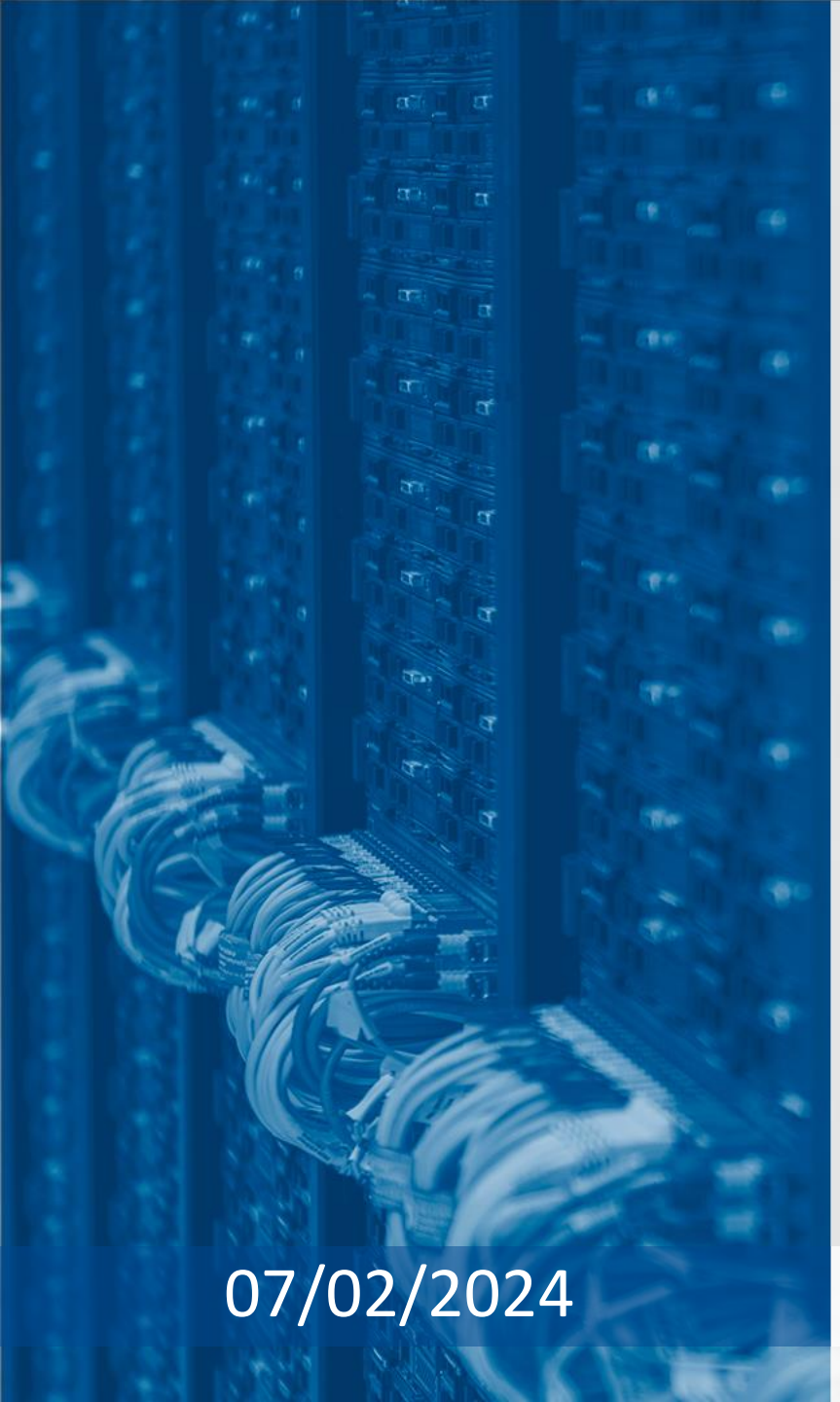


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Providentia v2.3.0
User Meeting

07/02/2024

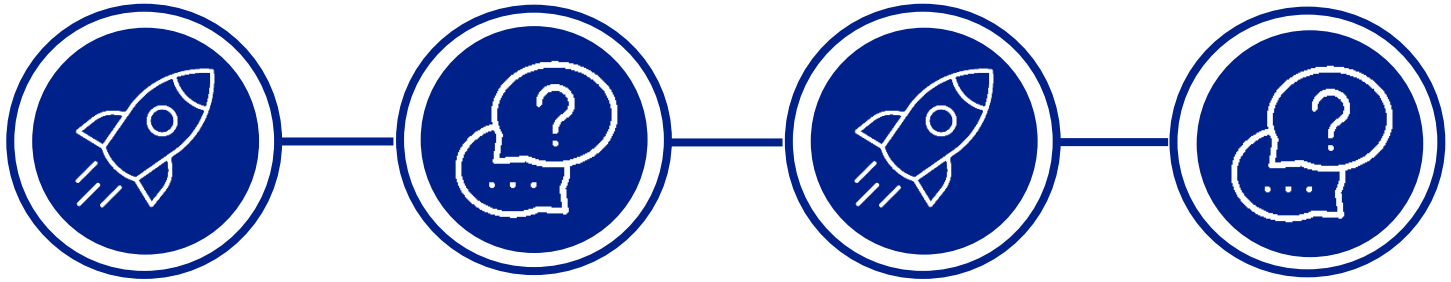
Alba Vilanova | Dene Bowdalo

Timeline

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Customisation features
15 min.

Interactive mode
30 min.



Q & A
5 min.

Q & A
10 min.

Customisation features

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Customisation features

1) Remove stations by statistics.

2) Define map bounds and colour map by pollutant.

3) Give a name to the observations label.

4) Set exceedances thresholds.

