<table>
<thead>
<tr>
<th>Recipe Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Porting a namelist (recipe) or diagnostic to ESMValTool v2.0</td>
<td>49</td>
</tr>
<tr>
<td>17</td>
<td>Contributing a CMORizing script for an observational dataset</td>
<td>55</td>
</tr>
<tr>
<td>18</td>
<td>GitHub Workflow</td>
<td>57</td>
</tr>
<tr>
<td>19</td>
<td>Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power</td>
<td>65</td>
</tr>
<tr>
<td>20</td>
<td>Clouds</td>
<td>69</td>
</tr>
<tr>
<td>21</td>
<td>Combined Climate Extreme Index</td>
<td>77</td>
</tr>
<tr>
<td>22</td>
<td>Indices based on area averages</td>
<td>81</td>
</tr>
<tr>
<td>23</td>
<td>Consecutive dry days</td>
<td>85</td>
</tr>
<tr>
<td>24</td>
<td>Cloud Regime Error Metric (CREM)</td>
<td>87</td>
</tr>
<tr>
<td>25</td>
<td>The Climate Variability Diagnostics Package (CVDP)</td>
<td>91</td>
</tr>
<tr>
<td>26</td>
<td>Diurnal temperature variation indicator: Difference between Tmax and Tmin for a specific day</td>
<td>95</td>
</tr>
<tr>
<td>27</td>
<td>EnsClus - Ensemble Clustering - a cluster analysis tool for climate model simulations</td>
<td>99</td>
</tr>
<tr>
<td>28</td>
<td>Extreme Events Indices - Computation of ETCCDI extreme indices and plotting</td>
<td>103</td>
</tr>
<tr>
<td>29</td>
<td>IPCC AR5 Chapter 9</td>
<td>107</td>
</tr>
<tr>
<td>30</td>
<td>Heat wave and cold wave duration</td>
<td>109</td>
</tr>
<tr>
<td>31</td>
<td>Hydroclimatic intensity and extremes (HyInt)</td>
<td>113</td>
</tr>
<tr>
<td>32</td>
<td>Landcover</td>
<td>119</td>
</tr>
<tr>
<td>33</td>
<td>Blocking metrics and indices, teleconnections and weather regimes (MiLES)</td>
<td>125</td>
</tr>
<tr>
<td>34</td>
<td>Modes of variability</td>
<td>129</td>
</tr>
<tr>
<td>35</td>
<td>Generic multi-model products</td>
<td>133</td>
</tr>
<tr>
<td>36</td>
<td>Recipes for evaluating models of the ocean</td>
<td>137</td>
</tr>
<tr>
<td>37</td>
<td>Performance metrics for essential climate parameters</td>
<td>153</td>
</tr>
<tr>
<td>38</td>
<td>Precipitation quantile bias</td>
<td>159</td>
</tr>
<tr>
<td>39</td>
<td>Runoff_ET</td>
<td>161</td>
</tr>
<tr>
<td>40</td>
<td>RainFARM stochastic downscaling</td>
<td>167</td>
</tr>
<tr>
<td>41</td>
<td>Shapeselect</td>
<td>169</td>
</tr>
<tr>
<td>42</td>
<td>Single Model Performance Index (SMPI)</td>
<td>171</td>
</tr>
<tr>
<td>43</td>
<td>SPEI</td>
<td>175</td>
</tr>
</tbody>
</table>
Part I

Preface
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.
CHAPTER 2

Objectives and approach

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.
CHAPTER 3

License

The ESMValTool is released under the Apache License, version 2.0 and citation of the ESMValTool paper (“Software Documentation Paper”) is kindly requested upon use alongside with the software doi (to be added for v2) and version number:

• Righi et al. to be added

Besides the above citation, users are kindly asked to register any journal articles (or other scientific documents) that use the software at the ESMValTool webpage (http://www.esmvaltool.org/). Citing the Software Documentation Paper and registering your paper(s) will serve to document the scientific impact of the Software, which is of vital importance for securing future funding. You should consider this an obligation if you have taken advantage of the ESMValTool, which represents the end product of considerable effort by the development team.
CHAPTER 4

Architecture

*Schematic of the system architecture.* shows a schematic of the ESMValTool architecture: to be written.
Fig. 1: Schematic of the system architecture.
CHAPTER 5

The ESMValTool core development team

5.1 Main contacts

A mailing list has been set up for all general and technical questions on the ESMValTool such as, for instance, questions on installation, application or development. You are encouraged to subscribe to the ESMValTool user mailing list by sending an email to Listserv@dlr.de with the following text:

subscribe ESMValTool-Usr

5.2 Core development team

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

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

5.3 Pull requests

This section describes the general workflow of how new diagnostics are integrated into the ESMValTool and the responsibilities of the developer contribution to the ESMValTool. To be written for v2.
5.3.1 Workflow core development team

The following workflow followed by the ESMValTool core development team takes place whenever a developer requests integration of a diagnostics set into the development branch.

To be written for v2.

5.3.2 Responsibilities of ESMValTool developers

To be written.
Part II

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

The ESMValTool supports three different installation methods:

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

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

6.1 Conda installation

A conda package will be available after the release of ESMValTool 2.

6.2 Docker installation

**Warning:** Docker section to be added

6.3 Install from source

6.3.1 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:
It is also possible to work in one of the ESMValTool private repositories, e.g.:

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

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

### 6.3.2 Prerequisites

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

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

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

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

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

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

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

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

Note that it is advisable to update conda to the latest version before installing ESMValTool, using the command

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

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

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

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

```
conda activate esmvaltool
```

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

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

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

### 6.3.3 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 .
```

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

```
julia esmvaltool/install/Julia/setup.jl
```

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

```
Rscript esmvaltool/install/R/setup.R
```

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

```
esmvaltool --help
```

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

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

```
python setup.py test --installation
```
The `config-user.yml` configuration file contains all the global level information needed by ESMValTool. The following shows the default settings from the `config-user.yml` file.

```yaml
# Diagnostics create plots? [true]/false
write_plots: true
# Diagnostics write NetCDF files? [true]/false
write_netcdf: true
# Set the console log level debug, [info], warning, error
log_level: info
# verbosity is deprecated and will be removed in the future
# verbosity: 1
# Exit on warning? true/[false]
exit_on_warning: false
# Plot file format? [ps]/pdf/png/eps/epsi
output_file_type: pdf
# Destination directory
output_dir: ./esmvaltool_output
# Auxiliary data directory (used for some additional datasets)
auxiliary_data_dir: ./auxiliary_data
# Use netCDF compression true/[false]
compress_netcdf: false
# Save intermediary cubes in the preprocessor true/[false]
save_intermediary_cubes: false
# Remove the preproc dir if all fine
remove_preproc_dir: true
# Run at most this many tasks in parallel null/[1]/2/3/4/...
# Set to null to use the number of available CPUs.
# Make sure your system has enough memory for the specified number of tasks.
max_parallel_tasks: 1
# Path to custom config-developer file, to customise project configurations.
# See config-developer.yml for an example. Set to None to use the default
config_developer_file: null
# Get profiling information for diagnostics
# Only available for Python diagnostics
```

(continues on next page)
profile_diagnostic: false

# Rootpaths to the data from different projects (lists are also possible)
rootpath:
  CMIP5: [~/cmip5_inputpath1, ~/cmip5_inputpath2]
  OBS: ~/obs_inputpath
  default: ~/default_inputpath

# Directory structure for input data: [default]/BADC/DKRZ/ETHZ/etc
# See config-developer.yml for definitions.
drs:
  CMIP5: default

Most of these settings are fairly self-explanatory, ie:

# Diagnostics create plots? [true]/false
write_plots: true
# Diagnostics write NetCDF files? [true]/false
write_netcdf: true

The write_plots setting is used to inform ESMValTool about your preference for saving figures. Similarly, the write_netcdf setting is a boolean which turns on or off the writing of netCDF files.

# Auxiliary data directory (used for some additional datasets)
auxiliary_data_dir: ./auxiliary_data

The auxiliary_data_dir setting is the path to place any required additional auxiliary data files. This method was necessary because certain Python toolkits such as cartopy will attempt to download data files at run time, typically geographic data files such as coastlines or land surface maps. This can fail if the machine does not have access to the wider internet. This location allows us to tell cartopy (and other similar tools) where to find the files if they can not be downloaded at runtime. To reiterate, this setting is not for model or observational datasets, rather it is for data files used in plotting such as coastline descriptions and so on.

Tip: You choose your config.yml file at run time, so you could have several available with different purposes. One for formalised run, one for debugging, etc...

:: _inputdata:
CHAPTER 8

Input data

8.1 Models

8.2 Observations

Observational and reanalysis products in the standard CF/CMOR format used in CMIP and required by the ESMValTool are available via the obs4mips (https://esgf-node.llnl.gov/projects/obs4mips/) and ana4mips (https://esgf.nccs.nasa.gov/projects/ana4mips/) projects, respectively. 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 using the cmorizers included in the ESMValTool. The cmorizers are dataset-specific scripts that can be run once to generate a local pool of observational datasets for usage with the ESMValTool. 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 dataset cmorized for ESMValTool v1 may not be working with v2, due to the much stronger constraints on metadata set by the Iris library.

To cmorize one or more datasets, run:

```bash
cmorize_obs -c [CONFIG_FILE] -o [DATASET_LIST]
```

The path to the raw data to be cmorized must be specified in the CONFIG_FILE as RAWOBS. Within this path, the data are expected to be organized in subdirectories corresponding to the data tier: Tier2 for freely-available datasets (other than obs4mips and ana4mips) and Tier3 for restricted datasets (i.e., dataset which requires a registration to be retrieved or provided upon request to the respective contact or PI). The cmorization follows the CMIP5 CMOR tables. The resulting output is saved in the output_dir, again following the Tier structure. The output file names follow the definition given in config-developer.yml for the OBS project: OBS_[dataset]_[type]_[version]_[mip]_[short_name]_YYYYMM/YYYYMM.nc, where type may be sat (satellite data), reanal (reanalysis data), ground (ground observations), clim (derived climatologies), campaign (aircraft campaign).

At the moment, cmorize_obs supports Python and NCL scripts.

A list of the datasets for which a cmorizers is available is provided in the following table.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Variables (MIP)</th>
<th>Tier</th>
<th>Script language</th>
</tr>
</thead>
<tbody>
<tr>
<td>AURA-TES</td>
<td>tro3 (Amon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>CDS-SATELLITE-</td>
<td>sm (Lmon), smStderr (Lmon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>SOIL-MOISTURE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDS-XCH4</td>
<td>xch4 (Amon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>CDS-XCO2</td>
<td>xco2 (Amon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>CERES-SYN1deg</td>
<td>rlds, rlscs, rlus, rluscs, rlut, rlutcs, rsds, rsds, rsds, rsuscs, rsut, rsuts (3hr) rlds, rlscs, rlus, rlut, rlutcs, rsds, rsut, rsuts (Amon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>CRU</td>
<td>tas, pr (Amon)</td>
<td>2</td>
<td>Python</td>
</tr>
<tr>
<td>ERA-Interim</td>
<td>clivi, clt, clwvi, hdfs, hur, hus, pr, prw, ps, ps, ta, tas, tauu, tauv, ts, ua, va, wap, zg (Amon) pr, ps, tas, tasmin, tasmx, zg (day), sftlf (fx), tos (Omon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-AEROSOL</td>
<td>abs550aer, od550aer, od550aerStderr, od550lt1aer, od870aer, od870aerStderr (aero)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-CLOUD</td>
<td>clivi, clt, cltStderr, clwvi (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-FIRE</td>
<td>burntArea (Lmon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-LANDCOVER</td>
<td>baresoilFrac, cropFrac, grassFrac, shrubFrac, treeFrac (Lmon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-OC</td>
<td>chl</td>
<td>2</td>
<td>Python</td>
</tr>
<tr>
<td>ESACCI-OZONE</td>
<td>toz, tozStderr, tro3prof, tro3profStderr (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-SOILMOISTURE</td>
<td>dos, dosStderr, sm, smStderr (Lmon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>ESACCI-SST</td>
<td>ts, tsStderr (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>GHCN</td>
<td>pr (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>HadCRUT3</td>
<td>tas, tasa (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>HadCRUT4</td>
<td>tas, tasa (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>HadISST</td>
<td>sic (OImon), tos (Omon), ts (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>LandFlux-EVAL</td>
<td>et, etStderr (Lmon)</td>
<td>3</td>
<td>Python</td>
</tr>
<tr>
<td>Landschuetzer2016</td>
<td>fgco2 (Omon), spco2 (Omon), dpco2 (Omon)</td>
<td>2</td>
<td>Python</td>
</tr>
<tr>
<td>MODIS</td>
<td>clivi, clt, clwvi, iwpStderr, lwpStderr (Amon), od550aer (aero)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>MTE</td>
<td>gpp, gppStderr (Lmon)</td>
<td>3</td>
<td>Python</td>
</tr>
<tr>
<td>NCEP</td>
<td>hur, hus, pr, ta, tas, ua, va, wap, zg (Amon) pr, slt, slt, va (day)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>NIWA-BS</td>
<td>toz, tozStderr (Amon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>PATMOS-x</td>
<td>clt (Amon)</td>
<td>2</td>
<td>NCL</td>
</tr>
<tr>
<td>UWisc</td>
<td>clwvi, lwpStderr (Amon)</td>
<td>3</td>
<td>NCL</td>
</tr>
<tr>
<td>WOA</td>
<td>no3, o2, po4, si (Oyr), so, thetao (Omon)</td>
<td>2</td>
<td>Python</td>
</tr>
</tbody>
</table>

:: _running:
To run ESMValTool, use the command

