995).

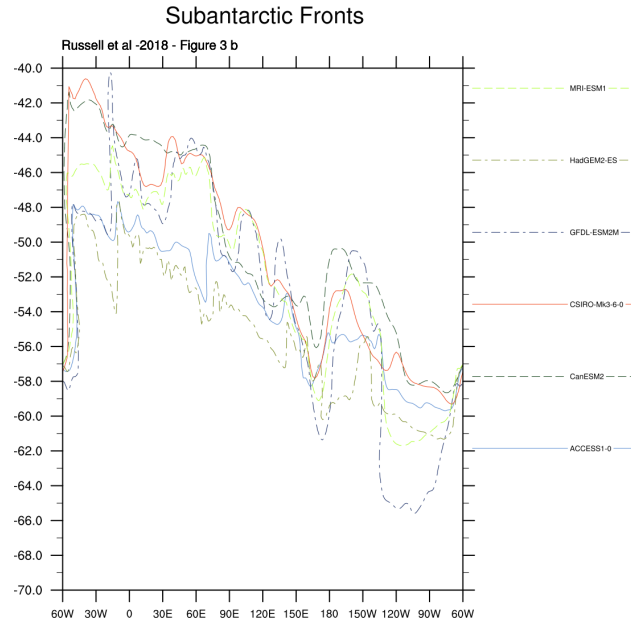


Fig. 16: Figure 3b: The latitudinal position of Polar Front using definitions from Orsi et al (1995).

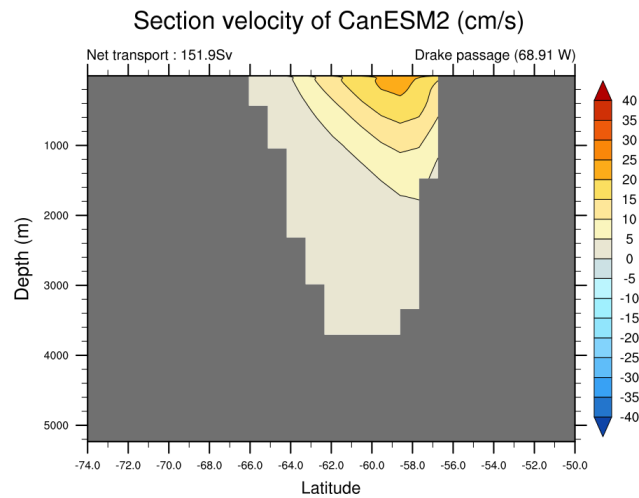


Fig. 17: Figure 4: Time averaged zonal velocity through Drake Passage (at 69W, in cm/s, eastward is positive). The total transport by the ACC is calculated if volcello file is available.

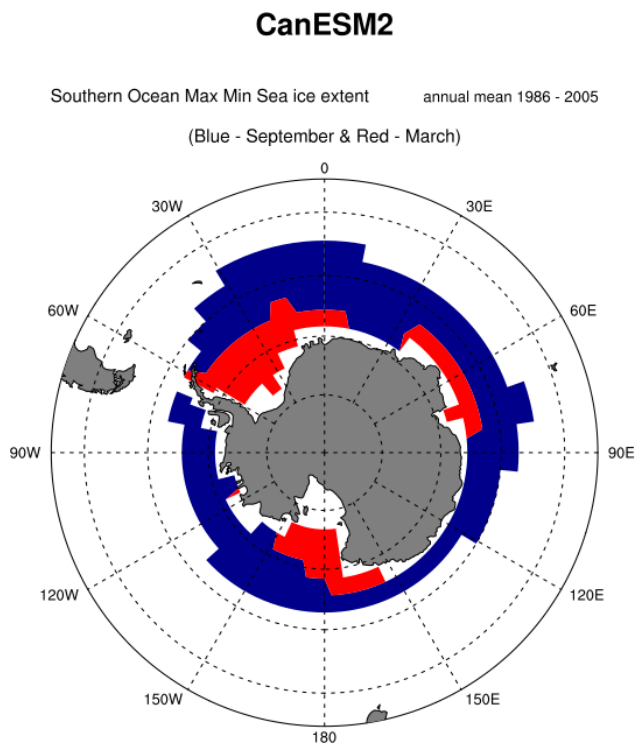


Fig. 18: Figure 5: Mean extent of sea ice for September(max) in blue and February(min) in red plotted as polar contour map.

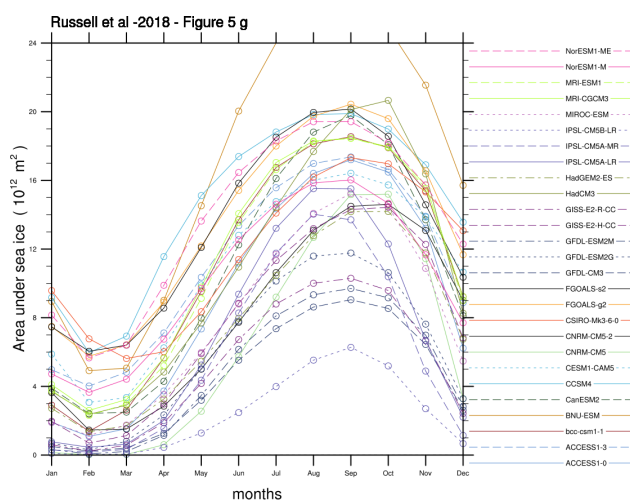


Fig. 19: Figure 5g: Annual cycle of sea ice area in southern ocean as a line plot (monthly climatology).

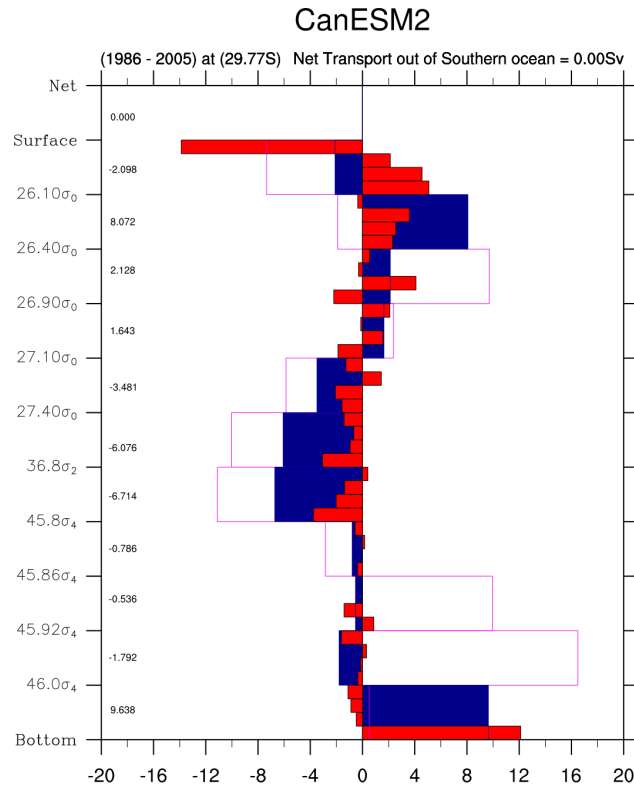


Fig. 20: Figure 6a: Density layer based volume transport (in Sv) across 30S based on the layer definitions in Talley (2008).

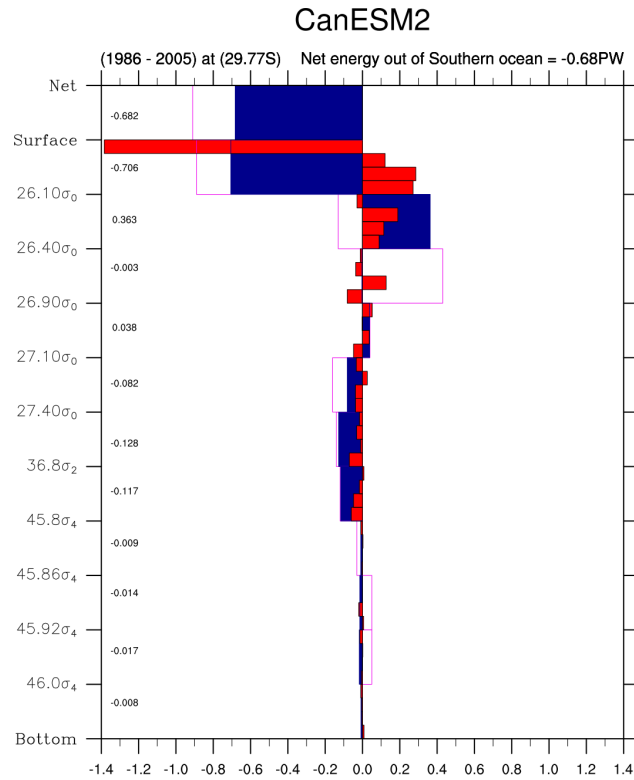


Fig. 21: Figure 6b: Density layer based heat transport(in PW) across 30S based on the layer definitions in Talley (2008).

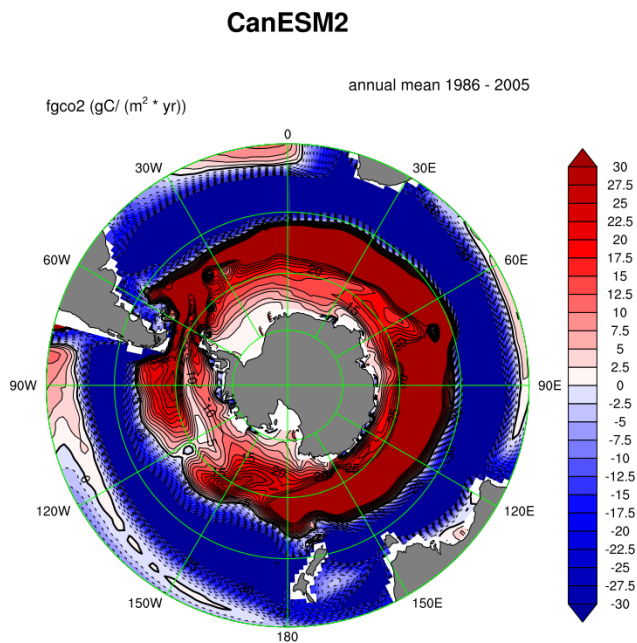


Fig. 22: Figure 7: Annual mean CO₂ flux (sea to air, gC/(yr * m²), positive (red) is out of the ocean) as a polar contour map.

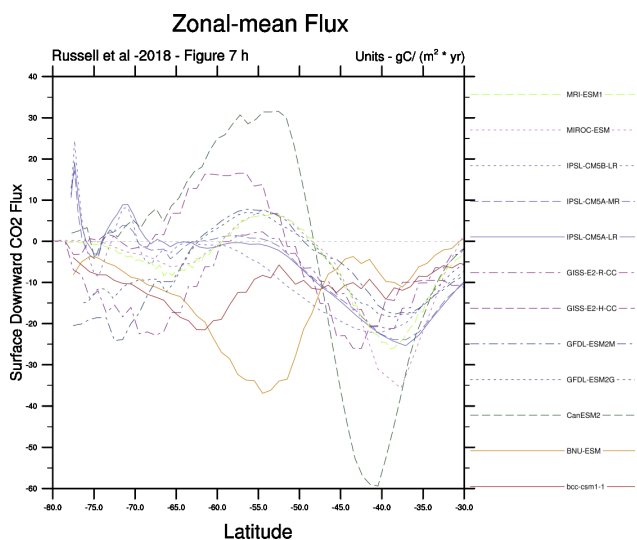


Fig. 23: Figure 7h: the time and zonal mean flux of CO₂ in gC/(yr * m²) plotted as a line plot.

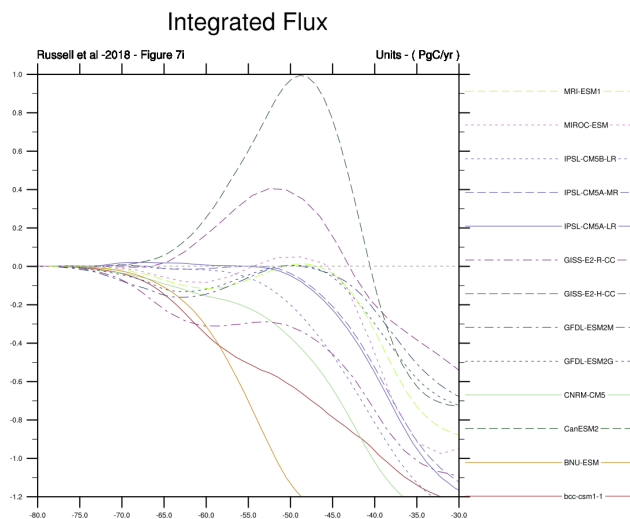


Fig. 24: Figure 7i is the cumulative integral of the net CO₂ flux from 90S to 30S (in PgC/yr) plotted as a line plot.

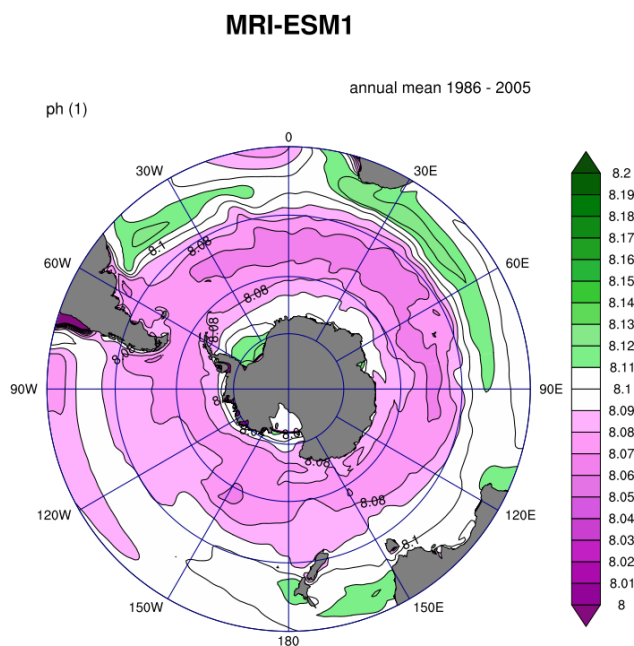


Fig. 25: Figure 8: Annual-mean surface pH plotted as a polar contour map.

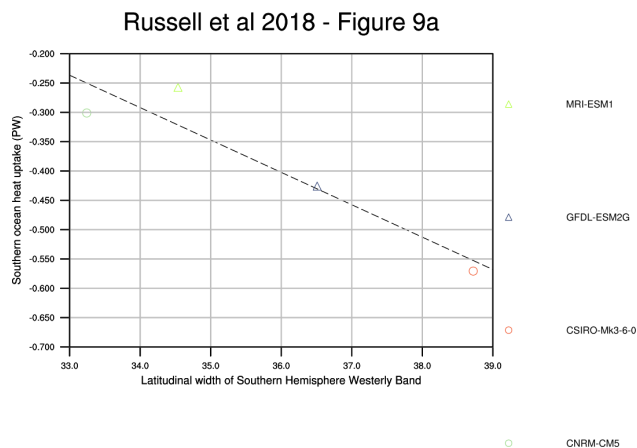


Fig. 26: Figure 9a: Scatter plot of the width of the Southern Hemisphere westerly wind band (in degrees of latitude) against the annual-mean integrated heat uptake south of 30S (in PW—negative uptake is heat lost from the ocean) along with the best fit line.

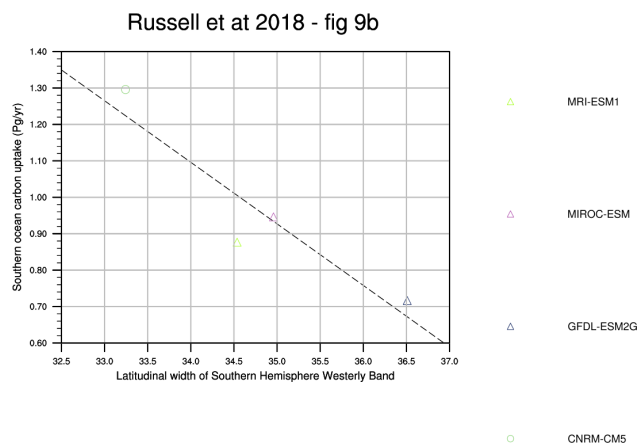


Fig. 27: Figure 9b: Scatter plot of the width of the Southern Hemisphere westerly wind band (in degrees of latitude) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the best fit line.

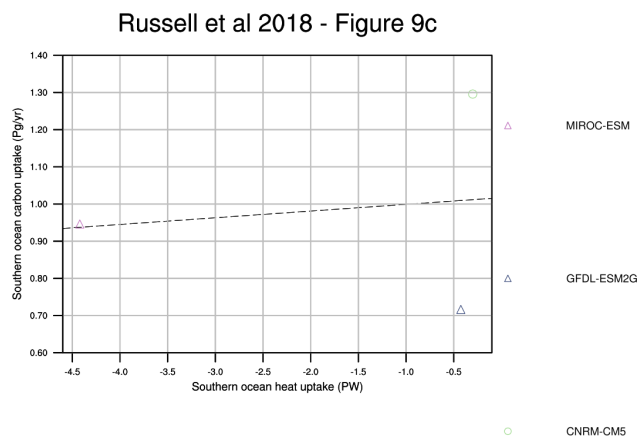


Fig. 28: Figure 9c: Scatter plot of the net heat uptake south of 30S (in PW) against the annual-mean integrated carbon uptake south of 30S (in Pg C/yr), along with the best fit line.

20.1 Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power

20.1.1 Overview

The goal of this diagnostic is to compute the wind capacity factor, taking as input the daily instantaneous surface wind speed, which is then extrapolated to obtain the wind speed at a height of 100 m as described in Lledó (2017).

The capacity factor is a normalized indicator of the suitability of wind speed conditions to produce electricity, irrespective of the size and number of installed turbines. This indicator is provided for three different classes of wind turbines (IEC, 2005) that are designed specifically for low, medium and high wind speed conditions.

The user can select the region, temporal range and season of interest.

The output of the recipe is a netcdf file containing the capacity factor for each of the three turbine classes.

20.1.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_capacity_factor.yml`

Diagnostics are stored in `diag_scripts/magic_bsc/`

- `capacity_factor.R`: calculates the capacity factor for the three turbine classes.
- `PC.R`: calculates the power curves for the three turbine classes.

20.1.3 User settings

User setting files are stored in `recipes/`

1. `recipe_capacity_factor.yml`

Required settings for script

- `power_curves`: (should not be changed)

20.1.4 Variables

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

20.1.5 Observations and reformat scripts

Main features of the selected turbines:

Turbine name	Rotor diameter (m)	Rated power (MW)	Cut-in speed (m/s)	Rated speed (m/s)	Cut-out speed (m/s)
Enercon E70 2.3MW	70	2.3	2.0	16.0	25.0
Gamesa G80 2.0MW	80	2.0	4.0	17.0	25.0
Gamesa G87 2.0MW	87	2.0	4.0	16.0	25.0
Vestas V100 2.0MW	100	2.0	3.0	15.0	20.0
Vestas V110 2.0MW	110	2.0	3.0	11.5	20.0

20.1.6 References

- IEC. (2005). International Standard IEC 61400-1, third edition, International Electrotechnical Commission. https://webstore.iec.ch/preview/info_iec61400-1%7Bed3.0%7Den.pdf
- Lledó, L. (2017). Computing capacity factor. Technical note BSC-ESS-2017-001, Barcelona Supercomputing Center. Available online at https://earth.bsc.es/wiki/lib/exe/fetch.php?media=library:external:bsc-ess-2017-001-c4e_capacity_factor.pdf [last accessed 11 October 2018]

20.1.7 Example plots

Wind capacity factor for five turbines: Enercon E70 (top-left), Gamesa G80 (middle-top), Gamesa G87 (top-right), Vestas V100 (bottom-left) and Vestas V110 (middle-bottom) using the IPSL-CM5A-MR simulations for the r1pl11 ensemble for the rcp8.5 scenario during the period 2021-2050.

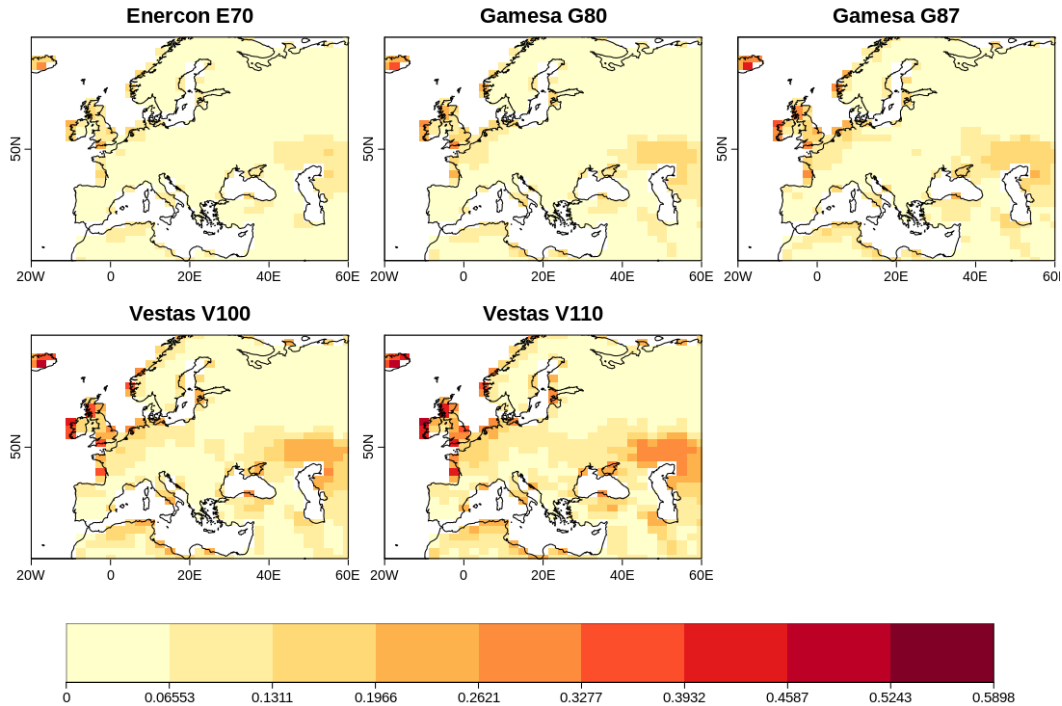
20.2 CMORizer recipes

20.2.1 Overview

These are CMORizer recipes calling CMORizer diagnostic scripts.

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.

CF from IPSL-CM5A-MR (2021-2050)



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](#).

20.2.2 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

20.2.3 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

20.2.4 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

20.2.5 References

- Hersbach, H., et al., Quarterly Journal of the Royal Meteorological Society, 730, 1999-2049, doi:10.1002/qj.3803, 2020.

20.3 Ensemble Clustering - a cluster analysis tool for climate model simulations (EnsClus)

20.3.1 Overview

EnsClus is a cluster analysis tool in Python, based on the k-means algorithm, for ensembles of climate model simulations.

Multi-model studies allow to investigate climate processes beyond the limitations of individual models by means of inter-comparison or averages of several members of an ensemble. With large ensembles, it is often an advantage to be able to group members according to similar characteristics and to select the most representative member for each cluster.

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).

20.3.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_ensclus.yml

Diagnostics are stored in diag_scripts/ensclus/

- ensclus.py

and subroutines

- ens_anom.py
- ens_eof_kmeans.py
- ens_plots.py
- eof_tool.py
- read_netcdf.py

- `sel_season_area.py`

20.3.3 User settings

Required settings for script

- `season`: season over which to perform seasonal averaging (DJF, DJFM, NDJFM, JJA)
- `area`: region of interest (EAT=Euro-Atlantic, PNA=Pacific North American, NH=Northern Hemisphere, EU=Europe)
- `extreme`: extreme to consider: XXth_percentile (XX can be set arbitrarily, e.g. 75th_percentile), mean (mean value over the period), maximum (maximum value over the period), std (standard deviation), trend (linear trend over the period)
- `numclus`: number of clusters to be computed
- `perc`: percentage of variance to be explained by PCs (select either this or `numpcs`, default=80)
- `numpcs`: number of PCs to retain (has priority over `perc` unless it is set to 0 (default))

Optional settings for script

- `max_plot_panels`: maximum number of panels (datasets) in a plot. When exceeded multiple plots are created. Default: 72

20.3.4 Variables

- chosen by user (e.g., precipitation as in the example)

20.3.5 Observations and reformat scripts

None.

20.3.6 References

- Straus, D. M., S. Corti, and F. Molteni: Circulation regimes: Chaotic variability vs. SST forced predictability. *J. Climate*, 20, 2251–2272, 2007. <https://doi.org/10.1175/JCLI4070.1>

20.3.7 Example plots

20.4 ESA CCI LST comparison to Historical Models

20.4.1 Overview

This diagnostic compares ESA CCI LST to multiple historical ensemble 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 difference 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 monthly data. A CMORizing script calculates the mean of the day time, and night time overpasses to give the all time

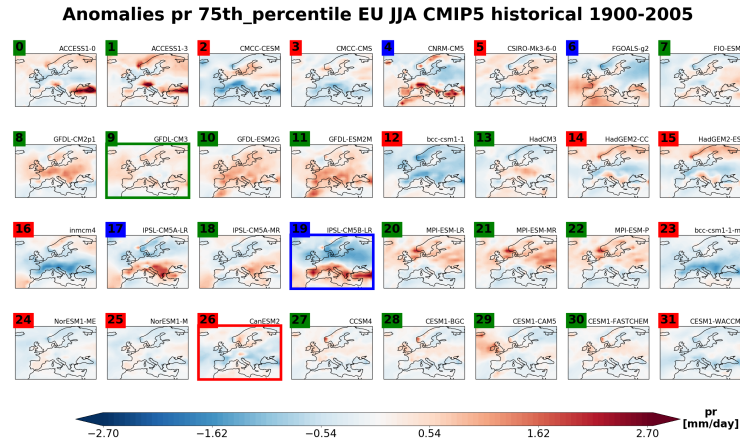


Fig. 1: Clustering based on the 75th percentile of historical summer (JJA) precipitation rate for CMIP5 models over 1900-2005. 3 clusters are computed, based on the principal components explaining 80% of the variance. The 32 models are grouped in three different clusters. The green cluster is the most populated with 16 ensemble members mostly characterized by a positive anomaly over central-north Europe. The red cluster counts 12 elements that exhibit a negative anomaly centered over southern Europe. The third cluster – labelled in blue- includes only 4 models showing a north-south dipolar precipitation anomaly, with a wetter than average Mediterranean counteracting dryer North-Europe. Ensemble members No.9, No.26 and No.19 are the “specimen” of each cluster, i.e. the model simulations that better represent the main features of that cluster. These ensemble members can eventually be used as representative of the whole possible outcomes of the multi-model ensemble distribution associated to the 32 CMIP5 historical integrations for the summer precipitation rate 75 th percentile over Europe when these outcomes are reduced from 32 to 3. The number of ensemble members of each cluster might provide a measure of the probability of occurrence of each cluster.

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.

20.4.2 Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/`

- `recipe_esacci_lst.yml`

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

- `lst.py`

20.4.3 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.

20.4.4 Variables

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

20.4.5 Observations and reformat scripts

This recipe and diagnostic is written to work with data created from the CMORizer `esmval-tool/cmorizers/obs/cmorize_obs_esacci_lst.py`. This takes the original 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 the possibility of biasing towards night time LST values being more prevalent because of how the cloud screening algorithms work.

20.4.6 References

- ESA CCI LST project <https://climate.esa.int/en/projects/land-surface-temperature/>

20.4.7 Example plots

20.5 Example recipes

20.5.1 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_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_julia.yml` produces a map plot with the mean temperature over the year 1997 plus a number that is configurable from the recipe.

The recipe `examples/recipe_decadal.yml` showcases how the `timerange` tag can be used to load datasets belonging to the DCP activity. Produces timeseries plots comparing the global mean temperature of a DCP dataset with an observational dataset.

For detailed instructions on obtaining input data, please refer to [Obtaining input data](#). However, in case you just quickly want to run through the example, you can use the following links to obtain the data from ESGF:

- [BCC-ESM1](#)
- [CanESM2](#)

Please refer to the terms of use for [CMIP5](#) and [CMIP6](#) data.

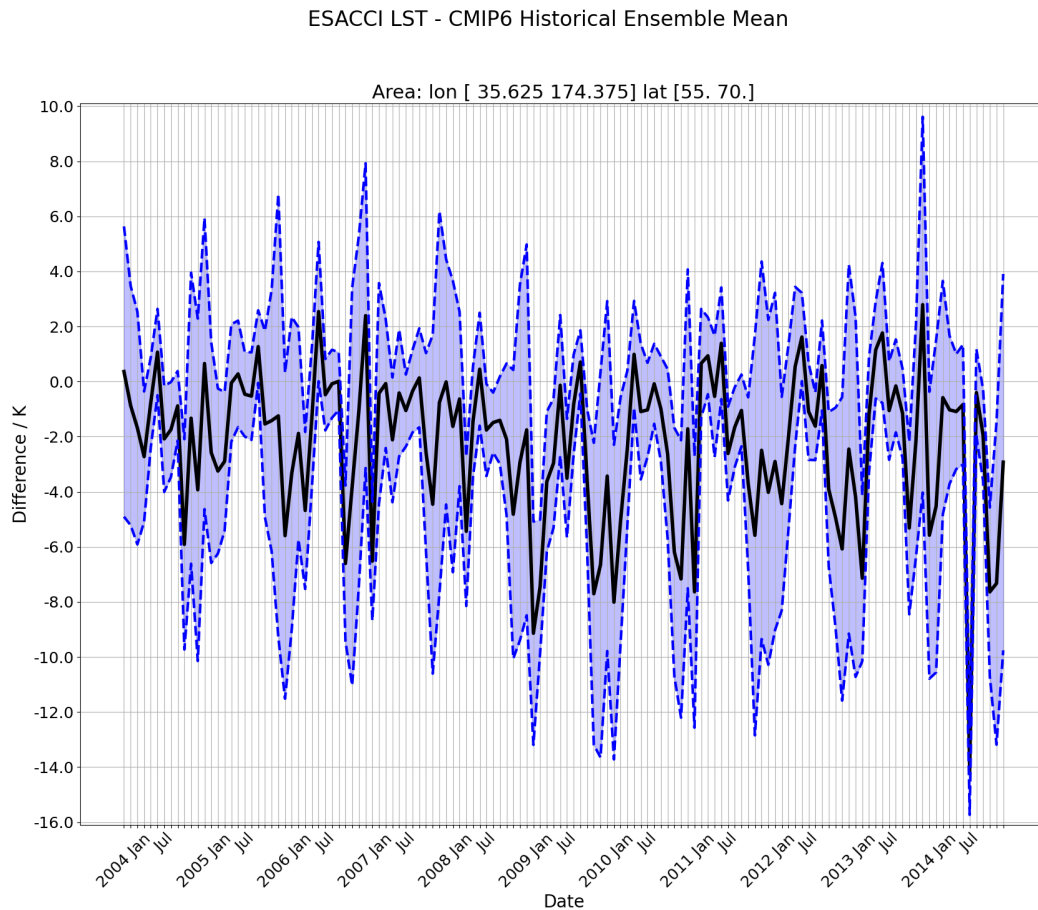


Fig. 2: Timeseries of the ESA CCI LST minus mean of CMIP6 ensembles. The selected region is 35E-175E, 55N-70N. The black line is the mean difference, and the blue shaded area denotes one standard deviation either way of the individual ensemble member's difference in LST. Models used for this are UKESM1 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.

20.5.2 Available recipes and diagnostics

Recipes are stored in `esmvaltool/recipes/`

- `examples/recipe_python.yml`
- `examples/recipe_extract_shape.yml`
- `examples/recipe_julia.yml`
- `examples/recipe_decadal.yml`

Diagnostics are stored in `esmvaltool/diag_scripts/`

- `examples/diagnostic.py`: visualize results and store provenance information
- `examples/diagnostic.jl`: visualize results and store provenance information
- `examples/decadal_example.py`: visualize results and store provenance information

20.5.3 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.

Optional settings for script

- `write_netcdf`: `true` (default) or `false`. This can be used to disable writing the results to netcdf files.

2. Script `examples/diagnostic.jl`

Required settings for script

- `parameter1`: example parameter, this number will be added to the mean (over time) value of the input data.

20.5.4 Variables

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

20.5.5 Example plots

20.6 Monitor

20.6.1 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.

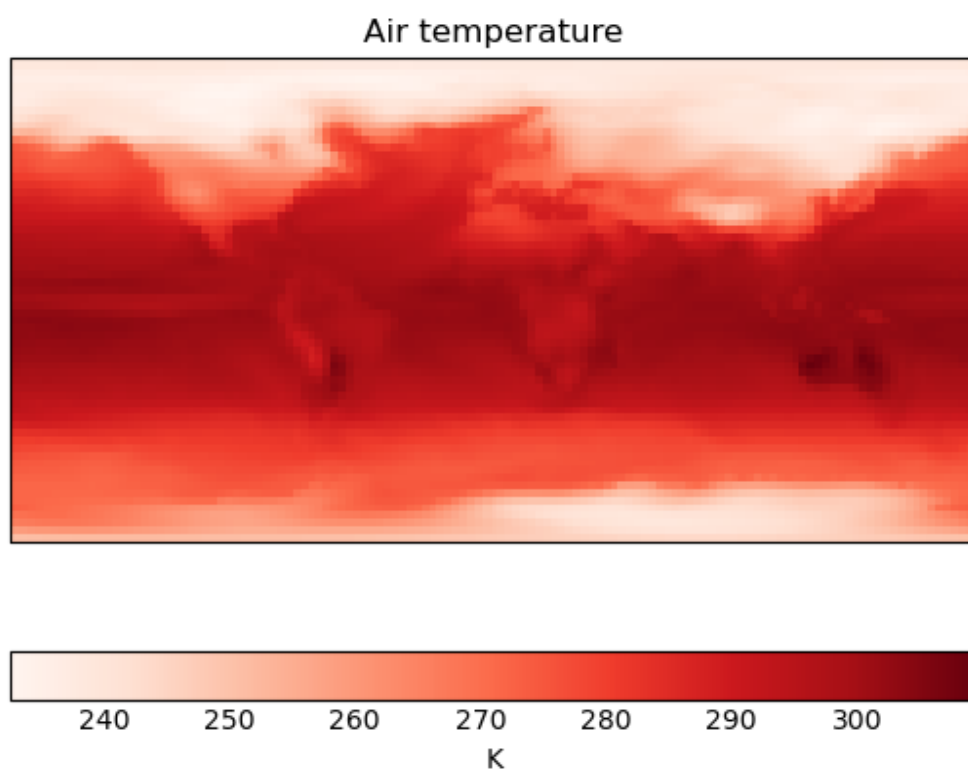


Fig. 3: Air temperature in January 2000 (BCC-ESM1 CMIP6).

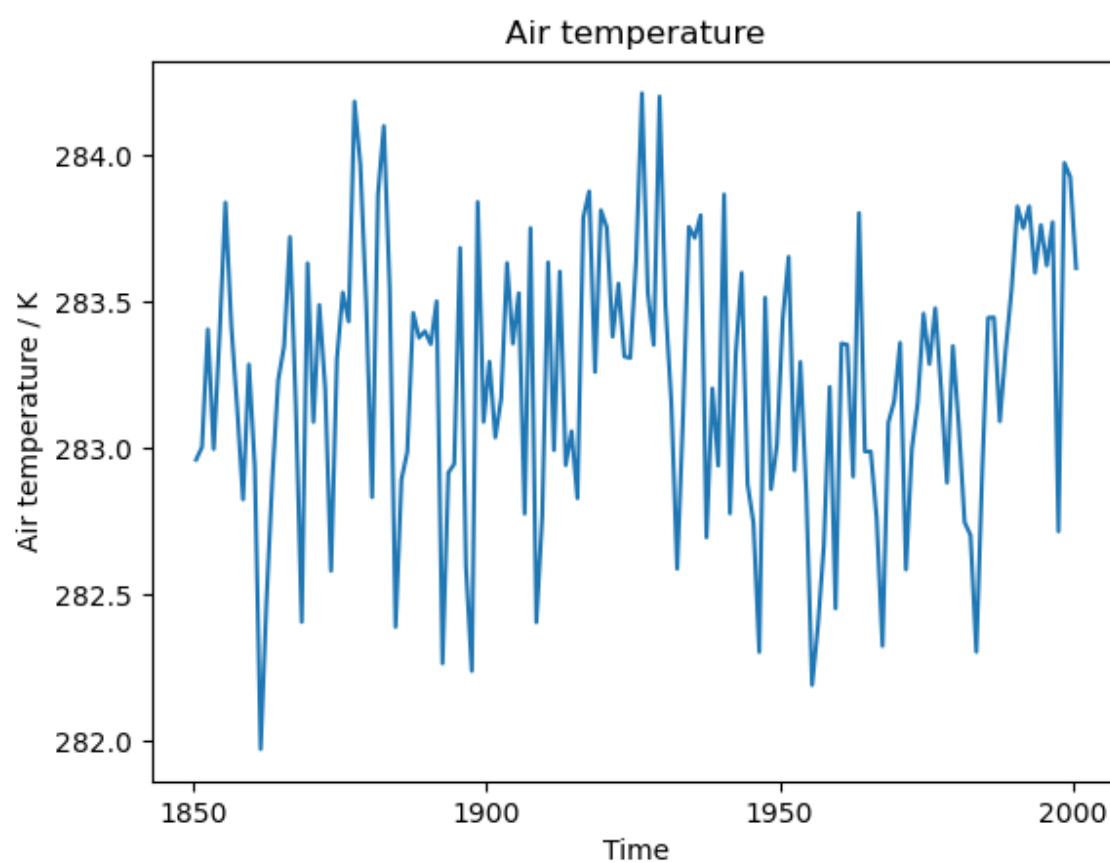


Fig. 4: Amsterdam air temperature (multimodel mean of CMIP5 CanESM2 and CMIP6 BCC-ESM1).

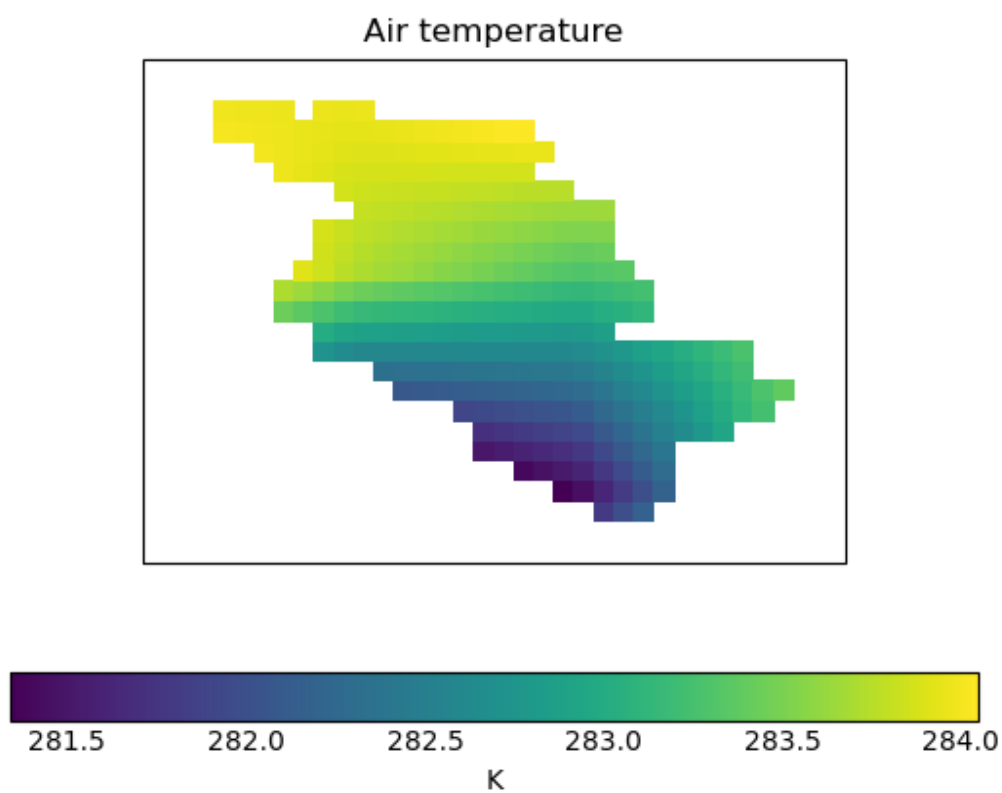


Fig. 5: Mean air temperature over the Elbe catchment during 2000-2002 according to CMIP5 CanESM2.

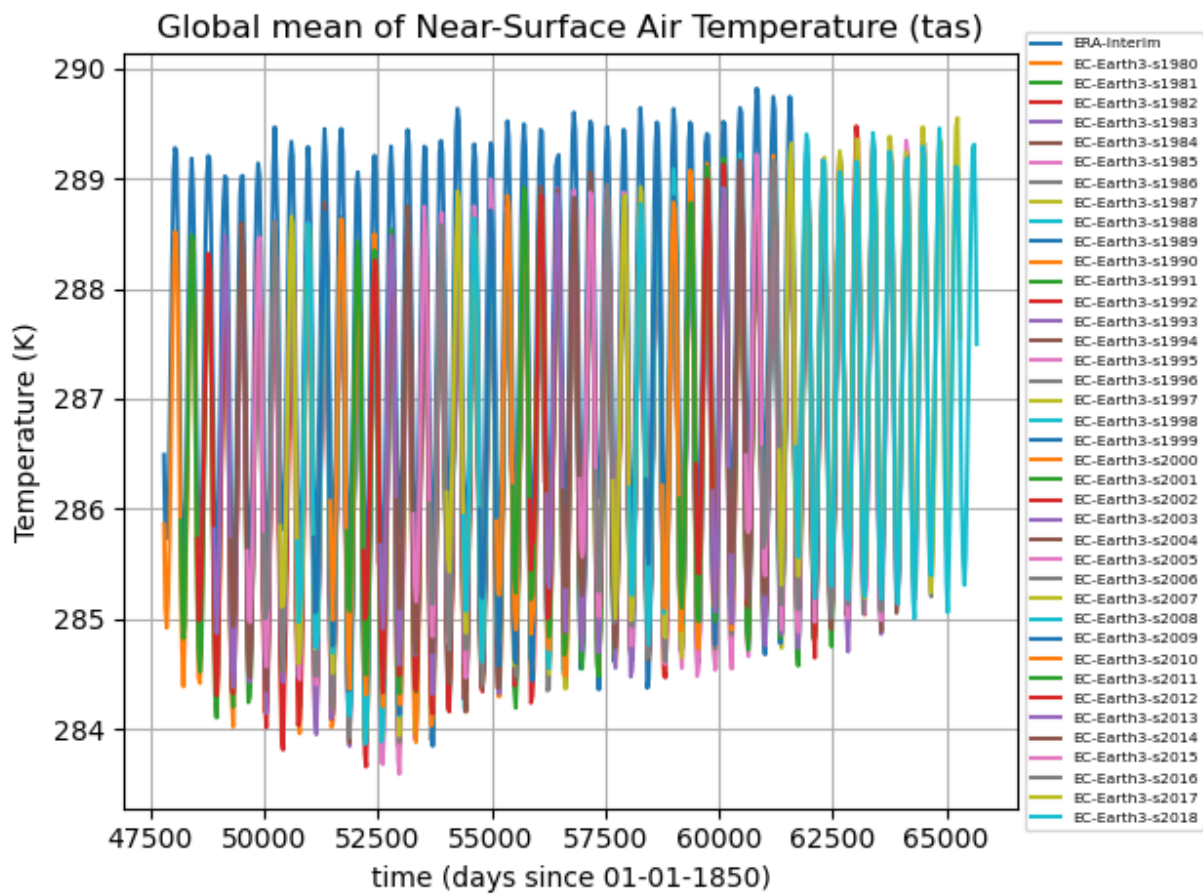


Fig. 6: Global mean temperature of CMIP6 dcppA-hindcast EC-Earth3 and OBS ERA-Interim.

20.6.2 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).

20.6.3 User settings

It is recommended to use a vector graphic file type (e.g., SVG) for the output files when running this recipe, i.e., run the recipe with the command line option `--output_file_type=svg` or use `output_file_type: svg` in your [User configuration file](#). 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).

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]
    supitle_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:

variables:

```

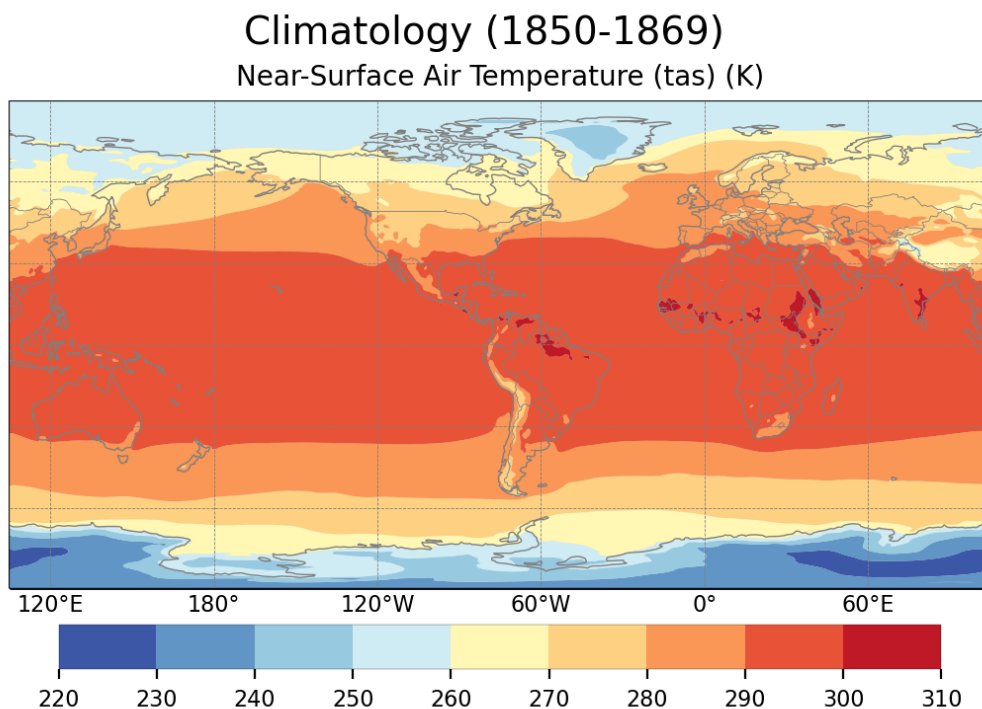
# Define default. Variable definitions completely override the default
# not just the values defined. If you want to override only the defined
# values, use yaml anchors as shown
default: &default
  colors: RdYlBu_r # Matplotlib colormap to use for the colorbar
  N: 20 # Number of map intervals to plot
  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 definitions are taken from variable's default
  # definition
  default:
    <<: *default
    bounds: 25-41,1
    extend: both
  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: SouthPolarStereographic
  projection_kwargs:
    central_longitude: 270
  smooth: true
  lon: [-120, -60, 0, 60, 120, 180]

```

20.6.4 Variables

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

20.6.5 Example plots



Global climatology of tas.

Seasonal climatology of pr, with a custom colorbar.

Monthly climatology of sivol, only for March and September.

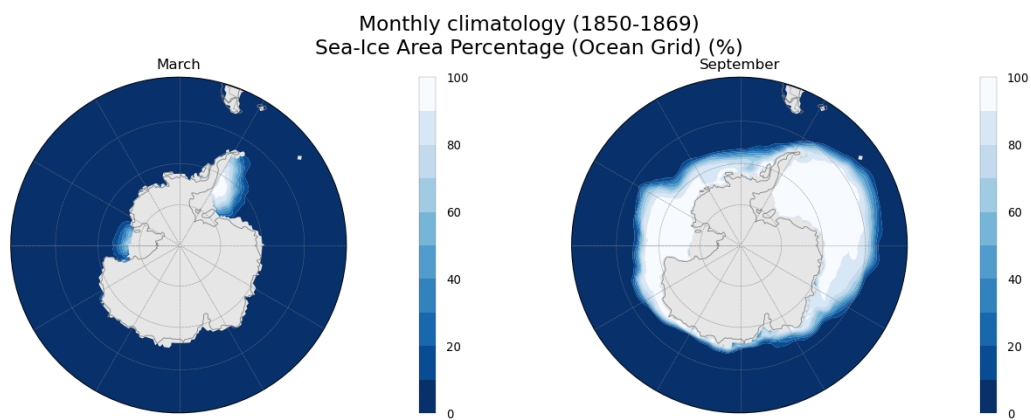
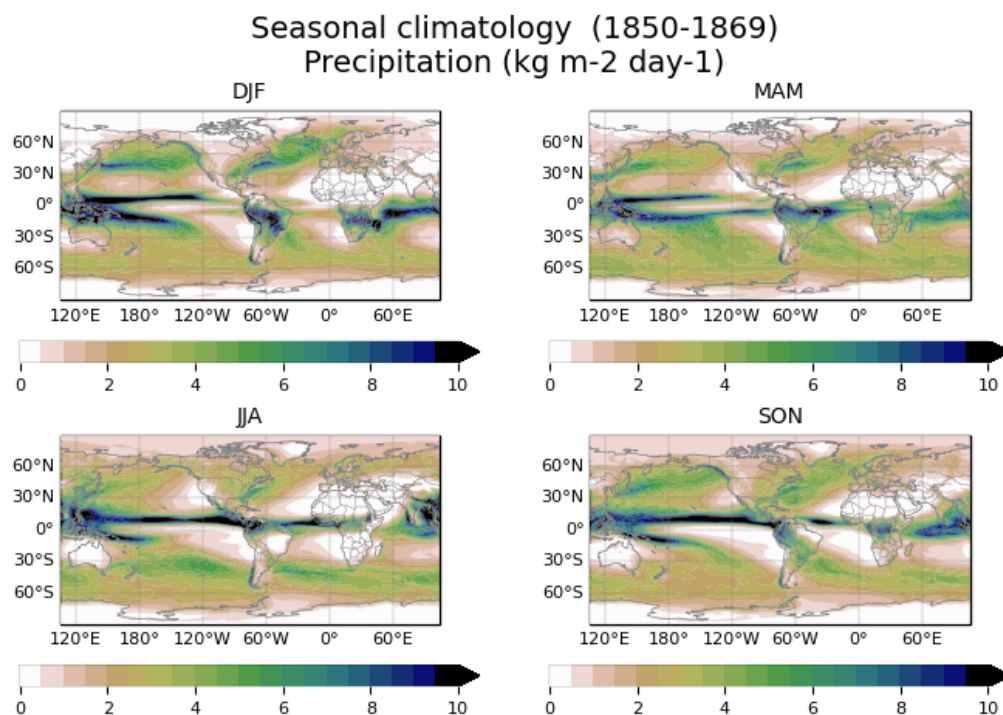
Timeseries of Niño 3.4 index, computed directly with the preprocessor.

