Technical standards for diagnostics

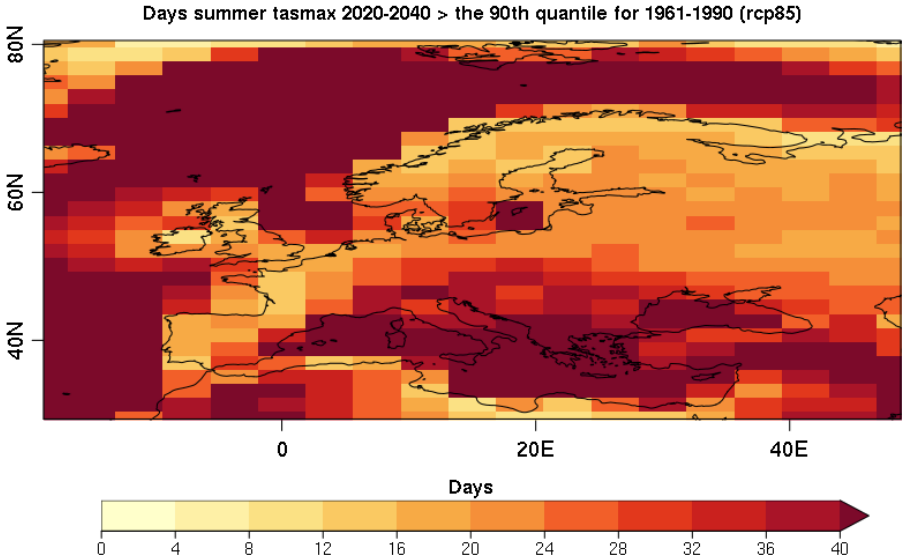
Javier Vegas Regidor (BSC)
Kim Serradell (BSC)
Stéphane Sénési (IPSL)

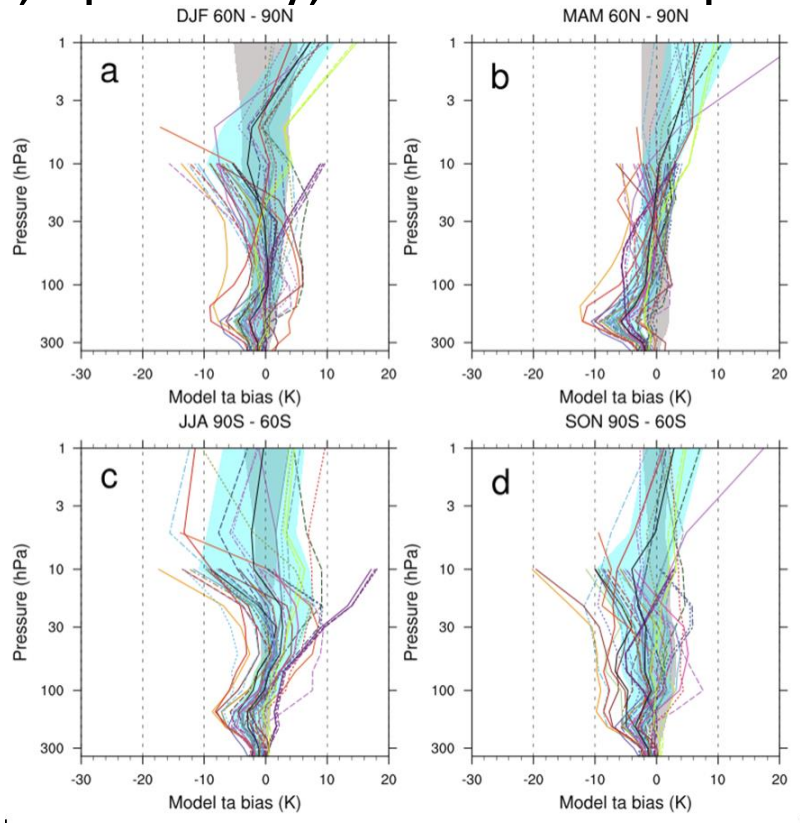
IS-ENES3 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