```
esmvaltool -c /path/to/config-user.yml examples/recipe_python.yml
```

This will run the example recipe_python.yml. The path to the recipe can either be the path to a recipe file, or a path relative to the esmvaltool/recipes directory of your installed ESMValTool. See the chapter User configuration file for an explanation of how to create your own config-user.yml file.

To get help on additional commands, please use

```
esmvaltool --help
```

### 9.1 Available diagnostics and metrics

See Recipes for a description of all available recipes.

:: _outputdata:
CHAPTER 10

Output

10.1 Preprocessed datasets
preproc/

10.2 Diagnostic output
work/

10.3 Plots
plots/

10.4 Log files
run/
Part III

ESMValTool Core
There are several configuration files in ESMValTool:

- config-user.yml
- config-developer.yml
- config-references.yml
- config-logging.yml

### 11.1 User configuration file

See Section

### 11.2 Developer configuration file

This configuration file describes the file system structure for several key projects (CMIP5, CMIP6) on several key machines (BADC, CP4CDS, DKRZ, ETHZ, SMHI, BSC).

The data directory structure of the CMIP5 project is set up differently at each site. The following code snippet is an example of several paths descriptions for the CMIP5 at various sites:

```yaml
CMIP5:
  input_dir:
    default: '/
    BADC: '[institute]/[dataset]/[exp]/[frequency]/[modeling_realm]/[mip]/[ensemble]/
        latest/[short_name]'
    CP4CDS: '[institute]/[dataset]/[exp]/[frequency]/[modeling_realm]/[mip]/
          [ensemble]/[short_name]/latest/
    DKRZ: '[institute]/[dataset]/[exp]/[frequency]/[modeling_realm]/[mip]/[ensemble]/
          [latestversion]/[short_name]/
    ETHZ: '[exp]/[mip]/[short_name]/[dataset]/[ensemble]/
```

(continues on next page)
As an example, the CMIP5 file path on BADC would be:

\[
\text{[institute]/[dataset]/[exp]/[frequency]/[modeling_realm]/[mip]/[ensemble]/latest/} \\
\rightarrow \text{[short_name]}
\]

When loading these files, ESMValTool replaces the placeholders with the true values. The resulting real path would look something like this:

MOHC/HadGEM2-CC/rcp85/mon/ocean/Omon/r1i1p1/latest/tos

### 11.3 References configuration file

The `config-references.yml` file is the full list of ESMValTool authors, references and projects. Each author, project and reference in the documentation section of a recipe needs to be in this file in the relevant section.

For instance, the recipe `recipe_ocean_example.yml` file contains the following documentation section:

```yaml
documentation
  authors:
    - demo_le
  maintainer:
    - demo_le
  references:
    - demora2018gmd
  projects:
    - ukesm
```

All four items here are named people, references and projects listed in the `config-references.yml` file.

### 11.4 Logging configuration file

**Warning:** Section to be added

:: _inputdata:
CHAPTER 12

Data finder

Documentation of the _data_finder.py module (incl. _download.py?)
Recipe

:: _preprocessor:
The ESMValTool preprocessor can be used to perform all types of climate data pre-processing needed before indices or diagnostics can be calculated. It is a base component for many other diagnostics and metrics shown on this portal. It can be applied to tailor the climate model data to the need of the user for its own calculations.

Features of the ESMValTool Climate data pre-processor are:

- Regridding
- Geographical area selection
- Aggregation of data
- Provenance tracking of the calculations
- Model statistics
- Multi-model mean
- and many more

### 14.1 Variable derivation

Documentation of _derive.py

### 14.2 Time manipulation

The _time.py module contains the following preprocessor functions:

- `extract_time`: Extract a time range from a cube.
- `extract_season`: Extract only the times that occur within a specific season.
- `extract_month`: Extract only the times that occur within a specific month.
- `time_average`: Take the weighted average over the time dimension.
• seasonal_mean: Produces a mean for each season (DJF, MAM, JJA, SON)
• annual_mean: Produces an annual or decadal mean.
• regrid_time: Aligns the time axis of each dataset to have common time points and calendars.

14.2.1 1. extract_time

This function subsets a dataset between two points in times. It removes all times in the dataset before the first time and after the last time point. The required arguments are relatively self explanatory:

• start_year
• start_month
• start_day
• end_year
• end_month
• end_day

These start and end points are set using the datasets native calendar. All six arguments should be given as integers - the named month string will not be accepted.

See also esmvalcore.preprocessor.extract_time().

14.2.2 2. extract_season

Extract only the times that occur within a specific season.

This function only has one argument: season. This is the named season to extract. ie: DJF, MAM, JJA, SON.

Note that this function does not change the time resolution. If your original data is in monthly time resolution, then this function will return three monthly datapoints per year.

If you want the seasonal average, then this function needs to be combined with the seasonal_mean function, below.

See also esmvalcore.preprocessor.extract_season().

14.2.3 3. extract_month

The function extracts the times that occur within a specific month. This function only has one argument: month. This value should be an integer between 1 and 12 as the named month string will not be accepted.

See also esmvalcore.preprocessor.extract_month().

14.2.4 4. time_average

This function takes the weighted average over the time dimension. This function requires no arguments and removes the time dimension of the cube.

See also esmvalcore.preprocessor.time_average().
14.2.5 5. seasonal_mean

This function produces a seasonal mean for each season (DJF, MAM, JJA, SON). Note that this function will not check for missing time points. For instance, if you are looking at the DJF field, but your datasets starts on January 1st, the first DJF field will only contain data from January and February.

We recommend using the extract_time to start the dataset from the following December and remove such biased initial datapoints.

See also esmvalcore.preprocessor.seasonal_mean().

14.2.6 6. annual_mean

This function produces an annual or a decadal mean. The only argument is the decadal boolean switch. When this switch is set to true, this function will output the decadal averages.

See also esmvalcore.preprocessor.annual_mean().

14.2.7 7. regrid_time

This function aligns the time points of each component dataset so that the dataset iris cubes can be subtracted. The operation makes the datasets time points common and sets common calendars; it also resets the time bounds and auxiliary coordinates to reflect the artificially shifted time points. Current implementation for monthly and daily data; the frequency is set automatically from the variable CMOR table unless a custom frequency is set manually by the user in recipe.

14.3 Area manipulation

The _area.py module contains the following preprocessor functions:

- extract_region: Extract a region from a cube based on lat/lon corners.
- zonal_means: Calculates the zonal or meridional means.
- area_statistics: Calculates the average value over a region.
- extract_named_regions: Extract a specific region from in the region coordinate.

14.3.1 1. extract_region

This function masks data outside a rectangular region requested. The boundaries of the region are provided as latitude and longitude coordinates in the arguments:

- start_longitude
- end_longitude
- start_latitude
- end_latitude

Note that this function can only be used to extract a rectangular region.

See also esmvalcore.preprocessor.extract_region().
14.3.2 2. zonal_means

The function calculates the zonal or meridional means. While this function is named `zonal_mean`, it can be used to apply several different operations in an zonal or meridional direction. This function takes two arguments:

- `coordinate`: Which direction to apply the operation: latitude or longitude
- `mean_type`: Which operation to apply: mean, std_dev, variance, median, min or max

See also `esmvalcore.preprocessor.zonal_means()`.

14.3.3 3. area_statistics

This function calculates the average value over a region - weighted by the cell areas of the region. This function takes the argument, `operator`: the name of the operation to apply. This function can be used to apply several different operations in the horizontal plane: mean, standard deviation, median, variance, minimum and maximum.

Note that this function is applied over the entire dataset. If only a specific region, depth layer or time period is required, then those regions need to be removed using other preprocessor operations in advance.

See also `esmvalcore.preprocessor.area_statistics()`.

14.3.4 4. extract_named_regions

This function extract a specific named region from the data. This function takes the following argument: `regions` which is either a string or a list of strings of named regions. Note that the dataset must have a `region` coordinate which includes a list of strings as values. This function then matches the named regions against the requested string.

See also `esmvalcore.preprocessor.extract_named_regions()`.

14.4 Volume manipulation

The `_volume.py` module contains the following preprocessor functions:

- `extract_volume`: Extract a specific depth range from a cube.
- `volume_statistics`: Calculate the volume-weighted average.
- `depth_integration`: Integrate over the depth dimension.
- `extract_transect`: Extract data along a line of constant latitude or longitude.
- `extract_trajectory`: Extract data along a specified trajectory.

14.4.1 1. extract_volume

Extract a specific range in the z-direction from a cube. This function takes two arguments, a minimum and a maximum (`z_min` and `z_max`, respectively) in the z direction.

Note that this requires the requested z-coordinate range to be the same sign as the iris cube. ie, if the cube has z-coordinate as negative, then `z_min` and `z_max` need to be negative numbers.

See also `esmvalcore.preprocessor.extract_volume()`.
14.4.2 2. volume_statistics

This function calculates the volume-weighted average across three dimensions, but maintains the time dimension. The following arguments are required:

This function takes the argument: operator, which defines the operation to apply over the volume.

No depth coordinate is required as this is determined by iris. This function works best when the fx_files provide the cell volume.

See also `esmvalcore.preprocessor.volume_statistics()`.

14.4.3 3. depth_integration

This function integrate over the depth dimension. This function does a weighted sum along the z-coordinate, and removes the z direction of the output cube. This preprocessor takes no arguments.

See also `esmvalcore.preprocessor.depth_integration()`.

14.4.4 4. extract_transect

This function extract data along a line of constant latitude or longitude. This function takes two arguments, although only one is strictly required. The two arguments are `latitude` and `longitude`. One of these arguments needs to be set to a float, and the other can then be either ignored or set to a minimum or maximum value. Ie: If we set latitude to 0 N and leave longitude blank, it would produce a cube along the equator. On the other hand, if we set latitude to 0 and then set longitude to `[40., 100.]` this will produce a transect of the equator in the indian ocean.

See also `esmvalcore.preprocessor.extract_transect()`.

14.4.5 5. extract_trajectory

This function extract data along a specified trajectory. The three arguments are: latitudes and longitudes are the coordinates of the trajectory.

If two points are provided, the `number_points` argument is used to set a the number of places to extract between the two end points.

If more than two points are provided, then extract_trajectory will produce a cube which has extrapolated the data of the cube to those points, and `number_points` is not needed.

Note that this function uses the expensive interpolate method, but it may be necceasiry for irregular grids.

See also `esmvalcore.preprocessor.extract_trajectory()`.

14.5 CMORization and dataset-specific fixes

Documentation of _reformat.py, check.py and fix.py

14.6 Vertical interpolation

Documentation of _regrid.py (part 1)
14.7 Masking

Documentation of _mask.py (part 1)

14.7.1 1. Introduction to masking

Certain metrics and diagnostics need to be computed and performed on restricted regions of the Globe; ESMValTool supports subsetting the input data on land mass, oceans and seas, ice. This is achieved by masking the model data and keeping only the values associated with grid points that correspond to e.g. land mass or oceans and seas; masking is done either by using standard mask files that have the same grid resolution as the model data (these files are usually produced at the same time with the model data and are called fx files) or, in the absence of these files, by using Natural Earth masks. Natural Earth masks, even if they are not model-specific, represent a good approximation since their grid resolution is almost always much higher than the model data, and they are constantly updated with changing geographical features.

14.7.2 2. Land-sea masking

In ESMValTool v2 land-seas-ice masking can be done in two places: in the preprocessor, to apply a mask on the data before any subsequent preprocessing step, and before running the diagnostic, or in the diagnostic phase. We present both these implementations below.

To mask out seas in the preprocessor step, simply add `mask_landsea` as a preprocessor step in the `preprocessor` of your choice section of the recipe, example:

```yaml
preprocessors:
  my_masking_preprocessor:
    mask_landsea:
      mask_out: sea
```

The tool will retrieve the corresponding `fx: stf0f` type of mask for each of the used variables and apply the mask so that only the land mass points are kept in the data after applying the mask; conversely, it will retrieve the `fx: sftlf` files when land needs to be masked out. `mask_out` accepts: land or sea as values. If the corresponding fx file is not found (some models are missing these type of files; observational data is missing them altogether), then the tool attempts to mask using Natural Earth mask files (that are vectorized rasters). Note that the resolutions for the Natural Earth masks are much higher than any usual CMIP model: 10m for land and 50m for ocean masks.

14.7.3 3. Ice masking

Note that for masking out ice the preprocessor is using a different function, this so that both land and sea or ice can be masked out without losing generality. To mask ice out one needs to add the preprocessing step much as above:

```yaml
preprocessors:
  my_masking_preprocessor:
    mask_landseaice:
      mask_out: ice
```

To keep only the ice, one needs to mask out landsea, so use that as value for `mask_out`. As in the case of `mask_landsea`, the tool will automatically retrieve the `fx: stgif` file corresponding the the used variable and extract the ice mask from it.
14.7.4 4. mask files

At the core of the land/sea/ice masking in the preprocessor are the mask files (whether it be fx type or Natural Earth type of files); these files (bar Natural Earth) can be retrieved and used in the diagnostic phase as well or solely. By specifying the $fx_files$: key in the variable in diagnostic in the recipe, and populating it with a list of desired files e.g.:

```python
variables:
    ta:
        preprocessor: my_masking_preprocessor
        fx_files: [sftlf, sftof, sftgif, areacello, areacella]
```

Such a recipe will automatically retrieve all the [sftlf, sftof, sftgif, areacello, areacella]-type fx files for each of the variables that are needed for and then, in the diagnostic phase, these mask files will be available for the developer to use them as they need to. They $fx_files$ attribute of the big variable nested dictionary that gets passed to the diagnostic is, in turn, a dictionary on its own, and members of it can be accessed in the diagnostic through a simple loop over the ‘config’ diagnostic variable items e.g.:

```python
for filename, attributes in config['input_data'].items():
    sftlf_file = attributes['fx_files']['sftlf']
    areacello_file = attributes['fx_files']['areacello']
```

14.7.5 5. Missing values masks

Missing (masked) values can be a nuisance especially when dealing with multimodel ensembles and having to compute multimodel statistics; different numbers of missing data from dataset to dataset may introduce biases and artificially assign more weight to the datasets that have less missing data. This is handled in ESMValTool via the missing values masks: two types of such masks are available: one for the multimodel case and another for the single model case.

The multimodel missing values mask (mask_fillvalues) is a preprocessor step that usually comes after all the single-model steps (regridding, area selection etc) have been performed; in a nutshell, it combines missing values masks from individual models into a multimodel missing values mask; the individual model masks are built according to common criteria: the user chooses a time window in which missing data points are counted, and if the number of missing data points relative to the number of total data points in a window is less than a chosen fractional threshold, the window is discarded i.e. all the points in the window are masked (set to missing).

```python
preprocessors:
    missing_values_preprocessor:
        mask_fillvalues:
            threshold_fraction: 0.95
            min_value: 19.0
            time_window: 10.0
```

In the example above, the fractional threshold for missing data vs. total data is set to 95% and the time window is set to 10.0 (units of the time coordinate units). Optionally, a minimum value threshold can be applied, in this case it is set to 19.0 (in units of the variable units).

A similar preprocessor step exists for the single-dataset: mask_window_threshold (with the same arguments as mask_fillvalues).

14.7.6 6. Min, max and interval masking

Thresholding on minimum and maximum accepted data values can also be performed: masks are constructed based on the results of thresholding; inside and outside interval thresholding and masking can also be performed. These functions are mask_above_threshold, mask_below_threshold, mask_inside_range, and mask_outside_range.
14.8 Horizontal regridding

Documentation of _regrid.py (part 2)

14.9 Masking of missing values

Documentation of _mask.py (part 2)

14.10 Multi-model statistics

Documentation of multimodel.py

14.11 Time-area statistics

Documentation of _area_pp.py and _volume_pp.py

14.12 Information on maximum memory required

In the most general case, we can set upper limits on the maximum memory the analysis will require:

\[Ms = (R + N) \times F_{\text{eff}} - F_{\text{eff}}\]  
when no multimodel analysis is performed;  
\[Mm = (2R + N) \times F_{\text{eff}} - 2F_{\text{eff}}\]  
when multimodel analysis is performed;

where

Ms: maximum memory for non-multimodel module  
Mm: maximum memory for multimodel module  
R: computational efficiency of module; \(R\) is typically 2-3  
N: number of datasets  
F_{\text{eff}}: average size of data per dataset where  
F_{\text{eff}} = e \times f \times F  
where \(e\) is the factor that describes how lazy the data is (\(e = 1\) for fully realized data) and \(f\) describes how much the data was shrunk by the immediately previous module eg time extraction, area selection or level extraction; note that for fix_data \(f\) relates only to the time extraction, if data is exact in time (no time selection) \(f = 1\) for fix_data

so for cases when we deal with a lot of datasets \((R + N = N)\), data is fully realized, assuming an average size of 1.5GB for 10 years of 3D netCDF data, \(N\) datasets will require

\[Ms = 1.5 \times (N - 1) \text{ GB}\]  
\[Mm = 1.5 \times (N - 2) \text{ GB}\]  
\[= \text{======}\]
Part IV

Guidelines for diagnostic developers
15.1 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.

15.2 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 pre-processing 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. A good example recipe is esmvaltool/recipes/examples/recipe_python.yml and a good example diagnostic is esmvaltool/diag_scripts/examples/diagnostic.py.

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.

Unfortunately not much documentation is available at this stage, so have a look at the other recipes and diagnostics for further inspiration.

15.3 Re-using existing code

Always make sure your code is or can be released under a license that is compatible with the Apache 2 license.

If you have existing code in a supported scripting language, you have two options for re-using it. If it is fairly mature and a large amount of code, the preferred way is to package and publish it on the official package repository for that language and add it as a dependency of esmvaltool. If it is just a few simple scripts or packaging is not possible (i.e. for NCL) you can simply copy and paste the source code into the esmvaltool/diag_scripts directory.
If you have existing code in a compiled language like C, C++, or Fortran that you want to re-use, the recommended way to proceed is to add Python bindings and publish the package on PyPI so it can be installed as a Python dependency. You can then call the functions it provides using a Python diagnostic.

### 15.4 Interfaces and provenance

When ESMValTool runs a recipe, it will first find all data and run the default preprocessor steps plus any additional preprocessing steps defined in the recipe. Next it will run the diagnostic script defined in the recipe and finally it will store provenance information. Provenance information is stored in the W3C PROV XML format and also plotted in an SVG file for human inspection. In addition to provenance information, a caption is also added to the plots.

In order to communicate with the diagnostic script, two interfaces have been defined, which are described below. 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.

#### 15.4.1 Using the interfaces from Python

Always use `esmvaltool.diag_scripts.shared.run_diagnostic()` to start your script and make use of a `esmvaltool.diag_scripts.shared.ProvenanceLogger` to log provenance. Have a look at the example Python diagnostic in `esmvaltool.diag_scripts/examples/diagnostic.py` for a complete example.

#### 15.4.2 Using the interfaces from NCL

Always call the `log_provenance` procedure after plotting from your NCL diag_script. You could find available shortcuts for statistics, domain, plottype, authors and references in the `config-references.yml` file.

```
l:og_provenance(nc-file, plot_file, caption, statistics, domain, plottype, authors, references, input-files)
```

Have a look at the example NCL diagnostic in `esmvaltool.diag_scripts/examples/diagnostic.ncl` for a complete example.

#### 15.4.3 Generic interface between backend and diagnostic

To provide the diagnostic script with the information it needs to run (e.g. location of input data, various settings), the backend creates a YAML file called `settings.yml` and provides the path to this file as the first command line argument to the diagnostic script.

The most interesting settings provided in this file are

```
run_dir: /path/to/recipe_output/run/diagnostic_name/script_name
work_dir: /path/to/recipe_output/work/diagnostic_name/script_name
plot_dir: /path/to/recipe_output/work/diagnostic_name/script_name
input_files:
  - /path/to/recipe_output/preproc/diagnostic_name/ta/metadata.yml
  - /path/to/recipe_output/preproc/diagnostic_name/pr/metadata.yml
```

Custom settings in the script section of the recipe will also be made available in this file.

There are three directories defined:

- `run_dir` use this for storing temporary files
• **work_dir** use this for storing NetCDF files containing the data used to make a plot

• **plot_dir** use this for storing plots

Finally, **input_files** is a list of YAML files, containing a description of the preprocessed data. Each entry in these YAML files is a path to a preprocessed file in NetCDF format, with a list of various attributes. An example preprocessor metadata.yml file could look like this:

```yaml
? /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_historical_r1i1p1_T2Ms_pr_2000-2002.nc
  : cmor_table: CMIP5
dataset: GFDL-ESM2G
diagnostic: diagnostic_name
der: 2002
ensemble: r1i1p1
exp: historical
filename: /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_historical_r1i1p1_T2Ms_pr_2000-2002.nc
frequency: mon
institute: [NOAA-GFDL]
long_name: Precipitation
mip: Amon
modeling_realm: [atmos]
preprocessor: preprocessor_name
project: CMIP5
recipe_dataset_index: 1
reference_dataset: MPI-ESM-LR
short_name: pr
standard_name: precipitation_flux
start_year: 2000
units: kg m$^{-2}$ s$^{-1}$
variable_group: pr

? /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_MPI-ESM-LR_Amon_historical_r1i1p1_T2Ms_pr_2000-2002.nc
  : cmor_table: CMIP5
dataset: MPI-ESM-LR
diagnostic: diagnostic_name
der: 2002
ensemble: r1i1p1
exp: historical
filename: /path/to/recipe_output/preproc/diagnostic1/pr/CMIP5_MPI-ESM-LR_Amon_historical_r1i1p1_T2Ms_pr_2000-2002.nc
frequency: mon
institute: [MPI-M]
long_name: Precipitation
mip: Amon
modeling_realm: [atmos]
preprocessor: preprocessor_name
project: CMIP5
recipe_dataset_index: 2
reference_dataset: MPI-ESM-LR
short_name: pr
standard_name: precipitation_flux
start_year: 2000
units: kg m$^{-2}$ s$^{-1}$
variable_group: pr
```

15.4. Interfaces and provenance
15.4.4 Generic interface between diagnostic and backend

After the diagnostic script has finished running, the backend will try to store provenance information. In order to link the produced files to input data, the diagnostic script needs to store a YAML file called `diagnostic_provenance.yml` in its `run_dir`.

For output file produced by the diagnostic script, there should be an entry in the `diagnostic_provenance.yml` file. The name of each entry should be the path to the output file. Each file entry should at least contain the following items:

- **ancestors** a list of input files used to create the plot
- **caption** a caption text for the plot
- **plot_file** if the diagnostic also created a plot file, e.g. in .png format.

Each file entry can also contain items from the categories defined in the file `esmvaltool/config_references.yml`. The short entries will automatically be replaced by their longer equivalent in the final provenance records. It is possible to add custom provenance information by adding custom items to entries.

An example `diagnostic_provenance.yml` file could look like this:

```
? /path/to/recipe_output/work/diagnostic_name/script_name/CMIP5_GFDL-ESM2G_Amon_historical_r1i1p1_T2Ms_pr_2000-2002_mean.nc
: ancestors:
  - /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_historical_r1i1p1_T2Ms_pr_2000-2002.nc
 authors: [ande_bo, righ_ma]
 caption: Average Precipitation between 2000 and 2002 according to GFDL-ESM2G.
 domains: [global]
 plot_file: /path/to/recipe_output/plots/diagnostic_name/script_name/CMIP5_GFDL-ESM2G_Amon_historical_r1i1p1_T2Ms_pr_2000-2002_mean.png
 plot_type: zonal
 references: [acknow_project]
 statistics: [mean]

? /path/to/recipe_output/work/diagnostic_name/script_name/CMIP5_MPI-ESM-LR_Amon_historical_r1i1p1_T2Ms_pr_2000-2002_mean.nc
: ancestors:
  - /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_MPI-ESM-LR_Amon_historical_r1i1p1_T2Ms_pr_2000-2002.nc
 authors: [ande_bo, righ_ma]
 caption: Average Precipitation between 2000 and 2002 according to MPI-ESM-LR.
 domains: [global]
 plot_file: /path/to/recipe_output/plots/diagnostic_name/script_name/CMIP5_MPI-ESM-LR_Amon_historical_r1i1p1_T2Ms_pr_2000-2002_mean.png
 plot_type: zonal
 references: [acknow_project]
 statistics: [mean]
```

You can check whether your diagnostic script successfully provided the provenance information to the backend by verifying that:

- for each output file in the `work_dir`, a file with the same name, but ending with `.provenance.xml` is created
- any NetCDF files created by your diagnostic script contain a ‘provenance’ global attribute
- any PNG plots created by your diagnostic script contain the provenance information in the ‘Image History’ attribute

Note that this is done automatically by the ESMValTool backend.
Porting a namelist (recipe) or diagnostic to ESMValTool v2.0

This guide summarizes the main steps to be taken in order to port an ESMValTool namelist (now called **recipe**) and the corresponding diagnostic(s) from v1.0 to v2.0, hereafter also referred as the “old” and the “new version”, respectively. The new ESMValTool version is being developed in the public git branch **version2_development**. 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).

### 16.1 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.

### 16.2 Create your own branch

Create your own branch from **version2_development** for each namelist (recipe) to be ported:

```bash
  git checkout version2_development
  git checkout -b version2_<recipe>
```

**version2_development** contains only v2.0 under the ./esmvaltool/ directory.

### 16.3 Convert xml to yml

In ESMValTool v2.0, the namelist (now recipe) is written in yaml format (Yet Another Markup Language format). It may be useful to activate the yaml syntax highlighting for the editor in use. This improves the readability of the recipe file and facilitates the editing, especially concerning the indentations which are essential in this format (like in python). Instructions can be easily found online, for example for [emacs](#) and [vim](#).
A xml2yml converter is available in `esmvaltool/utils/xml2yml/`, please refer to the corresponding README file for detailed instructions on how to use it.

Once the recipe is converted, a first attempt to run it can be done, possibly starting with a few datasets and one diagnostics and proceed gradually. The recipe file `.esmvaltool/recipes/recipe_perfmetrics_CMIP5.yml` can be used as an example, as it covers most of the common cases.

Do not forget to also rewrite the recipe header in a documentation section using the yaml syntax and, if possible, to add themes and realms item to each diagnostic section. All keys and tags used for this part must be defined in `.esmvaltool/config-references.yml`. See `.esmvaltool/recipes/recipe_perfmetrics_CMIP5.yml` for an example.

### 16.4 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 development 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.

### 16.5 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.

<table>
<thead>
<tr>
<th>Name in v1.0</th>
<th>Name in v2.0</th>
<th>Affected code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getenv(&quot;ESMValTool_wrk_dir&quot;)</code></td>
<td><code>config_user_info@work_dir</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>getenv(ESMValTool_att)</code></td>
<td><code>diag_script_info@att</code> or <code>config_user_info@att</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>xml</code></td>
<td><code>yml</code></td>
<td>all scripts</td>
</tr>
<tr>
<td><code>var_attr_ref(0)</code></td>
<td><code>variable_info@reference_dataset</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>var_attr_ref(1)</code></td>
<td><code>variable_info@alternative_dataset</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>models</code></td>
<td><code>input_file_info</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>models@name</code></td>
<td><code>input_file_info@dataset</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>verbosity</code></td>
<td><code>config_user_info@log_level</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>isfilepresent_esmval</code></td>
<td><code>fileexists</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>messaging.ncl</code></td>
<td><code>logging.ncl</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>info_output(arg1, arg2, arg3)</code></td>
<td><code>log_info(arg1) if arg3=1</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>info_output(arg1, arg2, arg3)</code></td>
<td><code>log_debug(arg1) if arg3&gt;1</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>verbosity = config_user_info@verbosity</code></td>
<td>remove this statement</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>enter_msg(arg1, arg2, arg3)</code></td>
<td><code>enter_msg(arg1, arg2)</code></td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td><code>leave_msg(arg1, arg2, arg3)</code></td>
<td><code>leave_msg(arg1, arg2)</code></td>
<td>all .ncl scripts</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name in v1.0</th>
<th>Name in v2.0</th>
<th>Affected code</th>
</tr>
</thead>
<tbody>
<tr>
<td>noop()</td>
<td>appropriate if-else statement</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>nooperation()</td>
<td>appropriate if-else statement</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>fullpaths</td>
<td>input_file_info@filename</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>get_output_dir(arg1, arg2)</td>
<td>config_user_info@plot_dir</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>get_work_dir</td>
<td>config_user_info@work_dir</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>instlist(arg1, arg2)</td>
<td>any(arg1.eq.arg2)</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>load interface_scripts/*.ncl</td>
<td>load $diag_scripts/.../interface_scripts/interface.ncl</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>&lt;varname&gt;_info.tmp</td>
<td>&lt;varname&gt;_info.ncl in preproc dir</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>ncl.interface</td>
<td>settings.ncl in run_dir and interface_scripts/interface.ncl</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>load diag_scripts/lib/ncl/</td>
<td>load $diag_scripts/shared/plot.ncl</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>load plot_scripts/ncl/</td>
<td>load $diag_scripts/shared/plot/</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>load diag_scripts/lib/ncl/rgb/</td>
<td>load $diag_scripts/shared/plot/rgb/</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>load diag_scripts/lib/ncl/styles/</td>
<td>load $diag_scripts/shared/plot/styles</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>load diag_scripts/lib/ncl/misc_function.ncl</td>
<td>load $diag_scripts/shared/plot/misc_function.ncl</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>LW_CRE, SW_CRE</td>
<td>lwcre, swcre</td>
<td>some yml recipes</td>
</tr>
<tr>
<td>check_min_max_models</td>
<td>check_min_max_datasets</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>get_ref_model_idx</td>
<td>get_ref_dataset_idx</td>
<td>all .ncl scripts</td>
</tr>
<tr>
<td>get_model_minus_ref</td>
<td>get_dataset_minus_ref</td>
<td>all .ncl scripts</td>
</tr>
</tbody>
</table>

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.

16.6 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 preprocessor 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 preproc): 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].

16.7 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.

16.8 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 meth-
ods, etc.). The files have to be saved in the work directory (defined in \texttt{cfg['work\_dir']}' and \texttt{config\_user\_info@work\_dir}, for the python and NCL diagnostics, respectively).

### 16.9 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: 	exttt{cdo} and 	exttt{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., \texttt{diff.nc}) with the difference between two given files:

```
ncdiff old_output.nc new_output.nc diff.nc
```

This file can be opened with \texttt{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 \texttt{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.

### 16.10 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 (\texttt{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;
- \texttt{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 \texttt{print} and \texttt{printVarSummary} statements must be avoided and replaced by the \texttt{log\_info} and \texttt{log\_debug} functions;
- for error and warning statements, the \texttt{error\_msg} function can be used, which automatically include an exit statement.
16.11 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.

16.12 Open a pull request

Create a pull request on github to merge your branch back to `version2_development`, provide a short description of what has been done and nominate one or more reviewers.

:: _CMORobs:
Contributing a CMORizing script for an observational dataset


18.1 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 (Section Main contacts) 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 benefitting from the possibilities of collaborating with other ESMValTool developers and having a backup of their work. FEATURE BRANCHES created in the PRIVATE repository are only visible to the ESMValTool development team but not to the public. The concept of a PRIVATE repository has proven to be very useful to efficiently share code during the development across institutions and projects in a common repository without having the contributions immediately accessible to the public.

Both, the PUBLIC and the PRIVATE repository, contain the following kinds of branches:

- *MASTER BRANCH* (official releases),
- *DEVELOPMENT BRANCH* (includes approved new contributions but version is not yet fully tested),
• FEATURE BRANCH (development branches for new features and diagnostics created by developers, the naming convention for FEATURE BRANCHES is <Project>_<myfeature>).

18.2 Access rights

• Write access to the MASTER and DEVELOPMENT BRANCH in both, the PUBLIC and the PRIVATE GitHub repositories, is restricted to the ESMValTool core development team.

• FEATURE BRANCHES in both the PUBLIC and the PRIVATE repository can be created by all members of the ESMValTool development team (i.e. members in the GitHub organization “ESMValGroup”). If needed, branches can be individually write-protected within each repository so that other developers cannot accidently push changes to these branches.

The MASTER BRANCH of the PRIVATE repository will be regularly synchronized with the MASTER BRANCH of the PUBLIC repository (currently by hand). This ensures that they are identical at all times (see schematic in Schematic diagram of the ESMValTool GitHub repositories.). 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.

![Schematic diagram of the ESMValTool GitHub repositories.](image)

**Fig. 1: Schematic diagram of the ESMValTool GitHub repositories.**

18.3 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](https://www.git-scm.com/docs/gittutorial) are recommended.
18.3.1 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/.

18.3.2 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>`.

![Branch management in GitHub](image)

- 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:
This will clone the ESMValTool repository at GitHub to a local folder. You can now query the status of your local working copy with:

```bash
git status
```

You will see that you are on a branch called master and your local working copy is up to date with the remote repository. With

```bash
git branch --all
```

you can list all available remote and local branches. Now switch to your feature branch by:

```bash
git checkout <NAME_OF_YOUR_FEATURE_BRANCH>
```

You can now start coding. To check your current developments you can use the command

```bash
git status
```

You can add new files and folders that you want to have tracked by Git using:

```bash
git add <NEW_FILE|FOLDER>
```

Commit your tracked changes to your local working copy via:

```bash
git commit -m "YOUR COMMIT MESSAGE"
```

You can inspect your changes with (use man git-log for all options):

```bash
git log
```

To share your work and to have an online backup, push your local development to your FEATURE BRANCH on GitHub:

```bash
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/

### 18.3.3 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:** Before creating a pull request, please make sure all requirements listed in Sections writing and NumPy’s Documentation are fully met (see also checklist in tab_checklist).
18.3.4 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.

18.4 General do-s and don’t-s

18.4.1 Do-s

- Create a FEATURE BRANCH and use exclusively this branch for developing the ESMValTool. The naming convention for FEATURE BRANCHES is <Project>_<myfeature>.
- Comment your code as much as possible and in English.
- Use short but self-explanatory variable names (e.g., model_input and reference_input instead of xm and xr).
- Consider a modular/functional programming style. This often makes code easier to read and deletes intermediate variables immediately. If possible, separate diagnostic calculations from plotting routines.
- Consider reusing or extending existing code. General-purpose code can be found in esmvaltool/diag_scripts/shared/.
- Comment all switches and parameters including a list of all possible settings/options in the header section of your code (see also Section std_diag).
- Use templates for recipes (Section std_recipe) and diagnostics (Section std_diag) 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.

18.4.2 Don’t-s

- Do not use other programming languages than the ones currently supported (NCL, Python, R). Contact the Core Development Team (Section Main contacts) 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.
Part V

Recipes
Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power

19.1 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.

19.2 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.

19.3 User settings

User setting files are stored in recipes/
1. *recipe_capacity_factor.yml*

*Required settings for script*

- `power_curves`: (should not be changed)

### 19.4 Variables

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

### 19.5 Observations and reformat scripts

*None*

### 19.6 References


### 19.7 Example plots
19.7. Example plots
Chapter 19. Capacity factor of wind power: Ratio of average estimated power to theoretical maximum power
20.1 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.

20.2 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
20.3 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: avg = sqrt(mean(var*var)), false: avg = mean(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)
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

4. 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 re-quired to mask grid cell as valid data

*Required settings (variables)*

- reference_dataset: name of reference data set

*Optional settings (variables)*

20.3. User settings in recipe
• long_name: description of variable
• units: variable units

Color tables
• variables “pr”, “pr-mmday”: diag_scripts/shared/plot/rgb/ipcc-precip-delta.rgb

5. Script clouds_taylor.ncl

Required settings (scripts)

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 re-quired to mask grid cell as valid data

Required settings (variables)

• reference_dataset: name of reference data set

Optional settings (variables)

none

20.4 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)
20.5 Observations and reformat scripts

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

- CERES-EBAF (obs4mips) - CERES TOA radiation fluxes (used for calculation of cloud forcing)
- GPCP-SG (obs4mips) - Global Precipitation Climatology Project total precipitation
- MODIS (obs4mips) - MODIS total cloud fraction

Reformat script: reformat_scripts/obs/reformat_obs_UWisc.ncl

20.6 References


20.7 Example plots
Fig. 1: 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. 2: Taylor diagram showing the 20-yr annual average performance of CMIP5 models for total cloud fraction as compared to MODIS satellite observations.
Fig. 3: 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. 4: 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.
21.1 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 monthly 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 is 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.

21.2 Available recipes and diagnostics

Recipes are stored in recipes/

• recipe_extreme_index.yml

Diagnostics are stored in diag_scripts/magic_bsc/

• extreme_index.r
21.3 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)

21.4 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)

21.5 Observations and reformat scripts

None

21.6 References


21.7 Example plots

Index for t90p 2020-2040 (rcp85 IPSL-CM5A-LR)
22.1 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 indices are:

- Nino 3
- Nino 3.4
- Nino 4
- North Atlantic Oscillation (NAO)
- Southern Oscillation Index (SOI)

22.2 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

22.3 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.

### 22.4 Variables

- psl, tasmax, tasmin, pr or sfcWind (atmos, monthly, longitude, latitude, time)
- tos (ocean, monthly, longitude, latitude, time)

### 22.5 Observations and reformat scripts

*None*

### 22.6 References


### 22.7 Example plots
Region NAO and Variable psl

Index

-0.5 0.0 0.5 1.0 1.5


Time (years)
Consecutive dry days

23.1 Overview

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

23.2 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

23.3 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.
23.4 Variables

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

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

24.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_williams09climdyn_CREM.yml

Diagnostics are stored in diag_scripts/crem/

- ww09_esmvaltool.py
24.3 User settings

None.

24.4 Variables

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

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

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

24.5 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.

24.6 References


24.7 Example plots
Fig. 1: Cloud Regime Error Metrics (CREMpd) from William and Webb (2009) applied to those CMIP5 AMIP simulations with the required data in the archive. A perfect score with respect to ISCCP is zero; the dashed red line is an indication of observational uncertainty.
25.1 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].