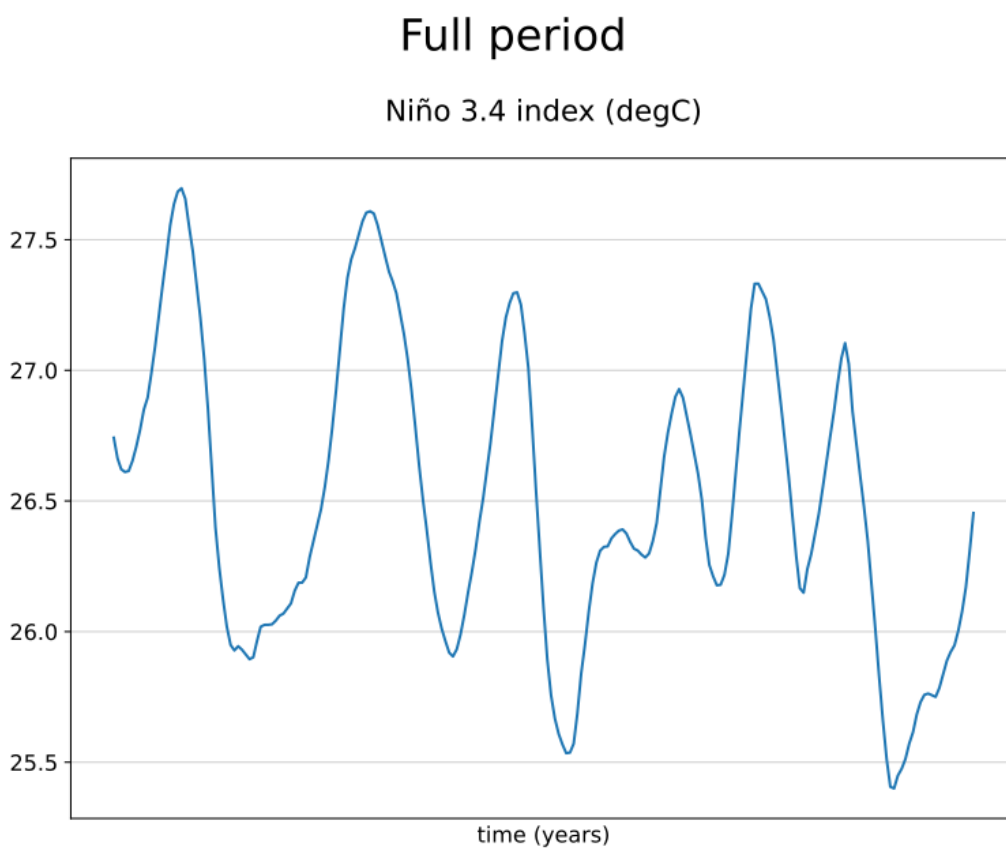
Annual cycle of tas.

Timeseries of tas including a reference dataset.

Global climatology of tas including a reference dataset.

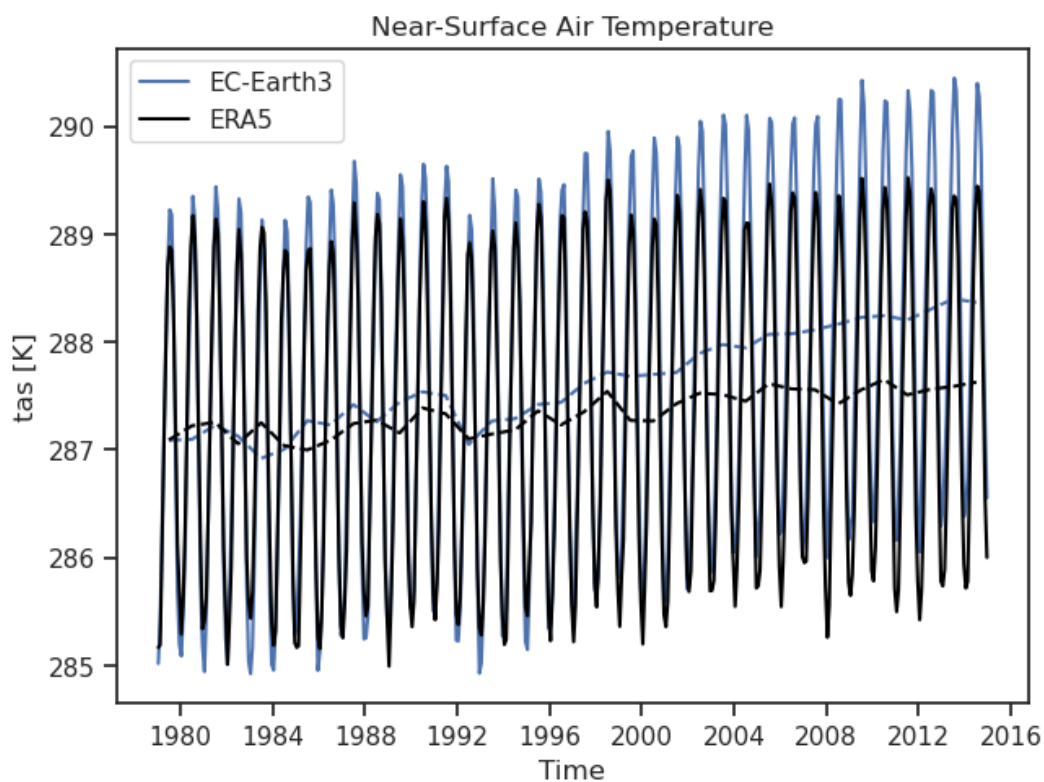
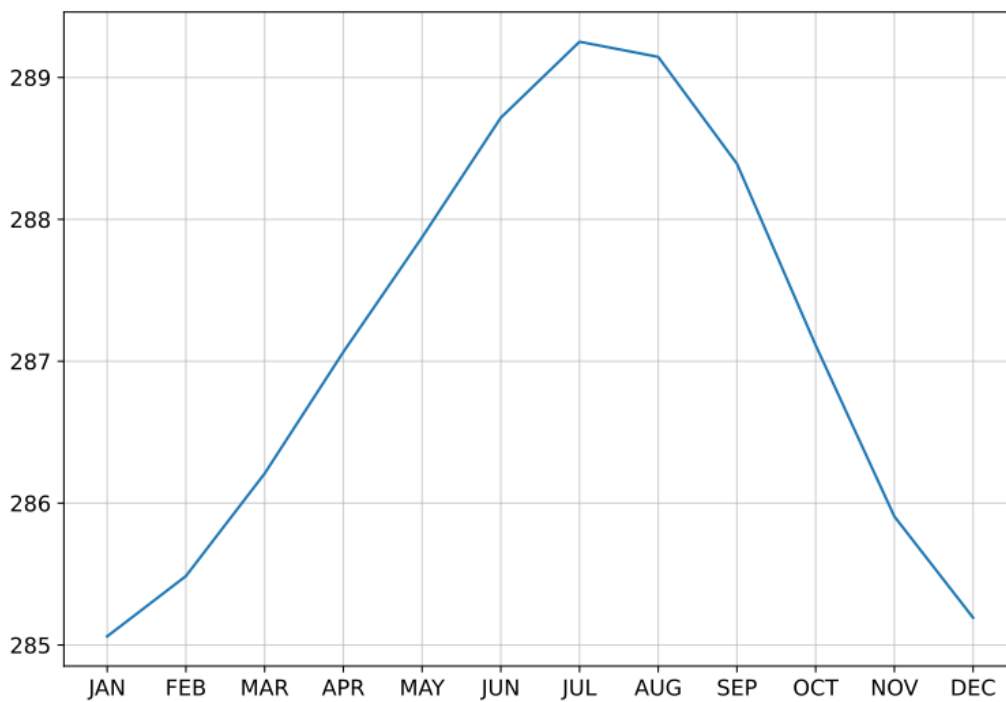
Vertical profile of ta including a reference dataset.

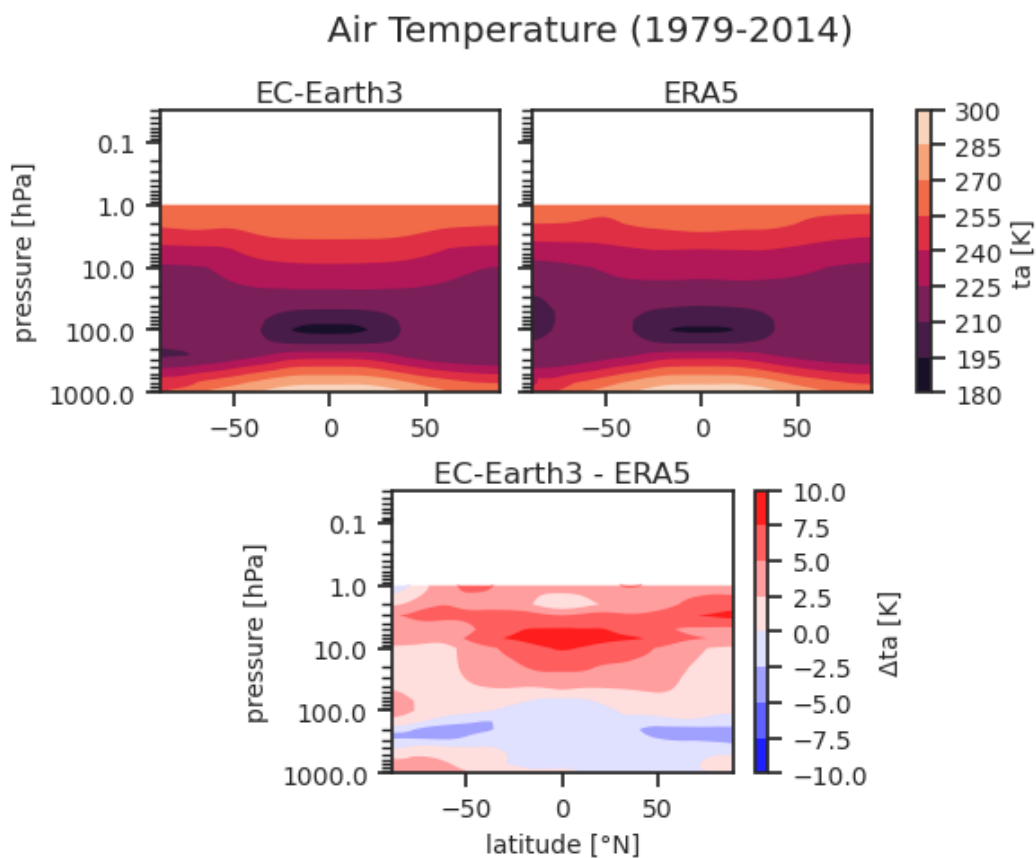
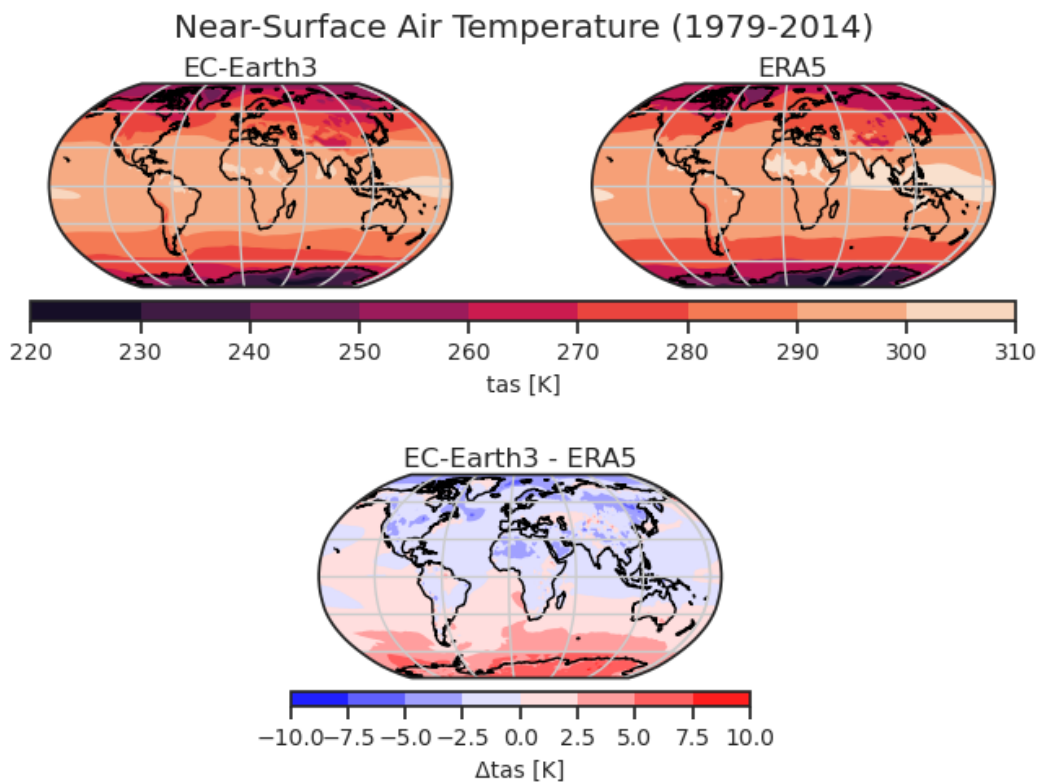




Annual cycle

Near-Surface Air Temperature (tas) (K)





20.7 Multi-model products

20.7.1 Overview

The goal of this diagnostic is to compute the multi-model ensemble mean for a set of models selected by the user for individual variables and different temporal resolutions (annual, seasonal, monthly).

After selecting the region (defined by the lowermost and uppermost longitudes and latitudes), the mean for the selected reference period is subtracted from the projections in order to obtain the anomalies for the desired period. In addition, the recipe computes the percentage of models agreeing on the sign of this anomaly, thus providing some indication on the robustness of the climate signal.

The output of the recipe consists of a colored map showing the time average of the multi-model mean anomaly and stippling to indicate locations where the percentage of models agreeing on the sign of the anomaly exceeds a threshold selected by the user. Furthermore, a time series of the area-weighted mean anomaly for the projections is plotted. For the plots, the user can select the length of the running window for temporal smoothing and choose to display the ensemble mean with a light shading to represent the spread of the ensemble or choose to display each individual models.

20.7.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_multimodel_products.yml`

Diagnostics are stored in `diag_scripts/magic_bsc/`

- `multimodel_products.R` - script for computing multimodel anomalies and their agreement.

20.7.3 User settings

User setting files are stored in `recipes/`

1. `recipe_multimodel_products.yml`

Required settings for script

- `colorbar_lim`: positive number specifying the range (`-colorbar_lim ... +colorbar_lim`) of the colorbar (0 = automatic colorbar scaling)
- `moninf`: integer specifying the first month of the seasonal mean period to be computed
- `monsup`: integer specifying the last month of the seasonal mean period to be computed, if it's null the anomaly of month indicated in `moninf` will be computed
- `agreement_threshold`: integer between 0 and 100 indicating the threshold in percent for the minimum agreement between models on the sign of the multi-model mean anomaly for the stippling to be plotted
- `running_mean`: integer indicating 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).

20.7.4 Variables

- any Amon variable (atmos, monthly mean, longitude latitude time)

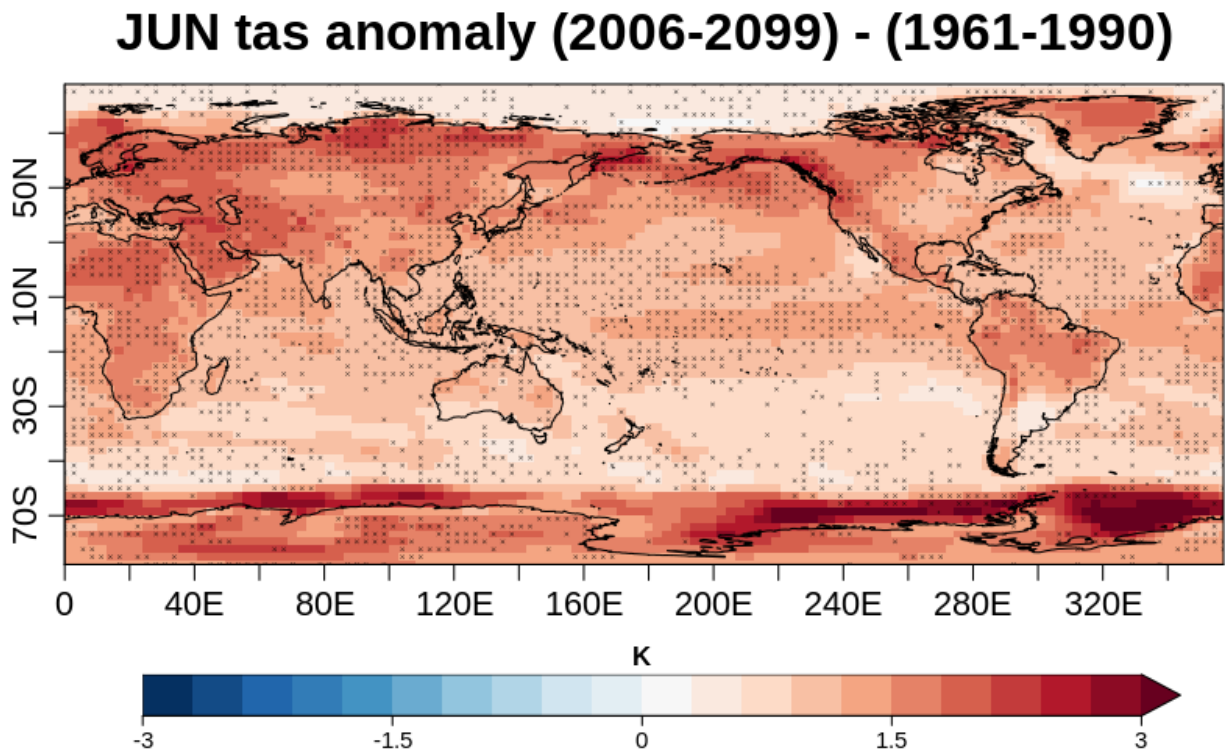
20.7.5 Observations and reformat scripts

None

20.7.6 References

- Hagedorn, R., Doblas-Reyes, F. J., Palmer, T. N., Nat E H Ag E D O R N, R. E., & Pa, T. N. (2005). The rationale behind the success of multi-model ensembles in seasonal forecasting-I. Basic concept, 57, 219–233. <https://doi.org/10.3402/tellusa.v57i3.14657>
- Weigel, A. P., Liniger, M. A., & Appenzeller, C. (2008). Can multi-model combination really enhance the prediction skill of probabilistic ensemble forecasts? Quarterly Journal of the Royal Meteorological Society, 134(630), 241–260. <https://doi.org/10.1002/qj.210>

20.7.7 Example plots



Multi-model mean anomaly of 2-m air temperature during the future projection 2006-2099 in June considering the reference period 1961-1990 (colours). Crosses indicate that the 80% of models agree in the sign of the multi-model mean anomaly. The models selected are BCC-CSM1-1, MPI-ESM-MR and MIROC5 in the r1i1p1 ensembles for the RCP 2.6 scenario.

20.8 Psyplot Diagnostics

20.8.1 Overview

These recipes showcase the use of the Psyplot diagnostic that provides a high-level interface to [Psyplot](#) for ESMValTool recipes.

20.8.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_psyplot.yml`

Diagnostics are stored in `diag_scripts/`

- `psyplot_diag.py`

20.8.3 Variables

Arbitrary variables are supported.

20.8.4 Observations and reformat scripts

Arbitrary datasets are supported.

20.8.5 References

- Sommer, (2017), The psyplot interactive visualization framework, Journal of Open Source Software, 2(16), 363, doi:10.21105/joss.00363

20.8.6 Example plots

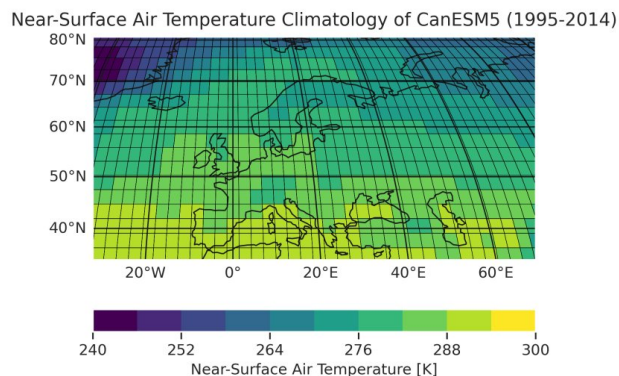


Fig. 7: Historical near-surface air temperature climatology over Europe simulated by CanESM5 between 1995 and 2014. The plot visualizes the individual rectangular grid cells of the model's regular grid.

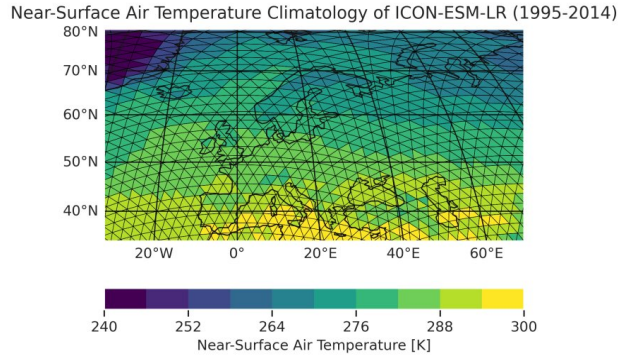


Fig. 8: Historical near-surface air temperature climatology over Europe simulated by ICON-ESM-LR between 1995 and 2014. The plot visualizes the individual triangular grid cells of the model's unstructured grid.

20.9 Capacity factor for solar photovoltaic (PV) systems

20.9.1 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.

20.9.2 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.

20.9.3 User settings

User setting files are stored in `recipes/`

1. `recipe_capacity_factor.yml`

Required settings for script

- `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.

20.9.4 Variables

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

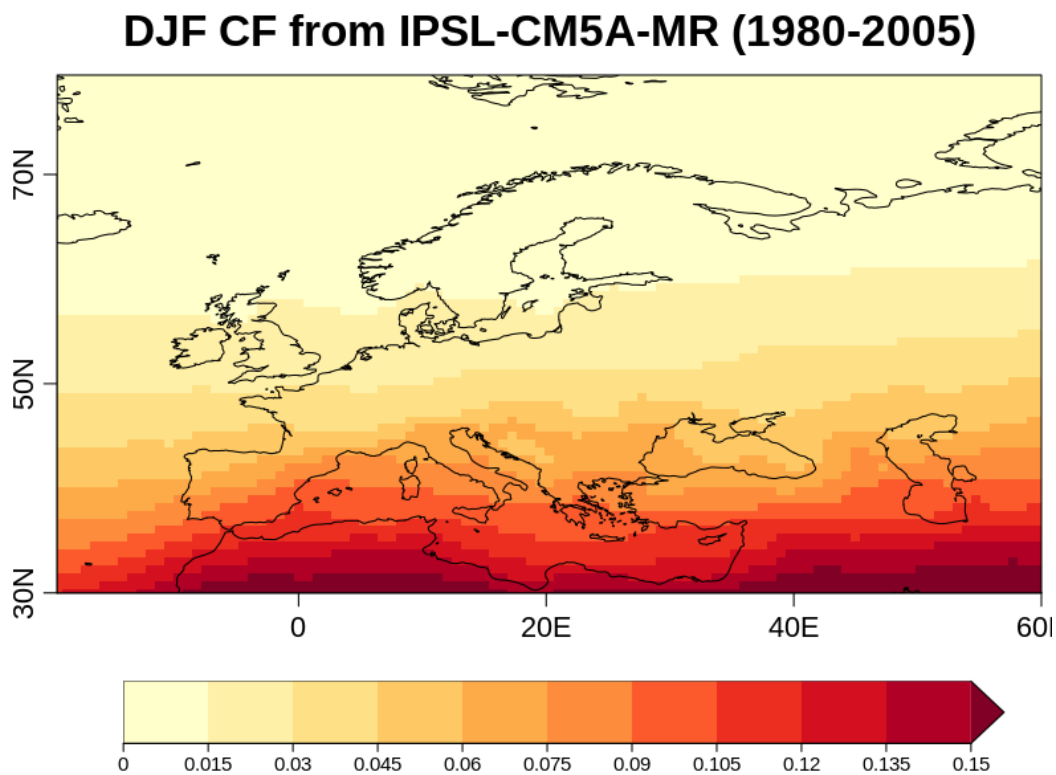
20.9.5 Observations and reformat scripts

- ERA-Interim

20.9.6 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.

20.9.7 Example plots



PV capacity factor calculated from IPSL-CM5-MR during the DJF season for 1980–2005.

20.10 RainFARM stochastic downscaling

20.10.1 Overview

Precipitation extremes and small-scale variability are essential drivers in many climate change impact studies. However, the spatial resolution currently achieved by global and regional climate models is still insufficient to correctly identify the fine structure of precipitation intensity fields. In the absence of a proper physically based representation, this scale gap can be at least temporarily bridged by adopting a stochastic rainfall downscaling technique (Rebora et al, 2006). With this aim, the Rainfall Filtered Autoregressive Model (RainFARM) was developed to apply the stochastic precipitation downscaling method to climate models. The RainFARM Julia library and command-line tool version (<https://github.com/jhardenberg/RainFARM.jl>) was implemented as recipe. The stochastic method allows to predict climate variables at local scale from information simulated by climate models at regional scale: It first evaluates the statistical distribution of precipitation fields at regional scale and then applies the relationship to the boundary conditions of the climate model to produce synthetic fields at the requested higher resolution. RainFARM exploits the nonlinear transformation of a Gaussian random precipitation field, conserving the information present in the fields at larger scale (Rebora et al., 2006; D'Onofrio et al., 2014).

20.10.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_rainfarm.yml

Diagnostics are stored in diag_scripts/rainfarm/

- rainfarm.jl

20.10.3 User settings

Required settings for script

- slope: spatial spectral slope (set to 0 to compute automatically from large scales)
- nens: number of ensemble members to be calculated
- nf: number of subdivisions for downscaling (e.g. 8 will produce output fields with linear resolution increased by a factor 8)
- conserv_glob: logical, if to conserve precipitation over full domain
- conserv_smooth: logical, if to conserve precipitation using convolution (if neither conserv_glob or conserv_smooth is chosen, box conservation is used)
- weights_climo: set to false or omit if no orographic weights are to be used, else set it to the path to a fine-scale precipitation climatology file. If a relative file path is used, *auxiliary_data_dir* will be searched for this file. The file is expected to be in NetCDF format and should contain at least one precipitation field. If several fields at different times are provided, a climatology is derived by time averaging. Suitable climatology files could be for example a fine-scale precipitation climatology from a high-resolution regional climate model (see e.g. Terzago et al. 2018), a local high-resolution gridded climatology from observations, or a reconstruction such as those which can be downloaded from the WORLDCLIM (<http://www.worldclim.org>) or CHELSA (<http://chelsa-climate.org>) websites. The latter data will need to be converted to NetCDF format before being used (see for example the GDAL tools (<https://www.gdal.org>)).

20.10.4 Variables

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

20.10.5 Observations and reformat scripts

None.

20.10.6 References

- Terzago et al. 2018, Nat. Hazards Earth Syst. Sci., 18, 2825-2840
- D'Onofrio et al. 2014, J of Hydrometeorology 15, 830-843
- Rebora et. al 2006, JHM 7, 724

20.10.7 Example plots

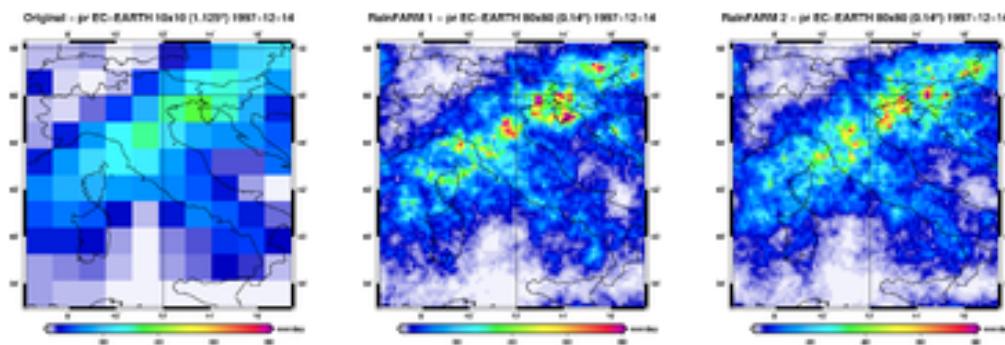


Fig. 9: Example of daily cumulated precipitation from the CMIP5 EC-EARTH model on a specific day, downscaled using RainFARM from its original resolution (1.125°) (left panel), increasing spatial resolution by a factor of 8 to 0.14° ; Two stochastic realizations are shown (central and right panel). A 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).

20.11 Sea Ice

20.11.1 Overview

The sea ice diagnostics include:

- (1) time series of Arctic and Antarctic sea ice area and extent (calculated as the total area (km^2) of grid cells with sea ice concentrations (sic) of at least 15%).
- (2) ice extent trend distributions for the Arctic in September and the Antarctic in February.
- (3) calculation of year of near disappearance of Arctic sea ice

- (4) scatter plots of (a) historical trend in September Arctic sea ice extent (SSIE) vs historical long-term mean SSIE; (b) historical SSIE mean vs 1st year of disappearance (YOD) RCP8.5; (c) historical SSIE trend vs YOD RCP8.5.

20.11.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_seaice.yml

Diagnostics are stored in diag_scripts/seaice/

- seaice_aux.ncl: contains functions for calculating sea ice area or extent from sea ice concentration and first year of disappearance
- seaice_ecs.ncl: scatter plots of mean/trend of historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a)
- seaice_trends.ncl: calculates ice extent trend distributions (similar to IPCC AR5 Chapter 9, Fig. 9.24c/d)
- seaice_tsline.ncl: creates a time series line plots of total sea ice area and extent (accumulated) for northern and southern hemispheres with optional multi-model mean and standard deviation. One value is used per model per year, either annual mean or the mean value of a selected month (similar to IPCC AR5 Chapter 9, Fig. 9.24a/b)
- seaice_yod.ncl: calculation of year of near disappearance of Arctic sea ice

20.11.3 User settings in recipe

1. Script seaice_ecs.ncl

Required settings (scripts)

- hist_exp: name of historical experiment (string)
- month: selected month (1, 2, ..., 12) or annual mean ("A")
- rcp_exp: name of RCP experiment (string)
- region: region to be analyzed ("Arctic" or "Antarctic")

Optional settings (scripts)

- fill_pole_hole: fill observational hole at North pole (default: False)
- styleset: color style (e.g. "CMIP5")

Optional settings (variables)

- reference_dataset: reference dataset

2. Script seaice_trends.ncl

Required settings (scripts)

- month: selected month (1, 2, ..., 12) or annual mean ("A")
- region: region to be analyzed ("Arctic" or "Antarctic")

Optional settings (scripts)

- fill_pole_hole: fill observational hole at North pole, Default: False

Optional settings (variables)

- ref_model: array of references plotted as vertical lines

3. Script `seaiice_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_1g: 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 `seaiice_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

20.11.4 Variables

- sic (ocean-ice, monthly mean, longitude latitude time)
- areacello (fx, longitude latitude)

20.11.5 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)

20.11.6 References

- Massonnet, F. et al., The Cryosphere, 6, 1383-1394, doi: 10.5194/tc-6-1383-2012, 2012.
- Stroeve, J. et al., Geophys. Res. Lett., 34, L09501, doi:10.1029/2007GL029703, 2007.

20.11.7 Example plots

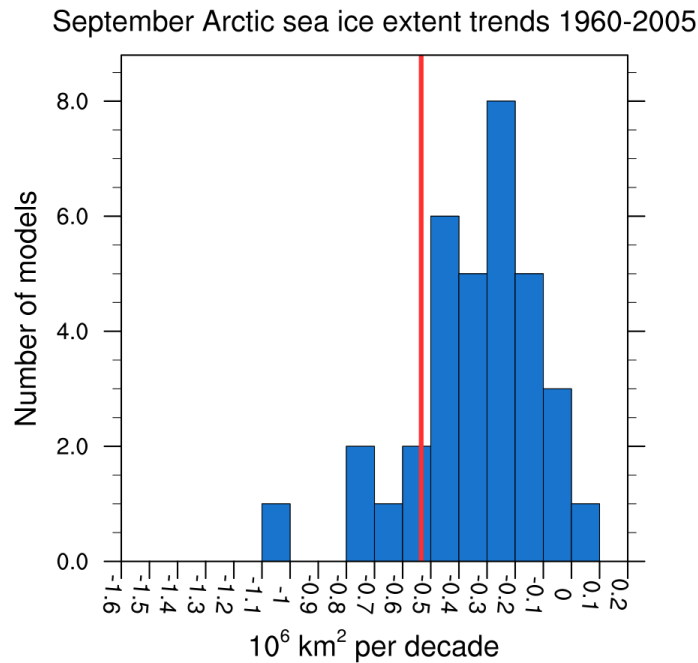


Fig. 10: Sea ice extent trend distribution for the Arctic in September (similar to IPCC AR5 Chapter 9, Fig. 9.24c). [seaice_trends.ncl]

20.12 Seaice drift

20.12.1 Overview

This recipe allows to quantify the relationships between Arctic sea-ice drift speed, concentration and thickness (Docquier et al., 2017). A decrease in concentration or thickness, as observed in recent decades in the Arctic Ocean (Kwok, 2018; Stroeve and Notz, 2018), leads to reduced sea-ice strength and internal stress, and thus larger sea-ice drift speed (Rampal et al., 2011). This in turn could provide higher export of sea ice out of the Arctic Basin, resulting in lower sea-ice concentration and further thinning. Olason and Notz (2014) investigate the relationships between Arctic sea-ice drift speed, concentration and thickness using satellite and buoy observations. They show that both seasonal and recent long-term changes in sea ice drift are primarily correlated to changes in sea ice concentration and thickness. This recipe allows to quantify these relationships in climate models.

In this recipe, four process-based metrics are computed based on the multi-year monthly mean sea-ice drift speed, concentration and thickness, averaged over the Central Arctic.

The first metric is the ratio between the modelled drift-concentration slope and the observed drift-concentration slope. The second metric is similar to the first one, except that sea-ice thickness is involved instead of sea-ice concentration. The third metric is the normalised distance between the model and observations in the drift-concentration space. The fourth metric is similar to the third one, except that sea-ice thickness is involved instead of sea-ice concentration.

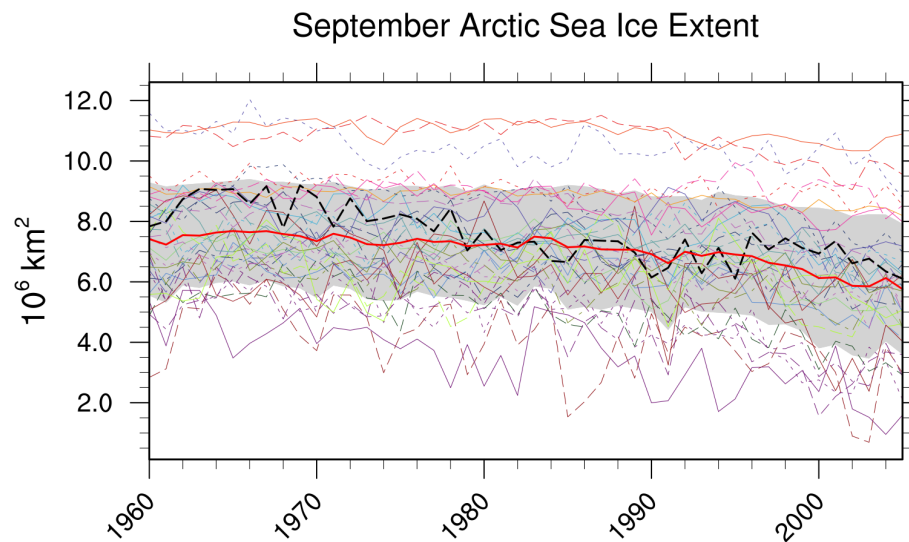


Fig. 11: Time series of total sea ice area and extent (accumulated) for the Arctic in September including multi-model mean and standard deviation (similar to IPCC AR5 Chapter 9, Fig. 9.24a). [seaice_tsline.ncl]

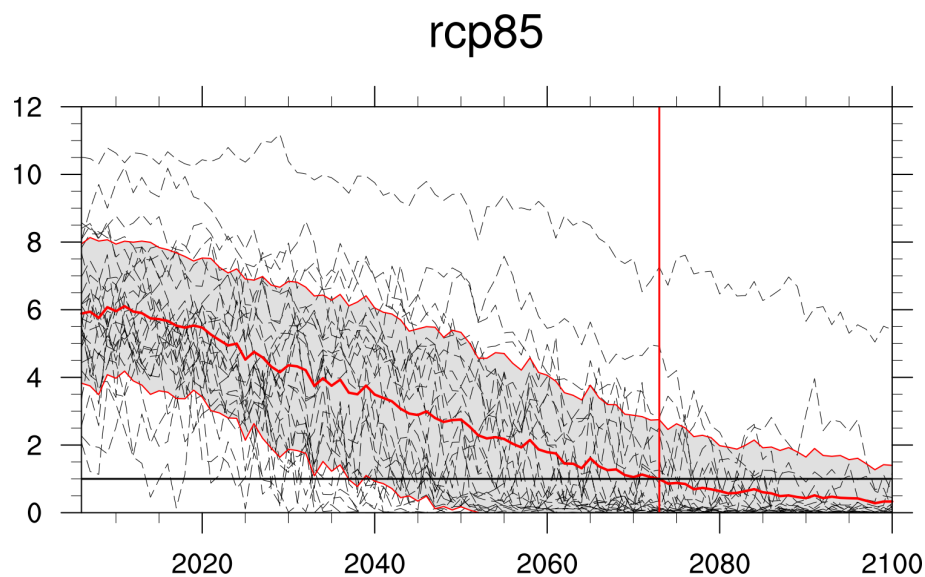


Fig. 12: Time series of September Arctic sea ice extent for individual CMIP5 models, multi-model mean and multi-model standard deviation, year of disappearance (similar to IPCC AR5 Chapter 12, Fig. 12.31e). [seaice_yod.ncl]

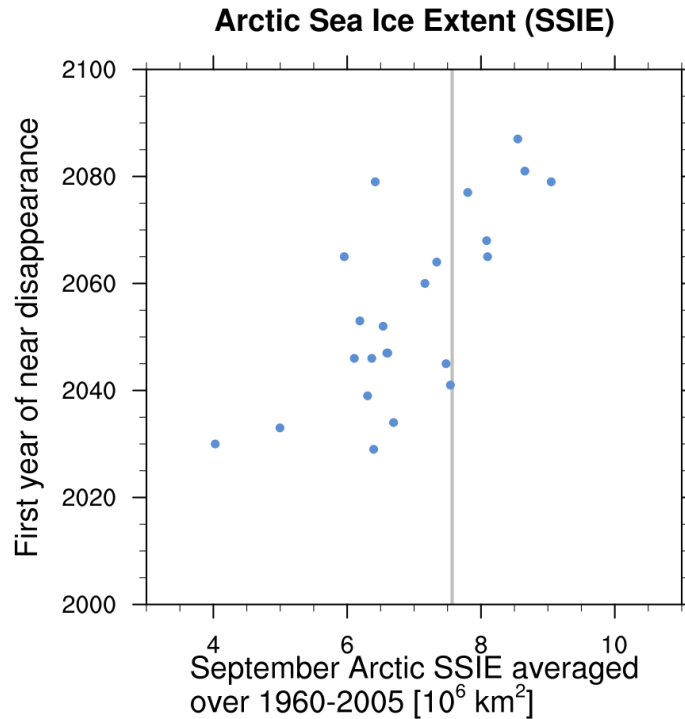


Fig. 13: Scatter plot of mean historical September Arctic sea ice extent vs 1st year of disappearance (RCP8.5) (similar to IPCC AR5 Chapter 12, Fig. 12.31a). [seaice_ecs.ncl]

20.12.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_seaice_drift.yml

Diagnostics are stored in diag_scripts/seaice_drift/

- seaice_drift.py: Compute metrics and plot results

20.12.3 User settings in recipe

1. Script diag_shapeselect.py

Required settings (scripts)

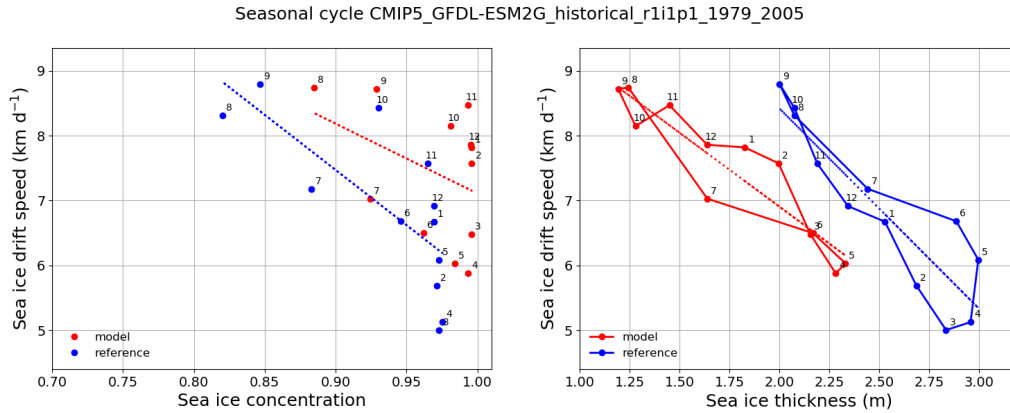
One of the following two combinations is required:

1. Latitude threshold:
 - latitude_threshold: metric will be computed north of this latitude value
2. Polygon:
 - polygon: metric will be computed inside the give polygon. Polygon is defined as a list of (lon, lat) tuple
 - polygon_name: name of the region defined by the polygon

20.12.4 Variables

- `sispeed`, `sithick`, `siconc` (daily)

20.12.5 Example plots



Scatter plots of modelled (red) and observed (blue) monthly mean sea-ice drift speed against sea-ice concentration (left panel) and sea-ice thickness (right panel) temporally averaged over the period 1979–2005 and spatially averaged over the SCICEX box.

20.13 Seaice feedback

20.13.1 Overview

In this recipe, one process-based diagnostic named the Ice Formation Efficiency (IFE) is computed based on monthly mean sea-ice volume estimated north of 80°N. The choice of this domain is motivated by the desire to minimize the influence of dynamic processes but also by the availability of sea-ice thickness measurements. The diagnostic intends to evaluate the strength of the negative sea-ice thickness/growth feedback, which causes late-summer negative anomalies in sea-ice area and volume to be partially recovered during the next growing season. A chief cause behind the existence of this feedback is the non-linear inverse dependence between heat conduction fluxes and sea-ice thickness, which implies that thin sea ice grows faster than thick sea ice. To estimate the strength of that feedback, anomalies of the annual minimum of sea-ice volume north of 80°N are first estimated. Then, the increase in sea-ice volume until the next annual maximum is computed for each year. The IFE is defined as the regression of this ice volume production onto the baseline summer volume anomaly.

20.13.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_seaice_feedback.yml`

Diagnostics are stored in `diag_scripts/seaice_feedback/`

- `negative_seaice_feedback.py`: scatterplot showing the feedback between seaice volume and seaice growth

20.13.3 User settings

script negative_seaice_feedback.py

Optional settings for script

- plot: dictionary containing plot options:
 - point_color: color of the plot points. (Default: black)
 - point_size: size of the plot points. (Default: 10)
 - show_values: show numerical values of feedback in plot. (Default: True)

20.13.4 Variables

- sit (seaice, monthly mean, time latitude longitude)

20.13.5 References

- Massonnet, F., Vancoppenolle, M., Goosse, H., Docquier, D., Fichefet, T. and Blanchard-Grigglesworth, E., 2018. Arctic sea-ice change tied to its mean state through thermodynamic processes. *Nature Climate Change*, 8: 599-603.

20.13.6 Example plots

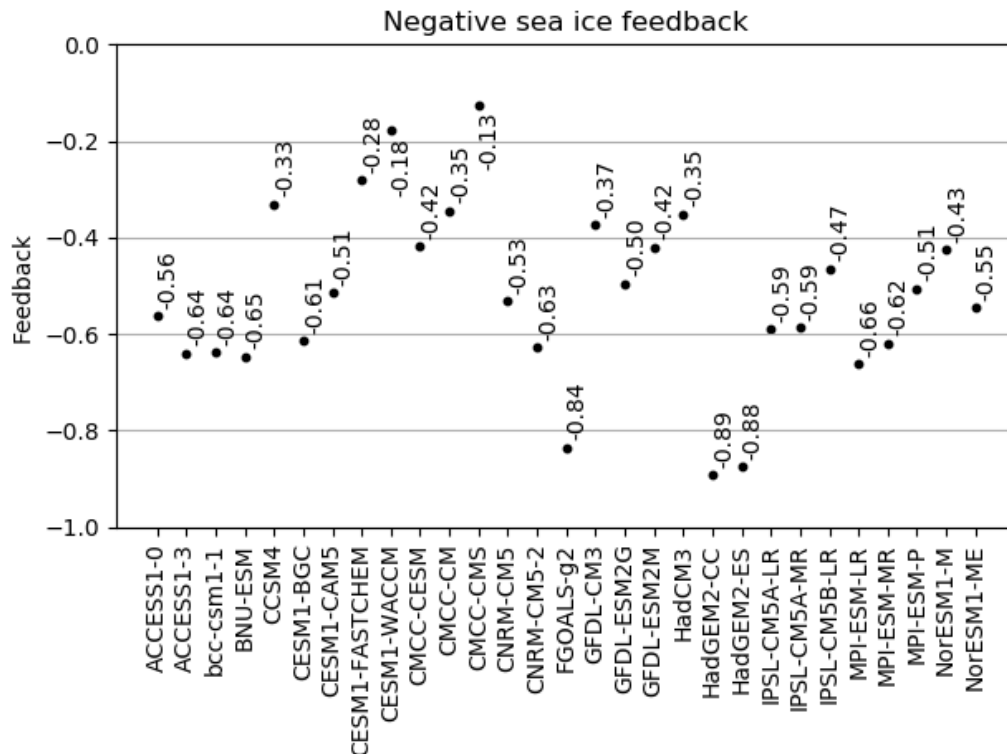


Fig. 14: Seaice negative feedback values (CMIP5 historical experiment 1979-2004).

20.14 Shapeselect

20.14.1 Overview

Impact modelers are often interested in data for irregular regions best defined by a shapefile. With the shapefile selector tool, the user can extract time series or CII data for a user defined region. The region is defined by a user provided shapefile that includes one or several polygons. For each polygon, a new timeseries, or CII, is produced with only one time series per polygon. The spatial information is reduced to a representative point for the polygon ('representative') or as an average of all grid points within the polygon boundaries ('mean_inside'). If there are no grid points strictly inside the polygon, the 'mean_inside' method defaults to 'representative' for that polygon. An option for displaying the grid points together with the shapefile polygon allows the user to assess which method is most optimal. In case interpolation to a high input grid is necessary, this can be provided in a pre-processing stage. Outputs are in the form of a NetCDF file, or as ascii code in csv format.

20.14.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_shapeselect.yml

Diagnostics are stored in diag_scripts/shapeselect/

- diag_shapeselect.py: calculate the average of grid points inside the user provided shapefile and returns the result as a NetCDF or Excel sheet.

20.14.3 User settings in recipe

1. Script diag_shapeselect.py

Required settings (scripts)

- shapefile: path to the user provided shapefile. A relative path is relative to the auxiliary_data_dir as configured in config-user.yml.
- weighting_method: the preferred weighting method 'mean_inside' - mean of all grid points inside polygon; 'representative' - one point inside or close to the polygon is used to represent the complete area.
- write_xlsx: true or false to write output as Excel sheet or not.
- write_netcdf: true or false to write output as NetCDF or not.

20.14.4 Variables

- pr,tas (daily)

20.14.5 Example plots

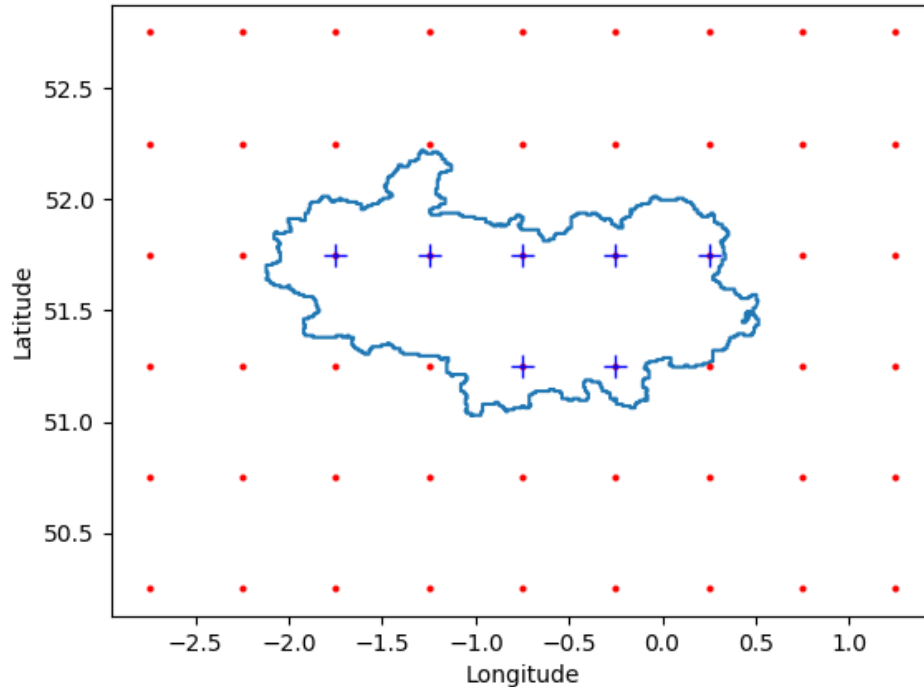


Fig. 15: Example of the selection of model grid points falling within (blue pluses) and without (red dots) a provided shapefile (blue contour).