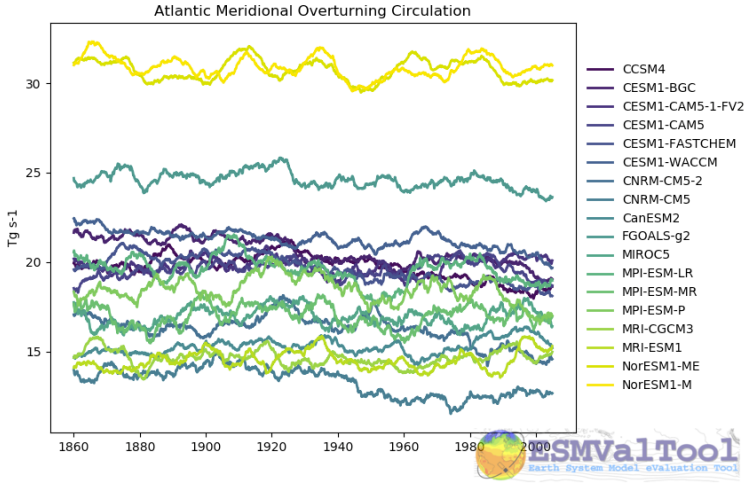
What we call a diagnostic?

The program that analyze the data and produce the final results.

Or, in other terms, the part of your software stack that you thoroughly describe in the methodology of your papers and, optionally, the code that produces the figures you show in it.







Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A familiar story

- I am very impressed with your presentation
- Thanks! It was a lot of work but it finally paid off!
- I am wondering if you will be interested in a collaboration. I have a couple of problems that I would like to solve by using your method.
- That would be great. I do not have much time, but I can share my scripts with you and help you to run them.
- Great!

Why

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A familiar nightmare

- Oh, so you used Python for this. Well, I usually work with R and Matlab, but I will give it a try.
- Can you help? It looks like it can not run in my machine?
- No worries, it found it requires Python 3.8 but I was using 3.7
- Oh, I see that your functions are expecting Iris cubes. Do you know how to get one from an xarray dataset? A colleague helped me getting the data loaded, but she uses xarray so I have the data in its format
- Sorry, I need your help again. You told me that I can activate a low pass filter, where can I find the option to control that?

So why a standard?

To reduce at maximum the complications of sharing the code between scientists / institutes / analysis tools

This implies the standard should provide a way to reduce the issues coming from

- Different software stacks due to versions / libraries / programming languages
- Different data organization
- Different preprocessing tools

Having a common interface also improves discoverability of features, as users know where to look to find them

How (WIP)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Basic introduction

The standard is based around three levels of definitions:

- **Must**: all diagnostic have to meet this conditions to comply with the standard.
- **Recommend**: diagnostics are encouraged to meet them but they are not mandatory. Nevertheless, if a list of compatible diagnostics is compiled, those who will meet them will be highlighted.
- **Suggest**: they will never be enforced, but the team consider them as nice to have. They mostly correspond to quality of life improvements.

The idea behind this division is that the bare minimum to ensure functionality is a **must**, that everything we want to encourage developers to do but is not functionally needed is **recommended** and good practices we know that may not be applicable to all diagnostics are **suggested**.

The diagnostic execution

Diagnostics **must** run as independent processes that receive a config file with all the options needed to run, including:

- Data input
- Output directories
- Any configuration required by the diagnostic itself

Why this solution?

- They can be installed in complete isolation, by using virtual environments or containers.
- All diagnostics would be run in the format `mydiag config.yaml`, which simplifies their integration into different tools
- If you keep your config, you keep track of the configuration of your run

It is **recommended** that the diagnostic supports the --help option. There are also some **recommendations** about the content of that output.

The configuration file

It is a yaml file that contains all the information needed to run the diagnostic (although there are others yaml files required for it to work).

Why yaml? Wide support for automatic reading in most programming languages but also really easy to read and write directly, so manual use is still possible.

In the same way it is done for the definitions, configurations in the yaml file have three levels:

- **Required**: they must always be present with the defined meaning and allowed values.
- **Reserved**: they can be omitted, but in the case that are present, they must have the defined meaning and allowed values.
- **Custom**: in all files, extra options can be added as needed by the diagnostics or the tools. The only requirement is that they do not collide with any `required` or `reserved` option.