25.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_cvdp.yml

Diagnostics are stored in diag_scripts/cvdp/

- cvdp_wrapper.py

25.3 User settings in recipe

Currently, the recipe must be used with a single dataset entry.

25.4 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)

25.5 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.

25.6 References

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

25.7 Example plots
NAM PR Regressions (Annual)
CHAPTER 26

Diurnal temperature variation indicator: Difference between Tmax and Tmin for a specific day

26.1 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) and energy consumption (Deandreis et al., N.D.).

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.

26.2 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.
26.3 User settings

User setting files are stored in recipes/

1. recipe_diurnal_temperature_index.yml

Required settings for script

• None

26.4 Variables

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

26.5 Observations and reformat scripts

None

26.6 References


26.7 Example plots

96 Chapter 26. Diurnal temperature variation indicator: Difference between Tmax and Tmin for a specific day
Number of days exceeding the DTR in 5 degrees during the period 2020 - 2040
Chapter 26. Diurnal temperature variation indicator: Difference between Tmax and Tmin for a specific day
EnsClus - Ensemble Clustering - a cluster analysis tool for climate model simulations

27.1 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).
27.2 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

27.3 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

27.4 Variables

- chosen by user (e.g., precipitation as in the example)

27.5 Observations and reformat scripts

None.
27.6 References

- Straus, D. M., S. Corti, and F. Molteni: Circulation regimes: Chaotic variability vs. SST forced predictability. 

27.7 Example plots

Clustering based on historical JJA precipitation rate (mm/day), 75th percentile, CMIP5 models, 3 clusters, 80% variance explained by PCs.
Extreme Events Indices - Computation of ETCCDI extreme indices and plotting

28.1 Overview

This diagnostic uses the standard climdex.pci.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.

28.2 Available recipes and diagnostics

Recipes are stored in recipes/
- recipe_extreme_events.yml

Diagnostics are stored in diag_scripts/extreme_events/
- ExtremeEvents.r

and subroutines
- common_climdex_preprocessing_for_plots.r
- make_Glecker_plot2.r
- make_timeseries_plot.r
- cfg_climdex.r
- cfg_extreme.r
28.3 User settings

Required settings for script

- **reference_datasets**: list containing the reference datasets to compare with
- **timeseries_idx**: list of indices to compute for timeseries plot. The syntax is “XXXETCCDI_TT”, where “TT” can be either “yr” or “mon” (yearly or monthly indices are computed) and “XXX” can be one of the following: “altcdd”, “altcsdi”, “altcwd”, “altwsdi”, “cdd”, “csdi”, “cwd”, “dtr”, “fd”, “gsl”, “id”, “prcptot”, “r10mm”, “r1mm”, “r20mm”, “r95p”, “r99p”, “rx1day”, “rx5day”, “sdii”, “su”, “tn10p”, “tn90p”, “tn”, “tx”, “txx”, “tx10p”, “tx90p”, “txn”, “tsx”, “wsdi”. The option “mon” for “TT” can only be used in combination with one of: “txx”, “txn”, “tnn”, “tn10p”, “tx10p”, “tn90p”, “tx90p”, “dtr”, “rx1day”, “rx5day”.
- **gleckler_idx**: list of indices to compute for Gleckler plot. Same syntax as above. The diagnostic computes all unique indices specified in either `gleckler_idx` or `timeseries_idx`. If at least one “mon” index is selected, the indices are computed but no plots are produced.
- **base_range**: a list of two years to specify the range to be used as “base range” for climdex (the period in which for example reference percentiles are computed)

Optional settings for script

- **regrid_dataset**: name of dataset to be used as common target for regridding. If missing the first reference dataset is used
- **mip_name**: string containing the name of the model ensemble, used for titles and labels in the plots (default: “CMIP”)
- **analysis_range**: a list of two years to specify the range to be used for the analysis in the plots. The input data will need to cover both `analysis_range` and `base_range`. If missing the full period covered by the input datasets will be used.
- **ts_plt**: (logical) if to produce the timeseries or not (default: true)
- **glc_plt**: (logical) if to produce the Gleckler or not (default: true)
- **climdex_parallel**: number of parallel threads to be used for climdex calculation (default: 4). Also the logical `false` can be passed to switch off parallel computation.
- **normalize**: (logical) if to detrend and normalize with the standard deviation for the datasets for use in the timeseries plot. When this option is used the data for the following indices are detrended and normalized in the timeseries plots: “altcdd”, “altcsdi”, “altcwd”, “altwsdi”, “cdd”, “csdi”, “cwd”, “dtr”, “fd”, “gsl”, “id”, “prcptot”, “r10mm”, “r1mm”, “r20mm”, “r95p”, “r99p”, “rx1day”, “rx5day”, “sdii”, “su”, “tn10p”, “tn90p”, “tx”, “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`: font size (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 lineweights (default: [2, 2, 2, 2])

• Gleckler plot:
  • gl_png_res: height for png figures (default: 480). The width of the figure is computed automatically.
  • gl_png_units: units for figure size (default: “px”)
  • gl_png_pointsize: fontsize (default: 12)
  • gl_png_bg: background color (default: “white”)
  • gl_mar_par: page margins vector (default: [10, 4, 3, 14])
  • gl_rmsespacer: spacing of RMSE column (default: 0.01)
  • gl_scaling_factor: scaling factor for colorscale height (default: 0.9)
  • gl_text_scaling_factor: scaling factor for text size (default: 1.0)
  • gl_xscale_spacer_rmse: horizontal position of colored 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)

28.4 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)

28.5 Observations and reformat scripts

None.

28.6 References


28.7 Example plots

Portrait plot of relative error metrics for the CMIP5 temperature and precipitation extreme indices. Reproduces Fig. 9.37 of the IPCC AR5 report, Chapter 9.

Timeseries of Consecutive Dry Days index for CMIP5 models.
Description will be ported from v1
30.1 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 exceed 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.

30.2 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.

30.3 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

30.4 Variables

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

30.5 Observations and reformat scripts

None

30.6 References


30.7 Example plots
Days summer tasmax 2020-2040 > the 90th quantile for 1961-1990 (rcp85)
31.1 Overview

The Earth’s hydrological cycle is of key importance both for the climate system and society. For example, the intensity and distribution of precipitation determines the availability or scarcity of fresh water in a certain region, and it is also related to the severity of hazardous events such as flooding or droughts. The simple investigation of average precipitation quantities can clearly hide some of the most relevant aspects of the hydrological cycle and its extremes (e.g., Giorgi et al., 2014). More in general, temperature and precipitation extremes have been the focus of recent climate studies attempting to capture the most relevant component of climate variability and impact on society in a changing climate (e.g., Alexander, 2016). A particular effort has been dedicated to developing and standardising indices that can be adopted for investigation studies with observations and climate models. This tool was developed to calculate a number of hydroclimatic and climate extremes indices and allow 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), 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, also 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 \texttt{lm} function and significance testing performed with a Student T test on non-null coefficients hypothesis. Trend coefficients are stored together with their statistics which include standard error, t value and Pr(>|t|). The tool can then produce a variety of types of plots including global and regional maps, maps of comparison between models and a reference dataset, timeseries with their spread, trend lines and summary plots of trend coefficients.

The hydroclimatic indices calculated by the diagnostic and included in the output are defined as follows:

- \text{PRY} = \text{mean annual precipitation}
- \text{INT} = \text{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. HY-INT = normalized(INT) \times \text{normalized}(DSL).

### 31.2 Available recipes and diagnostics

Recipes are stored in recipes/

- `recipe_hyint.yml` (evaluating the 6 hydroclimatic indices)

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`

### 31.3 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 (ordersensitive): “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

**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

31.4 Variables
   • pr (atmos, daily mean, longitude latitude time)

31.5 Observations and reformat scripts
   None.
31.6 References

- Giorgi et al., 2011, J. Climate 24, 5309-5324, doi:10.1175/2011JCLI3979.1

31.7 Example plots

Mean hydroclimatic intensity (figure type 1) for the EC-EARTH model historical + rcp8.5 projection over 1976-2099.

Timeseries for multiple indices and regions (figure type 12) for the ACCESS1-0 model historical + RCP8.5 projection over 1976-2099.

Multi-model trend coefficients over selected indices (figure type 14) for rcp85 2006-2099 future projection normalized to the 1976-2005 historical period.
31.7. Example plots
32.1 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.

32.2 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.

32.3 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.

### 32.4 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)

### 32.5 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.

<table>
<thead>
<tr>
<th>PFT</th>
<th>ESA-CCI Landcover Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>baresoilFrac</td>
<td>Bare_Soil</td>
</tr>
<tr>
<td>cropFrac</td>
<td>Managed_Grass</td>
</tr>
<tr>
<td>grassFrac</td>
<td>Natural_Grass</td>
</tr>
<tr>
<td>shrubFrac</td>
<td>Shrub_Broadleaf_Deciduous + Shrub_Broadleaf_Evergreen + Shrub_Needleleaf_Evergreen</td>
</tr>
<tr>
<td>treeFrac</td>
<td>Tree_Broadleaf_Deciduous + Tree_Broadleaf_Evergreen + Tree_Needleleaf_Deciduous + Tree_Needleleaf_Evergreen</td>
</tr>
</tbody>
</table>

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 coarsest 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.

### 32.6 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
32.7 Example plots

![Accumulated tree covered area](image)

Fig. 1: Accumulated tree covered area for different regions and experiments.
Fig. 2: Average grass covered fraction for different regions and experiments
Fig. 3: Biases in five major land cover fractions for different regions and one experiment.
CHAPTER 33

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

33.1 Overview

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

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

To this end, the MiLd-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:

33.1.1 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.

33.1.2 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.

33.1.3 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.

33.1.4 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
are saved. Only 4 regimes and DJF supported so far.

33.2 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.
33.3 User settings

1. miles_block.R
   Required settings for variables
   • reference_dataset: reference dataset for comparison
   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
   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
   Required or optional settings for script
   • None (the two parameters seasons and nclusters in the recipe should not be changed)

33.4 Variables

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

33.5 Observations and reformat scripts

• ERA-INTERIM

33.6 References


33.7 Example plots

Fig. 1: Blocking events frequency for EC-Earth model 1980-1989, compared to ERA-Interim.

Fig. 2: Teleconnection indices as Z500 empirical orthogonal functions for the North Atlantic (the figure shows EOF1)
34.1 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.

34.2 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

34.3 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).

34.4 Variables

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

34.5 Observations and reformat scripts

None

34.6 References


34.7 Example plots
34.7. Example plots

Cluster 1 (freq = 31.8%)

Cluster 2 (freq = 35.6%)

Cluster 3 (freq = 32.7%)

Temperature range: -15 to 15
35.1 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.

35.2 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.

35.3 User settings

User setting files are stored in recipes/
1. `recipe_multimodel_products.yml`

   **Required settings for script**
   - `moninf`: integer specifying the first month of the seasonal mean period to be computed
   - `monsup`: integer specifying the last month of the seasonal mean period to be computed, if it’s null the anomaly of month indicated in moninf will be computed
   - `agreement_threshold`: integer between 0 and 100 indicating the threshold in percent for the minimum agreement between models on the sign of the multi-model mean anomaly for the stipling to be plotted
   - `running_mean`: integer 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).

### 35.4 Variables

- any Amon variable (atmos, monthly mean, longitude latitude time)

### 35.5 Observations and reformat scripts

*None*

### 35.6 References


### 35.7 Example plots
JUN tas anomaly (2006-2099) - (1961-1990)
36.1 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 cannot 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. i.e: 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.

36.2 Available recipes

- recipe_ocean_amoc.yml
- recipe_ocean_example.yml
- recipe_ocean_scalar_fields.yml
- recipe_ocean_bgc.yml
- recipe_ocean_quadmap.yml
36.2.1 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:
The recipe_ocean_example.yml is an example recipe which shows several examples of how to manipulate marine model data using the ocean diagnostics tools.

While several of the diagnostics here have specific uses in evaluating models, it is meant to be a catch-all recipe demonstrating many different ways to evaluate models.

All example calculations are performed using the ocean temperature in a three dimensional field (thetao), or at the surface (tos). This recipe demonstrates the use of a range of preprocessors in a marine context, and also shows many of the standard model-only diagnostics (no observational component is included.)

This recipe includes examples of how to manipulate both 2D and 3D fields to produce:

- **Time series:**
  - Global surface area weighted mean time series
  - Volume weighted average time series within a specific depth range
  - Area weighted average time series at a specific depth
  - Area weighted average time series at a specific depth in a specific region.
  - Global volume weighted average time series
  - Regional volume weighted average time series

- **Maps:**
  - Global surface map (from 2D ad 3D initial fields)
Global surface map using re-gridding to a regular grid
Global map using re-gridding to a regular grid at a specific depth level
Regional map using re-gridding to a regular grid at a specific depth level

- Transects:
  - Produce various transect figure showing a re-gridded transect plot, and multi model comparisons
- Profile:
  - Produce a Global area-weighted depth profile figure
  - Produce a regional area-weighted depth profile figure

All the these fields can be expanded using a 36.2.3 recipe ocean bgc.yml

The recipe ocean bgc.yml is an example recipe which shows a several simple examples of how to manipulate marine biogeochemical model data.