20.15 Short test versions of scientific recipes to check for backward compatibility.

20.15.1 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.

20.15.2 Available recipes and diagnostics

Recipes are stored in `recipes/testing/`

- `recipe_deangelis15nat_fig1_fast.yml`

Diagnostics are stored in `diag_scripts/`

- `deangelis15nat/deangelis1b.py`

20.15.3 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 reduced 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*.

20.16 Toymodel

20.16.1 Overview

The goal of this diagnostic is to simulate single-model ensembles from an observational dataset to investigate the effect of observational uncertainty. For further discussion of this synthetic value generator, its general application to forecasts and its limitations, see Weigel et al. (2008). The output is a netcdf file containing the synthetic observations. Due to the sampling of the perturbations from a Gaussian distribution, running the recipe multiple times, with the same observation dataset and input parameters, will result in different outputs.

20.16.2 Available recipes and diagnostics

Recipes are stored in `recipes/`

- `recipe_toymodel.yml`

Diagnostics are stored in `diag_scripts/magic_bsc/`

- `toymodel.R`: generates a single model ensemble of synthetic observations

20.16.3 User settings

User setting files are stored in `recipes/`

1. `recipe_toymodel.yml`

Required settings for preprocessor

`extract_region:`

- `start_longitude`: minimum longitude
- `end_longitude`: maximum longitude
- `start_latitude`: minimum longitude
- `end_latitude`: maximum latitude

`extract_levels`: (for 3D variables)

- `levels`: [50000] # e.g. for 500 hPa level

Required settings for script

- `number_of_members`: integer specifying the number of members to be generated
- `beta`: the user defined underdispersion ($\text{beta} \geq 0$)

20.16.4 Variables

- any variable (atmos/ocean, daily-monthly, longitude, latitude, time)

20.16.5 Observations and reformat scripts

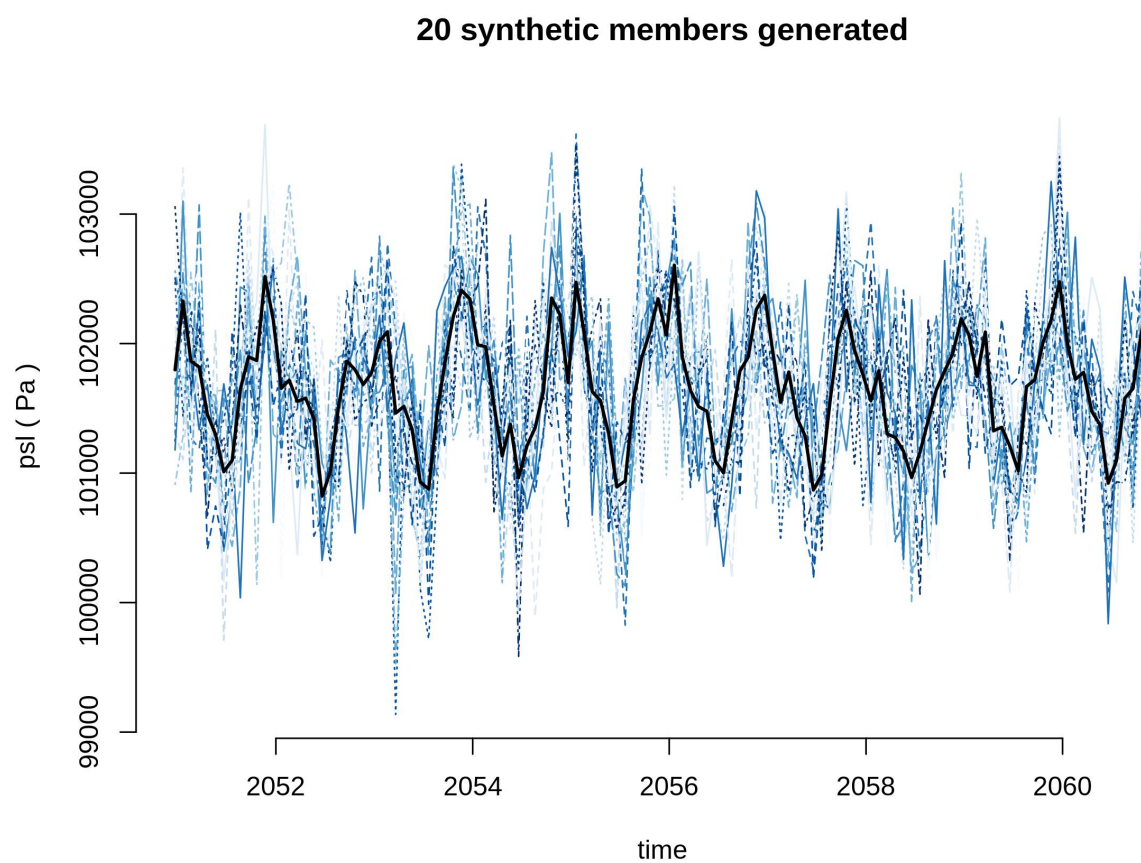
None

20.16.6 References

- Bellprat, O., Massonnet, F., Siegert, S., Prodhomme, C., Macias-Gómez, D., Guemas, V., & Doblas-Reyes, F. (2017). Uncertainty propagation in observational references to climate model scales. *Remote Sensing of Environment*, 203, 101-108.
- Massonnet, 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.

20.16.7 Example plots

Twenty synthetic single-model ensemble generated by the `recipe_toymodel.yml` (see Section 3.7.2) for the 2051-2060 monthly data of `r1i1p1` RCP 4.5 scenario of `BCC_CSM1-1` simulation.



Part VI

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](#) or, if you need longer term access or more computational resources, the [IS-ENES3 Trans-national Access call](#).

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.

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 --offline=False 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 re-use 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 [Synda](#) to maintain your own collection of ESGF data.

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*.

22.1 Using a CMORizer script

ESMValTool comes with a set of CMORizers readily available. The CMORizers are dataset-specific scripts that can be run once to generate a local pool of CMOR-compliant data. The necessary information to download and process the data is provided in the header of each CMORizing script. These scripts also serve as template to create new CMORizers for datasets not yet included. Note that datasets CMORized for ESMValTool v1 may not be working with v2, due to the much stronger constraints on metadata set by the iris library.

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 --config_file [CONFIG_FILE] [DATASET_LIST]
```

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 --config_file [CONFIG_FILE] [DATASET_LIST]
```

The path to the raw data to be CMORized must be specified in the *user configuration file* as RAWOBS. Within this path, the data are expected to be organized in subdirectories corresponding to the data tier: Tier2 for freely-available datasets (other than obs4MIPs and ana4mips) and Tier3 for restricted datasets (i.e., dataset which requires a registration to be retrieved or provided upon request to the respective contact or PI). The CMORization follows the [CMIP5 CMOR tables](#) 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 definition given in *config-developer file* for the OBS project:

```
[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), `reanal` (reanalysis data), `ground` (ground observations), `clim` (derived climatologies), `campaign` (aircraft campaign).

At the moment, `esmvaltool data format` supports Python and NCL scripts.

22.2 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
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	cli (AMon)	3	NCL
CDS-SATELLITE-ALBEDO	bdalb (Lmon), bhalb (Lmon)	3	Python
CDS-SATELLITE-LAI-FAPAR	fapar (Lmon), lai (Lmon)	3	Python

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
CDS-SATELLITE-SOIL-MOISTURE	sm (day), sm (Lmon)	3	NCL
CDS-UERRA	sm (E6hr)	3	Python
CDS-XCH4	xch4 (Amon)	3	NCL
CDS-XCO2	xco2 (Amon)	3	NCL
CERES-EBAF	rlut, rlutcs, rsut, rsutcs (Amon)	2	Python
CERES-SYN1deg	rlds, rldscs, rlus, rluscs, rlut, rlutcs, rsds, rsdscs, rsus, rsuscs, rsut, rsutcs (3hr) rlds, rldscs, rlus, rlut, rlutcs, rsds, rsdt, rsus, rsut, rsutcs (Amon)	3	NCL
CLARA-AVHRR	clt, clivi, lwp (Amon)	3	NCL
CLOUDSAT-L2	clw, clivi, lwp (Amon)	3	NCL
CowtanWay	tasa (Amon)	2	Python
CRU	tas, pr (Amon)	2	Python
CT2019	co2s (Amon)	2	Python
Duveiller2018	albDiffTr13	2	Python
E-OBS	tas, tasmin, tasmax, pr, psl (day, Amon)	2	Python
Eppley-VGPM-MODIS	intpp (Omon)	2	Python
ERA5 ¹	clt, evspsbl, evspsblpot, mrro, pr, prsn, ps, psl, ptype, rls, rlds, rlus, rlus ² , rsds, rsns, rsus ^{Page 445, 2} , rsdt, rss, uas, vas, tas, tasmax, tasmin, tdps, ts, tsn (E1hr/Amon), orog (fx)	3	n/a
ERA5-Land ^{Page 445, 1}	pr	3	n/a
ERA-Interim	clivi, clt, clwvi, evspsbl, hur, hus, pr, prsn, prw, ps, psl, rlds, rsds, rsdt, ta, tas, tauu, tauv, ts, ua, uas, va, vas, wap, zg (Amon), ps, rsdt (CFday), clt, pr, prsn, psl, rsds, rss, ta, tas, tasmax, tasmin, uas, va, vas, zg (day), evspsbl, tdps, ts, tsn, rss, tdps (Eday), tsn (Llmon), hfds, tos (Omon), orog, sftlf (fx)	3	Python
ERA-Interim-Land	sm (Lmon)	3	Python
ESACCI-AEROSOL	abs550aer, od550aer, od550aerStderr, od550lt1aer, od870aer, od870aerStderr (aero)	2	NCL
ESACCI-CLOUD	clivi, clt, cltStderr, lwp, rlut, rlutcs, rsut, rsutcs, rsdt, rlus, rsus, rsuscs (Amon)	2	NCL
ESACCI-FIRE	burntArea (Lmon)	2	NCL
ESACCI-LANDCOVER	baresoilFrac, cropFrac, grassFrac, shrubFrac, treeFrac (Lmon)	2	NCL
ESACCI-LST	ts (Amon)	2	Python
ESACCI-OC	chl (Omon)	2	Python
ESACCI-OZONE	toz, tozStderr, tro3prof, tro3profStderr (Amon)	2	NCL
ESACCI-SEA-SURFACE-SALINITY	sos (Omon)	2	Python
ESACCI-SOILMOISTURE	dos, dosStderr, sm, smStderr (Lmon)	2	NCL

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
ESACCI-SST	ts, tsStderr (Amon)	2	NCL
ESACCI-WATERVAPOUR	prw (Amon)	3	Python
ESRL	co2s (Amon)	2	NCL
FLUXCOM	gpp (Lmon)	3	Python
GCP2018	fgco2 (Omon), nbp (Lmon)	2	Python
GCP2020	fgco2 (Omon), nbp (Lmon)	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
GPCC	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
HadISST	sic (Olmon), tos (Omon), ts (Amon)	2	NCL
HALOE	tro3, hus (Amon)	2	NCL
HWSD	cSoil (Lmon), areacella (fx), sftlf (fx)	3	Python
ISCCP-FH	alb, prw, ps, rlds, rlus, rlut, rlutcs, rsds, rsdt, rsus, rsut, rsutcs, tas, ts (Amon)	2	NCL
JMA-TRANSCOM	nbp (Lmon), fgco2 (Omon)	3	Python
Kadow2020	tasa (Amon)	2	Python
LAI3g	lai (Lmon)	3	Python
LandFlux-EVAL	et, etStderr (Lmon)	3	Python
Landschuetzer2016	dpco2, fgco2, spco2 (Omon)	2	Python
MAC-LWP	lwp, lwpStderr (Amon)	3	NCL
MERRA2	sm (Lmon) clt, pr, evspsbl, hfss, hfsls, 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 (Amon)	3	Python
MLS-AURA	hur, hurStderr (day)	3	Python
MODIS	cliwi, clt, clwvi, iwpStderr, lwpStderr (Amon), od550aer (aero)	3	NCL
MSWEP ^{Page 445, 1}	pr	3	n/a
MTE	gpp, gppStderr (Lmon)	3	Python
NCEP	hur, hus, pr, ta, tas, ua, va, wap, zg (Amon) pr, rlut, ua, va (day)	2	NCL
NDP	cVeg (Lmon)	3	Python
NIWA-BS	toz, tozStderr (Amon)	3	NCL
NOAAGlobalTemp	tasa (Amon)	2	Python
NSIDC-0116-[nh sh]	usi, vsi (day)	3	Python
OSI-450-[nh sh]	sic (Olmon), sic (day)	2	Python
PATMOS-x	clt (Amon)	2	NCL
PERSIANN-CDR	pr (Amon), pr (day)	2	Python
PHC	thetao, so	2	Python
PIOMAS	sit (day)	2	Python
REGEN	pr (day, Amon)	2	Python

continues on next page

Table 1 – continued from previous page

Dataset	Variables (MIP)	Tier	Script language
Scripps-CO2-KUM	co2s (Amon)	2	Python
UWisc	clwvi, lwpStderr (Amon)	3	NCL
WFDE5	tas, pr (Amon, day)	2	Python
WOA	thetao, so, tos, sos (Omon) no3, o2, po4, si (Oyr)	2	Python

¹ CMORization is built into ESMValTool through the native6 project, so there is no separate CMORizer script.

² Derived on the fly from down & net radiation.

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). These projects are configured in the [config-developer](#) file.

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 provide a path in the [User configuration file](#) for the `native6` project data and/or the dedicated project used for the native dataset, e.g., `ICON`. Then, in the recipe, they can refer to those projects. For example:

```
datasets:
- {project: native6, dataset: ERA5, type: reanaly, version: '1', tier: 3, start_year: ↵
  ↵1990, end_year: 1990}
- {project: ICON, dataset: ICON, version: 42-0, component: atm, exp: amip, grid: R2B5, ↵
  ↵ensemble: r1i1, var_type: 2d}
```

For project `native6`, more examples can be found in the diagnostics `ERA5_native6` in the recipe [examples/recipe_check_obs.yml](#).

Part VII

Making a recipe or diagnostic

INTRODUCTION

This chapter contains instructions for developing your own recipes and/or diagnostics. It also contains a section describing how to use additional datasets with ESMValTool. While it is possible to use just the ESMValCore package and run any recipes/diagnostics you develop with just this package, it is highly recommended that you consider contributing the work you do back to the ESMValTool community. Among the advantages of contributing to the community are improved visibility of your work and support by the community with making and maintaining your diagnostic. See the *Community* chapter for a guide on how to contribute to the community.

25.1 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](#).

DIAGNOSTIC

26.1 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.

26.2 Functionality

my_little_diagnostic (or whatever the user will call their diagnostic) makes full use of ESMValTool's preprocessor output (both physical 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.

26.3 Example of config dictionary

To be added (use python-style code-block).

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. *Edit your configuration file*
3. *Store your dataset in the right place*
 - 3.1 *Downloader script (optional)*
4. *Create a cmorizer for the dataset*
 - 4.1 *Cmorizer script written in python*
 - 4.2 *Cmorizer script written in NCL*
5. *Run the cmorizing script*
6. *Naming convention of the observational data files*
7. *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](#).

27.1 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/cmip6/Tables/`, differentiated according to the MIP they belong to. The tables are a copy of the PCMDI guidelines. If you find the variable in one of these tables, you can proceed to the next section.

If your variable is not available in the standard CMOR tables, you need to write a custom CMOR table entry for the variable as outlined below and add it to `/esmvalcore/cmor/tables/custom/`.

To create a new custom CMOR table you need to follow these guidelines:

- Provide the `variable_entry`;
- Provide the `modeling_realm`;

- Provide the variable attributes, but leave `standard_name` blank. Necessary variable attributes are: `units`, `cell_methods`, `cell_measures`, `long_name`, `comment`.
- Provide some additional variable attributes. Necessary additional variable attributes are: `dimensions`, `out_name`, `type`. There are also additional variable attributes that can be defined here (see the already available cmorizers).

It is recommended to use an existing custom table as a template, to edit the content and save it as `CMOR_<short_name>.dat`.

27.2 2. Edit your configuration file

Make sure that beside the paths to the model simulations and observations, also the path to raw observational data to be cmorized (`RAWOBS`) is present in your configuration file.

27.3 3. Store your dataset in the right place

The folder `RAWOBS` needs the subdirectories `Tier1`, `Tier2` and `Tier3`. The different tiers describe the different levels of restrictions for downloading (e.g. providing contact information, licence agreements) and using the observations. The unformatted (raw) observations should then be stored then in the appropriate of these three folders.

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 4. *Create a cmorizer for the dataset*).

27.3.1 3.1 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 `RAWOBS` automatically. There are many downloading scripts available in `/esmval-tool/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 --config_file <config-user.yml> <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.

27.4 4. 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.

27.4.1 4.1 Cmorizer script written in python

Find here an example of a cmorizing script, written for the MTE dataset that is available at the MPI for Biogeochemistry in Jena: [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 6. *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 user configuration file, `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. 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.

27.4.2 4.2 Cmorizer script written in NCL

Find here an example of a cmorizing script, written for the ESACCI XCH4 dataset that is available on the Copernicus Climate Data Store: [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.

27.5 5. Run the cmorizing script

The cmorizing script for the given dataset can be run with:

```
esmvaltool data format --config_file <config-user.yml> <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 path given in the configuration file is the path where your cmorized dataset will be stored. The ESMValTool will create a folder with the correct tier information (see Section 2. [Edit your configuration file](#)) if that tier folder is not already available, and then a folder named after the 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_file <config-user.yml> <dataset-name>
```

Note that options from the `esmvaltool data download` and `esmvaltool data format` commands can be passed to the above command.

27.6 6. Naming convention of the observational data files

For the ESMValTool to be able to read the observations from the NetCDF file, the file name needs a very specific structure and order of information parts (very similar to the naming convention for observations in ESMValTool v1.0). The file name will be automatically correctly created if a cmorizing script has been used to create the netCDF file.

The correct structure of an observational data set is defined in `config-developer.yml`, and looks like the following:

```
OBS_[dataset]_[type]_[version]_[mip]_[short_name]_YYYYMM-YYYYMM.nc
```

For the example of the CDS-XCH4 data set, the correct structure of the file name looks then like this:

```
OBS_CDS-XCH4_sat_L3_Amon_xch4_200301-201612.nc
```

The different parts of the name are explained in more detail here:

- OBS: describes what kind of data can be expected in the file, in this case **observations**;
- CDS-XCH4: that is the name of the dataset. It has been named this way for illustration purposes (so that everybody understands it is the xch4 dataset downloaded from the CDS), but a better name would indeed be ESACCI-XCH4 since it is a ESA-CCI dataset;
- sat: describes the source of the data, here we are looking at satellite data (therefore **sat**), could also be **reanaly** for reanalyses;
- L3: describes the version of the dataset;
- Amon: is the information in which **mip** the variable is to be expected, and what kind of temporal resolution it has; here we expect **xch4** to be part of the atmosphere (A) and we have the dataset in a monthly resolution (**mon**);
- xch4: Is the name of the variable. Each observational data file is supposed to only include one variable per file;
- 200301-201612: Is the period the dataset spans with **200301** being the start year and month, and **201612** being the end year and month;

Note: There is a different naming convention for obs4MIPs data (see the exact specifications for the obs4MIPs data file naming convention in the `config-developer.yml` file).

27.7 7. Test the cmorized dataset

To verify that the cmorized data file is indeed correctly formatted, you can run a dedicated test recipe, that does not include any diagnostic, but only reads in the data file and has it processed in the preprocessor. Such a recipe is called `recipes/examples/recipe_check_obs.yml`. You just need to add a diagnostic for your dataset following the existing entries. Only the diagnostic of interest needs to be run, the others should be commented out for testing.

Part VIII

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](#).

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](#).

28.1 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.

28.2 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](#)*.

28.2.1 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

28.2.2 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

28.2.3 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*

28.3 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.

28.4 Code quality

To increase the readability and maintainability of 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.

28.4.1 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 check if your code adheres to the standard, go to the directory where the repository is cloned, e.g. `cd ESMValTool`, and run [prospector](#)

```
prospector esmvaltool/diag_scripts/your_diagnostic/your_script.py
```

In addition to [prospector](#), we also use [flake8](#) to automatically check for obvious bugs and formatting mistakes.

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, [flake8](#) is run by [CircleCI](#), see the section on [Tests](#) for more information.
2. [Codacy](#) is a service that runs [prospector](#) (and other code quality tools) on changed files and reports the results. Click the 'Details' link behind the Codacy check entry and then click 'View more details on Codacy Production' 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.

A pull request should preferably not introduce any new [prospector](#) issues. However, we understand that there is a limit to how much time can be spent on polishing code, so up to 10 new (non-trivial) issues is still an acceptable amount. Formatting issues are considered trivial and need to be addressed. Note that the automatic code quality checks by [prospector](#) are really helpful to improve the quality of your code, but they are not flawless. If you suspect [prospector](#) or [Codacy](#) may be wrong, please ask the [@ESMValGroup/tech-reviewers](#) by commenting on your pull request.

Note that running [prospector](#) locally will give you quicker and sometimes more accurate results than waiting for [Codacy](#).

Most formatting issues in Python code can be fixed automatically by running the commands

```
isort some_file.py
```

to sort the imports in [the standard way](#) using [isort](#) and

```
yapf -i some_file.py
```

to add/remove whitespace as required by the standard using [yapf](#),

```
docformatter -i some_file.py
```

to run [docformatter](#) which helps formatting the docstrings (such as line length, spaces).

28.4.2 NCL

Because there is no standard best practices document for NCL, we use [PEP8](#) for NCL code as well, with some minor adjustments to 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](#).

28.4.3 R

Best practices for R code are described in [The tidyverse style guide](#). We check adherence to this style guide by using [lintr](#) on CircleCI. Please use [styler](#) to automatically format your code according to this style guide. In the future we would also like to make use of [goodpractice](#) to assess the quality of R code.

28.4.4 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.

28.4.5 Any text file

A generic tool to check for common spelling mistakes is [codespell](#).

28.5 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](#).

28.5.1 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 what its capabilities are. For short functions, a one-line docstring is usually sufficient, but more complex functions might require slightly more extensive documentation.

28.5.2 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](https://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. Make sure that your newly added documentation builds without warnings or errors and looks correctly formatted. CircleCI 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.

28.5.3 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](#).

There is a script that generates the navigation menu shown on the left when you view the documentation. This script is called `doc/sphinx/source/gensidebar.py` in the ESMValTool repository and it should be identical to `doc/gensidebar.py` in the ESMValCore repository, or the sidebar will change when navigating from the ESMValTool documentation to the ESMValCore documentation and vice-versa.

28.6 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`.

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.

28.7 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 --format zenodo --outfile .zenodo.json
```

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.

28.8 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 files:

- `environment.yml` contains dependencies that cannot be installed from [PyPI/Julia package registry](#)
- `environment_osx.yml` contains development dependencies for MacOSX. Should be the same as `environment.yml`, but currently without multi language support.
- `esmvaltool/install/Julia/Project.toml` contains Julia dependencies that can be installed from the default Julia package registry
- `setup.py` contains all Python dependencies, regardless of their installation source

Note that packages may have a different name on [conda-forge](#) than on PyPI or CRAN.

Several test jobs on [CircleCI](#) related to the installation of the tool will only run if you change the dependencies. These will be skipped for most pull requests.

When reviewing a pull request where dependencies are added or removed, always check that the changes have been applied in all relevant files.

28.9 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.

MAKING A NEW DIAGNOSTIC OR RECIPE

29.1 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.

29.2 Creating a recipe and diagnostic script(s)

First create a recipe in `esmvaltool/recipes` to define the input data your analysis script needs and optionally preprocessing and other settings. Also create a script in the `esmvaltool/diag_scripts` directory and make sure it is referenced from your recipe. The easiest way to do this is probably to copy the example recipe and diagnostic script and adjust those to your needs.

If you have no preferred programming language yet, Python 3 is highly recommended, because it is most well supported. However, NCL, R, and Julia scripts are also supported.

Good example recipes for the different languages are:

- python: `esmvaltool/recipes/examples/recipe_python.yml`
- R: `esmvaltool/recipes/examples/recipe_r.yml`
- julia: `esmvaltool/recipes/examples/recipe_julia.yml`
- ncl: `esmvaltool/recipes/examples/recipe_ncl.yml`

Good example diagnostics are:

- python: `esmvaltool/diag_scripts/examples/diagnostic.py`
- R: `esmvaltool/diag_scripts/examples/diagnostic.R`
- julia: `esmvaltool/diag_scripts/examples/diagnostic.jl`
- ncl: `esmvaltool/diag_scripts/examples/diagnostic.ncl`

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.

29.3 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 re-use, the recommended way to proceed is to add Python bindings and publish the package on PyPI so it can be installed as a Python dependency. You can then call the functions it provides using a Python diagnostic.

29.4 Recipe and diagnostic documentation

This section describes how to document a recipe. For more general information on writing documentation, see [Documentation](#).

29.4.1 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.

29.4.2 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.

29.4.3 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.

29.5 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 `output_file_type` specified in the [User configuration file](#). 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.

29.6 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. 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.

29.6.1 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.

29.6.2 Provenance items provided by the diagnostic script

For each output file produced by the diagnostic script, ESMValCore supports the following additional information:

- `ancestors` a list of input files used to create the plot.
- `caption` a caption text for the plot

Note that the level of detail is limited, the only valid choices for `ancestors` are files produced by [ancestor tasks](#).

It is also possible to add more information for the implemented diagnostics using the following items:

- `authors` a list of authors
- `references` a list of references, see [Adding references](#) below
- `projects` a list of projects
- `domains` a list of spatial coverage of the dataset
- `plot_types` a list of plot types if the diagnostic created a plot, e.g. error bar
- `statistics` a list of types of the statistic, e.g. anomaly

Arbitrarily named other items are also supported.

Please see the (installed version of the) file `esmvaltool/config-references.yml` for all available information on each item, see [References configuration file](#) for an introduction. In this file, the information is written in the form of `key: value`. Note that we add the keys to the diagnostics. The keys will automatically be replaced by their values in the final provenance records. For example, in the `config-references.yml` there is a category for types of the plots:

```
plot_types:
  errorbar: error bar plot
```

In the diagnostics, we add the key as: `plot_types: [errorbar]` It is also possible to add custom provenance information by adding items to each category in this file.

In order to communicate with the diagnostic script, two interfaces have been defined, which are described in the [ESMValCore documentation](#). Note that for Python and NCL diagnostics much more convenient methods are available than directly reading and writing the interface files. For other languages these are not implemented (yet).

Depending on your preferred programming language for developing a diagnostic, see the instructions and examples below on how to add provenance information:

29.6.3 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)
```

And make use of a `esmvaltool.diag_scripts.shared.ProvenanceLogger` to log provenance:

```
with ProvenanceLogger(cfg) as provenance_logger:
    provenance_logger.log(diagnostic_file, provenance_record)
```

The `diagnostic_file` can be obtained using `esmvaltool.diag_scripts.shared.get_diagnostic_filename`.

The `provenance_record` is a dictionary of provenance items, for example:

```
provenance_record = {
    'ancestors': ancestor_files,
    'authors': [
        'andela_bouwe',
        'righi_mattia',
    ],
    'caption': caption,
    'domains': ['global'],
    'plot_types': ['zonal'],
    'references': [
        'acknow_project',
    ],
    'statistics': ['mean'],
}
```

Have a look at the example Python diagnostic in `esmvaltool/diag_scripts/examples/diagnostic.py` for a complete example.

29.6.4 Recording provenance in an NCL diagnostic script

Always call the `log_provenance` procedure after plotting from your NCL `diag_script`:

```
log_provenance(nc-file, plot_file, caption, statistics, domain, plottype, authors, references,
    ↪ input-files)
```

For example:

```
log_provenance(ncdf_outfile, \
    map@outfile, \
    "Mean of variable: " + var0, \
    "mean", \
    "global", \
    "geo", \
    ("/righi_mattia", "gottschaldt_klaus-dirk"/), \
    ("/acknow_author"/), \
    metadata_att_as_array(info0, "filename"))
```

Have a look at the example NCL diagnostic in `esmvaltool/diag_scripts/examples/diagnostic.ncl` for a complete example.

29.6.5 Recording provenance in a Julia diagnostic script

The provenance information is written in a `diagnostic_provenance.yml` that will be located in `run_dir`. For example a `provenance_record` can be stored in a yaml file as:

```
provenance_file = string(run_dir, "/diagnostic_provenance.yml")

open(provenance_file, "w") do io
    JSON.print(io, provenance_records, 4)
end
```

The `provenance_records` can be defined as a dictionary of provenance items. For example:

```
provenance_records = Dict{<T, Dict{String, Any}}{<T}()

provenance_record = Dict{String, Any}(
    "ancestors" => [input_file],
    "authors" => ["vonhardenberg_jost", "arnone_enrico"],
    "caption" => "Example diagnostic in Julia",
    "domains" => ["global"],
    "projects" => ["crescendo", "c3s-magic"],
    "references" => ["zhangllwcc"],
    "statistics" => ["other"],
)

provenance_records[output_file] = provenance_record
```

Have a look at the example Julia diagnostic in `esmvaltool/diag_scripts/examples/diagnostic.jl` for a complete example.

29.6.6 Recording provenance in an R diagnostic script

The provenance information is written in a `diagnostic_provenance.yml` that will be located in `run_dir`. For example a `provenance_record` can be stored in a yaml file as:

```
provenance_file <- paste0(run_dir, "/", "diagnostic_provenance.yml")
write_yaml(provenance_records, provenance_file)
```

The `provenance_records` can be defined as a list of provenance items. For example:

```
provenance_records <- list()

provenance_record <- list(
    ancestors = input_filenames,
    authors = list("hunter_alasdair", "perez-zanon_nuria"),
    caption = title,
    projects = list("c3s-magic"),
    statistics = list("other"),
)

provenance_records[[output_file]] <- provenance_record
```

29.7 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`.

29.8 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.yaml`, write the following comment below a pull request:

```
@esmvalbot Please run recipe_xyz.yaml
```

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.

29.9 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.

29.9.1 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
- Re-use 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

29.9.2 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

BROKEN 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. In such cases, the *Maintaining a recipe* 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 listed under “broken recipes” in the *Changelog*, 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, the affected recipe and diagnostics will be retired. This means the recipe and diagnostic code are removed from the ESMValTool main branch by the release manager and thus will not be included in future releases. Only 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.

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.

31.1 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](#).

31.2 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 RAWOBS path specified in your `config-user.yml`
2. If available, use the downloading script by running `esmvaltool data download --config_file <config-file> <dataset>`
3. Run the cmorization by running `esmvaltool data format <config-file> <dataset>`
4. Copy the resulting data to the OBS (for CMIP5 compliant data) or OBS6 (for CMIP6 compliant data) path specified in your `config-user.yml`
5. Run `recipes/examples/recipe_check_obs.yml` with the new dataset to check that the data can be used

31.3 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.

31.4 Data availability

Once your pull request has been approved by the reviewers, ask [@remi-kazeroni](#) to add the new dataset to the data pool at DKRZ and CEDA-Jasmin. He is also the person in charge of merging CMORizer pull requests.

31.5 Detailed checklist for reviews

This (non-exhaustive) checklist provides ideas for things to check when reviewing pull requests for new or updated CMORizer scripts.

31.5.1 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`.

31.5.2 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/`.

31.5.3 recipe_check_obs.yml

Check that new dataset has been added to the testing recipe `esmvaltool/recipes/examples/recipe_check_obs.yml`.

31.5.4 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

31.5.5 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

31.5.6 Config file

If present, check config file `<dataset>.yaml` in `esmvaltool/cmorizers/data/cmor_config/` for correctness. Use `yamllint` to check for syntax errors and common mistakes.

31.5.7 Run downloader script

If available, make sure the downloader script is working by running `esmvaltool data download --config_file <config-file> <dataset>`

31.5.8 Run CMORizer

Make sure CMORizer is working by running `esmvaltool data format --config_file <config-file> <dataset>`

31.5.9 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.yaml` for new dataset.

31.5.10 RAW data

Contact person in charge of ESMValTool data pool ([@remi-kazeroni](#)) and request to copy RAW data to RAWOBS/Tier2 (Tier3).

31.5.11 CMORized data

Contact person in charge of ESMValTool data pool ([@remi-kazoni](#)) and request to

- Merge the pull request
- Copy CMORized dataset to OBS/Tier2 (Tier3)
- Set file access rights for new dataset

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.

32.1 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.

32.2 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.

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.

33.1 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.

33.2 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.

33.3 3. Merge

Pull requests are merged by the [@ESMValGroup/esmvaltool-coreteam](#). Specifically, pull requests containing a *CMORizer script* can only be merged by [@remi-kazeroni](#), 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.

The core development team can be contacted by writing [@ESMValGroup/esmvaltool-coreteam](#) in a comment on an issue or pull request on GitHub.

33.4 Frequently asked questions

33.4.1 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.

33.4.2 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.

33.4.3 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.

33.4.4 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.

33.4.5 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.

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.

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](#)).

35.1 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.

35.2 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.

35.3 Convert xml to yaml

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 `xml2yaml` converter is available in `esmvaltool/utis/xml2yaml/`, 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.yaml` 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.

35.4 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.

35.5 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
<code>getenv("ESMValTool_wrk_dir")</code>	<code>config_user_info@work_dir</code>	all .ncl scripts
<code>getenv(ESMValTool_att)</code>	<code>diag_script_info@att</code> or <code>config_user_info@att</code>	all .ncl scripts
<code>xml</code>	<code>yaml</code>	all scripts
<code>var_attr_ref(0)</code>	<code>variable_info@reference_dataset</code>	all .ncl scripts
<code>var_attr_ref(1)</code>	<code>variable_info@alternative_dataset</code>	all .ncl scripts
<code>models</code>	<code>input_file_info</code>	all .ncl scripts
<code>models@name</code>	<code>input_file_info@dataset</code>	all .ncl scripts
<code>verbosity</code>	<code>config_user_info@log_level</code>	all .ncl scripts
<code>isfilepresent_esmval</code>	<code>fileexists</code>	all .ncl scripts
<code>messaging.ncl</code>	<code>logging.ncl</code>	all .ncl scripts
<code>info_output(arg1, arg2, arg3)</code>	<code>log_info(arg1)</code> if <code>arg3=1</code>	all .ncl scripts
<code>info_output(arg1, arg2, arg3)</code>	<code>log_debug(arg1)</code> if <code>arg3>1</code>	all .ncl scripts
<code>verbosity = config_user_info@verbosity</code>	remove this statement	all .ncl scripts
<code>enter_msg(arg1, arg2, arg3)</code>	<code>enter_msg(arg1, arg2)</code>	all .ncl scripts
<code>leave_msg(arg1, arg2, arg3)</code>	<code>leave_msg(arg1, arg2)</code>	all .ncl scripts
<code>noop()</code>	appropriate if-else statement	all .ncl scripts
<code>nooperation()</code>	appropriate if-else statement	all .ncl scripts
<code>fullpaths</code>	<code>input_file_info@filename</code>	all .ncl scripts
<code>get_output_dir(arg1, arg2)</code>	<code>config_user_info@plot_dir</code>	all .ncl scripts
<code>get_work_dir</code>	<code>config_user_info@work_dir</code>	all .ncl scripts
<code>inlist(arg1, arg2)</code>	<code>any(arg1.eq.arg2)</code>	all .ncl scripts
<code>load interface_scripts/*.ncl</code>	<code>load \$diag_scripts/./interface_scripts/interface.ncl</code>	all .ncl scripts
<code><varname>_info.tmp</code>	<code><varname>_info.ncl</code> in preproc dir	all .ncl scripts

continues on next page

Table 1 – continued from previous page

Name in v1.0	Name in v2.0	Affected code
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.

35.6 Move preprocessing from the diagnostic script to the backend

Many operations previously performed by the diagnostic scripts, are now included in the backend, including level extraction, regridding, masking, and multi-model statistics. If the diagnostics to be ported contains code performing any of such operations, the corresponding code has to be removed from the diagnostic script and the respective backend functionality can be used instead.

The backend operations are fully controlled by the `preprocessors` section in the recipe. Here, a number of preprocessor sets can be defined, with different options for each of the operations. The sets defined in this section are applied in the `diagnostics` section to preprocess a given variable.

It is recommended to proceed step by step, porting and testing each operation separately before proceeding with the next one. A useful setting in the user configuration file (`config-private.yml`) called `write_intermediary_cube` allows writing out the variable field after each preprocessing step, thus facilitating the comparison with the old version (e.g., after CMORization, level selection, after regridding, etc.). The CMORization step of the new backend exactly corresponds to the operation performed by the old backend (and stored in the `climo` directory, now called `preprec`): this is the very first step to be checked, by simply comparing the intermediary file produced by the new backend after CMORization with the output of the old backend in the `climo` directory (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>].

35.7 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.

35.8 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).

35.9 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.

35.10 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.

35.11 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.

35.12 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.

GITHUB WORKFLOW

36.1 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>).

36.2 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 accidentally 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.

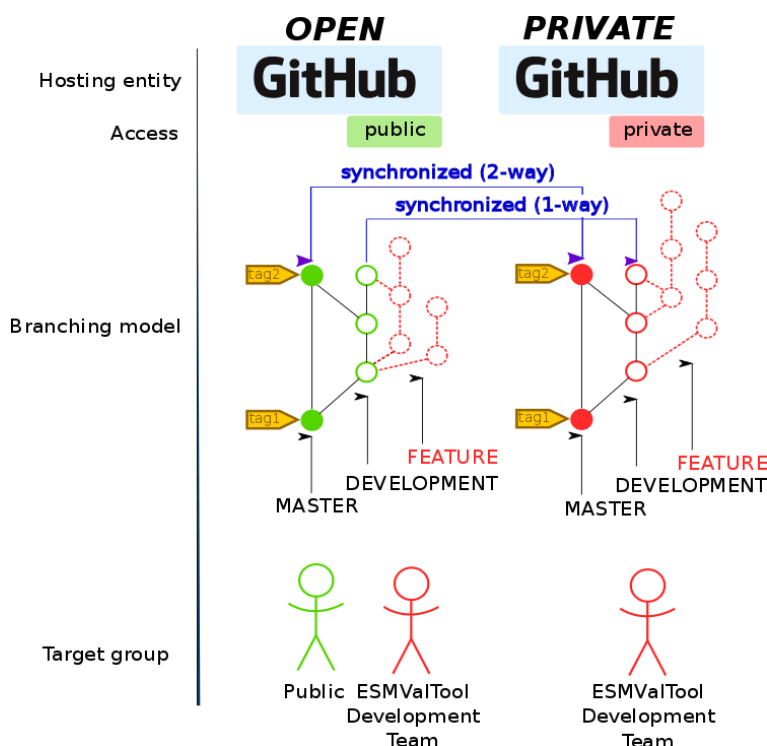


Fig. 1: Schematic diagram of the ESMValTool GitHub repositories.

36.3 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.

36.3.1 Getting started

First make sure that you have Git installed on your development machine. On shared machines, software is usually installed using the environment modules. Try e.g.

```
module avail git
```

if this is the case. You can ask your system administrator for assistance. You can test this with the command:

```
git --version
```

In order to properly identify your contributions to the ESMValTool you need to configure your local Git with some personal data. This can be done with the following commands:

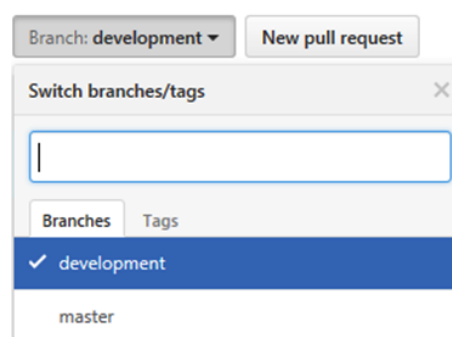
```
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL"
```

Note: For working on GitHub you need to create an account and login to <https://github.com/>.

36.3.2 Working with the ESMValTool GitHub Repositories

As a member of the ESMValTool development team you can create *FEATURE BRANCHES* in the PUBLIC as well as in the PRIVATE repository. We encourage all ESMValTool developers to use the following workflow for long-lived developments (>2 weeks).

- Login to GitHub.com
- On GitHub, go to the website of the ESMValTool repository (<https://github.com/ESMValGroup/ESMValTool-private> or <https://github.com/ESMValGroup/ESMValTool>)
- Click on the button create *FEATURE BRANCH*
- Select the “*DEVELOPMENT*” *BRANCH* and create a new *FEATURE BRANCH* for the diagnostic/feature you want to implement. Please follow the following naming convention for your new *FEATURE BRANCH*: <Project>_<myfeature>.



- Click the button “Clone or Download” and copy the URL shown there
- Open a terminal window and go to the folder where you would like to store your local copy of the ESMValTool source
- Type git clone, and paste the URL:

```
git clone <URL_FROM_CLIPBOARD>
```

This will clone the ESMValTool repository at GitHub to a local folder. You can now query the status of your local working copy with:

```
git status
```

You will see that you are on a branch called 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/>

36.3.3 Pull requests

Once your development is completely finished, go to the GitHub website of the ESMValTool repository and switch to your *FEATURE BRANCH*. You can then initiate a pull request by clicking on the button “New pull request”. Select the *DEVELOPMENT BRANCH* as “base branch” and click on “Create pull request”. Your pull request will then be tested, discussed and implemented into the *DEVELOPMENT 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.

36.3.4 GitHub issues

In case you encounter a bug or if you have a feature request or something similar you can open an issue on the PUBLIC ESMValTool GitHub repository.

36.4 General do-s and don't-s

36.4.1 Do-s

- Create a *FEATURE BRANCH* and use exclusively this branch for developing the ESMValTool. The naming convention for *FEATURE BRANCHES* is <Project>_<myfeature>.
- Comment your code as much as possible and in English.
- Use short but self-explanatory variable names (e.g., `model_input` and `reference_input` instead of `xm` and `xr`).
- Consider a modular/functional programming style. This often makes code easier to read and deletes intermediate variables immediately. If possible, separate diagnostic calculations from plotting routines.
- Consider reusing or extending existing code. General-purpose code can be found in `esmval-tool/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.

36.4.2 Don't-s

- Do not use other programming languages than the ones currently supported (Python, R, NCL, Julia). If you are unsure what language to use, Python is probably the best choice, because it has very good libraries available and is supported by a large community. Contact the [ESMValTool Core Development Team](#) if you wish to use another language, but remember that only open-source languages are supported by the ESMValTool.
- Do not develop without proper version control (see do-s above).
- Avoid large (memory, disk space) intermediate results. Delete intermediate files/variables or see modular/functional programming style.
- Do not use hard-coded pathnames or filenames.
- Do not mix developments / modifications of the ESMValTool framework and developments / modifications of diagnostics in the same *FEATURE BRANCH*.

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:

37.1 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
```

37.2 2. Make a branch to develop your recipe and diagnostic

```
git checkout main
git pull
git checkout -b my-awesome-diagnostic
```

37.3 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
```

37.4 4. Write and submit your paper

37.5 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
```

37.6 6. Make a pull request in the public repository

Go to <https://github.com/esmvalgroup/esmvaltool/pulls> and click the 'New pull request button'. Process reviewer comments and get it merged as described in *Review of pull requests*.

37.7 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>

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](#).

38.1 Overall Procedure

38.1.1 Timeline

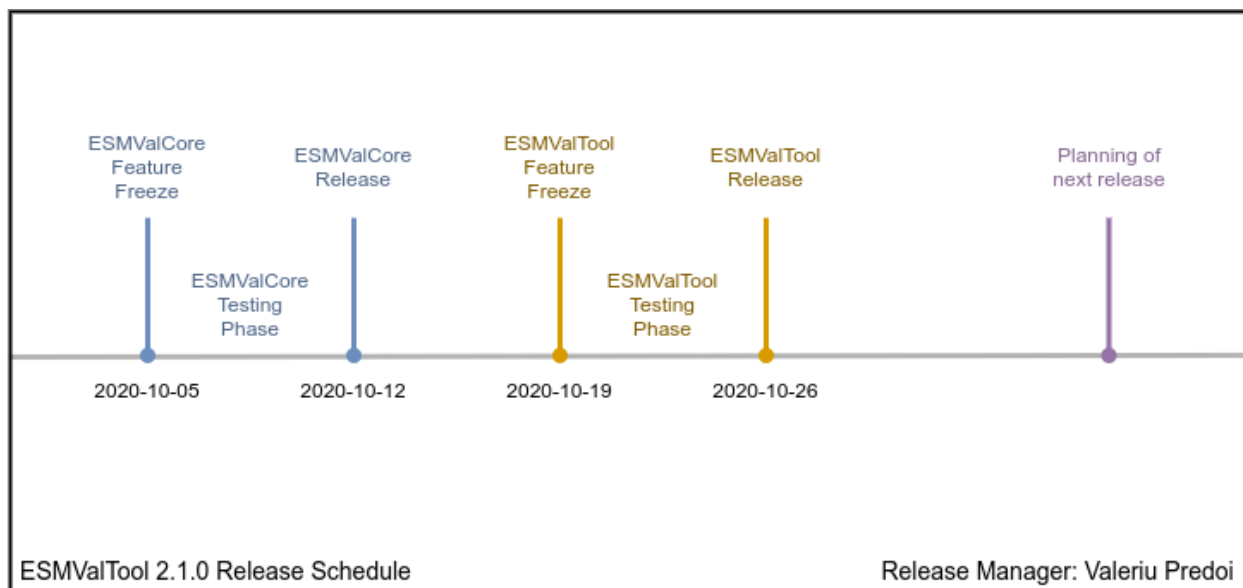


Fig. 1: Example of a Release Timeline (in this case for 2.1.0)

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
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

38.1.2 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

- 2.10.0 (Release Manager: TBA)

2023-10-02	ESMValCore feature freeze
2023-10-09	ESMValCore release
2023-10-16	ESMValTool feature freeze
2023-10-23	ESMValTool release

- 2.9.0 (Release Manager: TBA)

2023-06-05	ESMValCore feature freeze
2023-06-12	ESMValCore release
2023-06-19	ESMValTool feature freeze
2023-06-26	ESMValTool release

- 2.8.0 (Release Manager: TBA)

2023-02-06	ESMValCore feature freeze
2023-02-13	ESMValCore release
2023-02-20	ESMValTool feature freeze
2023-02-27	ESMValTool release

- 2.7.0 (Release Manager: [Valeriu Predoi](#))

2022-10-03	ESMValCore feature freeze
2022-10-10	ESMValCore release
2022-10-17	ESMValTool feature freeze
2022-10-24	ESMValTool release

Past releases

- 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

38.1.3 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

- 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](#).
- 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.
- Make the official ESMValCore release with the last release candidate.
- Ask the user engagement team to announce the release to the user mailing list, the development team mailing list, on twitter

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.

5. ESMValTool release

- Make the release by following *[How to make an ESMValTool release](#)*
- Ask the user engagement team to announce the release to the user mailing list, the development team mailing list, and on twitter

6. 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

38.2 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.

38.2.1 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.

38.3 Glossary

38.3.1 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.

38.3.2 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.

38.3.3 Release manager

The person in charge of making the release, both technically and organizationally. Appointed for a single release.

38.3.4 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.

38.4 How to make an ESMValTool release

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:

38.4.1 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).

38.4.2 2. Increase the version number

The version number is automatically generated from the information provided by git using [setuptools-scm](<https://pypi.org/project/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`.

38.4.3 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 backwards incompatible changes that the release may include. 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`.

38.4.4 4. Create a release branch

Create a branch off the `main` branch and push it to GitHub. 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.

38.4.5 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`.

38.4.6 6. Create and upload the PyPI package

The package is automatically uploaded to the [PyPI](#) by a GitHub action. If has failed for some reason, build and upload the package manually by following the instructions below.

Follow these steps to create a new Python package:

- 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 pep517 twine`
- Build the package: `python3 -m pep517.build --source --binary --out-dir dist/`. 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](#).

38.4.7 7. 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.

38.4.8 8. 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, R, and Julia,
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.

38.5 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

Part IX

Utilities

This section provides information on tools that are useful when developing ESMValTool. Tools that are specific to ESMValTool live in the [esmvaltool/utls](#) directory, while others can be installed using the usual package managers.

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 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`.