The configuration file

```
diagnostic_path: /home/jvegas/git/ESMValTool/esmvaltool/diag_scripts/examples/diagnostic.py
input_files:
- /home/jvegas/esmval_data/output/recipe_python_20210927_135059/preproc/map/tas/metadata.yml
tool: ESMValTool
version: 2.3.1
plot_dir: /home/jvegas/esmval_data/output/recipe_python_20210927_135059/plots/map/script1
run_dir: /home/jvegas/esmval_data/output/recipe_python_20210927_135059/run/map/script1
data_dir: /home/jvegas/esmval_data/output/recipe_python_20210927_135059/work/map/script1
auxiliary_data_dir: /home/jvegas/git/esmval/ESMValTool/auxiliary_data
log_level: info
quickplot:
  cmap: Reds
  plot_type: pcolormesh
output_file_type: png
recipe: recipe_python.yml
profile_diagnostic: false
```

The main config file: required keys

- **diagnostic_path** : `str`. Path to the diagnostic executable.
- **input_files**: `list`. List of absolute paths to the metadata files describing the data to be used by the diagnostic.
- **tool**: `str`. Name of the tool used to prepare the data for the diagnostic.
- **version**: `str` Version of the tool used to prepare the data for the diagnostic.
- **run_dir**: `str`. Path to the folder to use as current path for diagnostic execution. Must be writable by the diagnostic.
- **data_dir**: `str`. Path to the folder to store the diagnostic's generated data on. Must be writable by the diagnostic.
- **plot_dir**: `str`. Path to the folder to store the diagnostic's generated figures on. Must be writable by the diagnostic.

The main config file: reserved keys

- **write_plots**: `boolean`, default value is `true`. A flag to control if the diagnostic generates the associated figures with the results or not.
- **write_data**: `boolean`, default value is `true`. A flag to control if the diagnostic generates the associated data files with the results or not.
- **log_level**: `str`, default values is `info`. Sets the granularity of the diagnostic logs and console output. If used, diagnostic must accept at least `error` `warning`, `info` and `debug`
- **auxiliary_data_dir** : `str`. Path to the auxiliary directory where the diagnostic may find further input data.
- **max_proc_number** : `integer`. The maximum number of processors that the tool allows the diagnostic to request

The data files

In the main config files there is a list of paths pointing to files describing the data. This is done to keep the main config file as simple as possible and also to facilitate the reutilization of the same preprocessed format in several diagnostics.

This files are a dictionary of dictionaries, with the keys being the path to the file and the values the metadata associated to this file.

This internal dictionaries have only three **required** keys:

- **alias**: `str`. Unique name for the dataset. Any (`alias`, `variable`) pair must be unique
- **filename**: `str`. Absolute path to the data file.
- **variable**: `str`. Variable name to be used by the diagnostic. It does not have to match any of the CF conventions names in the file

However, there are lots of **reserved** keys matching CMOR and CF definitions (modeling_realm, ensemble, units, standard_name, long_name) among others

The script description file

This is a **recommended** file which describes the diagnostic itself (requirements and outputs) so tools can know enough about the diagnostics to help the users with them:

- A list of the mandatory keys, to check that the user provides all the required ones
- Information about the outputs generated (paths, labels to apply) to help users

Example ENSO

How are we sure this can work? Because it is already doing so!

ESMValTool is based in a very similar approach and we have successfully integrated the ENSO metrics package in it.

How we do it?

- Created a ESMValTool recipe that preprocess the data and get it ready for the ENSO package. In this case the preprocessing is fairly minimal, as almost all the work is done by the ENSO itself
- Created a new diagnostic script that is only a bridge between the ESMValTool and the ENSO package itself: gets the metadata from the ESMValTool files and translates then into the ENSO package format.
- Minimal work on the package was needed

Sharing the standard with the community

- The standard is being developed in https://github.com/IsEnesEval/diagstandards. Join the discussion.
- It will be released through github pages in https://iseneseval.github.io/diagstandards/
- To be released in February 2022: deliverable 5.2 (public)
- To be implemented in ESMValTool as part of WP9 Task 2
- Share the standard with other evaluation software developers to adopt it

is-enes
INFRASTRUCTURE FOR THE EUROPEAN NETWORK FOR EARTH SYSTEM MODELLING

THE CONSORTIUM

Coordinated by **CNRS-IPSL**, the IS-ENES3 project gathers **22 partners** in **11 countries**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°824084

Our website
https://is.enes.org/

Follow us on Twitter !
@ISENES_RI

Contact us at
is-enes@ipsl.fr

Follow our channel
IS-ENES3 H2020