This recipe includes the following fields:

- Global total volume-weighted average time series:
  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *
  - chlorophyll, iron, total alkalinity (no observations)
- Surface area-weighted average time series:
  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *
  - fgco2 (global total), integrated primary production, chlorophyll, iron, total alkalinity (no observations)
- Scalar fields time series:
  - mfo (including stuff like drake passage)
- Profiles:
  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *
  - chlorophyll, iron, total alkalinity (no observations)
- Maps + contours:
  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *
  - chlorophyll, iron, total alkalinity (no observations)
- Transects + contours:
  - temperature, salinity, nitrate, oxygen, silicate (vs WOA data) *
  - chlorophyll, iron, no observations)

* Note that Phosphate is also available as a WOA diagnostic, but I haven’t included it as HadGEM2-ES doesn’t include a phosphate field.

This recipe uses the World Ocean Atlas data, which can be downloaded from: https://www.nodc.noaa.gov/OC5/woa13/woa13data.html (last access 10/25/2018)

Instructions: Select the “All fields data links (1° grid)” netCDF file, which contain all fields.
36.2.4 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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>model1</td>
<td>model 1 minus model2</td>
</tr>
<tr>
<td>model2 minus obs</td>
<td>model1 minus obs</td>
</tr>
</tbody>
</table>

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:

![Sea Surface Temperature](image)

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.

36.2.5 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 built to be generic plotting tools which work with any field (i.e. `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.

### 36.3 Available diagnostics

Diagnostics are stored in the `diag_scripts` directory: `ocean`.  

---

**ESMValTool User’s and Developer’s Guide, Release 2.0a2**

---
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

### 36.3.1 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 `time_average` 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:
  time_average:
```

For a regional 2D field:

```
prep_map_2:
  extract_region:
    start_longitude: -80.
    end_longitude: 30.
    start_latitude: -80.
```

(continues on next page)
For a Global 3D field at the surface and 10m depth:

```yaml
prep_map_3:
  custom_order: true
  extract_levels:
    levels: [0., 10.]
    scheme: linear_horizontal_extrapolate_vertical
  time_average:
```

For a multi-model comparison mean of 2D global fields including contour thresholds.

```yaml
prep_map_4:
  custom_order: true
  time_average:
  regrid:
    target_grid: 1x1
    scheme: linear
```

And this also requires the threshold key in the diagnostic:

```yaml
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]
```

### 36.3.2 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
* model2 minus obs
* model1 minus model2
* 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:

```yaml
prep_quad_map:
  time_average:
```

and an example of an appropriate diagnostic section of the recipe would be:

---

36.3. Available diagnostics
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.

### 36.3.3 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:

<table>
<thead>
<tr>
<th>Model</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model minus Observations</td>
<td>Model over Observations</td>
</tr>
</tbody>
</table>

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:

```python
diag_map_1:
  variables:
    tos: # Temperature ocean surface
    preprocessor: prep_quad_map
    field: TO2Ms
    mip: Omon
  additional_datasets:
    # Filenname: 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,}
```
36.3.4 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:

```python
preprocessors:
  prep_profile:
    extract_volume:
      long1: 0.
      long2: 20.
      lat1: -30.
      lat2: 30.
      z_min: 0.
      z_max: 3000.
    area_statistics:
      operator: mean
```

36.3.5 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:

```python
area_statistics:
  operator: mean
```

For a global volume-weighted average 3D field:

```python
volume_statistics:
  operator: mean
```

For a global area-weighted surface of a 3D field:

```python
extract_levels:
  levels: [0., ]
  scheme: linear_horizontal_extrapolate_vertical
area_statistics:
  operator: mean
```
An example of the multi-model time series plots can be seen here:

![Atlantic Meridional Overturning Circulation](image)

### 36.3.6 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:

```yaml
- time_average:
- extract_slice:
  latitude: [-50., 50.]
  longitude: 332.
```
Here is an example of the transect figure:

And here is an example of the multi-model transect contour figure:
36.3.7 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 this 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:

```yaml
 timeseries_NHW_ice_extent: # North Hemisphere Winter ice_extent
custom_order: true
extract_time: &time_anchor # declare time here.
  start_year: 1960
  start_month: 12
  start_day: 1
  end_year: 2005
  end_month: 9
  end_day: 31
extract_season:
  season: DJF
extract_region:
```

(continues on next page)
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 \textit{extract\_time} preprocessor.

This preprocessor group produces a 2D field with a time component, allowing the diagnostic to investigate the time development of the sea ice extend.

The diagnostic section of the recipe should look like this:

```yaml
diag_ice_NHW:
  description: North Hemisphere Winter Sea Ice diagnostics
  variables:
    sic: # surface ice cover
      preprocessor: timeseries_NHW\_ice\_extent
      field: TO2M
      mip: OImon
  scripts:
    Global\_seaice\_timeseries:
      script: ocean/diagnostic\_seaice.py
      threshold: 15.
```

Note the the threshold here is 15\%, which is the standard cut off for the ice extent.

The sea ice diagnostic script produces three kinds of plots, using the methods:

- \textit{make\_map\_extent\_plots}: extent maps plots of individual models using a Polar Stereographic project.
- \textit{make\_map\_plots}: maps plots of individual models using a Polar Stereographic project.
- \textit{make\_ts\_plots}: time series plots of individual models

There are no multi model comparisons included here (yet).

### 36.3.8 \texttt{diagnostic\_tools.py}

The \texttt{diagnostic\_tools.py} is a module that contains several python tools used by the ocean diagnostics tools. These tools are:

- \texttt{folder}: produces a directory at the path provided and returns a string.
- \texttt{get\_input\_files}: loads a dictionary from the input files in the metadata.yml.
- \texttt{bgc\_units}: converts to sensible units where appropriate (ie Celsius, mmol/m3)
- \texttt{timecoord\_to\_float}: Converts time series to decimal time ie: Midnight on January 1st 1970 is 1970.0
- \texttt{add\_legend\_outside\_right}: a plotting tool, which adds a legend outside the axes.
- \texttt{get\_image\_format}: loads the image format, as defined in the global user config.yml.
- \texttt{get\_image\_path}: creates a path for an image output.
- \texttt{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 \texttt{diagnostic\_tools.py} module.
36.4 A note on the auxiliary data directory

Some of these diagnostic scripts may not function on machines with no access to the internet, as cartopy may try to download the shape files. The solution to this issue is the put the relevant cartopy shapefiles in a directory which is visible to esmvaltool, then link that path to ESMValTool via the `auxiliary_data_dir` variable in your config-user.yml file.

The cartopy masking files can be downloaded from: https://www.naturalearthdata.com/downloads/

In these recipes, cartopy uses the 1:10, physical coastlines and land files:

<table>
<thead>
<tr>
<th>Shapefile Type</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>coastline.dbf</td>
<td>110m_coastline.dbf</td>
</tr>
<tr>
<td>coastline.shp</td>
<td>110m_coastline.shp</td>
</tr>
<tr>
<td>coastline.shx</td>
<td>110m_coastline.shx</td>
</tr>
<tr>
<td>land.dbf</td>
<td>110m_land.dbf</td>
</tr>
<tr>
<td>land.shp</td>
<td>110m_land.shp</td>
</tr>
<tr>
<td>land.shx</td>
<td>110m_land.shx</td>
</tr>
</tbody>
</table>

36.5 Associated Observational datasets

The following observations datasets are used by these recipes:

36.5.1 World Ocean ATLAS

These data can be downloaded from: https://www.nodc.noaa.gov/OC5/woa13/woa13data.html (last access 10/25/2018) Select the “All fields data links (1° grid)” netCDF file, which contain all fields.

The following WOA datasets are used by the ocean diagnostics:

- Temperature
- Salinity
- Nitrate
- Phosphate
- Silicate
- Dissolved Oxygen

These files need to be reformatted using the `cmorize_obs_py` script with output name WOA.

36.5.2 Landschutzer 2014

These data can be downloaded from: ftp://ftp.nodc.noaa.gov/nodc/archive/arc0105/0160558/1.1/data/0-data/spco2_1998-2011_ETH_SOM-FFN_CDIAC_G05.nc (last access 02/28/2019)

The following variables are used by the ocean diagnostics:

- fgco2, Surface Downward Flux of Total CO2
- spco2, Surface Aqueous Partial Pressure of CO2
- dpco2, Delta CO2 Partial Pressure

The file needs to be reformatted using the `cmorize_obs_py` script with output name Landschutzer2014.
37.1 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).

37.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_perfmetrics_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.
37.3 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, nhex, shex, nthrop, 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.

2. Script collect.ncl

   Required settings (scripts)
   - metric: selected metric (RMSD, BIAS or taylor)
• label_bounds: for RMSD and BIAS metrics, min and max of the labelbar
• label_scale: for RMSD and BIAS metrics, bin width of the labelbar
• colormap: for RMSD and BIAS metrics, color table of the labelbar

Optional settings (scripts)
• label_lo: adds lower 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

37.4 Variables

• 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)

37.5 Observations and reformat scripts

Note: (1) obs4mips data can be used directly without any preprocessing; (2) see headers of cmorization scripts (in esmvaltool/utils/cmorizers/obs) for non-obs4mips data for download instructions.

• AIRS (hus - obs4mips)
• CERES-EBAF (rlut, rlutcs, rsut, rsutcs - obs4mips)
• ERA-Interim (tas, ta, ua, va, zg, hus - esmvaltool/utils/cmorizers/obs/cmorize_obsERA-Interim.ncl)
• ESACCI-AEROSOL (od550aer, od870aer, od550abs, od550lt1aer - esmvaltool/utils/cmorizers/obs/cmorize_obsESACCI-AEROSOL.ncl)
• ESACCI-CLOUD (clt - esmvaltool/utils/cmorizers/obs/cmorize_obsESACCI-CLOUD.ncl)
• ESACCI-OZONE (toz - esmvaltool/utils/cmorizers/obs/cmorize_obsESACCI-OZONE.ncl)
• ESACCI-SOILMOISTURE (sm - esmvaltool/utils/cmorizers/obs/cmorize_obsESACCI-SOILMOISTURE.ncl)
• ESACCI-SST (ts - esmvaltool/utils/cmorizers/obs/cmorize_obsESACCI-SST.ncl)
• GPCP-SG (pr - obs4mips)
• HadISST (tas - esmvaltool/utils/cmorizers/obs/cmorize_obsHadISST.ncl)
• MODIS (od550aer - esmvaltool/utils/cmorizers/obs/cmorize_obsMODIS.ncl)
• NCEP (tas, ta, ua, va, zg - esmvaltool/utils/cmorizers/obs/cmorize_obsNCEP.ncl)
• NIWA-BS (toz - esmvaltool/utils/cmorizers/obs/cmorize_obsNIWA-BS.ncl)
• PATMOS-x (clt - esmvaltool/utils/cmorizers/obs/cmorize_obsPATMOS-x.ncl)

37.6 References


37.7 Example plots
37.7. Example plots
38.1 Overview

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

38.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_quantilebias.yml

Diagnostics are stored in diag_scripts/quantilebias/

- quantilebias.R

38.3 User settings

Required settings for script
• perc_lev: quantile (in %), e.g. 50

### 38.4 Variables

• pr (atmos, monthly, longitude latitude time)

### 38.5 Observations and reformat scripts

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

### 38.6 References

39.1 Overview

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

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

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

The bias of the runoff coefficient is calculated via: $C_R = \frac{R_{\text{Model}} - R_{\text{GRDC}}}{P_{\text{WFDEI}}}$ 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.
39.2 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

39.3 User settings in recipe

1. Script catchment_analysis.py

   Required settings (scripts)
   - catchmentmask: netCDF file indicating the grid cell for a specific catchment. Mode 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.

39.4 Variables

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

39.5 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.

39.6 References

39.7 Example plots

Fig. 1: Catchment definitions used in the diagnostics.
Fig. 2: 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. 3: 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.
40.1 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).

40.2 Available recipes and diagnostics

Recipes are stored in recipes/
- recipe_rainfarm.yml

Diagnostics are stored in diag_scripts/rainfarm/
- rainfarm.R

40.3 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 if no orographic weights are to be used, else set it to the full path to a fine-scale precipitation climatology 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).

40.4 Variables

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

40.5 Observations and reformat scripts

None.

40.6 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
41.1 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.

41.2 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.

41.3 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.

41.4 Variables

• pr.tas (daily)
42.1 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 multimodel mean (black circle), with the diameter of each circle representing the range of $I^2$ values encompassed by the 5th and 95th percentiles of the bootstrap ensemble. The $I^2$ values vary around one, with values greater than one for underperforming models, and values less than one for more accurate models.

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

42.2 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.

### 42.3 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”)

### 42.4 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)
42.5 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.

- ERA-Interim (hfds, hus, psl, ta, tas, tauu, tauv, ua, va - esmvaltool/utils/cmorizers/obs/cmorize_obsERA-Interim.ncl)
- HadISST (sic, tos - reformat_scripts/obs/reformat_obs_HadISST.ncl)
- GPCP-SG (pr - obs4mips)

42.6 References


42.7 Example plots

![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).](image)

Fig. 1: 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).
43.1 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 groundwater levels, usually after extended periods of meteorological drought. GCMs normally do not simulate hydrological processes in sufficient detail to give deeper insights into hydrological drought processes. Neither do they properly describe agricultural droughts, when crops become affected by the hydrological drought. However, hydrological drought can be estimated by accounting for evapotranspiration, and thereby estimate the surface retention of water. The standardized precipitation-evapotranspiration index (SPEI; Vicente-Serrano et al., 2010) has been developed to also account for temperature effects on the surface water fluxes. Evapotranspiration is not normally calculated in GCMs, so SPEI often takes other inputs to estimate the evapotranspiration. Here, the Thornthwaite (Thornthwaite, 1948) method based on temperature is applied.

43.2 Available recipes and diagnostics

Recipes are stored in recipes/

- recipe_spei.yml

Diagnostics are stored in diag_scripts/droughtindex/

- diag_spi.r: calculate the SPI index
• diag_spei.r: calculate the SPEI index

43.3 User settings

1. Script diag_spi.py

   Required settings (script)
   • reference_dataset: dataset_name The reference data set acts as a baseline for calculating model bias.

2. Script diag_spei.py

   Required settings (script)
   • reference_dataset: dataset_name The reference data set acts as a baseline for calculating model bias.

43.4 Variables

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

43.5 References


43.6 Example plots
Fig. 1: (top) Probability distribution of the standardized precipitation-evapotranspiration index of a sub-set of the CMIP5 models, and (bottom) bias relative to the CRU reference data set.

Fig. 2: (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.
44.1 Overview

The goal of this diagnostic is to simulate single-model ensembles from an observational dataset to investigate the effect of observational uncertain. 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.

44.2 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

44.3 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

Required settings for script
• number_of_members: integer specifying the number of members to be generated
• beta: the user defined underdispersion (beta >= 0)

44.4 Variables

• psl(atmos, daily-monthly, longitude, latitude, time)
• tas(atmos, daily-monthly, longitude, latitude, time)

44.5 Observations and reformat scripts

None

44.6 References


44.7 Example plots
5 synthetic members generated
45.1 Overview

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

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

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

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

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

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

45.2 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;

45.3 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

45.4 Variables

• hfls (atmos, monthly mean, time latitude longitude)
• hfss (atmos, monthly mean, time latitude longitude)
• hus (atmos, monthly mean, time plev latitude longitude)
• pr (atmos, monthly mean, time latitude longitude)
• prsn (atmos, monthly mean, time latitude longitude)
• ps (atmos, monthly mean, time latitude longitude)
• rlds (atmos, monthly mean, time latitude longitude)
• rlus (atmos, monthly mean, time latitude longitude)
• rlut (atmos, monthly mean, time latitude longitude)
• rsds (atmos, monthly mean, time latitude longitude)
• rsdt (atmos, monthly mean, time latitude longitude)
• rsus (atmos, monthly mean, time latitude longitude)
• rsut (atmos, monthly mean, time latitude longitude)
• ta (atmos, daily mean, time plev latitude longitude)
• tas (atmos, daily mean, time latitude longitude)
• ts (atmos, monthly mean, time latitude longitude)
• ua (atmos, daily mean, time plev latitude longitude)
• uas (atmos, daily mean, time latitude longitude)
• va (atmos, daily mean, time plev latitude longitude)
• vas (atmos, daily mean, time latitude longitude)
• wap (atmos, daily mean, time plev latitude longitude)

45.5 References

45.6 Example plots

(a) total heat transports

(a) atmos heat transports

(a) ocean heat transports
Stratosphere-troposphere coupling and annular modes indices (ZMNAM)

46.1 Overview

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

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

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

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

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

46.2 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
46.3 User settings

None.

46.4 Variables

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

46.5 Observations and reformat scripts

None.

46.6 References


46.7 Example plots
Fig. 1: Example output: time series of the zonal-mean NAM index.

Fig. 2: Example output: regression map for a selected pressure level.
Part VI

ESMValTool Code API Documentation
ESMValTool is mostly used as a commandline tool. However, it is also possibly to use (parts of) EsmValTool as a library. This section documents the public API of ESMValTool.
CHAPTER 47

Preprocessor functions
CMOR functions

48.1 Checking compliance

Module for checking iris cubes against their CMOR definitions.

class esmvalcore.cmor.check.CMORCheck(cube, var_info, frequency=None, fail_on_error=False, automatic_fixes=False)

Bases: object

Class used to check the CMOR-compliance of the data.

It can also fix some minor errors and does some minor data homogeneization:

Parameters

• **cube** (*iris.cube.Cube*) – Iris cube to check.
• **var_info** (*variables_info.VariableInfo*) – Variable info to check.
• **frequency** (*str*) – Expected frequency for the data.
• **fail_on_error** (*bool*) – If true, CMORCheck stops on the first error. If false, it collects all possible errors before stopping.
• **automatic_fixes** (*bool*) – If True, CMORCheck will try to apply automatic fixes for any detected error, if possible.

**frequency**

Expected frequency for the data.

Type *str*

**automatic_fixes**

If True, CMORCheck will try to apply automatic fixes for any detected error, if possible.

Type *bool*
**check_data** *(logger=None)*
Check the cube data.

Performs all the tests that require to have the data in memory. Assumes that metadata is correct, so you must call check_metadata prior to this.

It will also report some warnings in case of minor errors.

*Raises* `CMORCheckError` – If errors are found. If fail_on_error attribute is set to True, raises as soon as an error is detected. If set to False, it perform all checks and then raises.

**check_metadata** *(logger=None)*
Check the cube metadata.

Perform all the tests that do not require to have the data in memory.

It will also report some warnings in case of minor errors and homogenize some data:

- Equivalent calendars will all default to the same name.
- Auxiliary coordinates year, month_number, day_of_month and day_of_year will be added for the time axis.

*Raises* `CMORCheckError` – If errors are found. If fail_on_error attribute is set to True, raises as soon as an error is detected. If set to False, it perform all checks and then raises.

**has_errors**
Check if there are reported errors.

*Returns* True if there are pending errors, False otherwise.

*Return type* bool

**has_warnings**
Check if there are reported warnings.

*Returns* True if there are pending warnings, False otherwise.

*Return type* bool

**report_error** *(message, *args)*
Report an error.

If fail_on_error is set to True, raises automatically. If fail_on_error is set to False, stores it for later reports.

*Parameters*

- `message (str: unicode)` – Message for the error.
- `*args` – arguments to format the message string.

**report_errors**
Report detected errors.

*Raises* `CMORCheckError` – If any errors were reported before calling this method.

**report_warning** *(message, *args)*
Report a warning.

If fail_on_error is set to True, logs it automatically. If fail_on_error is set to False, stores it for later reports.

*Parameters*

- `message (str: unicode)` – Message for the warning.
- `*args` – arguments to format the message string.
**report_warnings** *(logger)*

Report detected warnings to the given logger.

**Parameters**

- **logger**

**exception** esmvalcore.cmor.check.CMORCheckError

Bases: Exception

Exception raised when a cube does not pass the CMORCheck.

**args**

with_traceback()  
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

esmvalcore.cmor.check.cmor_check *(cube, cmor_table, mip, short_name, frequency)*

Check if cube conforms to variable’s CMOR definiton.

Equivalent to calling cmor_check_metadata and cmor_check_data consecutively.

**Parameters**

- **cube** *(iris.cube.Cube)* – Data cube to check.
- **cmor_table** *(basestring)* – CMOR definitions to use.
- **mip** – Variable’s mip.
- **short_name** *(basestring)* – Variable’s short name.
- **frequency** *(basestring)* – Data frequency.

esmvalcore.cmor.check.cmor_check_data *(cube, cmor_table, mip, short_name, frequency)*

Check if data conforms to variable’s CMOR definiton.

The checks performed at this step require the data in memory.

**Parameters**

- **cube** *(iris.cube.Cube)* – Data cube to check.
- **cmor_table** *(basestring)* – CMOR definitions to use.
- **mip** – Variable’s mip.
- **short_name** *(basestring)* – Variable’s short name
- **frequency** *(basestring)* – Data frequency

esmvalcore.cmor.check.cmor_check_metadata *(cube, cmor_table, mip, short_name, frequency)*

Check if metadata conforms to variable’s CMOR definiton.

None of the checks at this step will force the cube to load the data.

**Parameters**

- **cube** *(iris.cube.Cube)* – Data cube to check.
- **cmor_table** *(basestring)* – CMOR definitions to use.
- **mip** – Variable’s mip.
- **short_name** *(basestring)* – Variable’s short name.
- **frequency** *(basestring)* – Data frequency.
48.2 Fixing issues

Apply automatic fixes for known errors in cmorized data

All functions in this module will work even if no fixes are available for the given dataset. Therefore is recommended to apply them to all variables to be sure that all known errors are fixed.

```python
esmvalcore.cmor.fix.fix_data(cube, short_name, project, dataset, cmor_table=None, mip=None, frequency=None)
```

Fix cube data if fixes add present and check it anyway.

This method assumes that metadata is already fixed and checked.

This method collects all the relevant fixes for a given variable, applies them and checks resulting cube (or the original if no fixes were needed) metadata to ensure that it complies with the standards of its project CMOR tables.

**Parameters**

- `cube` *(iris.cube.Cube)* – Cube to fix
- `short_name` *(str)* – Variable’s short name
- `project` *(str)* –
- `dataset` *(str)* –
- `cmor_table` *(str, optional)* – CMOR tables to use for the check, if available
- `mip` *(str, optional)* – Variable’s MIP, if available
- `frequency` *(str, optional)* – Variable’s data frequency, if available

**Returns** Fixed and checked cube

**Return type** `iris.cube.Cube`

**Raises** `CMORCheckError` – If the checker detects errors in the data that it can not fix.

```python
esmvalcore.cmor.fix.fix_file(file, short_name, project, dataset, output_dir)
```

Fix files before ESMValTool can load them.

This fixes are only for issues that prevent iris from loading the cube or that cannot be fixed after the cube is loaded.

Original files are not overwritten.

**Parameters**

- `file` *(str)* – Path to the original file
- `short_name` *(str)* – Variable’s short name
- `project` *(str)* –
- `dataset` *(str)* –
- `output_dir` *(str)* – Output directory for fixed files

**Returns** Path to the fixed file

**Return type** `str`

```python
esmvalcore.cmor.fix.fix_metadata(cubes, short_name, project, dataset, cmor_table=None, mip=None, frequency=None)
```

Fix cube metadata if fixes are required and check it anyway.
This method collects all the relevant fixes for a given variable, applies them and checks the resulting cube (or the original if no fixes were needed) metadata to ensure that it complies with the standards of its project CMOR tables.

Parameters

- **cubes** (*iris.cube.CubeList*) – Cubes to fix
- **short_name** (*str*) – Variable’s short name
- **project** (*str*) –
- **dataset** (*str*) –
- **cmor_table** (*str, optional*) – CMOR tables to use for the check, if available
- **mip** (*str, optional*) – Variable’s MIP, if available
- **frequency** (*str, optional*) – Variable’s data frequency, if available

Returns Fixed and checked cube

Return type *iris.cube.Cube*

Raises **CMORCheckError** – If the checker detects errors in the metadata that it can not fix.

### 48.3 Using CMOR tables

CMOR information reader for ESMValTool.

Read variable information from CMOR 2 and CMOR 3 tables and make it easily available for the other components of ESMValTool

```python
class esmvalcore.cmor.table.CMIP5Info(cmor_tables_path, default=None)
    Bases: object

    Class to read CMIP5-like data request.

    Parameters cmor_tables_path (basestring) – Path to the folder containing the Tables folder with the json files

    get_table (table)
    Search and return the table info.

    Parameters table (basestring) – Table name

    Returns Return the TableInfo object for the requested table if found, returns None if not

    Return type TableInfo

    get_variable (table, short_name)
    Search and return the variable info.