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
```


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.

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.

42.1 Using *cylc*

A *cylc* suite for running all recipes is available in [esmvaltool/utils/testing/regression](#)

To prepare for using this tool:

1. Log in to Mistral or another system that uses [slurm](#)
2. Make sure the required CMIP and observational datasets are available and configured in `config-user.yml`
3. Make sure the required auxiliary data is available (see [recipe documentation](#))
4. Install ESMValTool
5. Update `config-user.yml` so it points to the right data locations

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`.
8. Create the `index.html` overview page by running `python esmvaltool/utils/testing/regression/ summarize.py ~/esmvaltool_output/`

42.2 Using Rose and cylc

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

42.2.1 Base suite

The base suite to run `esmvaltool` via *rose-cylc* is *u-bd684*; you can find this suite in the Met Office Rose repository at: <https://code.metoffice.gov.uk/svn/roses-u/b/d/6/8/4/trunk/>

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

42.2.2 Environment

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

`u-bd684/app/esmvaltool/env_setup`

with the correct pointers to `esmvaltool` installation, if desired.

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

42.2.3 Jasmin-example

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

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

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

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

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.

To use it, first install the package `imagehash`:

```
pip install imagehash
```

Next, go to the location where ESMValTool is installed and run

```
python esmvaltool/utils/testing/regression/compare.py ~/reference_output/ ~/output/  
↪ recipe_python_20220310_180417/
```

where the first argument is a reference run or a directory containing such runs and the second and following arguments are directories with runs to compare to the reference run(s).

To compare all results from the current version to the previous version, use 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, run the tool with the `--verbose` flag.

TESTING RECIPE SETTINGS

A tool for generating recipes with various diagnostic settings, to test if those work. Install ESMValTool in development mode (`pip install -e '[develop]'`) to make it available. To use it, run

```
test_recipe --help
```


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.

CONVERTING VERSION 1 NAMELISTS TO VERSION 2 RECIPES

The [xml2yaml](#) converter can turn the old xml namelists into new-style yaml 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 `xml2yaml.xsl` to produce a new yaml recipe.

After the conversion you need to manually check the mip information in the variables! Also, check the caveats below!

46.1 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:xml2yaml.xsl -s:namelist.xml -o:recipe.yaml
```

46.2 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

RECIPE FILLER

If you need to fill in a blank recipe with additional datasets, you can do that with the command *recipe_filler*. This runs a tool to obtain a set of additional datasets when given a blank recipe, and you can give an arbitrary number of data parameters. The blank recipe should contain, to the very least, a list of diagnostics, each with their variable(s). Example of running the tool:

```
recipe_filler recipe.yml
```

where *recipe.yml* is the recipe that needs to be filled with additional datasets; a minimal example of this recipe could be:

```
diagnostics:
  diagnostic:
    variables:
      ta:
        mip: Amon # required
        start_year: 1850 # required
        end_year: 1900 # required
```

47.1 Key features

- you can add as many variable parameters as are needed; if not added, the tool will use the "*" wildcard and find all available combinations;
- you can restrict the number of datasets to be looked for with the `dataset:` key for each variable, pass a list of datasets as value, e.g. `dataset: [MPI-ESM1-2-LR, MPI-ESM-LR];`
- you can specify a pair of experiments, e.g. `exp: [historical, rcp85]` for each variable; this will look for each available dataset per experiment and assemble an aggregated data stretch from each experiment to complete for the total data length specified by `start_year` and `end_year`; equivalent to ESMValTool's syntax on multiple experiments; this option needs an ensemble to be declared explicitly; it will return no entry if there are gaps in data;
- `start_year` and `end_year` are required and are used to filter out the datasets that don't have data in the interval; as noted above, the tool will not return datasets with partial coverage from `start_year` to `end_year`; if you want all possible years hence no filtering on years just use "*" for start and end years;
- `config-user: rootpath:` CMIPX may be a list, rootpath lists are supported;
- all major DRS paths (including default, BADC, ETHZ etc) are supported;
- speedup is achieved through CMIP mip tables lookup, so `mip` is required in recipe;

47.2 Caveats

- the tool doesn't yet work with derived variables; it will not return any available datasets;
- operation restricted to CMIP data only, OBS lookup is not available yet.

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`.

Part X

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.

SHARED DIAGNOSTIC CODE

49.1 Shared diagnostic script code

Code that is shared between multiple diagnostic scripts.

Classes:

<i>Datasets</i> (cfg)	Class to easily access a recipe's datasets in a diagnostic script.
<i>ProvenanceLogger</i> (cfg)	Open the provenance logger.
<i>Variable</i> (short_name, standard_name, ...)	Variable class containing all relevant information.
<i>Variables</i> ([cfg])	Class to easily access a recipe's variables in a diagnostic.

Functions:

<i>apply_supermeans</i> (ctrl, exper, obs_list)	Apply supermeans on data components ie MEAN on time.
<i>extract_variables</i> (cfg[, as_iris])	Extract basic variable information from configuration dictionary.
<i>get_cfg</i> ([filename])	Read diagnostic script configuration from settings.yml.
<i>get_control_exper_obs</i> (short_name, ...)	Get control, exper and obs datasets.
<i>get_diagnostic_filename</i> (basename, cfg[, ...])	Get a valid path for saving a diagnostic data file.
<i>get_plot_filename</i> (basename, cfg)	Get a valid path for saving a diagnostic plot.
<i>group_metadata</i> (metadata, attribute[, sort])	Group metadata describing preprocessed data by attribute.
<i>run_diagnostic</i> ()	Run a Python diagnostic.
<i>save_data</i> (basename, provenance, cfg, cube, ...)	Save the data used to create a plot to file.
<i>save_figure</i> (basename, provenance, cfg[, ...])	Save a figure to file.
<i>select_metadata</i> (metadata, **attributes)	Select specific metadata describing preprocessed data.
<i>sorted_group_metadata</i> (metadata_groups, sort)	Sort grouped metadata.
<i>sorted_metadata</i> (metadata, sort)	Sort a list of metadata describing preprocessed data.
<i>variables_available</i> (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:

<code>add_dataset(path[, data])</code>	Add dataset to class.
<code>add_to_data(data[, path])</code>	Add element to a dataset's data.
<code>get_data([path])</code>	Access a dataset's data.
<code>get_data_list(**dataset_info)</code>	Access the datasets' data in a list.
<code>get_dataset_info([path])</code>	Access a dataset's information.
<code>get_dataset_info_list(**dataset_info)</code>	Access dataset's information in a list.
<code>get_info(key[, path])</code>	Access a 'dataset_info's <i>key</i> .
<code>get_info_list(key, **dataset_info)</code>	Access <i>dataset_info</i> 's <i>key</i> values.
<code>get_path(**dataset_info)</code>	Access a dataset's path.
<code>get_path_list(**dataset_info)</code>	Access dataset's paths in a list.
<code>set_data(data[, path])</code>	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': [
        'author2@journal',
    ],
    '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:

<code>log(filename, record)</code>	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:

<code>count(value, /)</code>	Return number of occurrences of value.
<code>index(value[, start, stop])</code>	Return first index of value.

Attributes:

<code>long_name</code>	Alias for field number 2
<code>short_name</code>	Alias for field number 0
<code>standard_name</code>	Alias for field number 1
<code>units</code>	Alias for field number 3

count(*value, /*)

Return number of occurrences of value.

index(*value, start=0, stop=9223372036854775807, /*)

Return first index of value.

Raises ValueError if the value is not present.

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.

ExamplesGet 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:

<code>add_vars(**names)</code>	Add custom variables to the class.
<code>iris_dict(var)</code>	Access <code>iris</code> dictionary of the variable.
<code>long_name(var)</code>	Access long name.
<code>modify_var(var, **names)</code>	Modify an already existing variable of the class.
<code>short_name(var)</code>	Access short name.
<code>short_names()</code>	Get list of all <i>short_names</i> .
<code>standard_name(var)</code>	Access standard name.
<code>standard_names()</code>	Get list of all <i>standard_names</i> .
<code>units(var)</code>	Access units.
<code>var_name(var)</code>	Access var name.
<code>vars_available(*args)</code>	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_list`: list of dicts for OBS datasets (0, 1 or many)

Returns: control and experiment cubes and list of obs cubes

`esmvaltool.diag_scripts.shared.extract_variables(cfg, as_iris=False)`

Extract basic variable information from configuration dictionary.

Returns *short_name*, *standard_name*, *long_name* and *units* keys for each variable.

Parameters

- **cfg** (*dict*) – Diagnostic script configuration.
- **as_iris** (*bool*, *optional*) – Replace *short_name* by *var_name*, this can be used directly in *iris* classes.

Returns

Variable information in dict`s (values) for each `short_name` (key).

Return type

dict

`esmvaltool.diag_scripts.shared.get_cfg(filename=None)`

Read diagnostic script configuration from settings.yml.

`esmvaltool.diag_scripts.shared.get_control_exper_obs(short_name, input_data, cfg, cmip_type)`

Get control, exper and obs datasets.

This function is used when running recipes that need a clear distinction between a control dataset, an experiment dataset and have optional obs (OBS, obs4MIPs etc) datasets; such recipes include `recipe_validation`, and all the autoassess ones; `short_name`: variable short name `input_data`: dict containing the input data info `cfg`: config file as used in this module

`esmvaltool.diag_scripts.shared.get_diagnostic_filename(basename, cfg, extension='nc')`

Get a valid path for saving a diagnostic data file.

Parameters

- **basename** (*str*) – The basename of the file.
- **cfg** (*dict*) – Dictionary with diagnostic configuration.
- **extension** (*str*) – File name extension.

Returns

A valid path for saving a diagnostic data file.

Return type

str

`esmvaltool.diag_scripts.shared.get_plot_filename(basename, cfg)`

Get a valid path for saving a diagnostic plot.

Parameters

- **basename** (*str*) – The basename of the file.
- **cfg** (*dict*) – Dictionary with diagnostic configuration.

Returns

A valid path for saving a diagnostic plot.

Return type

`str`

`esmvaltool.diag_scripts.shared.group_metadata(metadata, attribute, sort=None)`

Group metadata describing preprocessed data by attribute.

Parameters

- **metadata** (`list of dict`) – A list of metadata describing preprocessed data.
- **attribute** (`str`) – The attribute name that the metadata should be grouped by.
- **sort** – See `sorted_group_metadata`.

Returns

A dictionary containing the requested groups.

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)
```

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

49.1.1 Iris helper functions

Convenience functions for *iris* objects.

Functions:

<code>check_coordinate(cubes, coord_name)</code>	Compare coordinate of cubes and raise error if not identical.
<code>convert_to_iris(dict_)</code>	Change all appearances of <i>short_name</i> to <i>var_name</i> .
<code>get_mean_cube(datasets)</code>	Get mean cube of a list of datasets.
<code>intersect_dataset_coordinates(cubes)</code>	Compare dataset coordinates of cubes and match them if necessary.
<code>iris_project_constraint(projects, input_data)</code>	Create <i>iris.Constraint</i> to select specific projects from data.
<code>prepare_cube_for_merging(cube, cube_label)</code>	Prepare single <i>iris.cube.Cube</i> in order to merge it later.
<code>unify_1d_cubes(cubes, coord_name)</code>	Unify 1D cubes by transforming them to identical coordinates.
<code>unify_time_coord(cube[, target_units])</code>	Unify time coordinate of cube in-place.
<code>var_name_constraint(var_name)</code>	<i>iris.Constraint</i> using <i>var_name</i> .

`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`.

`esmvaltool.diag_scripts.shared.iris_helpers.var_name_constraint(var_name)`
`iris.Constraint` using `var_name`.

Warning: Deprecated since version 2.6.0: This function has been deprecated in ESMValTool version 2.6.0 and is scheduled for removal in version 2.8.0. Please use the function `iris.NameConstraint` with the argument `var_name` instead: this is an exact replacement.

Parameters

var_name (*str*) – `var_name` used for the constraint.

Returns

Constraint.

Return type

`iris.Constraint`

49.1.2 Plotting

Module that provides common plot functions.

Functions:

<code>get_dataset_style(dataset[, style_file])</code>	Retrieve the style information for the given dataset.
<code>get_path_to_mpl_style([style_file])</code>	Get path to matplotlib style file.
<code>global_contourf(cube[, cbar_center, ...])</code>	Plot global filled contour plot.
<code>global_pcolormesh(cube[, cbar_center, ...])</code>	Plot global color mesh.
<code>multi_dataset_scatterplot(x_data, y_data, ...)</code>	Plot a multi dataset scatterplot.
<code>quickplot(cube, plot_type[, filename])</code>	Plot a cube using one of the <code>iris.quickplot</code> functions.
<code>scatterplot(x_data, y_data, filepath, **kwargs)</code>	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.

DIAGNOSTIC SCRIPTS

50.1 Emergent constraints diagnostics

This module provides various tools to evaluate emergent constraints for arbitrary input variables.

50.1.1 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?*

50.1.2 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()` (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()` (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()` (affects all plots).

50.1.3 Auxiliary scripts

Auxiliary functions for emergent constraints scripts

Convenience functions for emergent constraints diagnostics.

Functions:

<code>cdf(data, pdf)</code>	Calculate cumulative distribution function for a 1-dimensional PDF.
<code>check_metadata(metadata[, allowed_var_types])</code>	Check metadata.
<code>combine_groups(groups)</code>	Combine <code>list</code> of groups to a single <code>str</code> .
<code>constraint_info_array(x_data, y_data, ...[, ...])</code>	Get array with all relevant parameters of emergent constraint.
<code>create_simple_scatterplot(x_data, y_data, ...)</code>	Create simple scatterplot of an emergent relationship (without saving).
<code>export_csv(data_frame, attributes, basename, cfg)</code>	Export CSV file.
<code>get_caption(attributes, feature, label[, group])</code>	Construct caption from plotting attributes for (feature, label) pair.
<code>get_colors(cfg[, groups])</code>	Get color palette.
<code>get_constraint(x_data, y_data, obs_mean, obs_std)</code>	Get constraint on target variable.
<code>get_constraint_from_df(training_data, ...[, ...])</code>	Get constraint on target variable from <code>pandas.DataFrame</code> .
<code>get_groups(training_data[, add_combined_group])</code>	Extract groups from training data.
<code>get_input_data(cfg)</code>	Extract input data.
<code>get_input_files(cfg[, patterns, ignore_patterns])</code>	Get input files.
<code>get_provenance_record(attributes, tags, **kwargs)</code>	Get provenance record.
<code>get_xy_data_without_nans(data_frame, ...)</code>	Get (X, Y) data for (feature, label) combination without nans.
<code>pandas_object_to_cube(pandas_object[, ...])</code>	Convert pandas object to <code>iris.cube.Cube</code> .
<code>plot_individual_scatterplots(training_data, ...)</code>	Plot individual scatterplots for the different groups.
<code>plot_merged_scatterplots(training_data, ...)</code>	Plot merged scatterplots (all groups in one plot).
<code>plot_target_distributions(training_data, ...)</code>	Plot distributions of target variable for every feature.
<code>regression_line(x_data, y_data[, n_points])</code>	Return x and y coordinates of the regression line (mean and error).
<code>set_plot_appearance(axes, attributes, **kwargs)</code>	Set appearance of a plot.
<code>standard_prediction_error(x_data, y_data)</code>	Return a function to calculate standard prediction error.
<code>target_pdf(x_data, y_data, obs_mean, obs_std)</code>	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)) – Confidence 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)) – Confidence 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*

50.2 Machine Learning Regression (MLR) diagnostics

This module provides various tools to create and evaluate MLR models for arbitrary input variables.

50.2.1 Examples

- *Constraining uncertainty in projected gross primary production (GPP) with machine learning*: Use Gradient Boosted Regression Tree (GBRT) algorithm to constrain projected Gross Primary Production (GPP) in RCP 8.5 scenario using observations of process-based predictors.

50.2.2 Diagnostic scripts

Evaluate residuals

Simple evaluation of residuals (coming from MLR model output).

Description

This diagnostic evaluates residuals created by MLR models.

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict` s of metadata.

mse_plot: dict, optional

Additional options for plotting the mean square errors (MSE). Specify additional keyword arguments for `seaborn.boxplot()` by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`).

pattern: str, optional

Pattern matched against ancestor file names.

rmse_plot: dict, optional

Additional options for plotting the root mean square errors (RMSE). Specify additional keyword arguments for `seaborn.boxplot()` by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`).

savefig_kwargs: dict, optional

Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict, optional

Options for `seaborn.set()` (affects all plots).

weighted_samples: dict

If specified, use weighted root mean square error. The given keyword arguments are directly passed to `esmvaltool.diag_scripts.mlr.get_all_weights()` to calculate the sample weights. By default, area weights and time weights are used.

MLR main diagnostic

Main Diagnostic script to create MLR models.

Description

This diagnostic script creates Machine Learning Regression (MLR) models which use inter-model relations between process-based predictors (usually from the past/present climate) and a target variable (usually a projection of the future climate) to get a constrained prediction of the target variable. It provides an interface for using MLR models (subclasses of `esmvaltool.diag_scripts.mlr.models.MLRModel`).

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

efecv_kwargs: dict, optional

If specified, use these additional keyword arguments to perform a exhaustive feature elimination using cross-validation. May not be used together with `grid_search_cv_param_grid` or `rfecv_kwargs`.

grid_search_cv_kwargs: dict, optional

Keyword arguments for the grid search cross-validation, see https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

grid_search_cv_param_grid: dict or list of dict, optional

If specified, perform exhaustive parameter search using cross-validation instead of simply calling `esmvaltool.diag_scripts.mlr.models.MLRModel.fit()`. Contains parameters (keys) and ranges (values) for the exhaustive parameter search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`. May not be used together with `efecv_kwargs` or `rfecv_kwargs`.

group_metadata: str, optional

Group input data by an attribute. For every group element (set of datasets), an individual MLR model is calculated. Only affects feature and label datasets. May be used together with the option `pseudo_reality`.

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict` s of metadata.

mlr_model_type: str

MLR model type. The given model has to be defined in `esmvaltool.diag_scripts.mlr.models`.

only_predict: bool, optional (default: False)

If True, only use `esmvaltool.diag_scripts.mlr.models.MLRModel.predict()` and do not create any other output (CSV files, plots, etc.).

pattern: str, optional

Pattern matched against ancestor file names.

plot_partial_dependences: bool, optional (default: False)

Plot partial dependence of every feature in MLR model (computationally expensive).

predict_kwargs: dict, optional

Optional keyword arguments for the final regressor's `predict()` function.

pseudo_reality: list of str, optional

List of dataset attributes which are used to group input data for a pseudo- reality test (also known as *model-as-truth* or *perfect-model* setup). For every element of the group a single MLR model is fitted on all data **except** for that of the specified group element. This group element is then used as additional `prediction_input` and `prediction_reference`. This allows a direct assessment of the predictive power of the MLR model by comparing the MLR prediction output and the true labels (similar to splitting the input data in a training and test set, but not dividing the data randomly but using specific datasets, e.g. the different climate models). May be used together with the option `group_metadata`.

rfecv_kwargs: dict, optional

If specified, use these additional keyword arguments to perform a recursive feature elimination using cross-validation, see https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html. May not be used together with `efecv_kwargs` or `grid_search_cv_param_grid`.

save_mlr_model_error: str or int, optional

Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`!= 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

save_lime_importance: bool, optional (default: False)

Additionally save local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).

save_propagated_errors: bool, optional (default: False)

Additionally save propagated errors from `prediction_input_error` datasets.

select_metadata: dict, optional

Pre-select input data by specifying (key, value) pairs. Affects all datasets regardless of `var_type`.

Additional optional parameters are optional parameters for `esmvaltool.diag_scripts.mlr.models.MLRModel` given [here](#) or optional parameters of `esmvaltool.diag_scripts.mlr.mmm` if `mlr_model_type` is `'mmm'`.

Multi-model means (MMM)

Use simple multi-model mean for predictions.

Description

This diagnostic calculates the (unweighted) mean over all given datasets for a given target variable.

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

convert_units_to: str, optional

Convert units of the input data. Can also be given as dataset option.

dtype: str (default: 'float64')

Internal data type which is used for all calculations, see <https://docs.scipy.org/doc/numpy/user/basics.types.html> for a list of allowed values.

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict` s of metadata.

mlr_model_name: str, optional (default: 'MMM')

Human-readable name of the MLR model instance (e.g used for labels).

mmm_error_type: str, optional

If given, additionally saves estimated squared MMM model error. If the option is set to 'loo', the (constant) error is estimated as RMSEP using leave-one-out cross-validation. No other options are supported at the moment.

pattern: str, optional

Pattern matched against ancestor file names.

prediction_name: str, optional

Default prediction_name of output cubes if no 'prediction_reference' dataset is given.

weighted_samples: dict

If specified, use weighted mean square error to estimate prediction error. The given keyword arguments are directly passed to `esmvaltool.diag_scripts.mlr.get_all_weights()` to calculate the sample weights. By default, area weights and time weights are used.

Plotting functionalities

Plotting scripts for MLR models input/output.

Description

This diagnostic creates plots for MLR model input/output.

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Notes

All configuration options starting with `plot_` specify keyword arguments for a specific plot type. A certain plot type is only plotted if the corresponding option is given in the recipe (if no additional keyword arguments are desired, use `{}`).

Configuration options in recipe

additional_plot_kwargs_xy_plots: dict, optional

Optional keyword arguments (values) for single datasets used in X-Y plots. They keys may include a `var_type` or values of the attribute given by `group_by_attribute`.

alias: dict, optional

`str` to `str` mapping for nicer plot labels (e.g. `{'feature': 'Historical CMIP5 data'}`).

apply_common_mask: bool, optional (default: False)

Apply common mask to all datasets prior to plotting. Requires identical shapes for all datasets.

group_attribute_as_default_alias: bool, optional (default: True)

If `True`, use value of attribute given by `group_by_attribute` as default alias if possible. If `False`, use full group name (including `var_type`) as default alias.

group_by_attribute: str, optional (default: 'mlr_model_name')

By default, datasets are grouped using the `var_type` attribute. This option can be used to specify a further attribute to group datasets. This diagnostic expects a single dataset per group.

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict`s of metadata.

legend_kwargs: dict, optional

Optional keyword arguments of `matplotlib.pyplot.legend()` (affects only plots with legends).

map_plot_type: str, optional (default: 'pcolormesh')

Type of plot used for plotting maps. Must be one of `'pcolormesh'` or `'contourf'`.

pattern: str, optional

Pattern matched against ancestor file names.

plot_map: dict, optional

Specify additional keyword arguments for plotting global maps showing datasets by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`).

plot_map_abs_biases: dict, optional

Specify additional keyword arguments for plotting global maps showing absolute biases by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`).

plot_map_ratios: dict, optional

Specify additional keyword arguments for plotting global maps showing ratios of datasets by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`).

plot_map_rel_biases: dict, optional

Specify additional keyword arguments for plotting global maps showing relative biases of datasets by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`).

plot_xy: dict, optional

Specify additional keyword arguments for simple X-Y plots by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`). By default, plots data against dimensional coordinate (if available). Use `x_coord` (`str`) to use another coordinate as X-axis. Use `reg_line: True` to additionally plot a linear regression line.

plot_xy_with_errors: dict, optional

Specify additional keyword arguments for X-Y plots with error ranges `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`). By default, plots data against dimensional coordinate (if available). Use `x_coord` (`str`) to use another coordinate as X-axis.

print_corr: bool, optional (default: False)

Print and save Pearson correlation coefficient between all datasets at the end. Requires identical shapes for all datasets.

savefig_kwargs: dict, optional

Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict, optional

Options for `seaborn.set()` (affects all plots).

years_in_title: bool, optional (default: False)

Print years in default title of plots.

Postprocessing functionalities

Simple postprocessing of MLR model output.

Description

This diagnostic performs postprocessing operations for MLR model output (mean and error).

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Notes

Prior to postprocessing, this diagnostic groups input datasets according to `tag` and `prediction_name`. For each group, accepts datasets with three different `var_type` s:

- **prediction_output:** **Exactly one** necessary, refers to the mean prediction and serves as reference dataset (regarding shape).
- **prediction_output_error:** Arbitrary number of error datasets. If not given, error calculation is skipped. May be squared errors (marked by the attribute `squared`) or not. In addition, a single covariance dataset can be specified (`short_name` ending with `_cov`).
- **prediction_input:** Dataset used to estimate covariance structure of the mean prediction (i.e. matrix of Pearson correlation coefficients) for error estimation. At most one dataset allowed. Ignored when no `prediction_output_error` is given. This is only possible when (1) the shape of the `prediction_input` dataset is identical to the shape of the `prediction_output_error` datasets, (2) the number of dimensions of the `prediction_input` dataset is higher than the number of dimensions of the `prediction_output_error` datasets and they have identical trailing (rightmost) dimensions or (3) the number of dimensions of the `prediction_input` dataset is higher than the number of dimensions of `prediction_output_error` datasets and all dimensions of the `prediction_output_error` datasets are mapped to a corresponding dimension of the `prediction_input` using the `cov_estimate_dim_map` option (e.g. when `prediction_input` has shape (10, 5, 100, 20) and `prediction_output_error` has shape (5, 20), you can use `cov_estimate_dim_map: [1, 3]` to map the dimensions of `prediction_output_error` to dimension 1 and 3 of `prediction_input`).

All data with other `var_type` s is ignored (`feature`, `label`, etc.).

Real error calculation (using covariance dataset given as `prediction_output_error`) and estimation (using `prediction_input` dataset to estimate covariance structure) is only possible if the mean prediction cube is collapsed completely during postprocessing, i.e. all coordinates are listed for either `mean` or `sum`.

Configuration options in recipe

add_var_from_cov: bool, optional (default: True)

Calculate variances from covariance matrix (diagonal elements) and add those to (squared) error datasets. Set to False if variance is already given separately in prediction output.

area_weighted: bool, optional (default: True)

Calculate weighted averages/sums when collapsing over latitude and/or longitude coordinates using grid cell areas (calculated using grid cell bounds). Only possible for datasets on regular grids that contain `latitude` and `longitude` coordinates.

convert_units_to: str, optional

Convert units of the input data.

cov_estimate_dim_map: list of int, optional

Map dimensions of `prediction_output_error` datasets to corresponding dimensions of `prediction_input` used for estimating covariance. Only relevant if both dataset types are given. See notes above for more information.

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict` s of metadata.

landsea_fraction_weighted: str, optional

When given, calculate weighted averages/sums when collapsing over latitude and/or longitude coordinates using land/sea fraction (calculated using Natural Earth masks). Only possible if the datasets contains `latitude` and `longitude` coordinates. Must be one of 'land', 'sea'.

mean: list of str, optional

Perform mean over the given coordinates.

pattern: str, optional

Pattern matched against ancestor file names.

sum: list of str, optional

Perform sum over the given coordinates.

time_weighted: bool, optional (default: True)

Calculate weighted averages/sums for time (using time bounds).

Preprocessing functionalities

Simple preprocessing of MLR model input.

Description

This diagnostic performs preprocessing operations for datasets used as MLR model input in a desired way. It can also be used to process output of MLR models for plotting.

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

aggregate_by: dict, optional

Aggregate over given coordinates (dict values; given as `list` of `str`) using a desired aggregator (dict key; given as `str`). Allowed aggregators are 'max', 'mean', 'median', 'min', 'sum', 'std', 'var', and 'trend'.

apply_common_mask: bool, optional (default: False)

Apply common mask to all datasets. Requires identical shapes for all datasets.

area_weighted: bool, optional (default: True)

Use weighted aggregation when collapsing over latitude and/or longitude using `collapse`. Weights are estimated using grid cell bounds. Only possible for datasets on regular grids that contain latitude and longitude coordinates.

argsort: dict, optional

Calculate `numpy.ma.argsort()` along given coordinate to get ranking. The coordinate can be specified by the `coord` key. If `descending` is set to `True`, use descending order instead of ascending.

collapse: dict, optional

Collapse over given coordinates (dict values; given as `list` of `str`) using a desired aggregator (dict key; given as `str`). Allowed aggregators are 'max', 'mean', 'median', 'min', 'sum', 'std', 'var', and 'trend'.

convert_units_to: str, optional

Convert units of the input data. Can also be given as dataset option.

extract: dict, optional

Extract certain values (dict values, given as `int`, `float` or iterable of them) for certain coordinates (dict keys, given as `str`).

extract_ignore_bounds: bool, optional (default: False)

If True, ignore coordinate bounds when using `extract` or `extract_range`. If False, consider coordinate bounds when using `extract` or `extract_range`. For time coordinates, bounds are always ignored.

extract_range: dict, optional

Like `extract`, but instead of specific values extract ranges (dict values, given as iterable of exactly two `int`s or `float`s) for certain coordinates (dict keys, given as `str`).

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict`s of metadata.

landsea_fraction_weighted: str, optional

When given, use land/sea fraction for weighted aggregation when collapsing over latitude and/or longitude using collapse. Only possible if the dataset contains `latitude` and `longitude` coordinates and for regular grids. Must be one of `'land'`, `'sea'`.

mask: dict of dict

Mask datasets. Keys have to be `numpy.ma` conversion operations (see <https://docs.scipy.org/doc/numpy/reference/routines.ma.html>) and values all the keyword arguments of them.

n_jobs: int (default: 1)

Maximum number of jobs spawned by this diagnostic script. Use `-1` to use all processors. More details are given [here](#).

normalize_by_mean: bool, optional (default: False)

Remove total mean of the dataset in the last step (resulting mean will be 0.0). Calculates weighted mean if `area_weighted`, `time_weighted` or `landsea_fraction_weighted` are set and the cube contains the corresponding coordinates. Does not apply to error datasets.

normalize_by_std: bool, optional (default: False)

Scale total standard deviation of the dataset in the last step (resulting standard deviation will be 1.0).

output_attributes: dict, optional

Write additional attributes to netcdf files, e.g. `'tag'`.

pattern: str, optional

Pattern matched against ancestor file names.

ref_calculation: str, optional

Perform calculations involving reference dataset. Must be one of `merge` (simply merge two datasets by adding the data of the reference dataset as `iris.coords.AuxCoord` to the original dataset), `add` (add reference dataset), `divide` (divide by reference dataset), `multiply` (multiply with reference dataset), `subtract` (subtract reference dataset) or `trend` (use reference dataset as x axis for calculation of linear trend along a specified axis, see `ref_kwargs`).

ref_kwargs: dict, optional

Keyword arguments for calculations involving reference datasets. Allowed keyword arguments are:

- `matched_by` (`list` of `str`, default: `[]`): Use a given set of attributes to match datasets with their corresponding reference datasets (specified by `ref = True`).
- `collapse_over` (`str`, default: `'time'`): Coordinate which is collapsed. Only relevant when `ref_calculation` is set to `trend`.

return_trend_stderr: bool, optional (default: True)

Return standard error of slope in case of trend calculations (as `var_type prediction_input_error`).

scalar_operations: dict, optional

Operations involving scalars. Allowed keys are add, divide, multiply or subtract. The corresponding values (`float` or `int`) are scalars that are used with the operations.

time_weighted: bool, optional (default: True)

Use weighted aggregation when collapsing over time dimension using collapse. Weights are estimated using time bounds.

unify_coords_to: dict, optional

If given, replace coordinates of all datasets with that of a reference cube (if necessary and possible, broadcast beforehand). The reference dataset is determined by keyword arguments given to this option (keyword arguments must point to exactly one dataset).

Rescale data with emergent constraints

Rescale label data using a single emergent constraint.

Description

This diagnostic uses an emergent relationship between data marked as `var_type=label` (Y axis) and `var_type=feature` (X axis) together with an observation of the X axis (`var_type=prediction_input` and `var_type=prediction_input_error`) to calculate factors that are necessary to rescale each input point so that it matches the constraint. The rescaling is applied to data marked as `var_type=label_to_rescale`. All data needs the attribute tag which needs to be identical for label, `prediction_input`, `prediction_input_error` and `label_to_rescale`. Only a single tag for feature is possible.

Author

Manuel Schlund (DLR, Germany)

Project

CRESCENDO

Configuration options in recipe

group_by_attributes: list of str, optional (default: ['dataset'])

List of attributes used to separate different input points.

ignore: list of dict, optional

Ignore specific datasets by specifying multiple `dict` s of metadata.

legend_kwargs: dict, optional

Optional keyword arguments of `matplotlib.pyplot.legend()` (affects only plots with legends).

pattern: str, optional

Pattern matched against ancestor file names.

plot_emergent_relationship: dict, optional

If given, plot emergent relationship between X and Y data. Specify additional keyword arguments by `plot_kwargs` and plot appearance options by `pyplot_kwargs` (processed as functions of `matplotlib.pyplot`). Use `{}` to plot with default settings.

plot_kwargs_for_groups: dict, optional

Specify additional keyword arguments (values) for the different points defined by `group_by_attributes` (keys) used in plots.

savefig_kwargs: dict, optional

Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict, optional

Options for `seaborn.set()` (affects all plots).

50.2.3 Auxiliary scripts

Auxiliary functions for MLR scripts

Convenience functions for MLR diagnostics.

Functions:

<code>check_predict_kwargs(predict_kwargs)</code>	Check keyword argument for <code>predict()</code> functions.
<code>create_alias(dataset, attributes[, delimiter])</code>	Create alias key of a dataset using a list of attributes.
<code>datasets_have_mlr_attributes(datasets[, ...])</code>	Check (MLR) attributes of datasets.
<code>get_1d_cube(x_data, y_data[, x_kwargs, y_kwargs])</code>	Convert 2 arrays to <code>iris.cube.Cube</code> (with single coordinate).
<code>get_absolute_time_units(units)</code>	Convert time reference units to absolute ones.
<code>get_alias(dataset)</code>	Get alias for dataset.
<code>get_all_weights(cube[, area_weighted, ...])</code>	Get all desired weights for a cube.
<code>get_area_weights(cube[, normalize])</code>	Get area weights calculated from grid cell areas.
<code>get_horizontal_weights(cube[, ...])</code>	Get horizontal (latitude/longitude) weights of cube.
<code>get_input_data(cfg[, pattern, ...])</code>	Get input data and check MLR attributes if desired.
<code>get_landsea_fraction_weights(cube, area_type)</code>	Get land/sea fraction weights calculated from Natural Earth files.
<code>get_new_path(cfg, old_path)</code>	Convert old path to new diagnostic path.
<code>get_squared_error_cube(ref_cube, error_datasets)</code>	Get array of squared errors.
<code>get_time_weights(cube[, normalize])</code>	Get time weights of cube calculated from time bounds.
<code>ignore_warnings()</code>	Ignore warnings given by <code>WARNINGS_TO_IGNORE</code> .
<code>square_root_metadata(cube)</code>	Take the square root of the cube metadata.
<code>units_power(units, power)</code>	Raise a <code>cf.units.Unit</code> to given power preserving symbols.

`esmvaltool.diag_scripts.mlr.check_predict_kwargs(predict_kwargs)`

Check keyword argument for `predict()` functions.

Parameters

predict_kwargs (*keyword arguments, optional*) – Keyword arguments for a `predict()` function.

Raises

RuntimeError – `return_var` and `return_cov` are both set to `True` in the keyword arguments.

`esmvaltool.diag_scripts.mlr.create_alias(dataset, attributes, delimiter='-')`

Create alias key of a dataset using a list of attributes.

Parameters

- **dataset** (*dict*) – Metadata dictionary representing a single dataset.
- **attributes** (*list of str*) – List of attributes used to create the alias.
- **delimiter** (*str, optional (default: '-')*) – Delimiter used to separate different attributes in the alias.

Returns

Dataset alias.

Return type

str

Raises

AttributeError – dataset does not contain one of the attributes.

`esmvaltool.diag_scripts.mlr.datasets_have_mlr_attributes(datasets, log_level='debug', mode='full')`

Check (MLR) attributes of datasets.

Parameters

- **datasets** (*list of dict*) – Datasets to check.
- **log_level** (*str, optional (default: 'debug')*) – Verbosity level of the logger.
- **mode** (*str, optional (default: 'full')*) – Checking mode. Must be one of 'only_missing' (only check if attributes are missing), 'only_var_type' (check only *var_type*) or 'full' (check both).

Returns

True if all required attributes are available, False if not.

Return type

bool

Raises

ValueError – Invalid value for argument mode is given.

`esmvaltool.diag_scripts.mlr.get_1d_cube(x_data, y_data, x_kwargs=None, y_kwargs=None)`

Convert 2 arrays to `iris.cube.Cube` (with single coordinate).

Parameters

- **x_data** (*numpy.ndarray*) – Data for coordinate.
- **y_data** (*numpy.ndarray*) – Data for cube.
- **x_kwargs** (*dict*) – Keyword arguments passed to `iris.coords.AuxCoord`.
- **y_kwargs** (*dict*) – Keyword arguments passed to `iris.cube.Cube`.

Returns

1D cube with single auxiliary coordinate.

Return type

`iris.cube.Cube`

Raises

ValueError – Arrays are not 1D and do not have matching shapes.

`esmvaltool.diag_scripts.mlr.get_absolute_time_units(units)`

Convert time reference units to absolute ones.

This function converts reference time units (like 'days since YYYY') to absolute ones (like 'days').

Parameters

units (*cf_units.Unit*) – Time units to convert.

Returns

Absolute time units.

Return type

cf_units.Unit

Raises

ValueError – If conversion failed (e.g. input units are not time units).

`esmvaltool.diag_scripts.mlr.get_alias(dataset)`

Get alias for dataset.

Parameters

dataset (*dict*) – Dataset metadata.

Returns

Alias.

Return type

str

`esmvaltool.diag_scripts.mlr.get_all_weights(cube, area_weighted=True, time_weighted=True,
landsea_fraction_weighted=None, normalize=False)`

Get all desired weights for a cube.

Parameters

- **cube** (*iris.cube.Cube*) – Input cube.
- **area_weighted** (*bool*, *optional* (*default: True*)) – Use area weights calculated from grid cell areas using `iris.analysis.cartography.area_weights()`. Only works for regular grids.
- **time_weighted** (*bool*, *optional* (*default: True*)) – Use time weights calculated from time bounds.
- **landsea_fraction_weighted** (*str*, *optional*) – If given, use land/sea fraction weights calculated from Natural Earth files. Must be one of 'land', 'sea'. Only works for regular grids.
- **normalize** (*bool*, *optional* (*default: False*)) – Normalize weights with total area and total time range.

Returns

Area weights.

Return type

numpy.ndarray

Raises

- **iris.exceptions.CoordinateMultiDimError** – Dimension of latitude or longitude coordinate is greater than 1.
- **iris.exceptions.CoordinateNotFoundError** – Cube does not contain the coordinates latitude and longitude (if used with `area_weighted` or `landsea_fraction_weighted`) or cube does not contain the coordinate time (if used with `time_weighted`).

- **ValueError** – `landsea_fraction_weighted` is not one of `None`, `'land'`, `'sea'` or coordinates latitude and longitude share dimensions.

`esmvaltool.diag_scripts.mlr.get_area_weights(cube, normalize=False)`

Get area weights calculated from grid cell areas.

Note: Only works for regular grids. Uses `iris.analysis.cartography.area_weights()` for an approximate calculation of the grid cell areas.

Parameters

- **cube** (`iris.cube.Cube`) – Input cube.
- **normalize** (`bool`, optional (default: `False`)) – Normalize weights with total area.

Returns

Area weights.

Return type

`numpy.ndarray`

Raises

`iris.exceptions.CoordinateNotFoundError` – Cube does not contain the coordinates latitude and longitude.

`esmvaltool.diag_scripts.mlr.get_horizontal_weights(cube, area_weighted=True,
landsea_fraction_weighted=None,
normalize=False)`

Get horizontal (latitude/longitude) weights of cube.

Parameters

- **cube** (`iris.cube.Cube`) – Input cube.
- **area_weighted** (`bool`, optional (default: `True`)) – Use area weights calculated from grid cell areas using `iris.analysis.cartography.area_weights()`. Only works for regular grids.
- **landsea_fraction_weighted** (`str`, optional) – If given, use land/sea fraction weights calculated from Natural Earth files. Must be one of `'land'`, `'sea'`. Only works for regular grids.
- **normalize** (`bool`, optional (default: `False`)) – Normalize weights with sum of weights over latitude and longitude (i.e. if only `area_weighted` is given, this is equal to the total area).

Returns

Horizontal (latitude/longitude) weights.

Return type

`numpy.ndarray`

Raises

- **`iris.exceptions.CoordinateMultiDimError`** – Dimension of latitude or longitude coordinate is greater than 1.
- **`iris.exceptions.CoordinateNotFoundError`** – Cube does not contain the coordinates latitude and longitude.

- **ValueError** – `landsea_fraction_weighted` is not one of 'land', 'sea'.

```
esmvaltool.diag_scripts.mlr.get_input_data(cfg, pattern=None, check_mlr_attributes=True,
                                           ignore=None)
```

Get input data and check MLR attributes if desired.

Use `input_data` and `ancestors` to get all relevant input files.

Parameters

- **cfg** (*dict*) – Recipe configuration.
- **pattern** (*str*, *optional*) – Pattern matched against ancestor file names.
- **check_mlr_attributes** (*bool*, *optional* (*default: True*)) – If `True`, only returns datasets with valid MLR attributes. If `False`, returns all found datasets.
- **ignore** (*list of dict*, *optional*) – Ignore specific datasets by specifying multiple `dict`s` of metadata. By setting an attribute to ```None```, ignore all datasets which do not have that attribute.

Returns

List of input datasets.

Return type

`list of dict`

Raises

ValueError – No input data found or at least one dataset has invalid attributes.

```
esmvaltool.diag_scripts.mlr.get_landsea_fraction_weights(cube, area_type, normalize=False)
```

Get land/sea fraction weights calculated from Natural Earth files.

Note: The implementation of this feature is not optimal. For large cubes, calculating the land/sea fraction weights might be very slow. Only works for regular grids.

Parameters

- **cube** (*iris.cube.Cube*) – Input cube.
- **area_type** (*str*) – Area type. Must be one of 'land' (land fraction weighting) or 'sea' (sea fraction weighting).
- **normalize** (*bool*, *optional* (*default: False*)) – Normalize weights with total land/sea fraction.

Raises

- **iris.exceptions.CoordinateMultiDimError** – Dimension of latitude or longitude coordinate is greater than 1.
- **iris.exceptions.CoordinateNotFoundError** – Cube does not contain the coordinates latitude and longitude.
- **ValueError** – `area_type` is not one of 'land', 'sea' or coordinates latitude and longitude share dimensions.

```
esmvaltool.diag_scripts.mlr.get_new_path(cfg, old_path)
```

Convert old path to new diagnostic path.

Parameters

- **cfg** (*dict*) – Recipe configuration.
- **old_path** (*str*) – Old path.

Returns

New diagnostic path.

Return type

str

`esmvaltool.diag_scripts.mlr.get_squared_error_cube(ref_cube, error_datasets)`

Get array of squared errors.

Parameters

- **ref_cube** (*iris.cube.Cube*) – Reference cube (determines mask, coordinates and attributes of output).
- **error_datasets** (*list of dict*) – List of metadata dictionaries where each dictionary represents a single dataset.

Returns

Cube containing squared errors.

Return type

iris.cube.Cube

Raises

ValueError – Shape of a dataset does not match shape of reference cube.

`esmvaltool.diag_scripts.mlr.get_time_weights(cube, normalize=False)`

Get time weights of cube calculated from time bounds.

Parameters

- **cube** (*iris.cube.Cube*) – Input cube.
- **normalize** (*bool, optional (default: False)*) – Normalize weights with total time range.

Returns

Time weights.

Return type

numpy.ndarray

Raises

iris.exceptions.CoordinateNotFoundError – Cube does not contain the coordinate time.

`esmvaltool.diag_scripts.mlr.ignore_warnings()`

Ignore warnings given by WARNINGS_TO_IGNORE.

`esmvaltool.diag_scripts.mlr.square_root_metadata(cube)`

Take the square root of the cube metadata.

Parameters

cube (*iris.cube.Cube*) – Cube (will be modified in-place).

`esmvaltool.diag_scripts.mlr.units_power(units, power)`

Raise a **cf_units.Unit** to given power preserving symbols.

Raise `cf_units.Unit` to given power without expanding it first. For example, using `units_power(Unit('J'), 2)` gives `Unit('J2')`. In contrast, simply using `Unit('J')**2` would yield `'kg2 m4 s-4'`.

Parameters

- **units** (`cf_units.Unit`) – Input units.
- **power** (`int`) – Desired exponent.

Returns

Input units raised to given power.

Return type

`cf_units.Unit`

Raises

- **TypeError** – Argument `power` is not `int`-like.
- **ValueError** – Invalid unit given.

Custom extensions of sklearn functionalities

Custom expansions of `sklearn` functionalities.

Note: This module provides custom expansions of some `sklearn` classes and functions which are necessary to fit the purposes for the desired functionalities of the *MLR module*. As long-term goal we would like to include these functionalities to the `sklearn` package since we believe these additions might be helpful for everyone. This module serves as interim solution. To ensure that all features are properly working this module is also covered by extensive tests.

Parts of this code have been copied from `sklearn`.

License: BSD 3-Clause License

Copyright (c) 2007-2020 The scikit-learn developers. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```
class esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline(steps, *, memory=None,
                                                                verbose=False)
```

Bases: `Pipeline`

Expand `sklearn.pipeline.Pipeline`.

property classes_

The classes labels. Only exist if the last step is a classifier.

property coef_

Model coefficients.

Type

`numpy.ndarray`

decision_function(X)

Transform the data, and apply *decision_function* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *decision_function* method. Only valid if the final estimator implements *decision_function*.

Parameters

X (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.

Returns

y_score – Result of calling *decision_function* on the final estimator.

Return type

ndarray of shape (n_samples, n_classes)

property feature_importances_

Feature importances.

Type

`numpy.ndarray`

property feature_names_in_

Names of features seen during first step *fit* method.

fit(X, y=None, **fit_params)

Fit the model.

Fit all the transformers one after the other and transform the data. Finally, fit the transformed data using the final estimator.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable*, *default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****fit_params** (*dict of string -> object*) – Parameters passed to the *fit* method of each step, where each parameter name is prefixed such that parameter *p* for step *s* has key *s__p*.

Returns

self – Pipeline with fitted steps.

Return type`object`**fit_predict**(*X*, *y=None*, ***fit_params*)

Transform the data, and apply *fit_predict* with the final estimator.

Call *fit_transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *fit_predict* method. Only valid if the final estimator implements *fit_predict*.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable*, *default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****fit_params** (*dict of string -> object*) – Parameters passed to the *fit* method of each step, where each parameter name is prefixed such that parameter *p* for step *s* has key *s__p*.

Returns

y_pred – Result of calling *fit_predict* on the final estimator.

Return type`ndarray`**fit_target_transformer_only**(*y_data*, ***fit_kwargs*)

Fit only transform step of of target regressor.

fit_transform(*X*, *y=None*, ***fit_params*)

Fit the model and transform with the final estimator.

Fits all the transformers one after the other and transform the data. Then uses *fit_transform* on transformed data with the final estimator.

Parameters

- **X** (*iterable*) – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable*, *default=None*) – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****fit_params** (*dict of string -> object*) – Parameters passed to the *fit* method of each step, where each parameter name is prefixed such that parameter *p* for step *s* has key *s__p*.

Returns

Xt – Transformed samples.

Return type`ndarray of shape (n_samples, n_transformed_features)`**fit_transformers_only**(*x_data*, *y_data*, ***fit_kwargs*)

Fit only transform steps of Pipeline.

get_feature_names_out(*input_features=None*)

Get output feature names for transformation.

Transform input features using the pipeline.

Parameters

input_features (array-like of *str* or *None*, *default=None*) – Input features.

Returns

feature_names_out – Transformed feature names.

Return type

ndarray of *str* objects

get_params(*deep=True*)

Get parameters for this estimator.

Returns the parameters given in the constructor as well as the estimators contained within the *steps* of the *Pipeline*.

Parameters

deep (*bool*, *default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

mapping of string to any

inverse_transform(*Xt*)

Apply *inverse_transform* for each step in a reverse order.

All estimators in the pipeline must support *inverse_transform*.

Parameters

Xt (array-like of shape (*n_samples*, *n_transformed_features*)) – Data samples, where *n_samples* is the number of samples and *n_features* is the number of features. Must fulfill input requirements of last step of pipeline's *inverse_transform* method.

Returns

Xt – Inverse transformed data, that is, data in the original feature space.

Return type

ndarray of shape (*n_samples*, *n_features*)

property n_features_in_

Number of features seen during first step *fit* method.

property named_steps

Access the steps by name.

Read-only attribute to access any step by given name. Keys are steps names and values are the steps objects.

predict(*X*, ***predict_params*)

Transform the data, and apply *predict* with the final estimator.

Call *transform* of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls *predict* method. Only valid if the final estimator implements *predict*.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.

- ****predict_params** (*dict of string -> object*) – Parameters to the `predict` called at the end of all transformations in the pipeline. Note that while this may be used to return uncertainties from some models with `return_std` or `return_cov`, uncertainties that are generated by the transformations in the pipeline are not propagated to the final estimator.

New in version 0.20.

Returns

y_pred – Result of calling `predict` on the final estimator.

Return type

ndarray

predict_log_proba(*X*, ****predict_log_proba_params**)

Transform the data, and apply `predict_log_proba` with the final estimator.

Call `transform` of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls `predict_log_proba` method. Only valid if the final estimator implements `predict_log_proba`.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****predict_log_proba_params** (*dict of string -> object*) – Parameters to the `predict_log_proba` called at the end of all transformations in the pipeline.

Returns

y_log_proba – Result of calling `predict_log_proba` on the final estimator.

Return type

ndarray of shape (n_samples, n_classes)

predict_proba(*X*, ****predict_proba_params**)

Transform the data, and apply `predict_proba` with the final estimator.

Call `transform` of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls `predict_proba` method. Only valid if the final estimator implements `predict_proba`.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- ****predict_proba_params** (*dict of string -> object*) – Parameters to the `predict_proba` called at the end of all transformations in the pipeline.

Returns

y_proba – Result of calling `predict_proba` on the final estimator.

Return type

ndarray of shape (n_samples, n_classes)

score(*X*, *y=None*, *sample_weight=None*)

Transform the data, and apply `score` with the final estimator.

Call `transform` of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls `score` method. Only valid if the final estimator implements `score`.

Parameters

- **X** (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.
- **y** (*iterable*, *default=None*) – Targets used for scoring. Must fulfill label requirements for all steps of the pipeline.
- **sample_weight** (*array-like*, *default=None*) – If not None, this argument is passed as `sample_weight` keyword argument to the `score` method of the final estimator.

Returns

score – Result of calling `score` on the final estimator.

Return type

float

`score_samples(X)`

Transform the data, and apply `score_samples` with the final estimator.

Call `transform` of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls `score_samples` method. Only valid if the final estimator implements `score_samples`.

Parameters

X (*iterable*) – Data to predict on. Must fulfill input requirements of first step of the pipeline.

Returns

y_score – Result of calling `score_samples` on the final estimator.

Return type

ndarray of shape (n_samples,)

`set_params(**kwargs)`

Set the parameters of this estimator.

Valid parameter keys can be listed with `get_params()`. Note that you can directly set the parameters of the estimators contained in `steps`.

Parameters

****kwargs** (*dict*) – Parameters of this estimator or parameters of estimators contained in `steps`. Parameters of the steps may be set using its name and the parameter name separated by a `'_'`.

Returns

self – Pipeline class instance.

Return type

object

`steps: List[Any]`

`transform(X)`

Transform the data, and apply `transform` with the final estimator.

Call `transform` of each transformer in the pipeline. The transformed data are finally passed to the final estimator that calls `transform` method. Only valid if the final estimator implements `transform`.

This also works where final estimator is `None` in which case all prior transformations are applied.

Parameters

X (*iterable*) – Data to transform. Must fulfill input requirements of first step of the pipeline.

Returns

Xt – Transformed data.

Return type

ndarray of shape (n_samples, n_transformed_features)

transform_only(*x_data*)

Only perform transform steps of Pipeline.

transform_target_only(*y_data*)

Only perform transform steps of target regressor.

```
class esmvaltool.diag_scripts.mlr.custom_sklern.AdvancedRFE(estimator, *,
                                                            n_features_to_select=None, step=1,
                                                            verbose=0,
                                                            importance_getter='auto')
```

Bases: [RFE](#)

Expand [sklearn.feature_selection.RFE](#).

property classes_

Classes labels available when *estimator* is a classifier.

Return type

ndarray of shape (n_classes,)

decision_function(*X*)

Compute the decision function of *X*.

Parameters

X (*{array-like or sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

Returns

score – The decision function of the input samples. The order of the classes corresponds to that in the attribute `classes_`. Regression and binary classification produce an array of shape `[n_samples]`.

Return type

array, shape = `[n_samples, n_classes]` or `[n_samples]`

fit(*x_data*, *y_data*, ****fit_kwargs**)

Expand [fit\(\)](#) to accept kwargs.

fit_transform(*X*, *y=None*, ****fit_params**)

Fit to data, then transform it.

Fits transformer to *X* and *y* with optional parameters *fit_params* and returns a transformed version of *X*.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Input samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs), default=None*) – Target values (None for unsupervised transformations).
- ****fit_params** (*dict*) – Additional fit parameters.

Returns

X_new – Transformed array.

Return type

ndarray array of shape (n_samples, n_features_new)

get_feature_names_out(*input_features=None*)

Mask feature names according to selected features.

Parameters

input_features (array-like of *str* or *None*, *default=None*) – Input features.

- If *input_features* is *None*, then *feature_names_in_* is used as feature names in. If *feature_names_in_* is not defined, then the following input feature names are generated: `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
- If *input_features* is an array-like, then *input_features* must match *feature_names_in_* if *feature_names_in_* is defined.

Returns

feature_names_out – Transformed feature names.

Return type

ndarray of str objects

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool*, *default=True*) – If *True*, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

dict

get_support(*indices=False*)

Get a mask, or integer index, of the features selected.

Parameters

indices (*bool*, *default=False*) – If *True*, the return value will be an array of integers, rather than a boolean mask.

Returns

support – An index that selects the retained features from a feature vector. If *indices* is *False*, this is a boolean array of shape `[# input features]`, in which an element is *True* iff its corresponding feature is selected for retention. If *indices* is *True*, this is an integer array of shape `[# output features]` whose values are indices into the input feature vector.

Return type

array

inverse_transform(*X*)

Reverse the transformation operation.

Parameters

X (array of shape `[n_samples, n_selected_features]`) – The input samples.

Returns

X_r – *X* with columns of zeros inserted where features would have been removed by *transform()*.

Return type

array of shape [n_samples, n_original_features]

predict(*x_data*, ***predict_kwargs*)

Expand `predict()` to accept kwargs.

predict_log_proba(*X*)

Predict class log-probabilities for *X*.

Parameters

X (array of shape [n_samples, n_features]) – The input samples.

Returns

p – The class log-probabilities of the input samples. The order of the classes corresponds to that in the attribute `classes_`.

Return type

array of shape (n_samples, n_classes)

predict_proba(*X*)

Predict class probabilities for *X*.

Parameters

X ({array-like or sparse matrix} of shape (n_samples, n_features)) – The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

Returns

p – The class probabilities of the input samples. The order of the classes corresponds to that in the attribute `classes_`.

Return type

array of shape (n_samples, n_classes)

score(*X*, *y*, ***fit_params*)

Reduce *X* to the selected features and return the score of the estimator.

Parameters

- **X** (array of shape [n_samples, n_features]) – The input samples.
- **y** (array of shape [n_samples]) – The target values.
- ****fit_params** (*dict*) – Parameters to pass to the `score` method of the underlying estimator.

New in version 1.0.

Returns

score – Score of the underlying base estimator computed with the selected features returned by `rfe.transform(X)` and *y*.

Return type

float

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self – Estimator instance.

Return type

estimator instance

transform(X)

Reduce X to the selected features.

Parameters

X (array of shape *[n_samples, n_features]*) – The input samples.

Returns

X_r – The input samples with only the selected features.

Return type

array of shape *[n_samples, n_selected_features]*

```
class esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV(estimator, step=1,
                                                                min_features_to_select=1,
                                                                cv=None, scoring=None,
                                                                verbose=0, n_jobs=None)
```

Bases: [AdvancedRFE](#)

Expand [sklearn.feature_selection.RFECV](#).

property classes_

Classes labels available when *estimator* is a classifier.

Return type

ndarray of shape *(n_classes,)*

decision_function(X)

Compute the decision function of X.

Parameters

X (*{array-like or sparse matrix} of shape (n_samples, n_features)*) – The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

Returns

score – The decision function of the input samples. The order of the classes corresponds to that in the attribute [classes_](#). Regression and binary classification produce an array of shape *[n_samples]*.

Return type

array, shape = *[n_samples, n_classes]* or *[n_samples]*

fit(x_data, y_data, groups=None, **fit_kwargs)

Expand [fit\(\)](#) to accept kwargs.

fit_transform(X, y=None, **fit_params)

Fit to data, then transform it.

Fits transformer to X and y with optional parameters *fit_params* and returns a transformed version of X.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Input samples.

- **y** (array-like of shape *(n_samples,)* or *(n_samples, n_outputs)*, *default=None*) – Target values (None for unsupervised transformations).
- ****fit_params** (*dict*) – Additional fit parameters.

Returns

X_new – Transformed array.

Return type

ndarray array of shape *(n_samples, n_features_new)*

get_feature_names_out(*input_features=None*)

Mask feature names according to selected features.

Parameters

input_features (array-like of *str* or *None*, *default=None*) – Input features.

- If *input_features* is *None*, then *feature_names_in_* is used as feature names in. If *feature_names_in_* is not defined, then the following input feature names are generated: *["x0", "x1", ..., "x(n_features_in_ - 1)"]*.
- If *input_features* is an array-like, then *input_features* must match *feature_names_in_* if *feature_names_in_* is defined.

Returns

feature_names_out – Transformed feature names.

Return type

ndarray of *str* objects

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool*, *default=True*) – If *True*, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

dict

get_support(*indices=False*)

Get a mask, or integer index, of the features selected.

Parameters

indices (*bool*, *default=False*) – If *True*, the return value will be an array of integers, rather than a boolean mask.

Returns

support – An index that selects the retained features from a feature vector. If *indices* is *False*, this is a boolean array of shape *[# input features]*, in which an element is *True* iff its corresponding feature is selected for retention. If *indices* is *True*, this is an integer array of shape *[# output features]* whose values are indices into the input feature vector.

Return type

array

inverse_transform(*X*)

Reverse the transformation operation.

Parameters

X (array of shape $[n_samples, n_selected_features]$) – The input samples.

Returns

X_r – X with columns of zeros inserted where features would have been removed by `transform()`.

Return type

array of shape $[n_samples, n_original_features]$

predict(*x_data*, ***predict_kwargs*)

Expand `predict()` to accept kwargs.

predict_log_proba(X)

Predict class log-probabilities for X.

Parameters

X (array of shape $[n_samples, n_features]$) – The input samples.

Returns

p – The class log-probabilities of the input samples. The order of the classes corresponds to that in the attribute `classes_`.

Return type

array of shape $(n_samples, n_classes)$

predict_proba(X)

Predict class probabilities for X.

Parameters

X ({array-like or sparse matrix} of shape $(n_samples, n_features)$) – The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

Returns

p – The class probabilities of the input samples. The order of the classes corresponds to that in the attribute `classes_`.

Return type

array of shape $(n_samples, n_classes)$

score(X, y, ***fit_params*)

Reduce X to the selected features and return the score of the estimator.

Parameters

- **X** (array of shape $[n_samples, n_features]$) – The input samples.
- **y** (array of shape $[n_samples]$) – The target values.
- ****fit_params** (*dict*) – Parameters to pass to the `score` method of the underlying estimator.

New in version 1.0.

Returns

score – Score of the underlying base estimator computed with the selected features returned by `rfe.transform(X)` and y.

Return type

float

set_params(params)**

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as [Pipeline](#)). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self – Estimator instance.

Return type

estimator instance

transform(X)

Reduce X to the selected features.

Parameters

X (*array of shape [n_samples, n_features]*) – The input samples.

Returns

X_r – The input samples with only the selected features.

Return type

array of shape [n_samples, n_selected_features]

```
class esmvaltool.diag_scripts.mlrcustom_sklern.AdvancedTransformedTargetRegressor(regressor=None,
*,
trans-
former=None,
func=None,
in-
verse_func=None,
check_inverse=True)
```

Bases: [TransformedTargetRegressor](#)

Expand [sklearn.compose.TransformedTargetRegressor](#).

property coef_

Model coefficients.

Type

[numpy.ndarray](#)

property feature_importances_

Feature importances.

Type

[numpy.ndarray](#)

fit(x_data, y_data, **fit_kwargs)

Expand [fit\(\)](#) to accept kwargs.

fit_transformer_only(y_data, **fit_kwargs)

Fit only transformer step.

get_params(deep=True)

Get parameters for this estimator.

Parameters

deep (*bool*, *default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

dict

property n_features_in_

Number of features seen during *fit*.

predict(*x_data*, *always_return_1d=True*, ***predict_kwargs*)

Expand *predict()* to accept kwargs.

score(*X*, *y*, *sample_weight=None*)

Return the coefficient of determination of the prediction.

The coefficient of determination R^2 is defined as $(1 - \frac{u}{v})$, where u is the residual sum of squares $((y_true - y_pred)** 2).sum()$ and v is the total sum of squares $((y_true - y_true.mean()) ** 2).sum()$. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y , disregarding the input features, would get a R^2 score of 0.0.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape $(n_samples, n_samples_fitted)$, where *n_samples_fitted* is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for X.
- **sample_weight** (*array-like of shape (n_samples,)*, *default=None*) – Sample weights.

Returns

score – R^2 of *self.predict(X)* wrt. *y*.

Return type

float

Notes

The R^2 score used when calling *score* on a regressor uses *multioutput='uniform_average'* from version 0.23 to keep consistent with default value of *r2_score()*. This influences the *score* method of all the multioutput regressors (except for *MultiOutputRegressor*).

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as *Pipeline*). The latter have parameters of the form *<component>__<parameter>* so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self – Estimator instance.

Return type

estimator instance

```
class esmvaltool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer(grid_scores,  
                                                                           n_features,  
                                                                           ranking,  
                                                                           support)
```

Bases: `BaseEstimator`, `SelectorMixin`

Transformer step of a feature selection estimator.

fit(**, **__*)

Empty method.

fit_transform(*X, y=None, **fit_params*)

Fit to data, then transform it.

Fits transformer to *X* and *y* with optional parameters *fit_params* and returns a transformed version of *X*.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Input samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs), default=None*) – Target values (None for unsupervised transformations).
- ****fit_params** (*dict*) – Additional fit parameters.

Returns

X_new – Transformed array.

Return type

ndarray array of shape (n_samples, n_features_new)

get_feature_names_out(*input_features=None*)

Mask feature names according to selected features.

Parameters

input_features (*array-like of str or None, default=None*) – Input features.

- If *input_features* is *None*, then *feature_names_in_* is used as feature names in. If *feature_names_in_* is not defined, then the following input feature names are generated: [*"x0"*, *"x1"*, ..., *"x(n_features_in_ - 1)"*].
- If *input_features* is an array-like, then *input_features* must match *feature_names_in_* if *feature_names_in_* is defined.

Returns

feature_names_out – Transformed feature names.

Return type

ndarray of str objects

get_params(*deep=True*)

Get parameters for this estimator.

Parameters

deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

`dict`

get_support(*indices=False*)

Get a mask, or integer index, of the features selected.

Parameters

indices (*bool*, *default=False*) – If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support – An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

Return type

`array`

inverse_transform(*X*)

Reverse the transformation operation.

Parameters

X (*array of shape [n_samples, n_selected_features]*) – The input samples.

Returns

X_r – *X* with columns of zeros inserted where features would have been removed by `transform()`.

Return type

`array of shape [n_samples, n_original_features]`

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self – Estimator instance.

Return type

`estimator instance`

transform(*X*)

Reduce *X* to the selected features.

Parameters

X (*array of shape [n_samples, n_features]*) – The input samples.

Returns

X_r – The input samples with only the selected features.

Return type

`array of shape [n_samples, n_selected_features]`

```
esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted(estimator, x_data,
                                                                    y_data=None,
                                                                    groups=None,
                                                                    scoring=None, cv=None,
                                                                    n_jobs=None, verbose=0,
                                                                    fit_params=None,
                                                                    pre_dispatch='2*n_jobs',
                                                                    error_score=nan,
                                                                    sample_weights=None)
```

Expand `sklearn.model_selection.cross_val_score()`.

```
esmvaltool.diag_scripts.mlr.custom_sklearn.get_rfecv_transformer(rfecv_estimator)
```

Get transformer step of RFECV estimator.

```
esmvaltool.diag_scripts.mlr.custom_sklearn.perform_efecv(estimator, x_data, y_data, **kwargs)
```

Perform exhaustive feature selection.

MLRModel base class

Base class for MLR models.

Example recipe

The *MLR main diagnostic script* provides an interface for using MLR models in recipes. The following recipe shows a typical example on how to setup MLR recipes/diagnostics with the following properties:

1. Setup an MLR model with target variable `y` (using the tag `Y`) and three predictors `x1`, `x2` and `latitude` (with tags `X1`, `X2` and `latitude`, respectively). The target variable needs the attribute `var_type: label`; the predictors `x1` and `x2` the attribute `var_type: feature`. The coordinate feature `latitude` is added via the option `coords_as_features: [latitude]`.
2. Suppose `y` and `x1` are 3D fields (pressure, latitude, longitude); `x2` is a 2D field (latitude, longitude). Thus, it is necessary to add the attribute `broadcast_from: [1, 2]` to it (see `dim_map` parameter in `iris.util.broadcast_to_shape()` for details). In order to consider multiple climate models (`A`, `B` and `C`) at once, the option `group_datasets_by_attributes: [dataset]` is necessary. Otherwise the diagnostic will complain about duplicate data.
3. For the prediction, data from dataset `D` is used (with `var_type: prediction_input`). For the feature `X1` additional input error (with `var_type: prediction_input_error`) is used.

```
diag_feature_x1:
  variables:
    feature:
      ... # specify project, mip, start_year, end_year, etc.
      short_name: x1
      var_type: feature
      tag: X1
      additional_datasets:
        - {dataset: A, ...}
        - {dataset: B, ...}
        - {dataset: C, ...}
      prediction_input:
        ... # specify project, mip, start_year, end_year, etc.
```

(continues on next page)

(continued from previous page)

```

    short_name: x1
    var_type: prediction_input
    tag: X1
    additional_datasets:
      - {dataset: D, ...}
    prediction_input_error:
      ... # specify project, mip, start_year, end_year, etc.
      short_name: x1Stderr
      var_type: prediction_input_error
      tag: X1
      additional_datasets:
        - {dataset: D, ...}
    scripts:
      null

diag_feature_x2:
  variables:
    feature:
      ... # specify project, mip, start_year, end_year, etc.
      short_name: x2
      var_type: feature
      broadcast_from: [1, 2]
      tag: X2
      additional_datasets:
        - {dataset: A, ...}
        - {dataset: B, ...}
        - {dataset: C, ...}
    prediction_input:
      ... # specify project, mip, start_year, end_year, etc.
      short_name: x2
      var_type: prediction_input
      broadcast_from: [1, 2]
      tag: X2
      additional_datasets:
        - {dataset: D, ...}
    scripts:
      null

diag_label:
  variables:
    label:
      ... # specify project, mip, start_year, end_year, etc.
      short_name: y
      var_type: label
      tag: Y
      additional_datasets:
        - {dataset: A, ...}
        - {dataset: B, ...}
        - {dataset: C, ...}
    scripts:
      null

```

4. In this example, a **GBRT model** (with `mlr_model_type: gbr_sklearn`) is used. Parameters for this

are specified via `parameters_final_regressor`. Apart from the best-estimate prediction, the estimated MLR model error (`save_mlr_model_error: test`) and the propagated prediction input error (`save_propagated_errors: true`) are returned.

5. With `postprocess.py`, the global mean of the best estimate prediction and the corresponding errors (MLR model + propagated input error) are calculated.

```
diag_mlr_gbrt:
  scripts:
    mlr:
      script: mlr/main.py
      ancestors: [
        'diag_label/y',
        'diag_feature_*/*',
      ]
      coords_as_features: [latitude]
      group_datasets_by_attributes: [dataset]
      mlr_model_name: GBRT
      mlr_model_type: gbr_sklearn
      parameters_final_regressor:
        learning_rate: 0.1
        n_estimators: 100
      save_mlr_model_error: test
      save_propagated_errors: true
  postprocess:
    script: mlr/postprocess.py
    ancestors: ['diag_mlr_gbrt/mlr']
    ignore:
      - {var_type: null}
    mean: [pressure, latitude, longitude]
```

6. Plots of the global distribution (latitude, longitude) are created with `plot.py` after calculating the mean over the pressure coordinate using `preprocess.py`.

```
diag_plot:
  scripts:
    preprocess:
      script: mlr/preprocess.py
      ancestors: ['diag_mlr_gbrt/mlr']
      collapse: [pressure]
      ignore:
        - {var_type: null}
    plot:
      script: mlr/plot.py
      ancestors: ['diag_plot/preprocess']
    plot_map:
      plot_kwargs:
        cbar_label: 'Y'
        cbar_ticks: [0, 1, 2, 3]
        vmin: 0
        vmax: 3
```

All datasets must have the attribute `var_type` which specifies the type of the dataset. Possible values are `feature` (independent variables used for training/testing), `label` (dependent variables, y-axis), `prediction_input` (independent variables used for prediction of dependent variables, usually observational data), `prediction_input_error`

(standard error of the `prediction_input` data, optional) or `prediction_reference` (*true* values for the `prediction_input` data, optional). In addition, all datasets must have the attribute `tag`, which specifies the name of variable/diagnostic. All datasets can be converted to new units in the loading step by specifying the key `convert_units_to` in the respective dataset(s).

Training data

All groups (specified in `group_datasets_by_attributes`, if desired) given for label datasets must also be given for the feature datasets. Within these groups, all feature and label datasets must have the same shape, except the attribute `broadcast_from` is set to a list of suitable coordinate indices to map this dataset to regular datasets (see parameter `dim_map` in `iris.util.broadcast_to_shape()`).

Prediction data

All `tag`s specified for `prediction_input` datasets must also be given for the feature datasets (except `allow_missing_features` is set to `True`). Multiple predictions can be specified by `prediction_name`. Within these predictions, all `prediction_input` datasets must have the same shape, except the attribute `broadcast_from` is given. Errors in the prediction input data can be specified by `prediction_input_error`. If given, these errors are used to calculate errors in the final prediction using linear error propagation given by [LIME](#). Additionally, *true* values for `prediction_input` can be specified with `prediction_reference` datasets (together with the respective `prediction_name`). This allows an evaluation of the performance of the MLR model by calculating residuals (*true* minus predicted values).

Available MLR models

MLR models are subclasses of this base class. A list of all available MLR models can be found [here](#). To add a new MLR model, create a new file in `esmvaltool/diag_scripts/mlr/models/` with a child class of `esmvaltool.diag_scripts.mlr.models.MLRModel` decorated with `esmvaltool.diag_scripts.mlr.models.MLRModel.register_mlr_model()`.

Optional parameters for class initialization

accept_only_scalar_data: bool (default: False)

If set to `True`, only accept scalar input data. Should be used together with the option `group_datasets_by_attributes`.

allow_missing_features: bool (default: False)

Allow missing features in the training data.

cache_intermediate_results: bool (default: True)

Cache the intermediate results of the pipeline's transformers.

categorical_features: list of str

Names of features which are interpreted as categorical features (in contrast to numerical features).

coords_as_features: list of str

If given, specify a list of coordinates which should be used as features.

dtype: str (default: 'float64')

Internal data type which is used for all calculations, see <https://docs.scipy.org/doc/numpy/user/basics.types.html> for a list of allowed values.

fit_kwargs: dict

Optional keyword arguments for the pipeline's `fit()` function. These arguments have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

group_datasets_by_attributes: list of str

List of dataset attributes which are used to group input data for feature `s` and label `s`. For example, this is necessary if the MLR model should consider multiple climate models in the training phase. If this option is not given, specifying multiple datasets with identical `var_type` and `tag` entries results in an error. If given, all the input data is first grouped by the given attributes and then checked for uniqueness within this group. After that, all groups are stacked to form a single set of training data.

imputation_strategy: str (default: 'remove')

Strategy for the imputation of missing values in the features. Must be one of 'remove', 'mean', 'median', 'most_frequent' or 'constant'.

log_level: str (default: 'info')

Verbosity for the logger. Must be one of 'debug', 'info', 'warning' or 'error'.

mlr_model_name: str

Human-readable name of the MLR model instance (e.g used for labels).

n_jobs: int (default: 1)

Maximum number of jobs spawned by this class. Use -1 to use all processors. More details are given [here](#).

output_file_type: str (default: 'png')

File type for the plots.

parameters: dict

Parameters used for the whole pipeline. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

parameters_final_regressor: dict

Parameters used for the **final** regressor. If these parameters are updated using the function `update_parameters()`, the new names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

pca: bool (default: False)

Preprocess numerical input features using PCA. Parameters for this pipeline step can be given via the `parameters` argument.

plot_dir: str (default: ~/plots)

Root directory to save plots.

plot_units: dict

Replace specific units (keys) with other text (values) in plots.

savefig_kwargs: dict

Keyword arguments for `matplotlib.pyplot.savefig()`.

seaborn_settings: dict

Options for `seaborn.set()` (affects all plots).

standardize_data: bool (default: True)

Linearly standardize numerical input data by removing mean and scaling to unit variance.

sub_dir: str

Create additional subdirectory for output in `work_dir` and `plot_dir`.

test_size: float (default: 0.25)

If given, randomly exclude the desired fraction of input data from training and use it as test data.

weighted_samples: dict

If specified, use weighted samples in the loss function used for the training of the MLR model. The given keyword arguments are directly passed to `esmvaltool.diag_scripts.mlr.get_all_weights()` to calculate the sample weights. By default, no weights are used. Raises errors if the desired weights cannot be calculated for the data, e.g., when `time_weighted=True` is used but the data does not contain a dimension `time`.

work_dir: str (default: ~/work)

Root directory to save all other files (mainly *.nc files).

Classes:

<code>MLRModel(input_datasets, **kwargs)</code>	Base class for MLR models.
---	----------------------------

class `esmvaltool.diag_scripts.mlr.models.MLRModel(input_datasets, **kwargs)`

Bases: `object`

Base class for MLR models.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}_{pred_name}.csv')) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}.csv')) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: `True`)) – Return label files.
- **features** (*list of str*, optional (default: `None`)) – Features for which files should be returned. If `None`, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals(filename=None)`

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals_distribution(filename=None)`

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option *test_size* is not set to *False* during class initialization. If the option is set to 'logo', the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the *group_attributes*. Only possible if *group_datasets_by_attributes* is given. If the option is set to an integer *n* (*n* != 0), the (constant) error is estimated as RMSEP using *n*-fold cross-validation.
- **save_lime_importance** (*bool*, optional (default: *False*)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool*, optional (default: *False*)) – Additionally saves propagated errors from *prediction_input_error* datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments*, optional) – Additional options for the final regressors *predict()* function.

Raises

- **RuntimeError** – *return_var* and *return_cov* are both set to *True*.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for *save_mlr_model_error* is given.
- **ValueError** – *save_propagated_errors* is *True* and no *prediction_input_error* data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(*logo=False*)

Print all available regression metrics for training data.

Parameters

logo (*bool*, optional (default: *False*)) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when *group_datasets_by_attributes* was given during class initialization.

classmethod register_mlr_model(*mlr_model_type*)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(*kwargs*)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments*, optional) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(*params*)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments*, optional) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Base class for Gradient Boosted Regression models

Base class for Gradient Boosting Regression model.

Classes:

<i>GBRModel</i> (input_datasets, **kwargs)	Base class for Gradient Boosting Regression models.
--	---

class esmvaltool.diag_scripts.mlr.models.gbr_base.**GBRModel**(input_datasets, **kwargs)

Bases: *MLRModel*

Base class for Gradient Boosting Regression models.

Attributes:

<i>categorical_features</i>	Categorical features.
<i>data</i>	Input data of the MLR model.
<i>features</i>	Features of the input data.
<i>features_after_preprocessing</i>	Features of the input data after preprocessing.
<i>features_types</i>	Types of the features.
<i>features_units</i>	Units of the features.
<i>fit_kwargs</i>	Keyword arguments for <i>fit()</i> .
<i>group_attributes</i>	Group attributes of the input data.
<i>label</i>	Label of the input data.
<i>label_units</i>	Units of the label.
<i>mlr_model_type</i>	MLR model type.
<i>numerical_features</i>	Numerical features.
<i>parameters</i>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (keyword arguments, optional) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}_{pred_name}.csv')) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}.csv')) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: `True`)) – Return label files.
- **features** (*list of str*, optional (default: `None`)) – Features for which files should be returned. If `None`, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: `'1d_mlr_model'`)) – Name of the plot file.
- **n_points** (*int*, optional (default: `1000`)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_feature_importance(filename=None, color_coded=True)`

Plot feature importance.

This function uses properties of the GBR model based on the number of appearances of that feature in the regression trees and the improvements made by the individual splits (see Friedman, 2001).

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str*, optional (default: `'feature_importance'`)) – Name of the plot file.
- **color_coded** (*bool*, optional (default: `True`)) – If `True`, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If `False`, all bars are blue.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

- **filename** (*str*, optional (default: `'partial_dependence_{feature}'`)) – Name of the plot file.

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

- **filename** (*str*, optional (default: `'prediction_errors'`)) – Name of the plot file.

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_ml_model_error=None, save_lime_importance=False, save_propagated_errors=False, **kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_ml_model_error** (*str or int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with **var_type** set to **prediction_input_error** and setting **save_propagated_errors** to **True**). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option **test_size** is not set to **False** during class initialization. If the option is set to 'logo', the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the **group_attributes**. Only possible if **group_datasets_by_attributes** is given. If the option is set to an integer *n* (*n* != 0), the (constant) error is estimated as RMSEP using *n*-fold cross-validation.

- **save_lime_importance** (*bool*, optional (default: *False*)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool*, optional (default: *False*)) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments*, optional) – Additional options for the final regressors `predict()` function.

Raises

- **RuntimeError** – `return_var` and `return_cov` are both set to `True`.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for `save_mlr_model_error` is given.
- **ValueError** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(*logo=False*)

Print all available regression metrics for training data.

Parameters

logo (*bool*, optional (default: *False*)) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

classmethod register_mlr_model(*mlr_model_type*)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(****kwargs**)

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments*, optional) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(***params*)

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Base class for Linear models

Base class for linear Machine Learning Regression models.

Classes:

<code>LinearModel</code> (<code>input_datasets</code> , ** <code>kwargs</code>)	Base class for linear Machine Learning models.
---	--

class `esmvaltool.diag_scripts.mlr.models.linear_base.LinearModel`(`input_datasets`, **`kwargs`)

Bases: `MLRModel`

Base class for linear Machine Learning models.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type`numpy.ndarray`**classmethod create(mlr_model_type, *args, **kwargs)**

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type`dict`**efecv(**kwargs)**

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: *True*)) – Return label files.
- **features** (*list of str*, optional (default: *None*)) – Features for which files should be returned. If *None*, return files for all features.
- **prediction_names** (*list of str*, optional (default: *None*)) – Prediction names for which files should be returned. If *None*, return files for all prediction names.
- **prediction_reference** (*bool*, optional (default: *False*)) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type*, *impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – *data_type* is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type*, *impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – *data_type* is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type*, *impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str, optional (default: '1d_mlr_model')*) – Name of the plot file.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – MLR model is built from more than 1 feature.

plot_coefs(*filename=None*)

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str, optional (default: 'coefs')*) – Name of the plot file.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str, optional (default: 'feature_importance')*) – Name of the plot file.
- **color_coded** (*bool, optional (default: True)*) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

filename (*str, optional (default: 'partial_dependence_{feature}')*) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input

data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **`save_lime_importance`** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics(logo=False)`

Print all available regression metrics for training data.

Parameters

`logo` (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

`classmethod register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

`reset_pipeline()`

Reset regressor pipeline.

`rfecv(kwargs)`**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

`kwargs`** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

50.2.4 Available MLR models

Gradient Boosted Regression Trees (sklearn implementation)

Gradient Boosting Regression model (using `sklearn`).

Use `mlr_model_type: gbr_sklearn` to use this MLR model in the recipe.

Classes:

<code>SklearnGBRModel(input_datasets, **kwargs)</code>	Gradient Boosting Regression model (<code>sklearn</code> implementation).
--	--

class `esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel(input_datasets, **kwargs)`

Bases: `GBRModel`

Gradient Boosting Regression model (`sklearn` implementation).

Attributes:

<i>categorical_features</i>	Categorical features.
<i>data</i>	Input data of the MLR model.
<i>features</i>	Features of the input data.
<i>features_after_preprocessing</i>	Features of the input data after preprocessing.
<i>features_types</i>	Types of the features.
<i>features_units</i>	Units of the features.
<i>fit_kwargs</i>	Keyword arguments for <i>fit()</i> .
<i>group_attributes</i>	Group attributes of the input data.
<i>label</i>	Label of the input data.
<i>label_units</i>	Units of the label.
<i>mlr_model_type</i>	MLR model type.
<i>numerical_features</i>	Numerical features.
<i>parameters</i>	Parameters of the complete MLR model pipeline.

Methods:

<i>create</i> (mlr_model_type, *args, **kwargs)	Create desired MLR model subclass (factory method).
<i>efecv</i> (**kwargs)	Perform exhaustive feature elimination using cross-validation.
<i>export_prediction_data</i> ([filename])	Export all prediction data contained in <i>self._data</i> .
<i>export_training_data</i> ([filename])	Export all training data contained in <i>self._data</i> .
<i>fit</i> ()	Fit MLR model.
<i>get_ancestors</i> ([label, features, ...])	Return ancestor files.
<i>get_data_frame</i> (data_type[, impute_nans])	Return data frame of specified type.
<i>get_x_array</i> (data_type[, impute_nans])	Return x data of specific type.
<i>get_y_array</i> (data_type[, impute_nans])	Return y data of specific type.
<i>grid_search_cv</i> (param_grid, **kwargs)	Perform exhaustive parameter search using cross-validation.
<i>plot_1d_model</i> ([filename, n_points])	Plot lineplot that represents the MLR model.
<i>plot_feature_importance</i> ([filename, color_coded])	Plot feature importance.
<i>plot_partial_dependences</i> ([filename])	Plot partial dependences for every feature.
<i>plot_prediction_errors</i> ([filename])	Plot predicted vs.
<i>plot_residuals</i> ([filename])	Plot residuals of training and test (if available) data.
<i>plot_residuals_distribution</i> ([filename])	Plot distribution of residuals of training and test data (KDE).
<i>plot_residuals_histogram</i> ([filename])	Plot histogram of residuals of training and test data.
<i>plot_scatterplots</i> ([filename])	Plot scatterplots label vs.
<i>plot_training_progress</i> ([filename])	Plot training progress for training and (if possible) test data.
<i>predict</i> ([save_mlr_model_error, ...])	Perform prediction using the MLR model(s) and write *.nc files.
<i>print_correlation_matrices</i> ()	Print correlation matrices for all datasets.
<i>print_regression_metrics</i> ([logo])	Print all available regression metrics for training data.
<i>register_mlr_model</i> (mlr_model_type)	Add MLR model (subclass of this class) (decorator).
<i>reset_pipeline</i> ()	Reset regressor pipeline.
<i>rfecv</i> (**kwargs)	Perform recursive feature elimination using cross-validation.
<i>test_normality_of_residuals</i> ()	Perform Shapiro-Wilk test to normality of residuals.
<i>update_parameters</i> (**params)	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(*mlr_model_type*, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool, optional (default: True)*) – Return label files.
- **features** (*list of str, optional (default: None)*) – Features for which files should be returned. If None, return files for all features.
- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

`list of str`

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`pandas.DataFrame`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (*default: False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(data_type, impute_nans=False)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (*default: False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(param_grid, **kwargs)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. s__p is the parameter p for step s.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property `group_attributes`

Group attributes of the input data.

Type

`numpy.ndarray`

property `label`

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (`str`, optional (default: `'1d_mlr_model'`)) – Name of the plot file.
- **n_points** (`int`, optional (default: `1000`)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- `sklearn.exceptions.NotFittedError` – MLR model is not fitted.
- `ValueError` – MLR model is built from more than 1 feature.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance.

This function uses properties of the GBR model based on the number of appearances of that feature in the regression trees and the improvements made by the individual splits (see Friedman, 2001).

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (`str`, optional (default: `'feature_importance'`)) – Name of the plot file.

- **color_coded** (*bool*, optional (default: *True*)) – If *True*, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If *False*, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: *'partial_dependence_{feature}'*)) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: *'prediction_errors'*)) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: *'residuals'*)) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: *'residuals_distribution'*)) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: *'residuals_histogram'*)) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: *'scatterplot_{feature}'*)) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`plot_training_progress`(*filename=None*)

Plot training progress for training and (if possible) test data.

Parameters

`filename`(*str*, *optional* (default: `'training_progress'`)) – Name of the plot file.

`predict`(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **`save_mlr_model_error`** (*str* or *int*, *optional*) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`!= 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.
- **`save_lime_importance`** (*bool*, *optional* (default: `False`)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool*, *optional* (default: `False`)) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments*, *optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics`(*logo=False*)

Print all available regression metrics for training data.

Parameters

`logo` (*bool*, *optional* (default: `False`)) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

classmethod `register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(kwargs)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Gradient Boosted Regression Trees (xgboost implementation)

Gradient Boosting Regression model (using `xgboost`).

Use `mlr_model_type: gbr_xgboost` to use this MLR model in the recipe.

Classes:

`XGBoostGBRModel`(`input_datasets`, ****kwargs**)

Gradient Boosting Regression model (`xgboost` implementation).

```
class esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel(input_datasets, **kwargs)
```

Bases: [*GBRModel*](#)

Gradient Boosting Regression model (xgboost implementation).

Attributes:

<i>categorical_features</i>	Categorical features.
<i>data</i>	Input data of the MLR model.
<i>features</i>	Features of the input data.
<i>features_after_preprocessing</i>	Features of the input data after preprocessing.
<i>features_types</i>	Types of the features.
<i>features_units</i>	Units of the features.
<i>fit_kwargs</i>	Keyword arguments for <i>fit()</i> .
<i>group_attributes</i>	Group attributes of the input data.
<i>label</i>	Label of the input data.
<i>label_units</i>	Units of the label.
<i>mlr_model_type</i>	MLR model type.
<i>numerical_features</i>	Numerical features.
<i>parameters</i>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>plot_training_progress([filename])</code>	Plot training progress for training and (if possible) test data.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: *True*)) – Return label files.
- **features** (*list of str*, optional (default: *None*)) – Features for which files should be returned. If *None*, return files for all features.
- **prediction_names** (*list of str*, optional (default: *None*)) – Prediction names for which files should be returned. If *None*, return files for all prediction names.
- **prediction_reference** (*bool*, optional (default: *False*)) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type*, *impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type*, *impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type*, *impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

dict

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance.

This function uses properties of the GBR model based on the number of appearances of that feature in the regression trees and the improvements made by the individual splits (see Friedman, 2001).

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str*, optional (default: 'feature_importance')) – Name of the plot file.
- **color_coded** (*bool*, optional (default: True)) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

- **filename** (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_training_progress(*filename=None*)

Plot training progress for training and (if possible) test data.

Parameters

filename (*str*, optional (default: 'training_progress')) – Name of the plot file.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input

data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **`save_lime_importance`** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics(logo=False)`

Print all available regression metrics for training data.

Parameters

`logo` (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

`classmethod register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

`reset_pipeline()`

Reset regressor pipeline.

`rfecv(kwargs)`**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

`kwargs`** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Gaussian Process Regression (sklearn implementation)

Gaussian Process Regression model (using sklearn).

Use `mlr_model_type: gpr_sklearn` to use this MLR model in the recipe.

Classes:

<code>AdvancedGaussianProcessRegressor([kernel, ...])</code>	Expand <code>sklearn.gaussian_process.GaussianProcessRegressor</code> .
<code>SklearnGPRModel(input_datasets, **kwargs)</code>	Gaussian Process Regression model (sklearn implementation).

```
class esmvaltool.diag_scripts.mlr.models.gpr_sklearn.AdvancedGaussianProcessRegressor(kernel=None,
*,
alpha=1e-10,
optimizer='fmin_l_bfgs_b',
n_restarts_optimizer=10,
normalize_y=False,
copy_X_train=True,
random_state=None)
```

Bases: `GaussianProcessRegressor`

Expand `sklearn.gaussian_process.GaussianProcessRegressor`.

Methods:

<code>fit(X, y)</code>	Fit Gaussian process regression model.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>log_marginal_likelihood([theta, ...])</code>	Return log-marginal likelihood of theta for training data.
<code>predict(x_data[, return_var, return_cov])</code>	Expand <code>predict()</code> to accept <code>return_var</code> .
<code>sample_y(X[, n_samples, random_state])</code>	Draw samples from Gaussian process and evaluate at X.
<code>score(X, y[, sample_weight])</code>	Return the coefficient of determination of the prediction.
<code>set_params(**params)</code>	Set the parameters of this estimator.

fit(X, y)

Fit Gaussian process regression model.

Parameters

- **X** (*array-like of shape (n_samples, n_features) or list of object*) – Feature vectors or other representations of training data.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_targets)*) – Target values.

Returns

self – GaussianProcessRegressor class instance.

Return type

object

get_params(deep=True)

Get parameters for this estimator.

Parameters

deep (*bool*, *default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

dict

log_marginal_likelihood(theta=None, eval_gradient=False, clone_kernel=True)

Return log-marginal likelihood of theta for training data.

Parameters

- **theta** (*array-like of shape (n_kernel_params,) default=None*) – Kernel hyperparameters for which the log-marginal likelihood is evaluated. If None, the precomputed `log_marginal_likelihood` of `self.kernel_.theta` is returned.
- **eval_gradient** (*bool*, *default=False*) – If True, the gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta is returned additionally. If True, theta must not be None.
- **clone_kernel** (*bool*, *default=True*) – If True, the kernel attribute is copied. If False, the kernel attribute is modified, but may result in a performance improvement.

Returns

- **log_likelihood** (*float*) – Log-marginal likelihood of theta for training data.
- **log_likelihood_gradient** (*ndarray of shape (n_kernel_params,)*, *optional*) – Gradient of the log-marginal likelihood with respect to the kernel hyperparameters at position theta. Only returned when eval_gradient is True.

predict(*x_data*, *return_var=False*, *return_cov=False*)

Expand **predict()** to accept *return_var*.

sample_y(*X*, *n_samples=1*, *random_state=0*)

Draw samples from Gaussian process and evaluate at *X*.

Parameters

- **X** (*array-like of shape (n_samples_X, n_features)* or *list of object*) – Query points where the GP is evaluated.
- **n_samples** (*int*, *default=1*) – Number of samples drawn from the Gaussian process per query point.
- **random_state** (*int*, *RandomState instance or None*, *default=0*) – Determines random number generation to randomly draw samples. Pass an int for reproducible results across multiple function calls. See [Glossary](#).

Returns

y_samples – Values of *n_samples* samples drawn from Gaussian process and evaluated at query points.

Return type

ndarray of shape (*n_samples_X*, *n_samples*), or (*n_samples_X*, *n_targets*, *n_samples*)

score(*X*, *y*, *sample_weight=None*)

Return the coefficient of determination of the prediction.

The coefficient of determination R^2 is defined as $(1 - \frac{u}{v})$, where u is the residual sum of squares $((y_true - y_pred) ** 2).sum()$ and v is the total sum of squares $((y_true - y_true.mean()) ** 2).sum()$. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y , disregarding the input features, would get a R^2 score of 0.0.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (*n_samples*, *n_samples_fitted*), where *n_samples_fitted* is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,)* or (*n_samples*, *n_outputs*)) – True values for *X*.
- **sample_weight** (*array-like of shape (n_samples,)*, *default=None*) – Sample weights.

Returns

score – R^2 of `self.predict(X)` wrt. *y*.

Return type

float

Notes

The R^2 score used when calling `score` on a regressor uses `multioutput='uniform_average'` from version 0.23 to keep consistent with default value of `r2_score()`. This influences the `score` method of all the multioutput regressors (except for `MultiOutputRegressor`).

set_params(params)**

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** (*dict*) – Estimator parameters.

Returns

self – Estimator instance.

Return type

estimator instance

class `esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel(input_datasets, **kwargs)`

Bases: `MLRModel`

Gaussian Process Regression model (sklearn implementation).

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_kernel_info()</code>	Print information of the fitted kernel of the GPR model.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (keyword arguments, optional) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}_{pred_name}.csv')) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}.csv')) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: `True`)) – Return label files.
- **features** (*list of str*, optional (default: `None`)) – Features for which files should be returned. If `None`, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals(filename=None)`

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals_distribution(filename=None)`

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option *test_size* is not set to *False* during class initialization. If the option is set to 'logo', the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the *group_attributes*. Only possible if *group_datasets_by_attributes* is given. If the option is set to an integer *n* (*n* != 0), the (constant) error is estimated as RMSEP using *n*-fold cross-validation.
- **save_lime_importance** (*bool*, optional (default: *False*)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool*, optional (default: *False*)) – Additionally saves propagated errors from *prediction_input_error* datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments*, optional) – Additional options for the final regressors *predict()* function.

Raises

- **RuntimeError** – *return_var* and *return_cov* are both set to *True*.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for *save_mlr_model_error* is given.
- **ValueError** – *save_propagated_errors* is *True* and no *prediction_input_error* data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_kernel_info()

Print information of the fitted kernel of the GPR model.

print_regression_metrics(*logo=False*)

Print all available regression metrics for training data.

Parameters

logo (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when *group_datasets_by_attributes* was given during class initialization.

classmethod register_mlr_model(*mlr_model_type*)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(*kwargs*)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(*params*)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Huber Regression

Huber Regression model.

Use `mlr_model_type`: `huber` to use this MLR model in the recipe.

Classes:

<code>HuberRegressionModel(input_datasets, **kwargs)</code>	Huber Regression model.
---	-------------------------

class `esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel(input_datasets, **kwargs)`

Bases: [`LinearModel`](#)

Huber Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: *True*)) – Return label files.
- **features** (*list of str*, optional (default: *None*)) – Features for which files should be returned. If *None*, return files for all features.
- **prediction_names** (*list of str*, optional (default: *None*)) – Prediction names for which files should be returned. If *None*, return files for all prediction names.
- **prediction_reference** (*bool*, optional (default: *False*)) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type*, *impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type*, *impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type*, *impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str, optional (default: '1d_mlr_model')*) – Name of the plot file.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

plot_coefs(*filename=None*)

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str, optional (default: 'coefs')*) – Name of the plot file.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str, optional (default: 'feature_importance')*) – Name of the plot file.
- **color_coded** (*bool, optional (default: True)*) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

filename (*str, optional (default: 'partial_dependence_{feature}')*) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input

data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **`save_lime_importance`** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics(logo=False)`

Print all available regression metrics for training data.

Parameters

`logo` (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

`classmethod register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

`reset_pipeline()`

Reset regressor pipeline.

`rfecv(kwargs)`**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

`kwargs`** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Kernel Ridge Regression

Kernel Ridge Regression model.

Use `mlr_model_type: krr` to use this MLR model in the recipe.

Classes:

<code>KRRModel(input_datasets, **kwargs)</code>	Kernel Ridge Regression model.
---	--------------------------------

class `esmvaltool.diag_scripts.mlr.models.krr.KRRModel(input_datasets, **kwargs)`

Bases: `MLRModel`

Kernel Ridge Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <i>self._data</i> .
<code>export_training_data([filename])</code>	Export all training data contained in <i>self._data</i> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

numpy.ndarray

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

dict

efecv(kwargs)**

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (keyword arguments, optional) – Additional options for esmvaltool.
diag_scripts.mlr. custom_sklearn.cross_val_score_weighted().

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}_{pred_name}.csv')) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}.csv')) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: `True`)) – Return label files.
- **features** (*list of str*, optional (default: `None`)) – Features for which files should be returned. If `None`, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals(filename=None)`

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals_distribution(filename=None)`

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option *test_size* is not set to *False* during class initialization. If the option is set to 'logo', the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the *group_attributes*. Only possible if *group_datasets_by_attributes* is given. If the option is set to an integer *n* (*n* != 0), the (constant) error is estimated as RMSEP using *n*-fold cross-validation.
- **save_lime_importance** (*bool*, optional (default: *False*)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool*, optional (default: *False*)) – Additionally saves propagated errors from *prediction_input_error* datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments*, optional) – Additional options for the final regressors *predict()* function.

Raises

- **RuntimeError** – *return_var* and *return_cov* are both set to *True*.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for *save_mlr_model_error* is given.
- **ValueError** – *save_propagated_errors* is *True* and no *prediction_input_error* data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(*logo=False*)

Print all available regression metrics for training data.

Parameters

logo (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

classmethod register_mlr_model(*mlr_model_type*)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(*kwargs*)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(*params*)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

LASSO Regression

Lasso Regression model.

Use `mlr_model_type`: `lasso` to use this MLR model in the recipe.

Classes:

<code>LassoModel(input_datasets, **kwargs)</code>	Lasso Regression model.
---	-------------------------

class `esmvaltool.diag_scripts.mlr.models.lasso.LassoModel(input_datasets, **kwargs)`

Bases: `LinearModel`

Lasso Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type`numpy.ndarray`**classmethod create(mlr_model_type, *args, **kwargs)**

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type`dict`**efecv(**kwargs)**

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: *True*)) – Return label files.
- **features** (*list of str*, optional (default: *None*)) – Features for which files should be returned. If *None*, return files for all features.
- **prediction_names** (*list of str*, optional (default: *None*)) – Prediction names for which files should be returned. If *None*, return files for all prediction names.
- **prediction_reference** (*bool*, optional (default: *False*)) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type*, *impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type*, *impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type*, *impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str, optional (default: '1d_mlr_model')*) – Name of the plot file.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

plot_coefs(*filename=None*)

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str, optional (default: 'coefs')*) – Name of the plot file.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str, optional (default: 'feature_importance')*) – Name of the plot file.
- **color_coded** (*bool, optional (default: True)*) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

filename (*str, optional (default: 'partial_dependence_{feature}')*) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input

data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **`save_lime_importance`** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics(logo=False)`

Print all available regression metrics for training data.

Parameters

`logo` (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

`classmethod register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

`reset_pipeline()`

Reset regressor pipeline.

`rfecv(kwargs)`**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

`kwargs`** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

LASSO Regression with built-in CV

Lasso Regression model with built-in CV.

Use `mlr_model_type: lasso_cv` to use this MLR model in the recipe.

Classes:

<code>LassoCVModel(input_datasets, **kwargs)</code>	Lasso Regression model with built-in CV.
---	--

class `esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel(input_datasets, **kwargs)`

Bases: `LinearModel`

Lasso Regression model with built-in CV.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <i>self._data</i> .
<code>export_training_data([filename])</code>	Export all training data contained in <i>self._data</i> .
<code>fit()</code>	Print final alpha after successful fitting.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

numpy.ndarray

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

dict

efecv(kwargs)**

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklern.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Print final alpha after successful fitting.

property fit_kwargs

Keyword arguments for *fit()*.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool, optional (default: True)*) – Return label files.
- **features** (*list of str, optional (default: None)*) – Features for which files should be returned. If None, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_coefs(filename=None)`

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str*, optional (default: 'coefs')) – Name of the plot file.

`plot_feature_importance(filename=None, color_coded=True)`

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str*, optional (default: 'feature_importance')) – Name of the plot file.
- **color_coded** (*bool*, optional (default: True)) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out)

test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **save_lime_importance** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **RuntimeError** – `return_var` and `return_cov` are both set to `True`.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for `save_mlr_model_error` is given.
- **ValueError** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(logo=False)

Print all available regression metrics for training data.

Parameters

logo (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

classmethod register_mlr_model(mlr_model_type)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(kwargs)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

`test_normality_of_residuals()`

Perform Shapiro-Wilk test to normality of residuals.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`update_parameters(**params)`

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

`ValueError` – Invalid parameter for pipeline given.

LASSO Regression (using Least-angle Regression algorithm) with built-in CV

Lasso Regression model with built-in CV using LARS algorithm.

Use `mlr_model_type: lasso_lars_cv` to use this MLR model in the recipe.