    Parameters

    • table (basestring) – Table name
    • short_name (basestring) – Variable’s short name

    Returns Return the VariableInfo object for the requested variable if found, returns None if not

    Return type VariableInfo
```
class esmvalcore.cmor.table.CMIP6Info(cmor_tables_path, default=None)

Class to read CMIP6-like data request.
This uses CMOR 3 json format

Parameters
cmor_tables_path (basestring) – Path to the folder containing the Tables
folder with the json files

get_table (table)
Search and return the table info.

Parameters
table (basestring) – Table name

Returns Return the TableInfo object for the requested table if found, returns None if not

Return type TableInfo

get_variable (table, short_name)
Search and return the variable info.

Parameters
• table (basestring) – Table name
• short_name (basestring) – Variable’s short name

Returns Return the VariableInfo object for the requested variable if found, returns None if not

Return type VariableInfo

esmvalcore.cmor.table.CMOR_TABLES = {}
CMOR info objects.

Type dict of str, obj

class esmvalcore.cmor.table.CoordinateInfo(name)

Class to read and store coordinate information.

axis = None
Axis

long_name = None
Long name

out_name = None
Out name
This is the name of the variable in the file

read_json (json_data)
Read coordinate information from json.
Non-present options will be set to empty

Parameters
json_data (dict) – dictionary created by the json reader containing coordinate information

requested = None
Values requested

standard_name = None
Standard name

Chapter 48. CMOR functions
stored_direction = None
    Direction in which the coordinate increases

units = None
    Units

valid_max = None
    Maximum allowed value

valid_min = None
    Minimum allowed value

value = None
    Coordinate value

var_name = None
    Short name

class esmvalcore.cmor.table.CustomInfo(cmor_tables_path=None)
    Bases: esmvalcore.cmor.table.CMIP5Info

Class to read custom var info for ESMVal.

Parameters cmor_tables_path (basestring or None) – Full path to the table or name for the table if it is present in ESMValTool repository

get_table (table)
    Search and return the table info.

    Parameters table (basestring) – Table name

    Returns Return the TableInfo object for the requested table if found, returns None if not

    Return type TableInfo

get_variable (table, short_name)
    Search and return the variable info.

    Parameters

        • table (basestring) – Table name

        • short_name (basestring) – Variable’s short name

    Returns Return the VariableInfo object for the requested variable if found, returns None if not

    Return type VariableInfo

class esmvalcore.cmor.table.JsonInfo
    Bases: object

Base class for the info classes.

Provides common utility methods to read json variables

class esmvalcore.cmor.table.TableInfo (*args, **kwargs)
    Bases: dict

Container class for storing a CMOR table.

clear () → None. Remove all items from D.

copy () → a shallow copy of D

fromkeys ()
    Create a new dictionary with keys from iterable and values set to value.

48.3. Using CMOR tables
get()  
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D’s items

keys() → a set-like object providing a view on D’s keys

pop(k[, d]) → v, remove specified key and return the corresponding value.  
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a  
2-tuple; but raise KeyError if D is empty.

setdefault()  
Insert key with a value of default if key is not in the dictionary.  
Return the value for key if key is in the dictionary, else default.

update([E], **F) → None. Update D from dict/iterable E and F.  
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k]  
If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v  
In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D’s values

class esmvalcore.cmor.table.VariableInfo(table_type, short_name)  
Bases: esmvalcore.cmor.table.JsonInfo

Class to read and store variable information.

coordinates = None  
Coordinates

dimensions = None  
List of dimensions

frequency = None  
Data frequency

long_name = None  
Long name

modeling_realm = None  
Modeling realm

positive = None  
Increasing direction

read_json(json_data)  
Read variable information from json.  
Non-present options will be set to empty

Parameters json_data (dict) – dictionary created by the json reader containing variable  
information

short_name = None  
Short name

standard_name = None  
Standard name

units = None  
Data units
valid_max = None
    Maximum admitted value

valid_min = None
    Minimum admitted value

esmvalcore.cmor.table.read_cmor_tables(cfg_developer)
    Read cmor tables required in the configuration.

    Parameters cfg_developer (dict of str) – Parsed config-developer file
Shared diagnostic script code

49.1 Plotting

Module that provides common plot functions.

- `esmvaltool.diag_scripts.shared.plot.get_path_to_mpl_style(style_file=None)`
  Get path to matplotlib style file.

- `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.quickplot(cube, filename, plot_type, **kwargs)`
  Plot a cube using one of the iris.quickplot functions.

- `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.
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`, `markersize`, 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`, `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.scatterplot(x_data, y_data, filepath, **kwargs)

Plot a scatterplot.
Various diagnostic packages exist as part of ESMValTool.

ESMValTool diagnostic scripts.
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

51.1 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

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

Returns

Return type  iris.cube

esmvaltool.diag_scripts.ocean.diagnostic_tools.folder(name)

Make a directory out of a string or list or strings.
Take a string or a list of strings, convert it to a directory style, then make the folder and the string. Returns folder string and final character is always os.sep (‘’)

Parameters  name (list or string) – A list of nested directories, or a path to a directory.

Returns  Returns a string of a full (potentially new) path of the directory.

Return type  str

esmvaltool.diag_scripts.ocean.diagnostic_tools.get_array_range(arrays)

Determine the minimum and maximum values of a list of arrays.

Parameters  arrays (list of numpy.array) – A list of numpy.array.

Returns  A list of two values, the overall minumum and maximum values of the list of cubes.

Return type  list

esmvaltool.diag_scripts.ocean.diagnostic_tools.get_colour_from_cmap(number, total, cmap=’jet’)

Get a colour number of total from a cmap.
This function is used when several lines are created evenly along a colour map.

Parameters
• **number** *(int, float)* – The
  • **total** *(int)* –
  • **cmap** *(string, plt.cm)* – A colour map, either by name (string) or from matplotlib

```python
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*

```python
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*

```python
esmvaltool.diag_scripts.ocean.diagnostic_tools.get_decade(coord, value)
```
Determine the decade.
Called by iris.coord_categorisation.add_categorised_coord.

```python
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*

```python
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 dictionairy, 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 dictionairy, passed by ESMValTool.

•  **metadata** (*dict*) – the metadata dictionairy

Returns  List of thresholds

Return type  list

esmvaltool.diag_scripts.ocean.diagnostic_tools.make_cube_layer_dict(cube)
Take a cube and return a dictionairy layer:cube

Each item in the dict is a layer with a separate cube for each layer. ie: cubes[depth] = cube from specific layer

Cubes with no depth component are returned as dict, where the dict key is a blank empty string, and the value is

the cube.

Parameters  **cube** (*iris.cube.Cube*) – the opened dataset as a cube.

Returns  A dictionary of layer name : layer cube.

Return type  dict

esmvaltool.diag_scripts.ocean.diagnostic_tools.match_model_to_key(model_type,
cfg_dict, input_files_dict)

Match up model or observations dataset dictionairies from config file.
This function checks that the control_model, exper_model and observational_dataset dictionairies from the recipe are matched with the input file dictionairy 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 dictionairy, 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*
Part VII

Indices and tables
• genindex
• search
Python Module Index

e
esmvalcore.cmor, 199
esmvalcore.cmor.check, 199
esmvalcore.cmor.fix, 202
esmvalcore.cmor.table, 203
esmvaltool.diag_scripts, 211
esmvaltool.diag_scripts.ocean, 211
esmvaltool.diag_scripts.ocean.diagnostics, 213
esmvaltool.diag_scripts.shared.plot, 209
A
add_legend_outside_right() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 213
args (esmvalcore.cmor.check.CMORCheckError attribute), 201
automatic_fixes (esmvalcore.cmor.check.CMORCheck attribute), 199
axis (esmvalcore.cmor.table.CoordinateInfo attribute), 204

B
bgc_units() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 214

C
check_data() (esmvalcore.cmor.check.CMORCheck method), 199
clear() (esmvalcore.cmor.table.TableInfo method), 205
CMIP5Info (class in esmvalcore.cmor.table), 203
CMIP6Info (class in esmvalcore.cmor.table), 203
cmor_check() (in module esmvalcore.cmor.check), 201
cmor_check_data() (in module esmvalcore.cmor.check), 201
cmor_check_metadata() (in module esmvalcore.cmor.check), 201
CMOR_TABLES (in module esmvalcore.cmor.table), 204
CMORCheck (class in esmvalcore.cmor.check), 199
CMORCheckError, 201
CoordinateInfo (class in esmvalcore.cmor.table), 204
coordinates (esmvalcore.cmor.table.VariableInfo attribute), 206
copy() (esmvalcore.cmor.table.TableInfo method), 205
cube_time_to_float() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 214
CustomInfo (class in esmvalcore.cmor.table), 205

d
Decadal average() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 214
dimensions (esmvalcore.cmor.table.VariableInfo attribute), 206

E
esmvalcore.cmor (module), 199
esmvalcore.cmor.check (module), 199
esmvalcore.cmor.fix (module), 202
esmvalcore.cmor.table (module), 203
esmvaltool.diag_scripts (module), 211
esmvaltool.diag_scripts.ocean (module), 211
esmvaltool.diag_scripts.ocean.diagnostic_tools (module), 213
esmvaltool.diag_scripts.shared.plot (module), 209

F
fix_data() (in module esmvalcore.cmor.fix), 202
fix_file() (in module esmvalcore.cmor.fix), 202
fix_metadata() (in module esmvalcore.cmor.fix), 202
folder() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 214
frequency (esmvalcore.cmor.check.CMORCheck attribute), 199
frequency (esmvalcore.cmor.table.VariableInfo attribute), 206
fromkeys() (esmvalcore.cmor.table.TableInfo method), 205
get () (esmvalcore.cmor.table.TableInfo method), 205
get_array_range() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 214
get_colour_from_cmap() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 214
get_cube_range() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 215
get_cube_range_diff() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 215
get_decade() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 215
get_image_format() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 215
get_image_path() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 215
get_input_files() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 216
get_obs_projects() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 216
get_path_to_mpl_style() (in module esmvaltool.diag_scripts.shared.plot), 209
get_table() (esmvalcore.cmor.table.CMIP5Info method), 203
get_table() (esmvalcore.cmor.table.CMIP6Info method), 204
get_table() (esmvalcore.cmor.table.CustomInfo method), 205
get_variable() (esmvalcore.cmor.table.CMIP5Info method), 203
get_variable() (esmvalcore.cmor.table.CMIP6Info method), 204
get_variable() (esmvalcore.cmor.table.CustomInfo method), 205
guess_calendar_datetime() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 216
has_errors() (esmvalcore.cmor.check.CMORCheck method), 200
has_warnings() (esmvalcore.cmor.check.CMORCheck method), 200
items() (esmvalcore.cmor.table.TableInfo method), 206
JsonInfo (class in esmvalcore.cmor.table), 205
keys() (esmvalcore.cmor.table.TableInfo method), 206
load_thresholds() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 216
long_name (esmvalcore.cmor.table.CoordinateInfo attribute), 204
long_name (esmvalcore.cmor.table.VariableInfo attribute), 206
make_cube_layer_dict() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 216
match_model_to_key() (in module esmvaltool.diag_scripts.ocean.diagnostic_tools), 216
modeling_realm (esmvalcore.cmor.table.VariableInfo attribute), 206
multi_dataset_scatterplot() (in module esmvaltool.diag_scripts.shared.plot), 209
out_name (esmvalcore.cmor.table.CoordinateInfo attribute), 204
pop() (esmvalcore.cmor.table.TableInfo method), 206
popitem() (esmvalcore.cmor.table.TableInfo method), 206
positive (esmvalcore.cmor.table.VariableInfo attribute), 206
quickplot() (in module esmvaltool.diag_scripts.shared.plot), 209
read_cmor_tables() (in module esmvalcore.cmor.table), 207
read_json() (esmvalcore.cmor.table.CoordinateInfo method), 204
read_json() (esmvalcore.cmor.table.VariableInfo method), 206
report_error() (esmval-core.cmor.check.CMORCheck method), 200
report_errors() (esmval-core.cmor.check.CMORCheck method), 200
report_warning() (esmval-core.cmor.check.CMORCheck method), 200
report_warnings() (esmval-core.cmor.check.CMORCheck method), 200
requested (esmvalcore.cmor.table.CoordinateInfo attribute), 204

S
scatterplot() (in module esmval-tool.diag_scripts.shared.plot), 210
setdefault() (esmvalcore.cmor.table.TableInfo method), 206
short_name (esmvalcore.cmor.table.VariableInfo attribute), 206
standard_name (esmvalcore.cmor.table.CoordinateInfo attribute), 204
standard_name (esmvalcore.cmor.table.VariableInfo attribute), 206
stored_direction (esmvalcore.cmor.table.CoordinateInfo attribute), 204

T
TableInfo (class in esmvalcore.cmor.table), 205

U
units (esmvalcore.cmor.table.CoordinateInfo attribute), 205
units (esmvalcore.cmor.table.VariableInfo attribute), 206
update() (esmvalcore.cmor.table.TableInfo method), 206

V
valid_max (esmvalcore.cmor.table.CoordinateInfo attribute), 205
valid_max (esmvalcore.cmor.table.VariableInfo attribute), 206
valid_min (esmvalcore.cmor.table.CoordinateInfo attribute), 205
valid_min (esmvalcore.cmor.table.VariableInfo attribute), 207
value (esmvalcore.cmor.table.CoordinateInfo attribute), 207
values() (esmvalcore.cmor.table.TableInfo method), 205
var_name (esmvalcore.cmor.table.CoordinateInfo attribute), 205
VariableInfo (class in esmvalcore.cmor.table), 206

W
with_traceback() (esmval-core.cmor.check.CMORCheckError method), 201