Classes:

<code>LassoLarsCVModel(input_datasets, **kwargs)</code>	Lasso Regression model with built-in CV using LARS algorithm.
---	---

```
class esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel(input_datasets,
                                                                           **kwargs)
```

Bases: `LinearModel`

Lasso Regression model with built-in CV using LARS algorithm.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Print final α after successful fitting.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Print final alpha after successful fitting.

property fit_kwargs

Keyword arguments for *fit()*.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool, optional (default: True)*) – Return label files.
- **features** (*list of str, optional (default: None)*) – Features for which files should be returned. If None, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_coefs(filename=None)`

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str*, optional (default: 'coefs')) – Name of the plot file.

`plot_feature_importance(filename=None, color_coded=True)`

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str*, optional (default: 'feature_importance')) – Name of the plot file.
- **color_coded** (*bool*, optional (default: True)) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out)

test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **save_lime_importance** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **RuntimeError** – `return_var` and `return_cov` are both set to `True`.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for `save_mlr_model_error` is given.
- **ValueError** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(logo=False)

Print all available regression metrics for training data.

Parameters

logo (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

classmethod register_mlr_model(mlr_model_type)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(kwargs)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

`ValueError` – Invalid parameter for pipeline given.

Linear Regression

Linear Regression model.

Use `mlr_model_type`: `linear` to use this MLR model in the recipe.

Classes:

`LinearRegressionModel(input_datasets, **kwargs)` Linear Regression model.

```
class esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel(input_datasets,
                                                                    **kwargs)
```

Bases: `LinearModel`

Linear Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: *True*)) – Return label files.
- **features** (*list of str*, optional (default: *None*)) – Features for which files should be returned. If *None*, return files for all features.
- **prediction_names** (*list of str*, optional (default: *None*)) – Prediction names for which files should be returned. If *None*, return files for all prediction names.
- **prediction_reference** (*bool*, optional (default: *False*)) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type*, *impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – *data_type* is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type*, *impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – *data_type* is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type*, *impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str, optional (default: '1d_mlr_model')*) – Name of the plot file.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

plot_coefs(*filename=None*)

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str, optional (default: 'coefs')*) – Name of the plot file.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str, optional (default: 'feature_importance')*) – Name of the plot file.
- **color_coded** (*bool, optional (default: True)*) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

filename (*str, optional (default: 'partial_dependence_{feature}')*) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`plot_residuals(filename=None)`

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`plot_residuals_distribution(filename=None)`

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`plot_residuals_histogram(filename=None)`

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`plot_scatterplots(filename=None)`

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

`predict(save_mlr_model_error=None, save_lime_importance=False, save_propagated_errors=False, **kwargs)`

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input

data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **`save_lime_importance`** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics(logo=False)`

Print all available regression metrics for training data.

Parameters

`logo` (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

`classmethod register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

`reset_pipeline()`

Reset regressor pipeline.

`rfecv(kwargs)`**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

`kwargs`** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Random Forest Regression

Random Forest Regression model.

Use `mlr_model_type: rfr` to use this MLR model in the recipe.

Classes:

<code>RFRModel(input_datasets, **kwargs)</code>	Random Forest Regression model.
---	---------------------------------

class `esmvaltool.diag_scripts.mlr.models.rfr.RFRModel(input_datasets, **kwargs)`

Bases: `MLRModel`

Random Forest Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <i>self._data</i> .
<code>export_training_data([filename])</code>	Export all training data contained in <i>self._data</i> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

numpy.ndarray

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

dict

efecv(kwargs)**

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (keyword arguments, optional) – Additional options for esmvaltool.
diag_scripts.mlr. custom_sklearn.cross_val_score_weighted().

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}_{pred_name}.csv')) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}.csv')) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: `True`)) – Return label files.
- **features** (*list of str*, optional (default: `None`)) – Features for which files should be returned. If `None`, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals(filename=None)`

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals_distribution(filename=None)`

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option *test_size* is not set to *False* during class initialization. If the option is set to 'logo', the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the *group_attributes*. Only possible if *group_datasets_by_attributes* is given. If the option is set to an integer *n* (*n* != 0), the (constant) error is estimated as RMSEP using *n*-fold cross-validation.
- **save_lime_importance** (*bool*, optional (default: *False*)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool*, optional (default: *False*)) – Additionally saves propagated errors from *prediction_input_error* datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments*, optional) – Additional options for the final regressors *predict()* function.

Raises

- **RuntimeError** – *return_var* and *return_cov* are both set to *True*.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for *save_mlr_model_error* is given.
- **ValueError** – *save_propagated_errors* is *True* and no *prediction_input_error* data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(*logo=False*)

Print all available regression metrics for training data.

Parameters

logo (*bool*, optional (default: *False*)) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when *group_datasets_by_attributes* was given during class initialization.

classmethod register_mlr_model(*mlr_model_type*)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(*kwargs*)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments*, optional) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(*params*)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments*, optional) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Ridge Regression

Ridge Regression model.

Use `mlr_model_type`: `ridge` to use this MLR model in the recipe.

Classes:

<code>RidgeModel(input_datasets, **kwargs)</code>	Ridge Regression model.
---	-------------------------

class `esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel(input_datasets, **kwargs)`

Bases: `LinearModel`

Ridge Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: *True*)) – Return label files.
- **features** (*list of str*, optional (default: *None*)) – Features for which files should be returned. If *None*, return files for all features.
- **prediction_names** (*list of str*, optional (default: *None*)) – Prediction names for which files should be returned. If *None*, return files for all prediction names.
- **prediction_reference** (*bool*, optional (default: *False*)) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type*, *impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type*, *impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, optional (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type*, *impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool*, *optional* (default: *False*)) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments*, *optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

plot_1d_model(*filename=None, n_points=1000*)

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str, optional (default: '1d_mlr_model')*) – Name of the plot file.
- **n_points** (*int, optional (default: 1000)*) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

plot_coefs(*filename=None*)

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str, optional (default: 'coefs')*) – Name of the plot file.

plot_feature_importance(*filename=None, color_coded=True*)

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str, optional (default: 'feature_importance')*) – Name of the plot file.
- **color_coded** (*bool, optional (default: True)*) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

plot_partial_dependences(*filename=None*)

Plot partial dependences for every feature.

Parameters

filename (*str, optional (default: 'partial_dependence_{feature}')*) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

`sklearn.exceptions.NotFittedError` – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input

data (errors in that will be considered by including datasets with `var_type` set to `prediction_input_error` and setting `save_propagated_errors` to `True`). If the option is set to `'test'`, the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **`save_lime_importance`** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **`save_propagated_errors`** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- **`**kwargs`** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **`RuntimeError`** – `return_var` and `return_cov` are both set to `True`.
- **`sklearn.exceptions.NotFittedError`** – MLR model is not fitted.
- **`ValueError`** – An invalid value for `save_mlr_model_error` is given.
- **`ValueError`** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

`print_correlation_matrices()`

Print correlation matrices for all datasets.

`print_regression_metrics(logo=False)`

Print all available regression metrics for training data.

Parameters

`logo` (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

`classmethod register_mlr_model(mlr_model_type)`

Add MLR model (subclass of this class) (decorator).

`reset_pipeline()`

Reset regressor pipeline.

`rfecv(kwargs)`**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

`kwargs`** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Ridge Regression with built-in CV

Ridge Regression model with built-in CV.

Use `mlr_model_type: ridge_cv` to use this MLR model in the recipe.

Classes:

<code>RidgeCVModel(input_datasets, **kwargs)</code>	Ridge Regression model with built-in CV.
---	--

class `esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel(input_datasets, **kwargs)`

Bases: `LinearModel`

Ridge Regression model with built-in CV.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <i>self._data</i> .
<code>export_training_data([filename])</code>	Export all training data contained in <i>self._data</i> .
<code>fit()</code>	Print final alpha after successful fitting.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_coefs([filename])</code>	Plot linear coefficients of models.
<code>plot_feature_importance([filename, color_coded])</code>	Plot feature importance given by linear coefficients.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

numpy.ndarray

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

dict

efecv(kwargs)**

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `esmvaltool.diag_scripts.mlr.custom_skllearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}_{pred_name}.csv')*) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str, optional (default: '{data_type}.csv')*) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Print final alpha after successful fitting.

property fit_kwargs

Keyword arguments for *fit()*.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool, optional (default: True)*) – Return label files.
- **features** (*list of str, optional (default: None)*) – Features for which files should be returned. If None, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_coefs(filename=None)`

Plot linear coefficients of models.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

filename (*str*, optional (default: 'coefs')) – Name of the plot file.

`plot_feature_importance(filename=None, color_coded=True)`

Plot feature importance given by linear coefficients.

Note: The features plotted here are not necessarily the real input features, but the ones after preprocessing.

Parameters

- **filename** (*str*, optional (default: 'feature_importance')) – Name of the plot file.
- **color_coded** (*bool*, optional (default: True)) – If True, mark positive (linear) correlations with red bars and negative (linear) correlations with blue bars. If False, all bars are blue.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.

plot_prediction_errors(*filename=None*)

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals(*filename=None*)

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_distribution(*filename=None*)

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out)

test data set. Only possible if test data is available, i.e. the option `test_size` is not set to `False` during class initialization. If the option is set to `'logo'`, the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the `group_attributes`. Only possible if `group_datasets_by_attributes` is given. If the option is set to an integer `n` (`n != 0`), the (constant) error is estimated as RMSEP using `n`-fold cross-validation.

- **save_lime_importance** (*bool, optional (default: False)*) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool, optional (default: False)*) – Additionally saves propagated errors from `prediction_input_error` datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments, optional*) – Additional options for the final regressors `predict()` function.

Raises

- **RuntimeError** – `return_var` and `return_cov` are both set to `True`.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for `save_mlr_model_error` is given.
- **ValueError** – `save_propagated_errors` is `True` and no `prediction_input_error` data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(logo=False)

Print all available regression metrics for training data.

Parameters

logo (*bool, optional (default: False)*) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when `group_datasets_by_attributes` was given during class initialization.

classmethod register_mlr_model(mlr_model_type)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(kwargs)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(params)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments, optional*) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

Support Vector Regression

Support Vector Regression model.

Use `mlr_model_type: svr` to use this MLR model in the recipe.

Classes:

<code>SVRModel(input_datasets, **kwargs)</code>	Support Vector Regression model.
---	----------------------------------

class `esmvaltool.diag_scripts.mlr.models.svr.SVRModel(input_datasets, **kwargs)`

Bases: `MLRModel`

Support Vector Regression model.

Attributes:

<code>categorical_features</code>	Categorical features.
<code>data</code>	Input data of the MLR model.
<code>features</code>	Features of the input data.
<code>features_after_preprocessing</code>	Features of the input data after preprocessing.
<code>features_types</code>	Types of the features.
<code>features_units</code>	Units of the features.
<code>fit_kwargs</code>	Keyword arguments for <code>fit()</code> .
<code>group_attributes</code>	Group attributes of the input data.
<code>label</code>	Label of the input data.
<code>label_units</code>	Units of the label.
<code>mlr_model_type</code>	MLR model type.
<code>numerical_features</code>	Numerical features.
<code>parameters</code>	Parameters of the complete MLR model pipeline.

Methods:

<code>create(mlr_model_type, *args, **kwargs)</code>	Create desired MLR model subclass (factory method).
<code>efecv(**kwargs)</code>	Perform exhaustive feature elimination using cross-validation.
<code>export_prediction_data([filename])</code>	Export all prediction data contained in <code>self._data</code> .
<code>export_training_data([filename])</code>	Export all training data contained in <code>self._data</code> .
<code>fit()</code>	Fit MLR model.
<code>get_ancestors([label, features, ...])</code>	Return ancestor files.
<code>get_data_frame(data_type[, impute_nans])</code>	Return data frame of specified type.
<code>get_x_array(data_type[, impute_nans])</code>	Return x data of specific type.
<code>get_y_array(data_type[, impute_nans])</code>	Return y data of specific type.
<code>grid_search_cv(param_grid, **kwargs)</code>	Perform exhaustive parameter search using cross-validation.
<code>plot_1d_model([filename, n_points])</code>	Plot lineplot that represents the MLR model.
<code>plot_partial_dependences([filename])</code>	Plot partial dependences for every feature.
<code>plot_prediction_errors([filename])</code>	Plot predicted vs.
<code>plot_residuals([filename])</code>	Plot residuals of training and test (if available) data.
<code>plot_residuals_distribution([filename])</code>	Plot distribution of residuals of training and test data (KDE).
<code>plot_residuals_histogram([filename])</code>	Plot histogram of residuals of training and test data.
<code>plot_scatterplots([filename])</code>	Plot scatterplots label vs.
<code>predict([save_mlr_model_error, ...])</code>	Perform prediction using the MLR model(s) and write *.nc files.
<code>print_correlation_matrices()</code>	Print correlation matrices for all datasets.
<code>print_regression_metrics([logo])</code>	Print all available regression metrics for training data.
<code>register_mlr_model(mlr_model_type)</code>	Add MLR model (subclass of this class) (decorator).
<code>reset_pipeline()</code>	Reset regressor pipeline.
<code>rfecv(**kwargs)</code>	Perform recursive feature elimination using cross-validation.
<code>test_normality_of_residuals()</code>	Perform Shapiro-Wilk test to normality of residuals.
<code>update_parameters(**params)</code>	Update parameters of the whole pipeline.

property categorical_features

Categorical features.

Type

`numpy.ndarray`

classmethod create(mlr_model_type, *args, **kwargs)

Create desired MLR model subclass (factory method).

property data

Input data of the MLR model.

Type

`dict`

efecv(**kwargs)

Perform exhaustive feature elimination using cross-validation.

Parameters

****kwargs** (keyword arguments, optional) – Additional options for `esmvaltool.diag_scripts.mlr.custom_sklearn.cross_val_score_weighted()`.

export_prediction_data(*filename=None*)

Export all prediction data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}_{pred_name}.csv')) – Name of the exported files.

export_training_data(*filename=None*)

Export all training data contained in *self._data*.

Parameters

filename (*str*, optional (default: '{data_type}.csv')) – Name of the exported files.

property features

Features of the input data.

Type

`numpy.ndarray`

property features_after_preprocessing

Features of the input data after preprocessing.

Type

`numpy.ndarray`

property features_types

Types of the features.

Type

`pandas.Series`

property features_units

Units of the features.

Type

`pandas.Series`

fit()

Fit MLR model.

Note: Specifying keyword arguments for this function is not allowed here since `features_after_preprocessing` might be altered by that. Use the keyword argument `fit_kwargs` during class initialization instead.

property fit_kwargs

Keyword arguments for `fit()`.

Type

`dict`

get_ancestors(*label=True, features=None, prediction_names=None, prediction_reference=False*)

Return ancestor files.

Parameters

- **label** (*bool*, optional (default: `True`)) – Return label files.
- **features** (*list of str*, optional (default: `None`)) – Features for which files should be returned. If `None`, return files for all features.

- **prediction_names** (*list of str, optional (default: None)*) – Prediction names for which files should be returned. If None, return files for all prediction names.
- **prediction_reference** (*bool, optional (default: False)*) – Return prediction_reference files if available for given prediction_names.

Returns

Ancestor files.

Return type

list of str

Raises

ValueError – Invalid feature or prediction_name given.

get_data_frame(*data_type, impute_nans=False*)

Return data frame of specified type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

pandas.DataFrame

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_x_array(*data_type, impute_nans=False*)

Return x data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

numpy.ndarray

Raises

TypeError – data_type is invalid or data does not exist (e.g. test data is not set).

get_y_array(*data_type, impute_nans=False*)

Return y data of specific type.

Parameters

- **data_type** (*str*) – Data type to be returned. Must be one of 'all', 'train' or 'test'.
- **impute_nans** (*bool, optional (default: False)*) – Impute nans if desired.

Returns

Desired data.

Return type

`numpy.ndarray`

Raises

TypeError – `data_type` is invalid or data does not exist (e.g. test data is not set).

grid_search_cv(*param_grid*, ***kwargs*)

Perform exhaustive parameter search using cross-validation.

Parameters

- **param_grid** (*dict* or *list of dict*) – Parameter names (keys) and ranges (values) for the search. Have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.
- ****kwargs** (*keyword arguments, optional*) – Additional options for `sklearn.model_selection.GridSearchCV`.

Raises

ValueError – Final regressor does not supply the attributes `best_estimator_` or `best_params_`.

property group_attributes

Group attributes of the input data.

Type

`numpy.ndarray`

property label

Label of the input data.

Type

`str`

property label_units

Units of the label.

Type

`str`

property mlr_model_type

MLR model type.

Type

`str`

property numerical_features

Numerical features.

Type

`numpy.ndarray`

property parameters

Parameters of the complete MLR model pipeline.

Type

`dict`

`plot_1d_model(filename=None, n_points=1000)`

Plot lineplot that represents the MLR model.

Note: This only works for a model with a single feature.

Parameters

- **filename** (*str*, optional (default: '1d_mlr_model')) – Name of the plot file.
- **n_points** (*int*, optional (default: 1000)) – Number of sampled points for the single feature (using linear spacing between minimum and maximum value).

Raises

- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – MLR model is built from more than 1 feature.

`plot_partial_dependences(filename=None)`

Plot partial dependences for every feature.

Parameters

filename (*str*, optional (default: 'partial_dependence_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_prediction_errors(filename=None)`

Plot predicted vs. true values.

Parameters

filename (*str*, optional (default: 'prediction_errors')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals(filename=None)`

Plot residuals of training and test (if available) data.

Parameters

filename (*str*, optional (default: 'residuals')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

`plot_residuals_distribution(filename=None)`

Plot distribution of residuals of training and test data (KDE).

Parameters

filename (*str*, optional (default: 'residuals_distribution')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_residuals_histogram(*filename=None*)

Plot histogram of residuals of training and test data.

Parameters

filename (*str*, optional (default: 'residuals_histogram')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

plot_scatterplots(*filename=None*)

Plot scatterplots label vs. feature for every feature.

Parameters

filename (*str*, optional (default: 'scatterplot_{feature}')) – Name of the plot file.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

predict(*save_mlr_model_error=None*, *save_lime_importance=False*, *save_propagated_errors=False*, ***kwargs*)

Perform prediction using the MLR model(s) and write *.nc files.

Parameters

- **save_mlr_model_error** (*str* or *int*, optional) – Additionally saves estimated squared MLR model error. This error represents the uncertainty of the prediction caused by the MLR model itself and not by errors in the prediction input data (errors in that will be considered by including datasets with *var_type* set to *prediction_input_error* and setting *save_propagated_errors* to *True*). If the option is set to 'test', the (constant) error is estimated as RMSEP using a (hold-out) test data set. Only possible if test data is available, i.e. the option *test_size* is not set to *False* during class initialization. If the option is set to 'logo', the (constant) error is estimated as RMSEP using leave-one-group-out cross-validation using the *group_attributes*. Only possible if *group_datasets_by_attributes* is given. If the option is set to an integer *n* (*n* != 0), the (constant) error is estimated as RMSEP using *n*-fold cross-validation.
- **save_lime_importance** (*bool*, optional (default: *False*)) – Additionally saves local feature importance given by LIME (Local Interpretable Model-agnostic Explanations).
- **save_propagated_errors** (*bool*, optional (default: *False*)) – Additionally saves propagated errors from *prediction_input_error* datasets. Only possible when these are available.
- ****kwargs** (*keyword arguments*, optional) – Additional options for the final regressors *predict()* function.

Raises

- **RuntimeError** – *return_var* and *return_cov* are both set to *True*.
- **sklearn.exceptions.NotFittedError** – MLR model is not fitted.
- **ValueError** – An invalid value for *save_mlr_model_error* is given.
- **ValueError** – *save_propagated_errors* is *True* and no *prediction_input_error* data is available.

print_correlation_matrices()

Print correlation matrices for all datasets.

print_regression_metrics(*logo=False*)

Print all available regression metrics for training data.

Parameters

logo (*bool*, optional (default: *False*)) – Print regression metrics using `sklearn.model_selection.LeaveOneGroupOut` cross-validation. Only possible when *group_datasets_by_attributes* was given during class initialization.

classmethod register_mlr_model(*mlr_model_type*)

Add MLR model (subclass of this class) (decorator).

reset_pipeline()

Reset regressor pipeline.

rfecv(*kwargs*)**

Perform recursive feature elimination using cross-validation.

Note: This only works for final estimators that provide information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

Parameters

****kwargs** (*keyword arguments*, optional) – Additional options for `sklearn.feature_selection.RFECV`.

Raises

RuntimeError – Final estimator does not provide `coef_` or `feature_importances_` attribute.

test_normality_of_residuals()

Perform Shapiro-Wilk test to normality of residuals.

Raises

sklearn.exceptions.NotFittedError – MLR model is not fitted.

update_parameters(*params*)**

Update parameters of the whole pipeline.

Note: Parameter names have to be given for each step of the pipeline separated by two underscores, i.e. `s__p` is the parameter `p` for step `s`.

Parameters

****params** (*keyword arguments*, optional) – Parameters for the pipeline which should be updated.

Raises

ValueError – Invalid parameter for pipeline given.

50.3 Monitor Diagnostic

This module provides various tools to monitor climate model simulations. It can be used to plot arbitrary variables from arbitrary datasets.

50.3.1 Examples

- *Monitor*

50.3.2 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).

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` or `annual_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 [User configuration file](#)).

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

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 [User configuration file](#)).

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 show multiple datasets in one plot (incl. biases).

Description

This diagnostic can be used to visualize multiple datasets in one plot.

For some plot types, a reference dataset can be defined. For this, use the facet `reference_for_monitor_diags: true` in the definition of the dataset in the recipe. Note that at most one reference dataset per variable is supported.

Currently supported plot types (use the option `plots` to specify them):

- Time series (plot type `timeseries`): for each variable separately, all datasets are plotted in one single figure. Input data needs to be 1D with single dimension *time*.
- Maps (plot type `map`): for each variable and dataset, an individual map is plotted. If a reference dataset is defined, also include this dataset and a bias plot into the figure. Note that if a reference dataset is defined, all input datasets need to be given on the same horizontal grid (you can use the preprocessor `esmvalcore.preprocessor.regrid()` for this). Input data needs to be 2D with dimensions *latitude*, *longitude*.
- Vertical profiles (plot type `profile`): for each variable and dataset, an individual profile is plotted. If a reference dataset is defined, also include this dataset and a bias plot into the figure. 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). Input data needs to be 2D with dimensions *latitude*, *height/air_pressure*.

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`.

plots: dict, optional

Plot types plotted by this diagnostic (see list above). Dictionary keys must be `timeseries`, `map`, or `profile`. 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 correspond-

ing tags). `{plot_dir}` is replaced with the default ESMValTool plot directory (i.e., `output_dir/plots/diagnostic_name/script_name/`, see [User configuration file](#)).

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()` (affects all plots). By default, uses `style: ticks`.

Configuration options for plot type timeseries

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.

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]`.

Configuration options for plot type map

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}`. This option has no effect if no reference dataset is given.

cbar_kwargs: dict, optional

Optional keyword arguments for `matplotlib.pyplot.colorbar()`. By default, uses `orientation: horizontal`, `aspect: 30`.

cbar_kwargs_bias: dict, optional

Optional keyword arguments for `matplotlib.pyplot.colorbar()` for plotting biases. These keyword argu-

ments update (and potentially overwrite) the `cbar_kwargs` for the bias plot. This option has no effect if no reference dataset is given.

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 `vmix` 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, the use of the `plot_kwargs` `vmin` and `vmax` or `levels` is highly recommend when using this `common_cbar: True`. This option has no effect if no reference dataset is given.

fontsize: int, optional (default: 10)

Fontsize used for ticks, labels and titles. For the latter, use the given fontsize plus 2. Does not affect subtitles.

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.

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 cubes with coordinates latitude and longitude.

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}`.

plot_kwargs_bias: dict, optional

Optional keyword arguments for the plot function defined by `plot_func` for plotting biases. These keyword arguments update (and potentially overwrite) the `plot_kwargs` for the bias plot. This option has no effect if no reference dataset is given. See option `plot_kwargs` for more details. By default, uses `cmap: bwr`.

projection: str, optional (default: 'Robinson')

Projection used for the map 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 the default projection Robinson, 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 single argument for these functions. 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: 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`).

show_stats: bool, optional (default: True)

Show basic statistics on the plots.

Configuration options for plot type profile

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}. This option has no effect if no reference dataset is given.

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. This option has no effect if no reference dataset is given.

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 `vmix` 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, the use of the `plot_kwargs` `vmin` and `vmax` or `levels` is highly recommend when using this `common_cbar: true`. This option has no effect if no reference dataset is given.

fontsize: int, optional (default: 10)

Fontsize used for ticks, labels and titles. For the latter, use the given fontsize plus 2. Does not affect subtitles.

log_y: bool, optional (default: True)

Use logarithmic Y-axis.

plot_func: str, optional (default: 'contourf')

Plot function used to plot the profiles. Must be a function of `iris.plot` that supports plotting of 2D cubes with coordinates latitude and height/air_pressure.

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}`.

plot_kwargs_bias: dict, optional

Optional keyword arguments for the plot function defined by `plot_func` for plotting biases. These keyword arguments update (and potentially overwrite) the `plot_kwargs` for the bias plot. This option has no effect if no reference dataset is given. See option `plot_kwargs` for more details. By default, uses `cmap: bwr`.

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 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: True)

If True, use `rasterization` for profile plots to produce smaller files. This is only relevant for vector graphics (e.g., `output_file_type=pdf,svg,ps`).

show_y_minor_ticklabels: bool, optional (default: False)

Show tick labels for the minor ticks on the Y axis.

show_stats: bool, optional (default: True)

Show basic statistics on the plots.

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.

50.3.3 Base class for monitoring diagnostics

Base class for monitoring diagnostics

Base class for monitoring diagnostics.

Classes:

<code>MonitorBase</code> (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:

<code>get_plot_folder</code> (var_info)	Get plot storage folder from variable info.
<code>get_plot_name</code> (plot_type, var_info[, add_ext])	Get plot filename from variable info.
<code>get_plot_path</code> (plot_type, var_info[, add_ext])	Get plot full path from variable info.
<code>get_provenance_record</code> (ancestor_files, **kwargs)	Create provenance record for the diagnostic data and plots.
<code>plot_cube</code> (cube, filename[, linestyle])	Plot a timeseries from a cube.
<code>plot_timeseries</code> (cube, var_info[, period])	Plot timeseries from a cube.
<code>record_plot_provenance</code> (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.

50.4 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

50.4.1 Maps diagnostics

Diagnostic to produce images of a map with coastlines from a cube. These plots show latitude vs longitude and the cube value is used as the colour scale.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

An appropriate 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 to 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:

<code>main(cfg)</code>	Load the config file, and send it to the plot makers.
<code>make_map_contour(cfg, metadata, filename)</code>	Make a simple contour map plot for an individual model.
<code>make_map_plots(cfg, metadata, filename)</code>	Make a simple map plot for an individual model.
<code>multi_model_contours(cfg, metadata)</code>	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.

50.4.2 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.

An appropriate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_map:
    extract_levels:
      levels: [100., ]
      scheme: linear_extrap
    climate_statistics:
      operator: mean
```

This diagnostic also requires the `exper_model`, `exper_model` and `observational_dataset` keys in the recipe:

```
diagnostics:
  diag_name:
    ...
  scripts:
    Global_Ocean_map:
      script: ocean/diagnostic_maps_quad.py
      exper_model: {Model 1 dataset details}
      control_model: {Model 2 dataset details}
      observational_dataset: {Observational dataset details}
```

This tool is part of the ocean diagnostic tools package in the ESMValTool, and was based on the plots produced by the Ocean Assess/Marine Assess toolkit.

Author: Lee de Mora (PML)

ledm@pml.ac.uk

Functions:

<code>add_map_subplot(subplot, cube, nspace[, ...])</code>	Add a map subplot to the current pyplot figure.
<code>main(cfg)</code>	Load the config file, and send it to the plot maker.
<code>multi_model_maps(cfg, input_files)</code>	Make the four pane model vs model vs obs comparison plot.

```
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.add_map_subplot(subplot, cube, nspace,  
                                                                    title="", cmap="")
```

Add a map subplot to the current pyplot figure.

Parameters

- **subplot** (*int*) – The matplotlib.pyplot subplot number. (ie 221)
- **cube** (*iris.cube.Cube*) – the iris cube to be plotted.
- **nspace** (*numpy.array*) – An array of the ticks of the colour part.
- **title** (*str*) – A string to set as the subplot title.
- **cmap** (*str*) – A string to describe the matplotlib colour map.

```
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.main(cfg)
```

Load the config file, and send it to the plot maker.

Parameters

cfg (*dict*) – the opened global config dictionary, passed by ESMValTool.

```
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad.multi_model_maps(cfg, input_files)
```

Make the four pane model vs model vs obs comparison plot.

Parameters

- **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
- **input_files** (*dict*) – the metadata dictionary

50.4.3 Model vs Observations maps Diagnostic.

Diagnostic to produce comparison of model and data. The first kind of image shows four maps and the other shows a scatter plot.

The four pane image is a latitude vs longitude figures showing:

- Top left: model
- Top right: observations
- Bottom left: model minus observations
- Bottom right: model over observations

The scatter plots plot the matched model coordinate on the x axis, and the observational dataset on the y coordinate, then performs a linear regression of those data and plots the line of best fit on the plot. The parameters of the fit are also shown on the figure.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, a small number of depth layers, and a latitude and longitude coordinates.

An appropriate preprocessor for a 3D + time field would be:

```
preprocessors:  
  prep_map:  
    extract_levels:  
      levels: [100., ]  
      scheme: linear_extrap  
    climate_statistics:
```

(continues on next page)

(continued from previous page)

```

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:

<code>add_linear_regression(plot_axes, arr_x, arr_y)</code>	Add a straight line fit to an axis.
<code>add_map_subplot(subplot, cube, nspace[, ...])</code>	Add a map subplot to the current pyplot figure.
<code>main(cfg)</code>	Load the config file, and send it to the plot maker.
<code>make_model_vs_obs_plots(cfg, metadata, ...)</code>	Make a figure showing four maps and the other shows a scatter plot.
<code>make_scatter(cfg, metadata, model_filename, ...)</code>	Makes Scatter plots of model vs observational data.
<code>rounds_sig(value[, sig])</code>	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 dictionary, 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – the input files dictionary
- **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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – the input files dictionary
- **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

50.4.4 Profile diagnostics.

Diagnostic to produce figure 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 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 appropriate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_profile:
    extract_volume:
      long1: 0.
      long2: 20.
      lat1: -30.
      lat2: 30.
      z_min: 0.
      z_max: 3000.
    area_statistics:
      operator: mean
```

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:

<code>determine_profiles_str(cube)</code>	Determine a string from the cube, to describe the profile.
<code>main(cfg)</code>	Run the diagnostics profile tool.
<code>make_profiles_plots(cfg, metadata, filename)</code>	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 dictionary, passed by ESMValTool.

```
esmvaltool.diag_scripts.ocean.diagnostic_profiles.make_profiles_plots(cfg, metadata, filename,  
                                                                    obs_metadata={},  
                                                                    obs_filename="")
```

Make a profile plot for an individual model.

The optional observational dataset can also be added.

Parameters

- **cfg** (*dict*) – the opened global config dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary for a specific model.
- **filename** (*str*) – The preprocessed model file.
- **obs_metadata** (*dict*) – The metadata dictionary for the observational dataset.
- **obs_filename** (*str*) – The preprocessed observational dataset file.

50.4.5 Time series diagnostics

Diagnostic to produce figures of the time development of a field from cubes. 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 is a multi-model mean made by the `_multimodel.py` preprocessor. 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 appropriate preprocessor for a 3D+time field would be:

```
preprocessors:  
  prep_timeseries_1: # For Global Volume Averaged  
    volume_statistics:  
      operator: mean
```

An appropriate 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 appropriate 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:

<code>main(cfg)</code>	Load the config file and some metadata, then pass them the plot making tools.
<code>make_time_series_plots(cfg, metadata, filename)</code>	Make a simple time series plot for an individual model 1D cube.
<code>moving_average(cube, window)</code>	Calculate a moving average.
<code>multi_model_time_series(cfg, metadata)</code>	Make a time series plot showing several preprocessed datasets.
<code>timeplot(cube, **kwargs)</code>	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 dictionary, passed by ESMValTool.

`esmvaltool.diag_scripts.ocean.diagnostic_timeseries.make_time_series_plots(cfg, metadata, filename)`

Make a simple time series plot for an individual 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 measurement 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 preprocessed 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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

50.4.6 Transects diagnostics

Diagnostic to produce images of a transect. These plost show either latitude or longitude against depth, and the cube value is used as the colour scale.

Note that this diagnostic assumes that the preprocessors do the bulk of the hard work, and that the cube received by this diagnostic (via the settings.yml and metadata.yml files) has no time component, and one of the latitude or longitude coordinates has been reduced to a single value.

An approporate preprocessor for a 3D+time field would be:

```
preprocessors:
  prep_transect:
    climate_statistics:
```

(continues on next page)

(continued from previous page)

```

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:

<code>add_sea_floor(cube)</code>	Add a simple sea floor line from the cube mask.
<code>determine_set_y_logscale(cfg, metadata)</code>	Determine whether to use a log scale y axis.
<code>determine_transect_str(cube[, region])</code>	Determine the Transect String.
<code>main(cfg)</code>	Load the config file and some metadata, then pass them the plot making tools.
<code>make_cube_region_dict(cube)</code>	Take a cube and return a dictionary region: cube.
<code>make_depth_safe(cube)</code>	Make the depth coordinate safe.
<code>make_transect_contours(cfg, metadata, filename)</code>	Make a contour plot of the transect for an individual model.
<code>make_transects_plots(cfg, metadata, filename)</code>	Make a simple plot of the transect for an individual model.
<code>multi_model_contours(cfg, metadatas)</code>	Make a multi model comparison plot showing several transect contour plots.
<code>titlify(title)</code>	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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 dictionary, passed by ESMValTool.

```
esmvaltool.diag_scripts.ocean.diagnostic_transects.make_cube_region_dict(cube)
```

Take a cube and return a dictionary 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 dictionary 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 individual 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 individual 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 dictionary, passed by ESMValTool.
- **metadatas** (*dict*) – The metadatas dictionary 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.

50.4.7 Sea Ice Diagnostics.

Diagnostic to produce a series of images which are useful for evaluating the behaviour of the a sea ice model.

There are three kinds of plots shown here. 1. Sea ice Extent maps plots with a stereoscopic 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 appropriate 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 to 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:

<code>agregate_by_season(cube)</code>	Aggregate the cube into seasonal means.
<code>calculate_area_time_series(cube, plot_type, ...)</code>	Calculate the area of unmasked cube cells.
<code>create_ice_cmap([threshold])</code>	Create colour map with ocean blue below a threshold and white above.
<code>get_pole(cube)</code>	Figure out the hemisphere and returns it as a string (North or South).
<code>get_season(cube)</code>	Return a climatological season time string.
<code>get_time_string(cube)</code>	Return a climatological season string in the format: "year season".
<code>get_year(cube)</code>	Return the cube year as a string.
<code>main(cfg)</code>	Load the config file and metadata, then pass them the plot making tools.
<code>make_map_extent_plots(cfg, metadata, filename)</code>	Make an extent map plot showing several times for an individual model.
<code>make_map_plots(cfg, metadata, filename)</code>	Make a simple map plot for an individual model.
<code>make_polar_map(cube[, pole, cmap])</code>	Make a polar stereoscopic map plot.
<code>make_ts_plots(cfg, metadata, filename)</code>	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 dictionary, 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary 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 dictionary, passed by ESMValTool.
- **metadata** (*dict*) – The metadata dictionary for a specific model.
- **filename** (*str*) – The preprocessed model file.

50.4.8 Diagnostic tools

This module contains several python tools used elsewhere by the ocean diagnostics package.

This tool is part of the ocean diagnostic tools package in the ESMValTool.

Author: Lee de Mora (PML)

ledm@pml.ac.uk

Functions:

<code>add_legend_outside_right(plot_details, ax1)</code>	Add a legend outside the plot, to the right.
<code>bgc_units(cube, name)</code>	Convert the cubes into some friendlier units.
<code>cube_time_to_float(cube)</code>	Convert from time coordinate into decimal time.
<code>decadal_average(cube)</code>	Calculate the decadal_average.
<code>folder(name)</code>	Make a directory out of a string or list or strings.
<code>get_array_range(arrays)</code>	Determine the minimum and maximum values of a list of arrays..
<code>get_colour_from_cmap(number, total[, cmap])</code>	Get a colour <i>number</i> of <i>total</i> from a cmap.
<code>get_cube_range(cubes)</code>	Determine the minimum and maximum values of a list of cubes.
<code>get_cube_range_diff(cubes)</code>	Determine the largest deviation from zero in an list of cubes.
<code>get_decade(coord, value)</code>	Determine the decade.
<code>get_image_format(cfg[, default])</code>	Load the image format from the global config file.
<code>get_image_path(cfg, metadata[, prefix, ...])</code>	Produce a path to the final location of the image.
<code>get_input_files(cfg[, index])</code>	Load input configuration file as a Dictionary.
<code>get_obs_projects()</code>	Return a list of strings with the names of observations projects.
<code>guess_calendar_datetime(cube)</code>	Guess the cftime.datetime form to create datetimes.
<code>load_thresholds(cfg, metadata)</code>	Load the thresholds for contour plots from the config files.
<code>make_cube_layer_dict(cube)</code>	Take a cube and return a dictionary layer:cube
<code>match_model_to_key(model_type, cfg_dict, ...)</code>	Match up model or observations dataset dictionaries from config file.
<code>prepare_provenance_record(cfg, ...)</code>	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 preferred over Kelvin, etc.)

Parameters

- **cube** (*iris.cube.Cube*) – the opened dataset as a cube.
- **name** (*str*) – The string describing the data field.

Returns

the cube with the new units.

Return type*iris.cube.Cube*`esmvaltool.diag_scripts.ocean.diagnostic_tools.cube_time_to_float(cube)`

Convert from time coordinate into decimal time.

Takes an iris time coordinate and returns a list of floats. :param cube: the opened dataset as a cube. :type cube: *iris.cube.Cube*

Returns

List of floats showing the time coordinate in decimal time.

Return type*list*`esmvaltool.diag_scripts.ocean.diagnostic_tools.decadal_average(cube)`

Calculate the decadal_average.

Parameters

cube (*iris.cube.Cube*) – The input cube

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)`

Determines 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 minimum 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)`

Determine 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 minimum and maximum values of the list of cubes.

Return type

`list`

`esmvaltool.diag_scripts.ocean.diagnostic_tools.get_cube_range_diff(cubes)`

Determine 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 extension.

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 Dictionary.

Get a dictionary with input files from the `metadata.yml` files. This is a wrapper 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 dictionaries from config file.

This function checks that the `control_model`, `exper_model` and `observational_dataset` dictionaries 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.

50.5 Psyplot Diagnostic

Create arbitrary Psyplot plots.

50.5.1 Description

This diagnostic provides a high-level interface to Psyplot.

50.5.2 Author

Manuel Schlund (DLR, Germany)

Notes

For each input dataset, an individual plot is created. This diagnostic supports arbitrary variables of arbitrary datasets.

50.5.3 Configuration options in recipe

psyplot_func: str

Function used to plot the data. Must be a function of `psyplot.project.plot`. Run `python -c "from psyplot.project import plot; print(plot.show_plot_methods())"` to get a list of all currently supported plotting functions (make sure to run this command in your ESMValTool environment).

psyplot_kwargs: dict, optional

Optional keyword arguments for the plotting function given by `psyplot_func`. String arguments can include facets in curly brackets which will be derived from the corresponding dataset, e.g., `clabel: '{long_name} [{units}]'`, `title: '{long_name} Climatology of {dataset} ({start_year}-{end_year})'`.

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()` (affects all plots).

Part XI

Frequently Asked Questions

IS THERE A MAILING LIST?

Yes, you can subscribe to the ESMValTool user mailing list and join the discussion on general topics (installation, configuration, etc). See [User mailing list](#).

WHAT IS YAML?

While `.yaml` or `.yml` is a relatively common format, users may not have encountered this language before. The key information about this format is:

- yaml is a human friendly markup language;
- yaml is commonly used for configuration files (gradually replacing the venerable `.ini`);
- the syntax is relatively straightforward;
- indentation matters a lot (like Python)!
- yaml is case sensitive;

More information can be found in the [yaml tutorial](#) and [yaml quick reference card](#). ESMValTool uses the [yamllint](#) linter tool to check recipe syntax.

RE-RUNNING DIAGNOSTICS

If a diagnostic fails, you will get the message

INFO	To re-run this diagnostic script, run:
------	--

If you run the command in the stdout you will be able to re-run the diagnostic without having to re-run the whole preprocessor. If you add the `-f` argument (available only for Python diagnostics, check your options with `--help`) that will force an overwrite, and it will delete not just the failed diagnostic, but the contents of its `work_dir` and `plot_dir` directories - this is useful when needing to redo the whole work. Adding `-i` or `--ignore-existing` will not delete any existing files, and it can be used to skip work that was already done successfully, provided that the diagnostic script supports this.

ENTER INTERACTIVE MODE WITH IPYTHON

Sometimes it is useful to enter an interactive session to have a look what's going on. Insert a single line in the code where you want to enter IPython: `import IPython; IPython.embed()`

This is a useful functionality because it allows the user to *fix* things on-the-fly and after quitting the IPython console, code execution continues as per normal.

USE MULTIPLE CONFIG-USER.YML FILES

The user selects the configuration yaml file at run time. It's possible to have several configurations files. For instance, it may be practical to have one config file for debugging runs and another for production runs.

CREATE A SYMBOLIC LINK TO THE LATEST OUTPUT DIRECTORY

When running multiple times the same recipe, the tool creates separate output directories sorted by the time tag that they were created at; sometimes, when running quite a few times, it is not straightforward to detect which one is the *latest* output directory, so a symbolic link attached to it would make things more clear e.g.:

```
recipe_example_20190905_163431
recipe_example_20190905_163519
recipe_example_latest -> recipe_example_20190905_163519
```

You can do that by running the tool using the latest output as basis and creating a symbolic link to it so it gets picked up at every re-run iteration:

```
esmvaltool run recipe_example.yml; \
ln -sfT $(ls -ld ~/esmvaltool_output/recipe_example_* | tail -1) ~/esmvaltool_output/
↪recipe_example_latest
```


CAN ESMVALTOOL PLOT ARBITRARY MODEL OUTPUT?

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.

Moreover, recipe *Psyplot Diagnostics* and the corresponding diagnostic *psyplot_diag.py* provide a high-level interface to the *Psyplot* package which can be used to create a large variety of different plots.

Part XII

Changelog

58.1 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](#)

58.2 Backwards incompatible changes

- Remove installation of R dependencies from the help message ([#2761](#)) [Rémi Kazeroni](#)

58.3 Bug fixes

- Fix misplaced provenance records from IPCC AR5 Ch.12 diags ([#2758](#)) [Axel Lauer](#)
- Fix `esmvaltool.utils.testing.regression.compare` module to run with Python<3.10 too ([#2778](#)) [Valeriu Predoi](#)
- Fixed small bug that could lead to wrong pr units in `monitor/multi_datasets.py` ([#2788](#)) [Manuel Schlund](#)
- Pin `xgboost>1.6.1` so we avert documentation failing to build with *1.6.1* ([#2780](#)) [Valeriu Predoi](#)
- Pin `matplotlib-base<3.6.0` to avoid conflict from `mapgenerator` that fails doc builds ([#2830](#)) [Valeriu Predoi](#)
- Fixed wrong latitudes in NDP CMORizer ([#2832](#)) [Manuel Schlund](#)
- Fix indexer in Autoassess supermeans module use a tuple of `(slice(), idx, idx)` ([#2838](#)) [Valeriu Predoi](#)
- Replace xarray ufuncs with bogstandard numpy in `weighting/climwip/calibrate_sigmas.py` ([#2848](#)) [Valeriu Predoi](#)
- Fix units MERRA2 CMORizer ([#2850](#)) [Axel Lauer](#)
- Fix bug when using log-scale y-axis for ocean transects. ([#2862](#)) [Tomas Torsvik](#)

58.4 Community

- Add MO-paths to config file (#2784) [mo-tgeddes](#)

58.5 Deprecations

- Recipe *recipe_esacci_oc.yml* replace with new regrid scheme *nearest_extrapolate* (#2841) [Valeriu Predoi](#)

58.6 Documentation

- Update release schedule for v2.7 (#2747) [Bouwe Andela](#)
- Add Met Office installation method (#2751) [mo-tgeddes](#)
- Add release dates for 2023 (#2769) [Rémi Kazeroni](#)
- Made *maintainer* entry mandatory for published recipes (#2703) [Manuel Schlund](#)
- Use command with current command line opts for *cffconvert* in documentation (#2791) [Valeriu Predoi](#)
- Update CMORizer documentation with command options (#2795) [Rémi Kazeroni](#)
- Fixed broken link for monthly meetings (#2806) [Rémi Kazeroni](#)
- Update MO obs4MIPs paths in the user configuration file (#2813) [mo-tgeddes](#)
- Fix Windows incompatible file names in documentation of *recipe_climate_change_hotspot.yml* (#2823) [Lee de Mora](#)
- Update documentation for the Landschuetzer 2016 recipe. (#2801) [Tomas Torsvik](#)
- Fixed anaconda badge in README (#2866) [Valeriu Predoi](#)
- Update release strategy notes (#2734) [sloosvel](#)
- Add documentation on how to handle CMORizers for multiple dataset versions (#2730) [Rémi Kazeroni](#)
- Extending documentation: recipe maintainer + broken recipe policy (#2719) [Axel Lauer](#)

58.7 Diagnostics

- Recipe and diagnostics for : Tebaldi et al.,ESD, 2021 (#2052) [debe-kevin](#)
- Figures for IPCC AR6 WG1 Chapter 3 (Atmosphere) (#2533) [Lisa Bock](#)

58.8 Observational and re-analysis dataset support

- Update CERES-EBAF to Ed4.1 (#2752) [Axel Lauer](#)
- New CMORizer for CALIPSO-ICECLOUD (#2753) [Axel Lauer](#)
- New CMORizer for CLOUDSAT-L2 (#2754) [Axel Lauer](#)
- Update MERRA2 cmorizer with extra 2D and 3D variables (#2774) [Valeriu Predoi](#)

58.9 Automatic testing

- Pin *netcdf4* != 1.6.1 since that is spitting large numbers of SegFaults (#2796) [Valeriu Predoi](#)

58.10 Installation

- Increase esmvalcore version to 2.7.0 in environment files (#2860) [Valeriu Predoi](#)
- Add iris-esmf-regrid as a dependency (#2880) [Klaus Zimmermann](#)

58.11 Improvements

- Fix tebaldi21esd (#2749) [Axel Lauer](#)
- Added option to show basic statistics in plots of *monitor/multi_datasets.py* (#2790) [Manuel Schlund](#)
- Remove retracted datasets from *recipe_climate_change_hotspot* (#2854) [sloosvel](#)

59.1 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 automatized 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

59.2 Bug fixes

- Fix dtype for Marrmot recipe results (#2646) [SarahAlidoost](#)
- Adapt test_fix_coords to new version of cf-units (#2707) [Klaus Zimmermann](#)
- Fix nested axes in *recipe_martin18_grl* and *recipe_li17natcc* (#2712) [Lukas](#)
- Update common_climindex_preprocessing_for_plots.R (#2727) [Enrico Arnone](#)

59.3 Community

- Collecting github user names for config-references (#2677) [Lukas](#)

59.4 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. (#2655) Manuel Schlund

59.5 Documentation

- Documentation Improvements (#2580) stacristo
- Fixed broken label in the documentation (#2616) Rémi Kazeroni
- Add readthedocs configuration file (#2627) Bouwe Andela
- Update the command for building the documentation (#2622) Bouwe Andela
- Added DKRZ-Levante to `config-user-example.yml` (#2632) Rémi Kazeroni
- Improved documentation on native dataset support (#2635) Manuel Schlund
- Add documentation on building and uploading Docker images (#2662) Bouwe Andela
- Remove support for Mistral in `config-user-example.yml` (#2667) Rémi Kazeroni
- Add note to clarify that CORDEX support is work in progress (#2682) Bouwe Andela
- Restore accidentally deleted text from input data docs (#2683) Bouwe Andela
- Add running settings note in `recipe_wenzel16nat.yml` documentation (#2692) sloosvel
- Add a note on transferring permissions to the release manager (#2688) Bouwe Andela
- Update documentation on ESMValTool module at DKRZ (#2696) Rémi Kazeroni
- Add note on how to run `recipe_wenzel14jgr.yml` (#2717) sloosvel
- Added conda forge feedstock repo link in README (#2555) Valeriu Predoi

59.6 Diagnostics

- Compute bias instead of correlation in `compare_salinity.py` (#2642) sloosvel
- Update monitor diagnostics (#2608) Manuel Schlund
- Add new Psypplot diagnostic (#2653) Manuel Schlund
- Reduce memory usage of lisflood recipe (#2634) Stefan Verhoeven
- Provenance in ocean diagnostics (#2651) Tomas Lovato
- Extend monitor diagnostics with multi-dataset plots (#2657) Manuel Schlund
- Recipe and diagnostics to plot climate change hotspots: Cos et al., ESD 2022 (#2614) Pep Cos
- Update plots of consecutive dry days recipe (#2671) Bouwe Andela
- Fix the format of ids in Hype forcing files (#2679) SarahAlidoost
- WFlow diagnostic script: remove manual rechunking (#2680) Peter Kalverla

59.7 Observational and re-analysis dataset support

- Extending the HadCRUT5 cmorizer (#2509) [Lisa Bock](#)
- Cmorize Kadow2020 dataset (#2513) [Lisa Bock](#)
- Cmorize NOAAGlobalTemp dataset (#2515) [Lisa Bock](#)
- Add option to CMORize ts as tos in ESACCI data (#2731) [sloosvel](#)

59.8 Automatic testing

- Add a tool for comparing recipe runs to previous runs (#2613) [Bouwe Andela](#)
- Ignore NCL interface files when comparing recipe runs (#2673) [Bouwe Andela](#)
- Add a short version of recipe deangelis15nat for testing (#2685) [katjaweigel](#)
- Expanded recipe output comparison tool to better handle absolute paths in output (#2709) [Manuel Schlund](#)
- Update development infrastructure (#2663) [Bouwe Andela](#)

59.9 Installation

- Removed *package/meta.yaml* and all references to it (#2612) [Manuel Schlund](#)

59.10 Improvements

- Improved handling of weights in MLR diagnostics (#2625) [Manuel Schlund](#)
- Fixed order of variables in perfemetrics plot of Anav13jclim recipe (#2706) [Manuel Schlund](#)
- Added input file sorting to many diagnostic to make output exactly reproducible (#2710) [Manuel Schlund](#)
- Removed 'ancestors' attributes before saving netcdf files in emergent constraints diagnostics (#2713) [Manuel Schlund](#)

60.1 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

60.2 Backwards incompatible changes

- Streamline observations download (#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 (#2585) [Manuel Schlund](#). 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](#)).

60.3 Bug fixes

- Remove the use of *esmvalgroup* channel from the conda install Github Action workflow (#2420) Valeriu Predoi
- Ignore *.pymon-journal* file in test discovery (#2491) Klaus Zimmermann
- Relocate pytest-monitor outputted database *.pymon* so *.pymon-journal* file should not be looked for by *pytest* (#2501) Valeriu Predoi
- Re-establish Python 3.7 compatibility (#2506) Klaus Zimmermann
- Update intersphinx mapping (#2531) Klaus Zimmermann
- Fixed *KeyError* in *recipe_ocean_bgc.yml* (#2540) Manuel Schlund
- Corrected ESACCI-SEA-SURFACE-SALINITY from OBS to OBS6 (#2542) Axel Lauer
- Fixed *recipe_kcs.yml* (#2541) Manuel Schlund
- Fix MDER diagnostic regression_stepwise (#2545) Axel Lauer
- Fix for *recipe_wenzel16nat* (#2547) Axel Lauer
- Fixed *recipe_carvalhais14nat* and removed deprecated use of *np.float* (#2558) Manuel Schlund
- Fix *recipe_wenzel14jgr* (#2577) Rémi Kazeroni
- Fixed various recipes by removing faulty or non-available datasets (#2563) Manuel Schlund
- Remove missing CMIP5 data from 2 recipes (#2579) Rémi Kazeroni
- Fix *recipe_seaice* (#2578) Rémi Kazeroni
- Fix *recipe_climwip_brunner20esd* (#2581) Rémi Kazeroni

60.4 Deprecations

- Remove *--use-feature=2020-resolver* command line option for obsolete pip 2020 solver (#2493) Valeriu Predoi
- Renamed vertical regridding schemes in affected recipes (#2487) Manuel Schlund

60.5 Documentation

- Update release manager for v2.5 (#2429) Axel Lauer
- Mention ENES Climate Analytics service (#2438) Bouwe Andela
- Add recipe overview page (#2439) Bouwe Andela
- Fix pointer to Tutorial lesson on preprocessor from 05 to 06 (#2473) Valeriu Predoi
- Removed obsolete option *synda-download* from documentation (#2485) Manuel Schlund
- Update CMUG XCH4 docu figure (#2502) Axel Lauer
- Add Python=3.10 to package info, update Circle CI auto install and documentation for Python=3.10 (#2503) Manuel Schlund
- Unify user configuration file (#2507) Manuel Schlund
- Synchronized *config-user.yml* with version from ESMValCore (#2516) Manuel Schlund

- CITATION.cff fix and automatic validation of your citation metadata (#2517) Abel Siqueira
- Add backwards incompatible changes at the top of the release notes draft (#2431) Bouwe Andela
- Fixed intersphinx mapping of *scipy* (#2523) Manuel Schlund
- Add authors to citation cff (#2525) SarahAlidoost
- Update documentation on running a recipe (#2432) Bouwe Andela
- Fix recipe *hydrology/recipe_wflow.yml* (#2549) Rémi Kazeroni
- Update *draft_release_notes.py* for new release (#2553) Manuel Schlund
- Added stand with Ukraine badge (#2565) Valeriu Predoi
- Updated CREM docu (recipe_williams09climdyn.yml) (#2567) Axel Lauer
- First draft for v2.5.0 changelog (#2554) Manuel Schlund
- Replace nonfunctional Github Actions badge with cool one in README (#2582) Valeriu Predoi
- Updated changelog (#2589) Manuel Schlund
- Updated release strategy with current release and upcoming release (#2597) Manuel Schlund
- Increased ESMValTool version to 2.5.0 (#2600) Manuel Schlund

60.6 Diagnostics

- AutoAssess: Add new diagnostic for radiation budget (#2282) Jon Lillis
- CMUG Sea Surface Salinity dataset and diagnostic (#1832) Javier Vegas-Regidor
- Recipe with new diagnostics for ESA-CMUG H2O (#1834) katjaweigel
- Cleaned Schlund et al. (2020) recipe and fixed small bugs in corresponding diagnostic (#2484) Manuel Schlund
- Add ESA CCI LST cmorizer and diagnostic (#1897) morobking
- XCH4 ESA CMUG diagnostics (subset of the MPQB diagnostics) (#1960) Birgit Hassler
- Add support for ESACCI Ocean Color (Chlorophyll) observations (#2055) ulrikaw-cloud
- Updated *recipe_zmnam.yml* with hemisphere selection (#2230) fserva
- Add recipe and diagnostic scripts to compute figures of D9.4 of ISENES3 (#2441) sloosvel
- Save resampled climates from KCS diagnostic local_resampling.py (#2221) Emma Daniels
- Use years from KCS recipe (#2223) Emma Daniels
- Recipe to plot generic output from the preprocessor (#2184) Javier Vegas-Regidor
- Fixed provenance tracking for emergent constraint diagnostics (#2573) Manuel Schlund

60.7 Observational and re-analysis dataset support

- Ensure dummy data for `cmorize_obs_woa` test are written to the correct directory (#2451) [Emma Hogan](#)
- Add ESA CCI LST cmorizer and diagnostic (see previous section *Diagnostics*)

60.8 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 (#2449) [Valeriu Predoi](#)
- Fix individual pytest runs broken due to missing explicit imports from *iris* and adding a couple missing package markers (#2455) [Valeriu Predoi](#)
- Add Python=3.10 to Github Actions and switch to Python=3.10 for the Github Action that builds the PyPi package (#2488) [Valeriu Predoi](#)
- Switch all github actions from miniconda to mambaforge (#2498) [Klaus Zimmermann](#)
- Pin *flake8*<4 to have actual FLAKE8 error printed if tests fail and not garbage (#2492) [Valeriu Predoi](#)
- Implementing conda lock (#2193) [Valeriu Predoi](#)
- [Docker] Update Docker container builds with correct installations of Julia (#2530) [Valeriu Predoi](#)
- Update Linux condalock file (various pull requests) [github-actions\[bot\]](#)

60.9 Installation

- Comment out release candidate channel in `environment.yml` (#2417) [Klaus Zimmermann](#)
- Comment out rc channel in osx environment file (#2421) [Valeriu Predoi](#)
- Add *python-cdo* as conda-forge dependency in environment files to ensure *cdo* gets used from conda-forge and not pip (#2469) [Valeriu Predoi](#)
- Install rasterio from conda-forge and avoid issues from python=3.10 (#2479) [Valeriu Predoi](#)
- Updated dependencies with new ESMValCore version (#2599) [Manuel Schlund](#)

60.10 Improvements

- Remove use of OBS and use CMIP instead in `examples/recipe_ncl.yml` (#2494) [Valeriu Predoi](#)
- Expanded `recipe_preprocessor_test.yml` to account for new *multi_model_statistics* features (#2519) [Manuel Schlund](#)
- Updated piControl periods for recipes that use KACE-1-0-G (#2537) [Manuel Schlund](#)
- Reduced time range in `recipe_globwat.yml` (#2548) [Manuel Schlund](#)
- Removed models with missing data from `recipe_williams09climdyn.yml` (#2566) [Axel Lauer](#)
- Restored original versions of `recipe_schlund20esd.yml` and `recipe_meehl20sciadv.yml` (#2583) [Manuel Schlund](#)

61.1 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 at [Move to Mamba](#) and 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

61.2 Bug fixes

- Fixed *recipe_meehl20sciadv.yml* for ESMValCore 2.3 (#2253) [Manuel Schlund](#)
- Fix provenance of NCL figures created using the `log_provenance` function (#2279) [Bouwe Andela](#)
- Fix bug in ClimWIP *brunner19* recipe when plotting (#2226) [Lukas Brunner](#)
- Pin docutils <0.17 to fix sphinx build with rtd theme (#2312) [Klaus Zimmermann](#)
- Fix example recipes (#2338) [Valeriu Predoi](#)
- Do not add bounds to *plev* (*plev19*) in era interim cmorizer (#2328) [Valeriu Predoi](#)
- Fix problem with pip 21.3 that prevents installation from source (#2344) [Klaus Zimmermann](#)
- Add title to recipe embedded in `test_diagnostic_run.py` (#2353) [Klaus Zimmermann](#)
- Fix capitalization of *obs4MIPs* (#2368) [Bouwe Andela](#)
- Specify that *areacella* is needed for area statistics in the Python example recipe (#2371) [Bouwe Andela](#)
- Enabling variable *obs550lt1aer* in recipes (#2388) [Rémi Kazeroni](#)
- Update a diagnostic to new Iris version (#2390) [katjaweigel](#)
- Fixed bug in provenance tracking of *ecs_scatter.ncl* (#2391) [Manuel Schlund](#)
- Fix provenance issue in *pv_capacity_factor.R* (#2392) [katjaweigel](#)
- Remove obsolete `write_plots` option from R diagnostics (#2395) [Klaus Zimmermann](#)
- Fix arctic ocean diagnostic (#2397) [Klaus Zimmermann](#)

- Fix sea ice drift recipe and script (#2404) sloosvel
- Adapt diagnostic script to new version of iris (#2403) Klaus Zimmermann
- Fix ocean multimap (#2406) Klaus Zimmermann
- Fix diagnostic that uses *xarray*: *dtype* correctly set and harmonize *xarray* and *matplotlib* (#2409) Klaus Zimmermann
- Deactivate provenance logging for plots in thermodyn toolbox (#2414) Klaus Zimmermann

61.3 Deprecations

- Removed `write_plots` and `write_netcdf` from some NCL diagnostics (#2293) Manuel Schlund
- Fixed provenance logging of all python diagnostics by removing 'plot_file' entry (#2296) Manuel Schlund
- Do not deprecate classes `Variable`, `Variables` and `Datasets` on a specific version (#2286) Manuel Schlund
- Remove obsolete `write_netcdf` option from ncl diagnostic scripts (#2387) Klaus Zimmermann
- Remove write plots from ocean diagnostics (#2393) Valeriu Predoi
- More removals of instances of `write_plots` from Python diagnostics (appears to be the final removal from Py diags) (#2394) Valeriu Predoi

61.4 Documentation

- List Manuel Schlund as release manager for v2.5 (#2268) Bouwe Andela
- GlobWat fix download links and gdal command (#2334) Banafsheh Abdollahi
- Add titles to recipes authored by *predoi_valeriu* (#2333) Valeriu Predoi
- Added titles to recipes maintained by *lauer_axel* (#2332) Axel Lauer
- Update the documentation of the GRACE CMORizer (#2349) Rémi Kazeroni
- Add titles in BSC recipes (#2351) sloosvel
- Update `esmvalcore` dependency to 2.4.0rc1 (#2348) Klaus Zimmermann
- Add titles to recipes maintained by Peter Kalverla (#2356) Peter Kalverla
- Adding titles to the recipes with maintainer `hb326` (#2358) Birgit Hassler
- Add title for `zmnam` as for #2354 (#2363) fserva
- Added recipe titles the the ocean recipes. (#2364) Lee de Mora
- Update `recipe_thermodyn_diagtool.yml` - add title (#2365) ValerioLembo
- Fix provenance of figures of several R diagnostics (#2300) Bouwe Andela
- Adding titles to Mattia's recipes (#2367) Rémi Kazeroni
- Adding titles to wenzel recipes (#2366) Birgit Hassler
- Fix formatting of some recipe titles merged from PR 2364 (#2372) Klaus Zimmermann
- Adding titles to Bjoern's recipes (#2369) Rémi Kazeroni
- Add titles to ocean recipes (maintainer Lovato) (#2375) Tomas Lovato

- Add titles for three c3s-magic recipes (#2378) Klaus Zimmermann
- Add title for recipe maintained by Ruth Lorenz (#2379) Klaus Zimmermann
- Fix toymodel recipe (#2381) Javier Vegas-Regidor
- Added titles for recipes of maintainer *schlund_manuel* (#2377) Manuel Schlund
- Write `_plots` and titles for `deangelis15nat`, `li17natcc`, `martin18grl`, `pv_capacity_factor` (#2382) katjaweigel
- Add titles for some recipes (#2383) Klaus Zimmermann
- Adding titles for recipes by von Hardenberg and Arnone (#2384) Klaus Zimmermann
- Last two missing titles (#2386) Valeriu Predoi
- Update documentation on downloading data (#2370) Bouwe Andela
- Fix installation instructions for Julia (#2335) Klaus Zimmermann
- Fix provenance of Julia example diagnostic (#2289) Bouwe Andela
- Added notes on use of mamba in the installation documentation chapter (#2236) Valeriu Predoi
- Update version number for 2.4.0 release (#2410) Klaus Zimmermann
- Update release schedule for 2.4.0 (#2412) Klaus Zimmermann
- Update changelog for 2.4.0 release (#2411) Klaus Zimmermann

61.5 Diagnostics

- Add all available CMIP5 and CMIP6 models to `recipe_impact.yml` (#2251) Bouwe Andela
- Add Fig. 6, 7 and 9 of Bock20jgr (#2252) Lisa Bock
- Generalize *recipe_validation** diagnostic to work with identical control and experiment dataset names (#2284) Valeriu Predoi
- Add missing preprocessor to `recipe_gier2020bg` and adapt to available data (#2399) Bettina Gier
- Removed custom version of *AtmosphereSigmaFactory* in diagnostics (#2405) Manuel Schlund

61.6 Observational and re-analysis dataset support

- Replace `recipe_era5.yml` with `recipe_daily_era5.yml` (#2182) SarahAlidoost
- Update WOA cmorizer for WOA18 and WOA13v2 (#1812) Lisa Bock
- GLODAP v2.2016 ocean data cmorizer (#2185) Tomas Lovato
- Updated GCP CMORizer (#2295) Manuel Schlund

61.7 Automatic testing

- Add a cylc suite to run all recipes (#2219) Bouwe Andela
- Retire test with Python 3.6 from full development Github Actions test (#2229) Valeriu Predoi
- Remove Python 3.6 tests from GitHub Actions (#2264) Valeriu Predoi
- Unpin upper bound for iris (previously was at <3.0.4) (#2266) Valeriu Predoi
- Pin latest esmvalcore to allow use of the bugfix release 2.3.1 always (#2269) Valeriu Predoi
- Add apt update so Julia gets found and installed by Docker (#2290) Valeriu Predoi
- Use mamba for environment update and creation in the Docker container build on DockerHub (#2297) Valeriu Predoi
- Docker container experimental - run a full env solve with mamba instead of a conda update (#2306) Valeriu Predoi
- Full use of mamba in Github Actions source install test and use generic Python 3.7 (removing the very specific 3.7.10) (#2287) Valeriu Predoi
- Replace use of conda with mamba for conda_install test on Circle CI (#2237) Valeriu Predoi
- Update circleci configuration (#2357) Klaus Zimmermann

61.8 Installation

- Remove *mpich* from conda dependencies list (#2343) Valeriu Predoi

61.9 Improvements

- Add script for extracting a list of input files from the provenance (#2278) Bouwe Andela
- Update github actions (#2360) Klaus Zimmermann
- Removed 'write_plots' from all NCL diagnostics (#2331) Axel Lauer
- Update and modernize *config-user-example.yml* (#2374) Valeriu Predoi

This release includes

62.1 Bug fixes

- Indent block to pick up and raise exception if cmorizer data not found (TierX dir is not there) (#1877) Valeriu Predoi
- Skip recipe filler tests until we have a new release since GA tests are failing (#2089) Valeriu Predoi
- Fixed broken link to contributions in README (#2102) Manuel Schlund
- Fix recipe filler for the case the variable doesn't contain short_name (#2104) Valeriu Predoi
- Add fix for iris longitude bug to ClimWIP (#2107) Lukas Brunner
- Update for outdated link to reference Déandreis et al. (2014). (#2076) katjaweigel
- Fixed recipes for ESMValCore 2.3.0 (#2203) Manuel Schlund
- Fix the WFDE5 cmorizer (#2211) Rémi Kazeroni
- Fix broken CMORizer log message if no Tier directory exists (#2207) jmrgonza
- Fix bug in ClimWIP basic test recipe when plotting (#2225) Lukas Brunner
- Fix bug in ClimWIP advanced test recipe when plotting (#2227) Lukas Brunner
- Adjust time range for the *WFDE5* dataset in the *recipe_check_obs.yml* (#2232) Rémi Kazeroni
- Fix plot and provenance of recipe_consecdrydays (#2244) Bouwe Andela

62.2 Documentation

- Improving the README.md file with a more appealing look and bit more info (#2065) Valeriu Predoi
- Update plot title martin18grl (#2080) katjaweigel
- Update contribution guidelines (#2031) Bouwe Andela
- Update links in pull request template to point to latest documentation (#2083) Bouwe Andela
- Update release schedule (#2081) Bouwe Andela
- Updates to contribution guidelines (#2092) Bouwe Andela
- Update documentation for ERA5 with new variables (#2111) Lukas Brunner

- Add OSX installation instructions to docs (#2115) Barbara Vreede
- Instructions to use pre-installed versions on HPC clusters (#2197) Rémi Kazeroni
- Add functional Autoassess diagnostics: land surface metrics: permafrost, soil moisture, surface radiation (#2170) Valeriu Predoi
- Add citation info in *recipe_eady_growth_rate.yml* (#2188) sloosvel
- Update version number to 2.3.0 (#2213) Klaus Zimmermann
- Update release schedule for 2.3.0 (#2247) Klaus Zimmermann
- Changelog update to v2.3.0 (#2214) Klaus Zimmermann

62.3 Diagnostics

- Added figures 8 and 10 to *recipe_bock20jgr.yml* (#2074) Manuel Schlund
- Add hydrological forcing comparison recipe (#2013) Stef Smeets
- Added recipe for Meehl et al., Sci. Adv. (2020) (#2094) Manuel Schlund
- Add GlobWat recipe and diagnostic (#1808) Banafsheh Abdollahi
- Add ClimWIP recipe to reproduce Brunner et al. 2019 (#2109) Lukas Brunner
- Update Climwip recipe to reproduce brunner2020esd (#1859) Ruth Lorenz
- Update *recipe_thermodyn_diagtool.yml*: code improvements and more user options (#1391) ValerioLembo
- Remove model AWI-CM-1-1-MR from *recipe_impact.yml* (#2238) Bouwe Andela
- PV capacity factor for ESMValTool GMD paper (#2153) katjaweigel

62.4 Observational and re-analysis dataset support

- Cmorize wfde5 (#1991) mwjury
- Make cmorizer utils funcs public in *utilities.py* and add some numpy style docstrings (#2206) Valeriu Predoi
- CMORizer for CLARA-AVHRR cloud data (#2101) Axel Lauer
- Update of ESACCI-CLOUD CMORizer (#2144) Axel Lauer

62.5 Automatic testing

- Force latest Python in empty environment in conda install CI test (#2069) Valeriu Predoi
- Removed imports from private sklearn modules and improved test coverage of *custom_sklearn.py* (#2078) Manuel Schlund
- Move private *_(global)_stock_cube* from *esmvacore.preprocessor._regrid* to *cmorizer* (#2087) Valeriu Predoi
- Try mamba install *esmvaltool* (#2125) Valeriu Predoi
- Reinstate OSX Github Action tests (#2110) Valeriu Predoi
- Pin mpich to avoid default install of 3.4.1 and 3.4.2 with *external_0* builds (#2220) Valeriu Predoi

- Include test sources in distribution (#2234) Klaus Zimmermann
- Pin *iris*<3.0.4 to ensure we still (sort of) support Python 3.6 (#2246) Valeriu Predoi

62.6 Installation

- Fix conda build by skipping documentation test (#2058) Javier Vegas-Regidor
- Update pin on esmvalcore pick up esmvalcore=2.3.0 (#2200) Valeriu Predoi
- Pin Python to 3.9 for development installation (#2208) Bouwe Andela

62.7 Improvements

- Add EUCP and IS-ENES3 projects to config-references (#2066) Peter Kalverla
- Fix flake8 tests on CircleCI (#2070) Bouwe Andela
- Added recipe filler. (#1707) Lee de Mora
- Update use of fx vars to new syntax (#2145) sloosvel
- Add recipe for climate impact research (#2072) Peter Kalverla
- Update references “master” to “main” (#2172) Axel Lauer
- Force git to ignore VSCode workspace files (#2186) Javier Vegas-Regidor
- Update to new ESMValTool logo (#2168) Axel Lauer
- Python cmorizers for CDR1 and CDR2 ESACCI H2O (TCWV=prw) data. (#2152) katjaweigel
- Remove obsolete conda package (closes #2100) (#2103) Klaus Zimmermann

63.1 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

63.2 Bug fixes

- Bugfix: time weights in `time_operations` (#1956) [Axel Lauer](#)
- Fix issues with bibtex references (#1955) [Stef Smeets](#)
- Fix ImportError for `configure_logging` (#1976) [Stef Smeets](#)
- Add required functional parameters for extract time in `recipe_er5.yml` (#1978) [Valeriu Predoi](#)
- Revert “Fix ImportError for `configure_logging`” (#1992) [Bouwe Andela](#)
- Fix import of `esmvalcore_logging` module in `cmorize_obs.py` (#2020) [Valeriu Predoi](#)
- Fix logging import in `cmorize_obs` again since last merge was nulled by pre-commit hooks (#2022) [Valeriu Predoi](#)
- Refactor the functions in `derive_evspblpot` due to new iris (#2023) [SarahAlidoost](#)
- Avoid importing private ESMValCore functions in CMORizer (#2027) [Bouwe Andela](#)
- Fix `extract_seasons` in validation recipe (#2054) [Javier Vegas-Regidor](#)

63.3 Deprecations

- Deprecate classes `Variable`, `Variables` and `Datasets` (#1944) [Manuel Schlund](#)
- Python 3.9: remove `pynio` as dependency and replace with `rasterio` and `pin Matplotlib>3.3.1` and `pin cartopy>=0.18` (#1997) [Valeriu Predoi](#)
- Removed `write_plots` and `write_netcdf` in some python diagnostics (#2036) [Manuel Schlund](#)

63.4 Documentation

- Update instructions on making a release (#1867) Bouwe Andela
- Update review.rst (#1917) Axel Lauer
- Add guidance on how to review a pull request (#1872) Bouwe Andela
- Adding tutorial links to documentation (#1927) Birgit Hassler
- Added bibtex file for schlund20jgr (#1928) Manuel Schlund
- Documentation contact added the actual email for the mailing list (#1938) Valeriu Predoi
- Make CircleCI badge specific to main branch (#1831) Bouwe Andela
- Documentation on how to move code from a private repository to a public repository (#1920) Birgit Hassler
- Refine pull request review guidelines (#1924) Stef Smeets
- Update release schedule (#1948) Klaus Zimmermann
- Improve contact info and move to more prominent location (#1950) Bouwe Andela
- Add some maintainers to some recipes that are missing them (#1970) Valeriu Predoi
- Update core team info (#1973) Axel Lauer
- Combine installation from source instructions and add common issues (#1971) Bouwe Andela
- Update iris documentation URL for sphinx (#2003) Bouwe Andela
- Fix iris documentation link(s) with new iris3 location on readthedocs (#2012) Valeriu Predoi
- Document how to run tests for installation verification (#1847) Valeriu Predoi
- List Remi Kazeroni as a code owner and sole merger of CMORizers (#2017) Bouwe Andela
- Install documentation: mention that we build conda package with python>=3.7 (#2030) Valeriu Predoi
- Recipe and documentation update for ERA5-Land. (#1906) katjaweigel
- Update changelog and changelog tool for v2.2.0 (#2043) Javier Vegas-Regidor
- Final update to the changelog for v2.2.0 (#2056) Javier Vegas-Regidor

63.5 Diagnostics

- Add mapplot diagnostic to ClimWIP (#1864) Lukas Brunner
- Add the option to weight variable groups in ClimWIP (#1856) Lukas Brunner
- Implementation of ensemble member recognition to the ClimWIP diagnostic (#1852) Lukas Brunner
- Restructure ClimWIP (#1919) Lukas Brunner
- Diagnostic for recipe_eyring13jgr.yml Fig. 12 (#1922) Lisa Bock
- Added changes in shared functions necessary for schlund20esd (#1967) Manuel Schlund
- Adding recipe and diagnostics for Gier et al 2020 (#1914) Bettina Gier
- Added recipe, diagnostics and documentation for Schlund et al., ESD (2020) (#2015) Manuel Schlund
- Add PRIMavera Eady Growth Rate diagnostic (#1285) sloosvel
- Implement shape parameter calibration for ClimWIP (#1905) Lukas Brunner

63.6 Observational and re-analysis dataset support

- Extended ESRL cmorizer (#1937) Bettina Gier
- Cmorizer for GRACE data (#1694) bascrezee
- Cmorizer for latest ESACCI-SST data (#1895) Valeriu Predoi
- Fix longitude in ESRL cmorizer (#1988) Bettina Gier
- Selectively turn off fixing bounds for coordinates during cmorization with utilities.py (#2014) Valeriu Predoi
- Cmorize hadcrut5 (#1977) mwjury
- Cmorize gpcc masking (#1995) mwjury
- Cmorize_utils_save_1mon_Amon (#1990) mwjury
- Cmorize gpcc fix (#1982) mwjury
- Fix flake8 raised by develop test in cmorize_obs_gpcc.py (#2038) Valeriu Predoi

63.7 Automatic testing

- Switched miniconda conda setup hooks for Github Actions workflows (#1913) Valeriu Predoi
- Fix style issue (#1929) Bouwe Andela
- Fix mlr test with solution that works for CentOS too (#1936) Valeriu Predoi
- Temporary deactivation Github Actions on OSX (#1939) Valeriu Predoi
- Fix conda installation test on CircleCI (#1952) Bouwe Andela
- Github Actions: change time for cron job that installs from conda (#1969) Valeriu Predoi
- CI upload relevant artifacts for test job (#1999) Valeriu Predoi
- Github Actions test that runs with the latest ESMValCore main (#1989) Valeriu Predoi
- Introduce python 39 in Github Actions tests (#2029) Valeriu Predoi
- Remove test for conda package installation on Python 3.6 (#2033) Valeriu Predoi
- Update codacy coverage reporter to fix coverage (#2039) Bouwe Andela

63.8 Installation

- Simplify installation of R development dependencies (#1930) Bouwe Andela
- Fix docker build (#1934) Bouwe Andela
- Use new conda environment for installing ESMValTool in Docker containers (#1993) Bouwe Andela
- Fix conda build (#2026) Bouwe Andela

63.9 Improvements

- Allow multiple references for a cmorizer script (#1953) SarahAlidoost
- Add GRACE to the recipe check_obs (#1963) Rémi Kazeroni
- Align ESMValTool to ESMValCore=2.2.0 (adopt iris3, fix environment for new Core release) (#1874) Stef Smeets
- Make it possible to use write_plots and write_netcdf from recipe instead of config-user.yml (#2018) Bouwe Andela
- Revise lisflood and hype recipes (#2035) SarahAlidoost
- Set version to 2.2.0 (#2042) Javier Vegas-Regidor

This release includes

64.1 Improvements

- Fix the conda build on CircleCI (#1883) [Bouwe Andela](#)
- Pin matplotlib to <3.3 and add compilers (#1898) [Bouwe Andela](#)
- Pin esmvaltool subpackages to the same version and build as the esmvaltool conda package (#1899) [Bouwe Andela](#)

64.2 Documentation

- Release notes v2.1.1 (#1932) [Valeriu Predoi](#)

This release includes

65.1 Diagnostics

- Add extra steps to diagnostic to make output of hydrology/recipe_lisflood.yml usable by the LISFLOOD model (#1737) Jaro Camphuijsen
- Recipe to reproduce the 2014 KNMI Climate Scenarios (kcs). (#1667) Peter Kalverla
- Implement the climwip weighting scheme in a recipe and diagnostic (#1648) Jaro Camphuijsen
- Remove unreviewed autoassess recipes (#1840) Valeriu Predoi
- Changes in shared scripts for Schlund et al., JGR: Biogeosciences, 2020 (#1845) Manuel Schlund
- Updated derivation test recipe (#1790) Manuel Schlund
- Support for multiple model occurrence in perf main (#1649) Bettina Gier
- Add recipe and diagnostics for Schlund et al., JGR: Biogeosciences, 2020 (#1860) Manuel Schlund
- Adjust recipe_extract_shape.yml to recent changes in the example diagnostic.py (#1880) Bouwe Andela

65.2 Documentation

- Add pip installation instructions (#1783) Bouwe Andela
- Add installation instruction for R and Julia dependencies tot pip install (#1787) Bouwe Andela
- Avoid autodocsumm 0.2.0 and update documentation build dependencies (#1794) Bouwe Andela
- Add more information on working on cluster attached to ESGF node (#1821) Bouwe Andela
- Add release strategy to community documentation (#1809) Klaus Zimmermann
- Update esmvaltool run command everywhere in documentation (#1820) Bouwe Andela
- Add more info on documenting a recipe (#1795) Bouwe Andela
- Improve the Python example diagnostic and documentation (#1827) Bouwe Andela
- Improve description of how to use draft_release_notes.py (#1848) Bouwe Andela
- Update changelog for release 2.1 (#1886) Valeriu Predoi

65.3 Improvements

- Fix R installation in WSL (#1789) Javier Vegas-Regidor
- Add pre-commit for linting/formatting (#1796) Stef Smeets
- Speed up tests on CircleCI and use pytest to run them (#1804) Bouwe Andela
- Move pre-commit excludes to top-level and correct order of lintr and styler (#1805) Stef Smeets
- Remove isort setup to fix formatting conflict with yapf (#1815) Stef Smeets
- GitHub Actions (#1806) Valeriu Predoi
- Fix yapf-isort import formatting conflict (#1822) Stef Smeets
- Replace vmprof with vprof as the default profiler (#1829) Bouwe Andela
- Update ESMValCore v2.1.0 requirement (#1839) Javier Vegas-Regidor
- Pin iris to version 2 (#1881) Bouwe Andela
- Pin eccodes to not use eccodes=2.19.0 for cdo to work fine (#1869) Valeriu Predoi
- Increase version to 2.1.0 and add release notes (#1868) Valeriu Predoi
- Github Actions Build Packages and Deploy tests (conda and PyPi) (#1858) Valeriu Predoi

65.4 Observational and re-analysis dataset support

- Added CMORizer for Scripps-CO2-KUM (#1857) Manuel Schlund

This release includes

66.1 Bug fixes

- Fix pep8-naming errors and fix znmam diagnostic (#1702) Bouwe Andela
- Fix keyword argument in cmorize_obs (#1721) Mattia Righi
- Fixed JMA-TRANSCOM CMORizer (#1735) Manuel Schlund
- Fix bug in extract_doi_value (#1734) bascrezee
- Fix small errors in the arctic_ocean diagnostic (#1722) Nikolay Koldunov
- Flatten ancestor lists for diag_spei.R and diag_spi.R. (#1745) katjaweigel
- Fix for recipe_ocean_ice_extent.yml (#1744) Mattia Righi
- Fix recipe_combined_indices.yml provenance (#1746) Javier Vegas-Regidor
- Fix provenance in recipe_multimodel_products (#1747) Javier Vegas-Regidor
- Exclude FGOALS-g2 due to ESMValCore issue #728 (#1749) Mattia Righi
- Fix recipe_modes_of_variability (#1753) Javier Vegas-Regidor
- Flatten lists for ancestors for hyint to prevent nested lists. (#1752) katjaweigel
- Fix bug in cmorize_obs_eppley_vgpm_modis.py (#1729) (#1759) Tomas Lovato
- Correct mip for clltkiscp in example derive preprocessor recipe (#1768) Bouwe Andela
- Update date conversion in recipe_hype.yml (#1769) Bouwe Andela
- Fix recipe_correlation.yml (#1767) Bouwe Andela
- Add attribute positive: down to plev coordinate in ERA-Interim CMORizer (#1771) Bouwe Andela
- Fix sispeed in recipe_preprocessor_derive_test (#1772) Javier Vegas-Regidor
- Fix extreme events and extreme index ancestors (#1774) katjaweigel
- Correct date in output filenames of ERA5 CMORizer recipe (#1773) Bouwe Andela
- Exclude WOA from multi-model stats in recipe_ocean_bgc (#1778) Mattia Righi

66.2 Diagnostics

- Enhancement of the hyint recipe to include etccdi indices (#1133) Enrico Arnone
- Add lazy regridding for wflow diagnostic (#1630) Bouwe Andela
- Miles default domains to include lat=0 (#1626) Jost von Hardenberg
- Miles: selection of reference dataset based on experiment (#1632) Jost von Hardenberg
- New recipe/diagnostic: recipe_li17natcc.yml for Axels GMD Paper (#1567) katjaweigel
- New recipe/diagnostics: recipe_deangelis_for_gmdpart4.yml for Axels GMD Paper (#1576) katjaweigel
- EWaterCycle: Add recipe to prepare input for LISFLOOD (#1298) Stefan Verhoeven
- Use area weighted regridding in wflow diagnostic (#1643) Bouwe Andela
- Workaround for permetrics recipe until Iris3 (#1674) Mattia Righi
- C3S_511_MPQB_bas-features (#1465) bascrezee
- Additional Land perfmetrics (#1641) Bettina Gier
- Necessary diagnostic from eyring06jgr for the release of version2 (#1686) Birgit Hassler
- Drought characteristics based on Martin2018 and SPI for gmd paper (#1689) katjaweigel
- Additional features and bugfixes for recipe anav13clim (#1723) Bettina Gier
- Gmd laueretal2020 revisions (#1725) Axel Lauer
- Wenzel16nature (#1692) zechlau
- Add mask albedolandcover (#1673) bascrezee
- IPCC AR5 fig. 9.3 (seasonality) (#1726) Axel Lauer
- Added additional emergent constraints on ECS (#1585) Manuel Schlund
- A diagnostic to evaluate the turnover times of land ecosystem carbon (#1395) koir-su
- Removed multi_model_statistics step in recipe_oceans_example.yml as a workaround (#1779) Valeriu Predoi

66.3 Documentation

- Extend getting started instructions to obtain config-user.yml (#1642) Peter Kalverla
- Extend information about native6 support on RTD (#1652) Peter Kalverla
- Update citation of ESMValTool paper in the doc (#1664) Mattia Righi
- Updated references to documentation (now docs.esmvaltool.org) (#1679) Axel Lauer
- Replace dead link with ESGF link. (#1681) Mattia Righi
- Add all European grants to Zenodo (#1682) Bouwe Andela
- Update Sphinx to v3 or later (#1685) Bouwe Andela
- Small fix to number of models in ensclus documentation (#1691) Jost von Hardenberg
- Move draft_release_notes.py from ESMValCore to here and update (#1701) Bouwe Andela
- Improve the installation instructions (#1634) Valeriu Predoi
- Improve description of how to implement provenance in diagnostic (#1750) SarahAlidoost

- Update command line interface documentation and add links to ESMValCore configuration documentation (#1776) Bouwe Andela
- Documentation on how to find shapefiles for hydrology recipes (#1777) Jaro Camphuijsen

66.4 Improvements

- Pin flake8<3.8.0 (#1635) Valeriu Predoi
- Update conda package path in more places (#1636) Bouwe Andela
- Remove curly brackets around issue number in pull request template (#1637) Bouwe Andela
- Fix style issue in test (#1639) Bouwe Andela
- Update Codacy badges (#1662) Bouwe Andela
- Support extra installation methods in R (#1360) Javier Vegas-Regidor
- Add ncdf4.helpers package as a dependency again (#1678) Bouwe Andela
- Speed up conda installation (#1677) Bouwe Andela
- Update CMORizers and recipes for ESMValCore v2.0.0 (#1699) SarahAlidoost
- Update setup.py for PyPI package (#1700) Bouwe Andela
- Cleanup recipe headers before the release (#1740) Mattia Righi
- Add colortables as esmvaltool subcommand (#1666) Javier Vegas-Regidor
- Increase version to v2.0.0 (#1756) Bouwe Andela
- Update job script (#1757) Mattia Righi
- Read authors and description from .zenodo.json (#1758) Bouwe Andela
- Update docker recipe to install from source (#1651) Javier Vegas-Regidor

66.5 Observational and re-analysis dataset support

- Cmorize aphro ma (#1555) mwjury
- Respectable testing for cmorizers/obs/utilities.py and cmorizers/obs/cmorize_obs.py (#1517) Valeriu Predoi
- Fix start year in recipe_check_obs (#1638) Mattia Righi
- Cmorizer for the PERSIANN-CDR precipitation data (#1633) Birgit Hassler
- Cmorize eobs (#1554) mwjury
- Update download cds satellite lai fapar (#1654) bascrezee
- Added monthly mean vars (ta, va, zg) to era5 cmorizer via recipe (#1644) Evgenia Galytska
- Make format time check more flexible (#1661) Mattia Righi
- Exclude od550ltlaer from recipe_check_obs.yml (#1720) Mattia Righi
- PERSIANN-CDR cmorizer update: adding the capability to save monthly mean files (#1728) Birgit Hassler
- Add standard_name attribute to lon and lat in cmorize_obs_esacci_oc.py (#1760) Tomas Lovato
- Allow for incomplete months on daily frequency in cmorizer ncl utilities (#1754) Mattia Righi

- Fix AURA-TES cmorizer ([#1766](#)) [Mattia Righi](#)

This release includes

67.1 Bug fixes

- Fix HALOE plev coordinate (#1590) [Mattia Righi](#)
- Fix tro3 units in HALOE (#1591) [Mattia Righi](#)

67.2 Diagnostics

- Apply sea ice negative feedback (#1299) [Javier Vegas-Regidor](#)
- Add Russell18jgr ocean diagnostics (#1592) [Bouwe Andela](#)
- Refactor marrmot recipe and diagnostic to use ERA5 daily data made by new cmorizer (#1600) [SarahAlidoost](#)
- In recipe_wflow, use daily ERA5 data from the new cmorizer. (#1599) [Peter Kalverla](#)
- In wflow diagnostic, calculate PET after(!) interpolation and lapse rate correction (#1618) [Jerom Aerts](#)
- Fixed wenz14jgr (#1562) [zechlau](#)
- Update portrait_plot.ncl (#1625) [Bettina Gier](#)

67.3 Documentation

- Restructure documentation (#1587) [Bouwe Andela](#)
- Add more links to documentation (#1595) [Bouwe Andela](#)
- Update links in readme (#1598) [Bouwe Andela](#)
- Minor improvements to installation documentation (#1608) [Bouwe Andela](#)
- Add info for new mailing list to documentation. (#1607) [Björn Brötz](#)
- Update making a release documentation (#1627) [Bouwe Andela](#)

67.4 Improvements

- Avoid broken pytest-html plugin (#1583) Bouwe Andela
- Remove reference section in config-references.yml (#1545) SarahAlidoost
- Various improvements to development infrastructure (#1570) Bouwe Andela
- Install scikit-learn from conda, remove libunwind as a direct dependency (#1611) Valeriu Predoi
- Create conda subpackages and enable tests (#1624) Bouwe Andela

67.5 Observational and re-analysis dataset support

- Cmorizer for HALOE (#1581) Mattia Righi
- Add CMORizer for CT2019 (#1604) Manuel Schlund

For older releases, see the release notes on <https://github.com/ESMValGroup/ESMValTool/releases>.

Part XIII

Indices and tables

- [genindex](#)
- [search](#)

PYTHON MODULE INDEX

e

esmvaltool.diag_scripts.emergent_constraints, 575
 esmvaltool.diag_scripts.emergent_constraints.cox_signature, 571
 esmvaltool.diag_scripts.emergent_constraints.ecs_scatter, 572
 esmvaltool.diag_scripts.emergent_constraints.multiple_constraints, 573
 esmvaltool.diag_scripts.emergent_constraints.single_constraint, 574
 esmvaltool.diag_scripts.ml, 594
 esmvaltool.diag_scripts.ml.custom_sklearn, 600
 esmvaltool.diag_scripts.ml.evaluate_residuals, 584
 esmvaltool.diag_scripts.ml.main, 585
 esmvaltool.diag_scripts.ml.mmm, 586
 esmvaltool.diag_scripts.ml.models, 616
 esmvaltool.diag_scripts.ml.models.gbr_base, 629
 esmvaltool.diag_scripts.ml.models.gbr_sklearn, 645
 esmvaltool.diag_scripts.ml.models.gbr_xgboost, 653
 esmvaltool.diag_scripts.ml.models.gpr_sklearn, 662
 esmvaltool.diag_scripts.ml.models.huber, 673
 esmvaltool.diag_scripts.ml.models.krr, 681
 esmvaltool.diag_scripts.ml.models.lasso, 689
 esmvaltool.diag_scripts.ml.models.lasso_cv, 697
 esmvaltool.diag_scripts.ml.models.lasso_lars_cv, 705
 esmvaltool.diag_scripts.ml.models.linear, 713
 esmvaltool.diag_scripts.ml.models.linear_base, 637
 esmvaltool.diag_scripts.ml.models.rfr, 721
 esmvaltool.diag_scripts.ml.models.ridge, 729
 esmvaltool.diag_scripts.ml.models.ridge_cv, 737
 esmvaltool.diag_scripts.ml.models.svr, 745
 esmvaltool.diag_scripts.ml.plot, 587
 esmvaltool.diag_scripts.ml.postprocess, 589
 esmvaltool.diag_scripts.ml.preprocess, 591
 esmvaltool.diag_scripts.ml.rescale_with_emergent_constraints, 593
 esmvaltool.diag_scripts.monitor.compute_eof, 755
 esmvaltool.diag_scripts.monitor.monitor, 753
 esmvaltool.diag_scripts.monitor.monitor_base, 760
 esmvaltool.diag_scripts.monitor.multi_datasets, 756
 esmvaltool.diag_scripts.ocean.diagnostic_maps, 761
 esmvaltool.diag_scripts.ocean.diagnostic_maps_quad, 763
 esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs, 764
 esmvaltool.diag_scripts.ocean.diagnostic_profiles, 767
 esmvaltool.diag_scripts.ocean.diagnostic_seaice, 773
 esmvaltool.diag_scripts.ocean.diagnostic_timeseries, 768
 esmvaltool.diag_scripts.ocean.diagnostic_tools, 776
 esmvaltool.diag_scripts.ocean.diagnostic_transects, 770
 esmvaltool.diag_scripts.psychplot_diag, 782
 esmvaltool.diag_scripts.shared, 551
 esmvaltool.diag_scripts.shared.iris_helpers, 564
 esmvaltool.diag_scripts.shared.plot, 567

INDEX

A

`add_dataset()` (*esmvaltool.diag_scripts.shared.Datasets* method), 552

`add_legend_outside_right()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 777

`add_linear_regression()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 765

`add_map_subplot()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_maps_quad*), 763

`add_map_subplot()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs*), 765

`add_sea_floor()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_transects*), 771

`add_to_data()` (*esmvaltool.diag_scripts.shared.Datasets* method), 552

`add_vars()` (*esmvaltool.diag_scripts.shared.Variables* method), 558

`AdvancedGaussianProcessRegressor` (*class in esmvaltool.diag_scripts.mlr.models.gpr_sklearn*), 662

`AdvancedPipeline` (*class in esmvaltool.diag_scripts.mlr.custom_sklearn*), 600

`AdvancedRFE` (*class in esmvaltool.diag_scripts.mlr.custom_sklearn*), 606

`AdvancedRFECV` (*class in esmvaltool.diag_scripts.mlr.custom_sklearn*), 609

`AdvancedTransformedTargetRegressor` (*class in esmvaltool.diag_scripts.mlr.custom_sklearn*), 612

`agregate_by_season()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 774

`apply_supermeans()` (*in module esmvaltool.diag_scripts.shared*), 560

B

`bgc_units()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_tools*), 778

C

`calculate_area_time_series()` (*in module esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 774

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel* property), 630

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* property), 646

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* property), 655

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* property), 666

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel* property), 674

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.krr.KRRModel* property), 682

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.lasso.LassoModel* property), 690

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* property), 698

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* property), 706

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel* property), 714

`categorical_features` (*esmvaltool.diag_scripts.mlr.models.linear_base.LinearModel* property), 638

categorical_features (*esmvaltool.diag_scripts.mlr.models.MLRModel* property), 622
categorical_features (*esmvaltool.diag_scripts.mlr.models.rfr.RFRModel* property), 722
categorical_features (*esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel* property), 730
categorical_features (*esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* property), 738
categorical_features (*esmvaltool.diag_scripts.mlr.models.svr.SVRModel* property), 746
cdf() (in module *esmvaltool.diag_scripts.emergent_constraints*), 576
check_coordinate() (in module *esmvaltool.diag_scripts.shared.iris_helpers*), 564
check_metadata() (in module *esmvaltool.diag_scripts.emergent_constraints*), 576
check_predict_kwargs() (in module *esmvaltool.diag_scripts.mlr*), 594
classes_ (*esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* property), 601
classes_ (*esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* property), 606
classes_ (*esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* property), 609
coef_ (*esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* property), 601
coef_ (*esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* property), 612
combine_groups() (in module *esmvaltool.diag_scripts.emergent_constraints*), 577
constraint_info_array() (in module *esmvaltool.diag_scripts.emergent_constraints*), 577
convert_to_iris() (in module *esmvaltool.diag_scripts.shared.iris_helpers*), 565
count() (*esmvaltool.diag_scripts.shared.Variable* method), 557
create() (*esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel* class method), 630
create() (*esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* class method), 647
create() (*esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* class method), 655
create() (*esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* class method), 666
create() (*esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel* class method), 674
create() (*esmvaltool.diag_scripts.mlr.models.krr.KRRModel* class method), 682
create() (*esmvaltool.diag_scripts.mlr.models.lasso.LassoModel* class method), 690
create() (*esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* class method), 698
create() (*esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* class method), 706
create() (*esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel* class method), 714
create() (*esmvaltool.diag_scripts.mlr.models.linear_base.LinearModel* class method), 638
create() (*esmvaltool.diag_scripts.mlr.models.MLRModel* class method), 622
create() (*esmvaltool.diag_scripts.mlr.models.rfr.RFRModel* class method), 722
create() (*esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel* class method), 730
create() (*esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* class method), 738
create() (*esmvaltool.diag_scripts.mlr.models.svr.SVRModel* class method), 746
create_alias() (in module *esmvaltool.diag_scripts.mlr*), 594
create_ice_cmap() (in module *esmvaltool.diag_scripts.ocean.diagnostic_seaice*), 775
create_ice_simple_scatterplot() (in module *esmvaltool.diag_scripts.emergent_constraints*), 577
create_ice_weighted() (in module *esmvaltool.diag_scripts.mlr.custom_sklearn*), 615
cube_time_to_float() (in module *esmvaltool.diag_scripts.ocean.diagnostic_tools*), 778
data (*esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel* property), 630
data (*esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* property), 647
data (*esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* property), 655
data (*esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* property), 666
data (*esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel* property), 674
data (*esmvaltool.diag_scripts.mlr.models.krr.KRRModel* property), 682
data (*esmvaltool.diag_scripts.mlr.models.lasso.LassoModel* property), 690
data (*esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* property), 698
data (*esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* property), 706

data (esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel property), 714

data (esmvaltool.diag_scripts.mlr.models.linear_base.LinearBaseModel property), 638

data (esmvaltool.diag_scripts.mlr.models.MLRModel property), 622

data (esmvaltool.diag_scripts.mlr.models.rfr.RFRModel property), 722

data (esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel property), 730

data (esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel property), 738

data (esmvaltool.diag_scripts.mlr.models.svr.SVRModel property), 746

Datasets (class in esmvaltool.diag_scripts.shared), 551

datasets_have_mlr_attributes() (in module esmvaltool.diag_scripts.mlr), 595

decadal_average() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 778

decision_function() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline method), 601

decision_function() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV method), 606

decision_function() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV method), 609

determine_profiles_str() (in module esmvaltool.diag_scripts.ocean.diagnostic_profiles), 767

determine_set_y_logscale() (in module esmvaltool.diag_scripts.ocean.diagnostic_transects), 771

determine_transect_str() (in module esmvaltool.diag_scripts.ocean.diagnostic_transects), 771

E

efecv() (esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel method), 630

efecv() (esmvaltool.diag_scripts.mlr.models.gbr_sklearn.StochasticGBModel method), 647

efecv() (esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostModel method), 655

efecv() (esmvaltool.diag_scripts.mlr.models.gpr_sklearn.StochasticGPModel method), 666

efecv() (esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 674

efecv() (esmvaltool.diag_scripts.mlr.models.krr.KRRModel method), 682

efecv() (esmvaltool.diag_scripts.mlr.models.lasso.LassoModel method), 690

efecv() (esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 698

efecv() (esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 706

efecv() (esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 714

efecv() (esmvaltool.diag_scripts.mlr.models.linear_base.LinearBaseModel method), 638

efecv() (esmvaltool.diag_scripts.mlr.models.MLRModel method), 622

efecv() (esmvaltool.diag_scripts.mlr.models.rfr.RFRModel method), 722

efecv() (esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel method), 730

efecv() (esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 738

efecv() (esmvaltool.diag_scripts.mlr.models.svr.SVRModel method), 746

efecv() (esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 738

efecv() (esmvaltool.diag_scripts.mlr.models.svr.SVRModel method), 746

esmvaltool.diag_scripts.emergent_constraints module, 575

esmvaltool.diag_scripts.emergent_constraints.cox18nature module, 571

esmvaltool.diag_scripts.emergent_constraints.ecs_scatter module, 572

esmvaltool.diag_scripts.emergent_constraints.multiple_constraints module, 573

esmvaltool.diag_scripts.emergent_constraints.single_constraints module, 574

esmvaltool.diag_scripts.mlr module, 594

esmvaltool.diag_scripts.mlr.custom_sklearn module, 600

esmvaltool.diag_scripts.mlr.evaluate_residuals module, 584

esmvaltool.diag_scripts.mlr.main module, 585

esmvaltool.diag_scripts.mlr.mmm module, 586

esmvaltool.diag_scripts.mlr.models module, 616

esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel module, 629

esmvaltool.diag_scripts.mlr.models.gbr_sklearn.StochasticGBModel module, 645

esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostModel module, 653

esmvaltool.diag_scripts.mlr.models.gpr_sklearn.StochasticGPModel module, 662

esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel module, 673

esmvaltool.diag_scripts.mlr.models.krr.KRRModel module, 681

esmvaltool.diag_scripts.mlr.models.lasso.LassoModel module, 689

esmvaltool.diag_scripts.mlr.models.lasso_cv module, 697	esmvaltool.diag_scripts.shared.plot module, 567
esmvaltool.diag_scripts.mlr.models.lasso_lars_cv module, 705	export_csv() (in module esmval- tool.diag_scripts.emergent_constraints), 578
esmvaltool.diag_scripts.mlr.models.linear module, 713	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.gbr_base.GBRModel method), 631
esmvaltool.diag_scripts.mlr.models.linear_base module, 637	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel method), 647
esmvaltool.diag_scripts.mlr.models.rfr module, 721	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel method), 656
esmvaltool.diag_scripts.mlr.models.ridge module, 729	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel method), 667
esmvaltool.diag_scripts.mlr.models.ridge_cv module, 737	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 675
esmvaltool.diag_scripts.mlr.models.svr module, 745	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.krr.KRRModel method), 683
esmvaltool.diag_scripts.mlr.plot module, 587	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.lasso.LassoModel method), 691
esmvaltool.diag_scripts.mlr.postprocess module, 589	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 699
esmvaltool.diag_scripts.mlr.preprocess module, 591	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 707
esmvaltool.diag_scripts.mlr.rescale_with_emergent_constraints module, 593	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 715
esmvaltool.diag_scripts.monitor.compute_eofs module, 755	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.linear_base.LinearModel method), 639
esmvaltool.diag_scripts.monitor.monitor module, 753	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.MLRModel method), 622
esmvaltool.diag_scripts.monitor.monitor_base module, 760	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.rfr.RFRModel method), 723
esmvaltool.diag_scripts.monitor.multi_datasets module, 756	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.ridge.RidgeModel method), 731
esmvaltool.diag_scripts.ocean.diagnostic_maps module, 761	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 739
esmvaltool.diag_scripts.ocean.diagnostic_maps_quad module, 763	export_prediction_data() (esmval- tool.diag_scripts.mlr.models.svr.SVRModel method), 746
esmvaltool.diag_scripts.ocean.diagnostic_model_vs_obs module, 764	export_training_data() (esmval-
esmvaltool.diag_scripts.ocean.diagnostic_profiles module, 767	
esmvaltool.diag_scripts.ocean.diagnostic_seaice module, 773	
esmvaltool.diag_scripts.ocean.diagnostic_timeseries module, 768	
esmvaltool.diag_scripts.ocean.diagnostic_toolbox module, 776	
esmvaltool.diag_scripts.ocean.diagnostic_transects module, 770	
esmvaltool.diag_scripts.psyplot_diag module, 782	
esmvaltool.diag_scripts.shared module, 551	
esmvaltool.diag_scripts.shared.iris_helpers module, 564	

`tool.diag_scripts.mlr.models.gbr_base.GBRModel``feature_importances_` (esmval-
`method`), 631 `tool.diag_scripts.mlr.custom_sklearn.AdvancedTransformedTarget`
`export_training_data()` (esmval- `property`), 612
`tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel``feature_names_in_` (esmval-
`method`), 647 `tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline`
`export_training_data()` (esmval- `property`), 601
`tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel``feature_names_in_` (esmval-
`method`), 656 `tool.diag_scripts.mlr.models.gbr_base.GBRModel`
`export_training_data()` (esmval- `features` (esmval-
`tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel``property`), 647
`method`), 667 `features` (esmval-
`export_training_data()` (esmval- `property`), 656
`tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel`
`method`), 675 `features` (esmval-
`tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel``property`), 667
`export_training_data()` (esmval- `features` (esmval-
`tool.diag_scripts.mlr.models.huber.HuberRegressionModel``property`), 675
`method`), 683 `features` (esmval-
`export_training_data()` (esmval- `property`), 683
`tool.diag_scripts.mlr.models.krr.KRRModel`
`method`), 691 `features` (esmval-
`export_training_data()` (esmval- `property`), 699
`tool.diag_scripts.mlr.models.lasso.LassoModel`
`method`), 699 `features` (esmval-
`export_training_data()` (esmval- `property`), 707
`tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel`
`method`), 707 `features` (esmval-
`export_training_data()` (esmval- `property`), 715
`tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel`
`method`), 715 `features` (esmval-
`export_training_data()` (esmval- `property`), 639
`tool.diag_scripts.mlr.models.linear.LinearRegressionModel`
`method`), 715 `features` (esmval-
`export_training_data()` (esmval- `property`), 623
`tool.diag_scripts.mlr.models.linear_base.LinearModel`
`method`), 639 `features` (esmval-
`export_training_data()` (esmval- `property`), 723
`tool.diag_scripts.mlr.models.linear_base.LinearModel`
`method`), 639 `features` (esmval-
`export_training_data()` (esmval- `property`), 731
`tool.diag_scripts.mlr.models.MLRModel`
`method`), 623 `features` (esmval-
`export_training_data()` (esmval- `property`), 739
`tool.diag_scripts.mlr.models.rfr.RFRModel`
`method`), 723 `features` (esmval-
`export_training_data()` (esmval- `property`), 747
`tool.diag_scripts.mlr.models.ridge.RidgeModel`
`method`), 731 `features_after_preprocessing` (esmval-
`export_training_data()` (esmval- `tool.diag_scripts.mlr.models.gbr_base.GBRModel`
`method`), 731 `property`), 631
`tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel` `features_after_preprocessing` (esmval-
`method`), 739 `tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel`
`export_training_data()` (esmval- `property`), 647
`tool.diag_scripts.mlr.models.svr.SVRModel` `features_after_preprocessing` (esmval-
`method`), 747 `tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel`
`extract_variables()` (in module esmval- `property`), 656
`tool.diag_scripts.shared`), 561 `features_after_preprocessing` (esmval-
`feature_importances_` (esmval- `tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel`
`tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline` `property`), 667
`property`), 601 `features_after_preprocessing` (esmval-
`features_after_preprocessing` (esmval-
`tool.diag_scripts.mlr.models.huber.HuberRegressionModel`
`property`), 675
`features_after_preprocessing` (esmval-

<code>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</code> (property), 715	<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.linear_base.LinearModel</code> method), 639
<code>features_units</code> (<code>esmvaltool.diag_scripts.mlr.models.MLRModel</code> property), 639	<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.rfr.RFRModel</code> method), 723
<code>features_units</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel</code> property), 623	<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</code> method), 731
<code>features_units</code> (<code>esmvaltool.diag_scripts.mlr.models.rfr.RFRModel</code> property), 723	<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.svr.SVRModel</code> method), 747
<code>features_units</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel</code> property), 731	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel</code> property), 631
<code>features_units</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</code> property), 739	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</code> property), 648
<code>features_units</code> (<code>esmvaltool.diag_scripts.mlr.models.svr.SVRModel</code> property), 747	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> property), 656
<code>FeatureSelectionTransformer</code> (class in <code>esmvaltool.diag_scripts.mlr.custom_sklearn</code>), 614	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> property), 667
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 601	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel</code> property), 675
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 606	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.krr.KRRModel</code> property), 683
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 609	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.lasso.LassoModel</code> property), 691
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 612	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</code> property), 699
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 614	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code> property), 707
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel</code> method), 631	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel</code> property), 715
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</code> method), 647	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.linear_base.LinearModel</code> property), 639
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> method), 656	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.MLRModel</code> property), 623
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.gpr_sklearn.AdvancedPipeline</code> method), 663	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.rfr.RFRModel</code> property), 723
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> method), 667	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel</code> property), 731
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel</code> method), 675	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</code> property), 739
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.krr.KRRModel</code> method), 683	<code>fit_kwargs</code> (<code>esmvaltool.diag_scripts.mlr.models.svr.SVRModel</code> property), 747
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.lasso.LassoModel</code> method), 691	<code>fit_predict()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 602
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</code> method), 699	<code>fit_target_transformer_only()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 602
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code> method), 707	<code>fit_transform()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 602
<code>fit()</code> (<code>esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel</code> method), 715	<code>fit_transform()</code> (<code>esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 602

`tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE`
`method)`, 606

`fit_transform()` (`esmval-` `get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV` `tool.diag_scripts.mlr.models.linear.LinearRegressionModel`
`method)`, 609 `method)`, 715

`fit_transform()` (`esmval-` `get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer` `tool.diag_scripts.mlr.models.linear_base.LinearModel`
`method)`, 614 `method)`, 639

`fit_transformer_only()` (`esmval-` `get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.custom_sklearn.AdvancedTransformedTool` `tool.diag_scripts.mlr.models.MLRModel`
`method)`, 612 `method)`, 623

`fit_transformers_only()` (`esmval-` `get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline` `tool.diag_scripts.mlr.models.rfr.RFRModel`
`method)`, 602 `method)`, 723

`folder()` (`in` `module` `esmval-` `get_ancestors()` (`esmval-`
`tool.diag_scripts.ocean.diagnostic_tools)`, `tool.diag_scripts.mlr.models.ridge.RidgeModel`
 778 `method)`, 731

G

`GBRModel` (`class` `in` `esmval-`
`tool.diag_scripts.mlr.models.gbr_base)`, 629

`get_1d_cube()` (`in` `module` `esmval-`
`tool.diag_scripts.mlr)`, 595

`get_absolute_time_units()` (`in` `module` `esmval-`
`tool.diag_scripts.mlr)`, 595

`get_alias()` (`in` `module` `esmvaltool.diag_scripts.mlr)`,
 596

`get_all_weights()` (`in` `module` `esmval-`
`tool.diag_scripts.mlr)`, 596

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_base.GBRModel`
`method)`, 631

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel`
`method)`, 648

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel`
`method)`, 656

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGBRModel`
`method)`, 667

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.huber.HuberRegressionModel`
`method)`, 675

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.krr.KRRModel`
`method)`, 683

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.lasso.LassoModel`
`method)`, 691

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel`
`method)`, 699

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel`
`method)`, 707

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.linear.LinearRegressionModel`
`method)`, 715

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.linear_base.LinearModel`
`method)`, 639

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.MLRModel`
`method)`, 623

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.rfr.RFRModel`
`method)`, 723

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.ridge.RidgeModel`
`method)`, 731

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel`
`method)`, 739

`get_ancestors()` (`esmval-`
`tool.diag_scripts.mlr.models.svr.SVRModel`
`method)`, 747

`get_area_weights()` (`in` `module` `esmval-`
`tool.diag_scripts.mlr)`, 597

`get_array_range()` (`in` `module` `esmval-`
`tool.diag_scripts.ocean.diagnostic_tools)`,
 778

`get_caption()` (`in` `module` `esmval-`
`tool.diag_scripts.emergent_constraints)`,
 578

`get_cfg()` (`in` `module` `esmvaltool.diag_scripts.shared)`,
 561

`get_color()` (`in` `module` `esmval-`
`tool.diag_scripts.emergent_constraints)`,
 578

`get_color_from_cmap()` (`in` `module` `esmval-`
`tool.diag_scripts.ocean.diagnostic_tools)`,
 779

`get_constraint()` (`in` `module` `esmval-`
`tool.diag_scripts.emergent_constraints)`,
 579

`get_constraint_from_df()` (`in` `module` `esmval-`
`tool.diag_scripts.emergent_constraints)`, 579

`get_control_exper_obs()` (`in` `module` `esmval-`
`tool.diag_scripts.shared)`, 561

`get_cube_range()` (`in` `module` `esmval-`
`tool.diag_scripts.ocean.diagnostic_tools)`,
 779

`get_cube_range_diff()` (`in` `module` `esmval-`
`tool.diag_scripts.ocean.diagnostic_tools)`,
 779

`get_data()` (`esmvaltool.diag_scripts.shared.Datasets`
`method)`, 553

<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> method), 632	(<i>esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> method), 632	<code>get_dataset_info_list()</code> (<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 554	(<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 554
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> method), 648	(<i>esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> method), 648	<code>get_dataset_style()</code> (in module <i>esmval-tool.diag_scripts.shared.plot</i>), 567	(in module <i>esmval-tool.diag_scripts.shared.plot</i>), 567
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> method), 657	(<i>esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> method), 657	<code>get_decade()</code> (in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 579	(in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 579
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGBRModel</i> method), 668	(<i>esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGBRModel</i> method), 668	<code>get_diagnostic_filename()</code> (in module <i>esmval-tool.diag_scripts.shared</i>), 561	(in module <i>esmval-tool.diag_scripts.shared</i>), 561
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> method), 676	(<i>esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> method), 676	<code>get_feature_names_out()</code> (in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</i> method), 602	(in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</i> method), 602
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.krr.KRRModel</i> method), 684	(<i>esmval-tool.diag_scripts.mlr.models.krr.KRRModel</i> method), 684	<code>get_feature_names_out()</code> (in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE</i> method), 607	(in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE</i> method), 607
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.lasso.LassoModel</i> method), 692	(<i>esmval-tool.diag_scripts.mlr.models.lasso.LassoModel</i> method), 692	<code>get_feature_names_out()</code> (in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV</i> method), 610	(in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV</i> method), 610
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVMModel</i> method), 700	(<i>esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVMModel</i> method), 700	<code>get_feature_names_out()</code> (in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer</i> method), 614	(in module <i>esmval-tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer</i> method), 614
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVMModel</i> method), 708	(<i>esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVMModel</i> method), 708	<code>get_groups()</code> (in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 579	(in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 579
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVMModel</i> method), 708	(<i>esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVMModel</i> method), 708	<code>get_horizontal_weights()</code> (in module <i>esmval-tool.diag_scripts.mlr</i>), 597	(in module <i>esmval-tool.diag_scripts.mlr</i>), 597
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> method), 716	(<i>esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> method), 716	<code>get_image_format()</code> (in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 779	(in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 779
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel</i> method), 640	(<i>esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel</i> method), 640	<code>get_image_path()</code> (in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 779	(in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 779
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.MLRModel</i> method), 624	(<i>esmval-tool.diag_scripts.mlr.models.MLRModel</i> method), 624	<code>get_info()</code> (<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 554	(<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 554
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 724	(<i>esmval-tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 724	<code>get_info_list()</code> (<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 555	(<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 555
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel</i> method), 732	(<i>esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel</i> method), 732	<code>get_input_data()</code> (in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 579	(in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 579
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVMModel</i> method), 740	(<i>esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVMModel</i> method), 740	<code>get_input_data()</code> (in module <i>esmval-tool.diag_scripts.mlr</i>), 598	(in module <i>esmval-tool.diag_scripts.mlr</i>), 598
<code>get_data_frame()</code> (<i>esmval-tool.diag_scripts.mlr.models.svr.SVRModel</i> method), 748	(<i>esmval-tool.diag_scripts.mlr.models.svr.SVRModel</i> method), 748	<code>get_input_files()</code> (in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 580	(in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 580
<code>get_data_list()</code> (<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 553	(<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 553	<code>get_input_files()</code> (in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 780	(in module <i>esmval-tool.diag_scripts.ocean.diagnostic_tools</i>), 780
<code>get_dataset_info()</code> (<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 553	(<i>esmval-tool.diag_scripts.shared.Datasets</i> method), 553	<code>get_landsea_fraction_weights()</code> (in module <i>esmval-tool.diag_scripts.mlr</i>), 598	(in module <i>esmval-tool.diag_scripts.mlr</i>), 598
		<code>get_mean_cube()</code> (in module <i>esmval-tool.diag_scripts.shared.iris_helpers</i>), 565	(in module <i>esmval-tool.diag_scripts.shared.iris_helpers</i>), 565
		<code>get_new_path()</code> (in module <i>esmval-</i>	(in module <i>esmval-</i>

<code>tool.diag_scripts.mlr)</code> , 598	<code>tool.diag_scripts.mlr)</code> , 599
<code>get_obs_projects()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_tools</code>), 780	<code>get_support()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE</code> method), 607
<code>get_params()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> method), 603	<code>get_support()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV</code> method), 610
<code>get_params()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE</code> method), 607	<code>get_support()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformers</code> method), 615
<code>get_params()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV</code> method), 610	<code>get_time_string()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 775
<code>get_params()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedTransformedToolDiagScripts</code> method), 612	<code>get_time_weights()</code> (in module <code>esmval-tool.diag_scripts.mlr)</code> , 599
<code>get_params()</code> (<code>esmval-tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformers</code> method), 614	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel</code> method), 632
<code>get_params()</code> (<code>esmval-tool.diag_scripts.mlr.models.gpr_sklearn.AdvancedGaussianMethodSklearnRegressor</code> method), 663	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</code> method), 648
<code>get_path()</code> (<code>esmvaltool.diag_scripts.shared.Datasets</code> method), 555	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> method), 657
<code>get_path_list()</code> (<code>esmval-tool.diag_scripts.shared.Datasets</code> method), 555	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> method), 668
<code>get_path_to_mpl_style()</code> (in module <code>esmval-tool.diag_scripts.shared.plot</code>), 567	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel</code> method), 676
<code>get_plot_filename()</code> (in module <code>esmval-tool.diag_scripts.shared</code>), 561	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.krr.KRRModel</code> method), 684
<code>get_plot_folder()</code> (<code>esmval-tool.diag_scripts.monitor.monitor_base.MonitorBase</code> method), 760	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.lasso.LassoModel</code> method), 692
<code>get_plot_name()</code> (<code>esmval-tool.diag_scripts.monitor.monitor_base.MonitorBase</code> method), 760	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</code> method), 700
<code>get_plot_path()</code> (<code>esmval-tool.diag_scripts.monitor.monitor_base.MonitorBase</code> method), 760	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code> method), 708
<code>get_pole()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 775	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel</code> method), 716
<code>get_provenance_record()</code> (<code>esmval-tool.diag_scripts.monitor.monitor_base.MonitorBase</code> static method), 761	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel</code> method), 640
<code>get_provenance_record()</code> (in module <code>esmval-tool.diag_scripts.emergent_constraints</code>), 580	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.MLRModel</code> method), 624
<code>get_rfecv_transformer()</code> (in module <code>esmval-tool.diag_scripts.mlr.custom_sklearn</code>), 616	<code>get_x_array()</code> (<code>esmval-tool.diag_scripts.mlr.models.rfr.RFRModel</code> method), 724
<code>get_season()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 775	
<code>get_squared_error_cube()</code> (in module <code>esmval-</code>	

<code>get_x_array()</code>	(esmval- tool.diag_scripts.mlr.models.ridge.RidgeModel method), 732	<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.svr.SVRModel method), 748
<code>get_x_array()</code>	(esmval- tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 740	<code>get_year()</code>	(in module esmval- tool.diag_scripts.ocean.diagnostic_seaice), 775
<code>get_x_array()</code>	(esmval- tool.diag_scripts.mlr.models.svr.SVRModel method), 748	<code>global_contourf()</code>	(in module esmval- tool.diag_scripts.shared.plot), 567
<code>get_xy_data_without_nans()</code>	(in module esmval- tool.diag_scripts.emergent_constraints), 580	<code>global_pcolormesh()</code>	(in module esmval- tool.diag_scripts.shared.plot), 568
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.gbr_base.GBRModel method), 632	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.gbr_base.GBRModel method), 633
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel method), 649	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel method), 649
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel method), 657	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel method), 658
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel method), 668	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel method), 669
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 676	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 677
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.krr.KRRModel method), 684	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.krr.KRRModel method), 685
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.lasso.LassoModel method), 692	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.lasso.LassoModel method), 693
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 700	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 701
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 708	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 709
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 716	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 717
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.linear_base.LinearModel method), 640	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.linear_base.LinearModel method), 641
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.MLRModel method), 624	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.MLRModel method), 625
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.rfr.RFRModel method), 724	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.rfr.RFRModel method), 725
<code>get_y_array()</code>	(esmval- tool.diag_scripts.mlr.models.ridge.RidgeModel method), 732	<code>grid_search_cv()</code>	(esmval- tool.diag_scripts.mlr.models.ridge.RidgeModel method), 733
<code>get_y_array()</code>	(esmval-		

`grid_search_cv()` (*esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* method), 741
`grid_search_cv()` (*esmval-tool.diag_scripts.mlr.models.svr.SVRModel* method), 749
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel* property), 633
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* property), 649
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* property), 658
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* property), 669
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel* property), 677
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.krr.KRRModel* property), 685
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.lasso.LassoModel* property), 693
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* property), 701
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* property), 709
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel* property), 717
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel* property), 641
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.MLRModel* property), 625
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.rfr.RFRModel* property), 725
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel* property), 733
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* property), 741
`group_attributes` (*esmval-tool.diag_scripts.mlr.models.svr.SVRModel* property), 749
`group_metadata()` (in module *esmval-tool.diag_scripts.shared*), 562
`guess_calendar_datetime()` (in module *esmval-tool.diag_scripts.ocean.diagnostic_tools*), 780

H

`HuberRegressionModel` (class in *esmval-tool.diag_scripts.mlr.models.huber*), 673
`ignore_warnings()` (in module *esmval-tool.diag_scripts.mlr*), 599
`Index()` (*esmvaltool.diag_scripts.shared.Variable* method), 557
`intersect_dataset_coordinates()` (in module *esmvaltool.diag_scripts.shared.iris_helpers*), 565
`inverse_transform()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* method), 603
`inverse_transform()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE* method), 607
`inverse_transform()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV* method), 610
`inverse_transform()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer* method), 615
`iris_dict()` (*esmvaltool.diag_scripts.shared.Variables* method), 559
`iris_project_constraint()` (in module *esmval-tool.diag_scripts.shared.iris_helpers*), 565

K

`KRRModel` (class in *esmval-tool.diag_scripts.mlr.models.krr*), 681

L

`label` (*esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel* property), 633
`label` (*esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* property), 649
`label` (*esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* property), 658
`label` (*esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* property), 669
`label` (*esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel* property), 677
`label` (*esmvaltool.diag_scripts.mlr.models.krr.KRRModel* property), 685
`label` (*esmvaltool.diag_scripts.mlr.models.lasso.LassoModel* property), 693

<code>main()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_timeseries</code>), 769	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel</code> property), 633
<code>main()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_transects</code>), 771	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</code> property), 650
<code>make_cube_layer_dict()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_tools</code>), 781	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> property), 658
<code>make_cube_region_dict()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_transects</code>), 772	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> property), 669
<code>make_depth_safe()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_transects</code>), 772	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel</code> property), 677
<code>make_map_contour()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_maps</code>), 762	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.krr.KRRModel</code> property), 685
<code>make_map_extent_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 776	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.lasso.LassoModel</code> property), 693
<code>make_map_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_maps</code>), 762	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</code> property), 701
<code>make_map_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 776	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code> property), 709
<code>make_model_vs_obs_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_model_vs_obs</code>), 766	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel</code> property), 717
<code>make_polar_map()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 776	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel</code> property), 641
<code>make_profiles_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_profiles</code>), 768	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.MLRModel</code> property), 625
<code>make_scatter()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_model_vs_obs</code>), 766	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.rfr.RFRModel</code> property), 725
<code>make_time_series_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_timeseries</code>), 769	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel</code> property), 733
<code>make_transect_contours()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_transects</code>), 772	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</code> property), 741
<code>make_transects_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_transects</code>), 772	<code>mlr_model_type</code> (<code>esmval-tool.diag_scripts.mlr.models.svr.SVRModel</code> property), 749
<code>make_ts_plots()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_seaice</code>), 776	<code>MLRModel</code> (class in <code>esmvaltool.diag_scripts.mlr.models</code>), 621
<code>match_model_to_key()</code> (in module <code>esmval-tool.diag_scripts.ocean.diagnostic_tools</code>), 781	<code>modify_var()</code> (<code>esmval-tool.diag_scripts.shared.Variables</code> method), 559
	module

esmvaltool.diag_scripts.emergent_constraints, 593
 575
 esmvaltool.diag_scripts.emergent_constraints.cox18nature, 556
 571
 esmvaltool.diag_scripts.emergent_constraints.ecs_scatter, 553
 572
 esmvaltool.diag_scripts.emergent_constraints.multi_constraints, 571
 573
 esmvaltool.diag_scripts.emergent_constraints.single_constraint, 556
 574
 esmvaltool.diag_scripts.mlr, 594
 esmvaltool.diag_scripts.mlr.custom_sklearn, 761
 600
 esmvaltool.diag_scripts.mlr.evaluate_residuals, 763
 584
 esmvaltool.diag_scripts.mlr.main, 585
 esmvaltool.diag_scripts.mlr.mmm, 586
 esmvaltool.diag_scripts.mlr.models, 616
 esmvaltool.diag_scripts.mlr.models.gbr_base, 773
 629
 esmvaltool.diag_scripts.mlr.models.gbr_sklearn, 768
 645
 esmvaltool.diag_scripts.mlr.models.gbr_xgboost, 776
 653
 esmvaltool.diag_scripts.mlr.models.gpr_sklearn, 770
 662
 esmvaltool.diag_scripts.mlr.models.huber, 551
 673
 esmvaltool.diag_scripts.mlr.models.krr, 564
 681
 esmvaltool.diag_scripts.mlr.models.lasso, MonitorBase (class in esmval-
 689 tool.diag_scripts.monitor.monitor_base),
 esmvaltool.diag_scripts.mlr.models.lasso_cv, 760
 697
 esmvaltool.diag_scripts.mlr.models.lasso_lars_cv, moving_average() (in module esmval-
 705 tool.diag_scripts.ocean.diagnostic_timeseries),
 769
 esmvaltool.diag_scripts.mlr.models.linear, multi_dataset_scatterplot() (in module esmval-
 713 tool.diag_scripts.shared.plot), 568
 esmvaltool.diag_scripts.mlr.models.linear_base, base_model_contours() (in module esmval-
 637 tool.diag_scripts.ocean.diagnostic_maps),
 esmvaltool.diag_scripts.mlr.models.rfr, 762
 721
 esmvaltool.diag_scripts.mlr.models.ridge, multi_model_contours() (in module esmval-
 729 tool.diag_scripts.ocean.diagnostic_transects),
 773
 esmvaltool.diag_scripts.mlr.models.ridge_cv, multi_model_maps() (in module esmval-
 737 tool.diag_scripts.ocean.diagnostic_maps_quad),
 764
 esmvaltool.diag_scripts.mlr.models.svr, multi_model_time_series() (in module esmval-
 745 tool.diag_scripts.ocean.diagnostic_timeseries),
 770
 esmvaltool.diag_scripts.mlr.plot, 587
 esmvaltool.diag_scripts.mlr.postprocess, 589
 589
 esmvaltool.diag_scripts.mlr.preprocess, N
 591
 esmvaltool.diag_scripts.mlr.rescale_with_emergent_constraints, n_features_in_ (esmval-
 esmvaltool.diag_scripts.mlr.rescale_with_emergent_constraints, esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline

`property`), 603
`n_features_in_` (`esmval-`
`tool.diag_scripts.mlr.custom_sklearn.AdvancedTransformedTargetRegressor`
`property`), 613
`named_steps` (`esmval-`
`tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline`
`property`), 603
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_base.GBRModel`
`property`), 633
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel`
`property`), 650
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel`
`property`), 658
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel`
`property`), 669
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.gpr_xgboost.XGBoostGPRModel`
`property`), 677
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.huber.HuberRegressionModel`
`property`), 685
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.krr.KRRModel`
`property`), 685
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel`
`property`), 669
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.huber.HuberRegressionModel`
`property`), 677
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.krr.KRRModel`
`property`), 685
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.lasso.LassoModel`
`property`), 693
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel`
`property`), 701
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel`
`property`), 709
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.linear.LinearRegressionModel`
`property`), 717
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.linear_base.LinearModel`
`property`), 641
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.MLRModel`
`property`), 625
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel`
`property`), 701
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel`
`property`), 709
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.linear.LinearRegressionModel`
`property`), 717
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.linear_base.LinearModel`
`property`), 641
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.MLRModel`
`property`), 625
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.rfr.RFRModel`
`property`), 725
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.ridge.RidgeModel`
`property`), 733
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel`
`property`), 741
`numerical_features` (`esmval-`
`tool.diag_scripts.mlr.models.svr.SVRModel`
`property`), 749
`perform_efecv()` (in module `esmval-`
`tool.diag_scripts.mlr.custom_sklearn`), 616
`plot_1d_model()` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_base.GBRModel`
`method`), 633
`plot_1d_model()` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel`
`method`), 650
`plot_1d_model()` (`esmval-`
`tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel`
`method`), 659
`plot_1d_model()` (`esmval-`
`tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel`
`method`), 669
`plot_1d_model()` (`esmval-`
`tool.diag_scripts.mlr.models.huber.HuberRegressionModel`
`method`), 678

<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.krr.KRRModel method), 685</i>	<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 742</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso.LassoModel method), 694</i>	<code>plot_cube()</code> <i>(esmval-tool.diag_scripts.monitor.monitor_base.MonitorBase method), 761</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 701</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel method), 634</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 709</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel method), 650</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 718</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel method), 659</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel method), 642</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 678</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.MLRModel method), 625</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso.LassoModel method), 694</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.rfr.RFRModel method), 725</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 702</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel method), 734</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 710</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 741</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 718</i>
<code>plot_1d_model()</code> <i>(esmval-tool.diag_scripts.mlr.models.svr.SVRModel method), 749</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel method), 642</i>
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 678</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel method), 734</i>
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso.LassoModel method), 694</i>	<code>plot_feature_importance()</code> <i>(esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 742</i>
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 702</i>	<code>plot_individual_scatterplots()</code> (in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 581
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 710</i>	<code>plot_merged_scatterplots()</code> (in module <i>esmval-tool.diag_scripts.emergent_constraints</i>), 581
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 718</i>	<code>plot_partial_dependences()</code> <i>(esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel method), 634</i>
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel method), 642</i>	<code>plot_partial_dependences()</code> <i>(esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel method), 651</i>
<code>plot_coefs()</code> <i>(esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel method), 734</i>	<code>plot_partial_dependences()</code> <i>(esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel method), 659</i>
	<code>plot_partial_dependences()</code> <i>(esmval-</i>

<i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> <i>method</i>), 670	<i>tool.diag_scripts.mlr.models.krr.KRRModel</i> <i>method</i>), 686
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> <i>method</i>), 678	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> <i>method</i>), 695
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> <i>method</i>), 686	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> <i>method</i>), 702
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> <i>method</i>), 694	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> <i>method</i>), 710
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> <i>method</i>), 702	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> <i>method</i>), 719
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> <i>method</i>), 710	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> <i>method</i>), 643
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> <i>method</i>), 718	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.MLRModel</i> <i>method</i>), 626
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> <i>method</i>), 642	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> <i>method</i>), 726
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.MLRModel</i> <i>method</i>), 626	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.ridge.RidgeModel</i> <i>method</i>), 735
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> <i>method</i>), 726	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</i> <i>method</i>), 742
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.ridge.RidgeModel</i> <i>method</i>), 734	<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.svr.SVRModel</i> <i>method</i>), 750
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</i> <i>method</i>), 742	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> <i>method</i>), 634
<i>plot_partial_dependences()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.svr.SVRModel</i> <i>method</i>), 750	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> <i>method</i>), 651
<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> <i>method</i>), 634	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> <i>method</i>), 660
<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> <i>method</i>), 651	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> <i>method</i>), 670
<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> <i>method</i>), 659	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> <i>method</i>), 679
<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> <i>method</i>), 670	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> <i>method</i>), 686
<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> <i>method</i>), 679	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> <i>method</i>), 695
<i>plot_prediction_errors()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> <i>method</i>), 670	<i>plot_residuals()</i> (<i>esmval-</i> <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> <i>method</i>), 686

<i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> <i>method</i>), 703	<i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> <i>method</i>), 719	
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> <i>method</i>), 711	<i>(esmval- plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> <i>method</i>), 643	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> <i>method</i>), 719	<i>(esmval- plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.MLRModel</i> <i>method</i>), 626	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> <i>method</i>), 643	<i>(esmval- plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> <i>method</i>), 726	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.MLRModel</i> <i>method</i>), 626	<i>(esmval- plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.ridge.RidgeModel</i> <i>method</i>), 735	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> <i>method</i>), 726	<i>(esmval- plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</i> <i>method</i>), 743	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.ridge.RidgeModel</i> <i>method</i>), 735	<i>(esmval- plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.svr.SVRModel</i> <i>method</i>), 750	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</i> <i>method</i>), 743	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> <i>method</i>), 635	<i>(esmval-</i>
<i>plot_residuals()</i> <i>tool.diag_scripts.mlr.models.svr.SVRModel</i> <i>method</i>), 750	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> <i>method</i>), 651	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> <i>method</i>), 635	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> <i>method</i>), 660	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> <i>method</i>), 651	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> <i>method</i>), 670	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> <i>method</i>), 660	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> <i>method</i>), 679	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> <i>method</i>), 670	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> <i>method</i>), 686	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> <i>method</i>), 679	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> <i>method</i>), 695	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> <i>method</i>), 686	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> <i>method</i>), 703	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> <i>method</i>), 695	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> <i>method</i>), 711	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> <i>method</i>), 703	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> <i>method</i>), 719	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> <i>method</i>), 711	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> <i>method</i>), 643	<i>(esmval-</i>
<i>plot_residuals_distribution()</i> <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> <i>method</i>), 719	<i>(esmval- plot_residuals_histogram()</i> <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> <i>method</i>), 643	<i>(esmval-</i>

<i>tool.diag_scripts.mlr.models.MLRModel</i> method), 626	<i>tool.diag_scripts.mlr.models.ridge.RidgeModel</i> method), 735
<i>plot_residuals_histogram()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 726	<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</i> method), 743
<i>plot_residuals_histogram()</i> (esmval- <i>tool.diag_scripts.mlr.models.ridge.RidgeModel</i> method), 735	<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.svr.SVRModel</i> method), 751
<i>plot_residuals_histogram()</i> (esmval- <i>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</i> method), 743	<i>plot_target_distributions()</i> (in module <i>esmval- tool.diag_scripts.emergent_constraints</i>), 582
<i>plot_residuals_histogram()</i> (esmval- <i>tool.diag_scripts.mlr.models.svr.SVRModel</i> method), 750	<i>plot_timeseries()</i> (esmval- <i>tool.diag_scripts.monitor.monitor_base.MonitorBase</i> method), 761
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> method), 635	<i>plot_training_progress()</i> (esmval- <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> method), 652
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> method), 651	<i>plot_training_progress()</i> (esmval- <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> method), 660
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> method), 660	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</i> method), 603
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> method), 660	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE</i> method), 608
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> method), 671	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV</i> method), 611
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> method), 679	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.custom_sklearn.AdvancedTransformer</i> method), 613
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> method), 687	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.gbr_base.GBRModel</i> method), 635
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> method), 695	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</i> method), 652
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> method), 703	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</i> method), 660
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> method), 711	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.gpr_sklearn.AdvancedGaussianProcess</i> method), 664
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> method), 719	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</i> method), 671
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> method), 643	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</i> method), 679
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.MLRModel</i> method), 627	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.krr.KRRModel</i> method), 687
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.lasso.LassoModel</i> method), 695
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</i> method), 703
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</i> method), 711
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</i> method), 719
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.linear_base.LinearModel</i> method), 643
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.MLRModel</i> method), 627
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 627
<i>plot_scatterplots()</i> (esmval- <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 727	<i>predict()</i> (esmval <i>tool.diag_scripts.mlr.models.rfr.RFRModel</i> method), 627

<code>method</code>), 727	<code>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code>
<code>predict()</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel</code> <code>method</code>), 712	<code>method</code>), 712
<code>method</code>), 735	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict()</code> (<code>esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</code> <code>method</code>), 743	<code>tool.diag_scripts.mlr.models.linear.LinearRegressionModel</code> <code>method</code>), 720
<code>predict()</code> (<code>esmvaltool.diag_scripts.mlr.models.svr.SVRModel</code> <code>method</code>), 751	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict_log_proba()</code> (<code>esmval-</code> <code>method</code>), 644	<code>tool.diag_scripts.mlr.models.linear_base.LinearModel</code> <code>method</code>), 644
<code>tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> <code>method</code>), 604	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict_log_proba()</code> (<code>esmval-</code> <code>method</code>), 627	<code>tool.diag_scripts.mlr.models.MLRModel</code> <code>method</code>), 627
<code>tool.diag_scripts.mlr.custom_sklearn.AdvancedRF</code> <code>method</code>), 608	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict_log_proba()</code> (<code>esmval-</code> <code>method</code>), 727	<code>tool.diag_scripts.mlr.models.rfr.RFRModel</code> <code>method</code>), 727
<code>tool.diag_scripts.mlr.custom_sklearn.AdvancedRF</code> <code>method</code>), 611	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict_proba()</code> (<code>esmval-</code> <code>method</code>), 736	<code>tool.diag_scripts.mlr.models.ridge.RidgeModel</code> <code>method</code>), 736
<code>tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline</code> <code>method</code>), 604	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict_proba()</code> (<code>esmval-</code> <code>method</code>), 744	<code>tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel</code> <code>method</code>), 744
<code>tool.diag_scripts.mlr.custom_sklearn.AdvancedRF</code> <code>method</code>), 608	<code>print_correlation_matrices()</code> (<code>esmval-</code>
<code>predict_proba()</code> (<code>esmval-</code> <code>method</code>), 751	<code>tool.diag_scripts.mlr.models.svr.SVRModel</code> <code>method</code>), 751
<code>tool.diag_scripts.mlr.custom_sklearn.AdvancedRF</code> <code>method</code>), 611	<code>print_kernel_info()</code> (<code>esmval-</code>
<code>prepare_cube_for_merging()</code> (in module <code>esmval-</code> <code>method</code>), 672	<code>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> <code>method</code>), 672
<code>tool.diag_scripts.shared.iris_helpers</code>), 566	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>prepare_provenance_record()</code> (in module <code>esmval-</code> <code>method</code>), 636	<code>tool.diag_scripts.mlr.models.gbr_base.GBRModel</code> <code>method</code>), 636
<code>tool.diag_scripts.ocean.diagnostic_tools</code>), 781	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 636	<code>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</code> <code>method</code>), 652
<code>tool.diag_scripts.mlr.models.gbr_base.GBRModel</code> <code>method</code>), 636	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 652	<code>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> <code>method</code>), 661
<code>tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel</code> <code>method</code>), 652	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 661	<code>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> <code>method</code>), 661
<code>tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel</code> <code>method</code>), 661	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 671	<code>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> <code>method</code>), 672
<code>tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel</code> <code>method</code>), 671	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 680	<code>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</code> <code>method</code>), 680
<code>tool.diag_scripts.mlr.models.huber.HuberRegressionModel</code> <code>method</code>), 680	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 687	<code>tool.diag_scripts.mlr.models.krr.KRRModel</code> <code>method</code>), 688
<code>tool.diag_scripts.mlr.models.krr.KRRModel</code> <code>method</code>), 687	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 696	<code>tool.diag_scripts.mlr.models.lasso.LassoModel</code> <code>method</code>), 696
<code>tool.diag_scripts.mlr.models.lasso.LassoModel</code> <code>method</code>), 696	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 704	<code>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</code> <code>method</code>), 704
<code>tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel</code> <code>method</code>), 704	<code>print_regression_metrics()</code> (<code>esmval-</code>
<code>print_correlation_matrices()</code> (<code>esmval-</code> <code>method</code>), 712	<code>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code> <code>method</code>), 712
<code>tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel</code> <code>method</code>), 712	<code>print_regression_metrics()</code> (<code>esmval-</code>

tool.diag_scripts.mlr.models.linear.LinearRegressionModel class method), 704
method), 720

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* class method), 712

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel* class method), 712

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.MLRModel* class method), 720

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel* class method), 644

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.rfr.RFRModel* class method), 628

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel* class method), 628

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* class method), 728

print_regression_metrics() (*esmval-tool.diag_scripts.mlr.models.svr.SVRModel* class method), 736

ProvenanceLogger (class in *esmval-tool.diag_scripts.shared*), 556

Q

quickplot() (in module *esmval-tool.diag_scripts.shared.plot*), 569

R

record_plot_provenance() (*esmval-tool.diag_scripts.monitor.monitor_base.MonitorBase* class method), 761

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel* class method), 636

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* class method), 652

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* class method), 661

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* class method), 672

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel* class method), 680

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.krr.KRRModel* class method), 688

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.lasso.LassoModel* class method), 696

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* class method), 704

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* class method), 712

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel* class method), 720

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel* class method), 644

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.MLRModel* class method), 628

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.rfr.RFRModel* class method), 628

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel* class method), 736

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* class method), 728

register_mlr_model() (*esmval-tool.diag_scripts.mlr.models.svr.SVRModel* class method), 752

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel* class method), 636

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* class method), 653

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* class method), 661

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* class method), 672

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel* class method), 680

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.krr.KRRModel* class method), 688

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.lasso.LassoModel* class method), 696

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* class method), 704

reset_pipeline() (*esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* class method), 712

method), 712

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 720

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel method), 644

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.MLRModel method), 628

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.rfr.RFRModel method), 728

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel method), 736

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 744

reset_pipeline() (esmval-tool.diag_scripts.mlr.models.svr.SVRModel method), 752

rfevcv() (esmvaltool.diag_scripts.mlr.models.gbr_base.GBRModel method), 636

rfevcv() (esmvaltool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel method), 653

rfevcv() (esmvaltool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel method), 661

rfevcv() (esmvaltool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel method), 672

rfevcv() (esmvaltool.diag_scripts.mlr.models.huber.HuberRegressionModel method), 680

rfevcv() (esmvaltool.diag_scripts.mlr.models.krr.KRRModel method), 688

rfevcv() (esmvaltool.diag_scripts.mlr.models.lasso.LassoModel method), 696

rfevcv() (esmvaltool.diag_scripts.mlr.models.lasso_cv.LassoCVModel method), 704

rfevcv() (esmvaltool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel method), 712

rfevcv() (esmvaltool.diag_scripts.mlr.models.linear.LinearRegressionModel method), 720

rfevcv() (esmvaltool.diag_scripts.mlr.models.linear_base.LinearModel method), 644

rfevcv() (esmvaltool.diag_scripts.mlr.models.MLRModel method), 628

rfevcv() (esmvaltool.diag_scripts.mlr.models.rfr.RFRModel method), 728

rfevcv() (esmvaltool.diag_scripts.mlr.models.ridge.RidgeModel method), 736

rfevcv() (esmvaltool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel method), 744

rfevcv() (esmvaltool.diag_scripts.mlr.models.svr.SVRModel method), 752

RFRModel (class in esmval-tool.diag_scripts.mlr.models.rfr), 721

RidgeCVModel (class in esmval-tool.diag_scripts.mlr.models.ridge_cv), 737

RidgeModel (class in esmval-tool.diag_scripts.mlr.models.ridge), 729

rounds_sig() (in module esmval-tool.diag_scripts.ocean.diagnostic_model_vs_obs), 766

run_diagnostic() (in module esmval-tool.diag_scripts.shared), 562

S

sample_y() (esmvaltool.diag_scripts.mlr.models.gpr_sklearn.AdvancedGaussianProcessRegression method), 664

save_data() (in module esmval-tool.diag_scripts.shared), 562

save_figure() (in module esmval-tool.diag_scripts.shared), 563

scatterplot() (in module esmval-tool.diag_scripts.shared.plot), 569

score() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline method), 604

score() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedRFE method), 608

score() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV method), 611

score() (esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedTransformedTarget method), 613

score() (esmvaltool.diag_scripts.mlr.models.gpr_sklearn.AdvancedGaussianProcessRegression method), 664

score_samples() (esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline method), 605

select_metadata() (in module esmval-tool.diag_scripts.shared), 563

set_data() (esmvaltool.diag_scripts.shared.Datasets method), 555

set_params() (esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline method), 605

set_params() (esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE method), 608

set_params() (esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV method), 611

set_params() (esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedTransformedTarget method), 613

set_params() (esmval-tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer method), 615

`set_params()` (*esmval-tool.diag_scripts.mlr.models.gpr_sklearn.AdvancedPipeline* method), 662
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGPRModel* method), 672
`set_plot_appearance()` (*in module esmval-tool.diag_scripts.emergent_constraints*), 582
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.huber.HuberRegressionModel* method), 681
`short_name` (*esmvaltool.diag_scripts.shared.Variable* attribute), 558
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.krr.KRRModel* method), 688
`short_name()` (*esmval-tool.diag_scripts.shared.Variables* method), 559
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.lasso.LassoModel* method), 697
`short_names()` (*esmval-tool.diag_scripts.shared.Variables* method), 559
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel* method), 705
`SklearnGBRModel` (*class in esmval-tool.diag_scripts.mlr.models.gbr_sklearn*), 645
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel* method), 713
`SklearnGPRModel` (*class in esmval-tool.diag_scripts.mlr.models.gpr_sklearn*), 665
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.linear.LinearRegressionModel* method), 721
`sorted_group_metadata()` (*in module esmval-tool.diag_scripts.shared*), 563
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.linear_base.LinearModel* method), 645
`sorted_metadata()` (*in module esmval-tool.diag_scripts.shared*), 563
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.MLRModel* method), 628
`square_root_metadata()` (*in module esmval-tool.diag_scripts.mlr*), 599
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.rfr.RFRModel* method), 728
`standard_name` (*esmval-tool.diag_scripts.shared.Variable* attribute), 558
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.ridge.RidgeModel* method), 737
`standard_name()` (*esmval-tool.diag_scripts.shared.Variables* method), 560
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel* method), 745
`standard_names()` (*esmval-tool.diag_scripts.shared.Variables* method), 560
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.svr.SVRModel* method), 752
`standard_prediction_error()` (*in module esmval-tool.diag_scripts.emergent_constraints*), 583
`steps` (*esmvaltool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* attribute), 605
`timeplot()` (*in module esmval-tool.diag_scripts.ocean.diagnostic_timeseries*), 770
`SVRModel` (*class in esmval-tool.diag_scripts.mlr.models.svr*), 745
`titlify()` (*in module esmval-tool.diag_scripts.ocean.diagnostic_transects*), 773
T
`target_pdf()` (*in module esmval-tool.diag_scripts.emergent_constraints*), 583
`transform()` (*esmval-tool.diag_scripts.mlr.models.gbr_base.GBRModel* method), 636
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel* method), 653
`transform()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline* method), 605
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel* method), 699
`transform()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFE* method), 609
`test_normality_of_residuals()` (*esmval-tool.diag_scripts.mlr.custom_sklearn.AdvancedRFECV* method), 699

method), 612

transform() (esmval-
tool.diag_scripts.mlr.custom_sklearn.FeatureSelectionTransformer
method), 615

transform_only() (esmval-
tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline
method), 606

transform_target_only() (esmval-
tool.diag_scripts.mlr.custom_sklearn.AdvancedPipeline
method), 606

U

unify_1d_cubes() (in module esmval-
tool.diag_scripts.shared.iris_helpers), 566

unify_time_coord() (in module esmval-
tool.diag_scripts.shared.iris_helpers), 566

units (esmvaltool.diag_scripts.shared.Variable
attribute), 558

units() (esmvaltool.diag_scripts.shared.Variables
method), 560

units_power() (in module esmval-
tool.diag_scripts.mlr), 599

update_parameters() (esmval-
tool.diag_scripts.mlr.models.gbr_base.GBRModel
method), 636

update_parameters() (esmval-
tool.diag_scripts.mlr.models.gbr_sklearn.SklearnGBRModel
method), 653

update_parameters() (esmval-
tool.diag_scripts.mlr.models.gbr_xgboost.XGBoostGBRModel
method), 662

update_parameters() (esmval-
tool.diag_scripts.mlr.models.gpr_sklearn.SklearnGBRModel
method), 672

update_parameters() (esmval-
tool.diag_scripts.mlr.models.huber.HuberRegressionModel
method), 681

update_parameters() (esmval-
tool.diag_scripts.mlr.models.krr.KRRModel
method), 688

update_parameters() (esmval-
tool.diag_scripts.mlr.models.lasso.LassoModel
method), 697

update_parameters() (esmval-
tool.diag_scripts.mlr.models.lasso_cv.LassoCVModel
method), 705

update_parameters() (esmval-
tool.diag_scripts.mlr.models.lasso_lars_cv.LassoLarsCVModel
method), 713

update_parameters() (esmval-
tool.diag_scripts.mlr.models.linear.LinearRegressionModel
method), 721

update_parameters() (esmval-
tool.diag_scripts.mlr.models.linear_base.LinearModel
method), 645

update_parameters() (esmval-
tool.diag_scripts.mlr.models.MLRModel
method), 628

update_parameters() (esmval-
tool.diag_scripts.mlr.models.rfr.RFRModel
method), 728

update_parameters() (esmval-
tool.diag_scripts.mlr.models.ridge.RidgeModel
method), 737

update_parameters() (esmval-
tool.diag_scripts.mlr.models.ridge_cv.RidgeCVModel
method), 745

update_parameters() (esmval-
tool.diag_scripts.mlr.models.svr.SVRModel
method), 752

V

var_name() (esmvaltool.diag_scripts.shared.Variables
method), 560

var_name_constraint() (in module esmval-
tool.diag_scripts.shared.iris_helpers), 566

Variable (class in esmvaltool.diag_scripts.shared), 557

Variables (class in esmvaltool.diag_scripts.shared), 558

variables_available() (in module esmval-
tool.diag_scripts.shared), 564

vars_available() (esmval-
tool.diag_scripts.shared.Variables
method), 560

X

XGBModel (class in esmval-
tool.diag_scripts.mlr.models.gbr_xgboost), 653