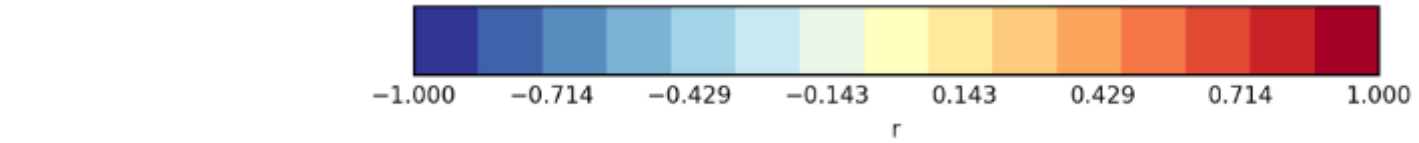
5) Show model domain in maps (with plot option *domain*).

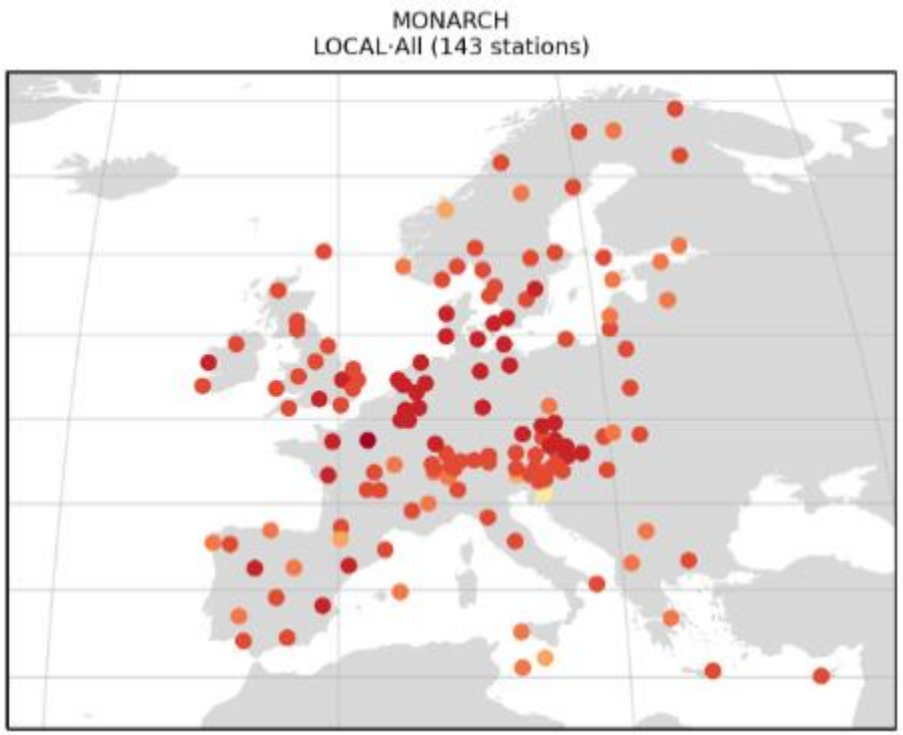
6) Hide points in timeseries and scatter (with plot option *hidedata*).

7) Set custom plot characteristics.

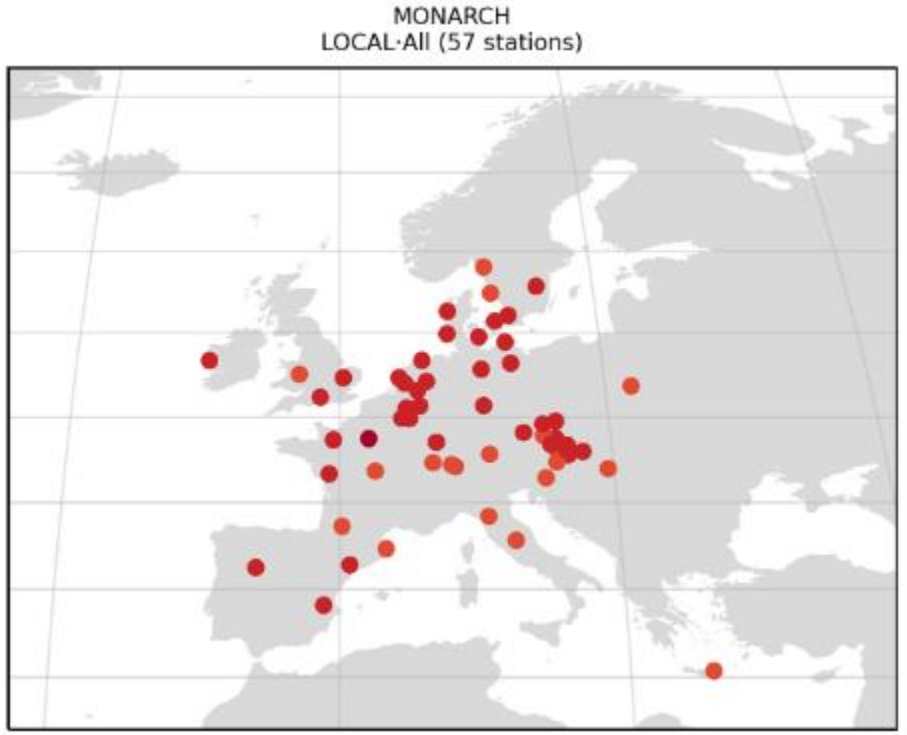
8) Create statistical timeseries (new in master!).

Remove stations by statistics





"r": ["<0.7"]



Remove stations by statistics

If you want to automatically remove stations that have certain statistical values, you will need to add your criteria in the file *settings/remove_extreme_stations.json*. An example of this exists for CAMS:

```
"CAMS": {"r": ["<0.3"],
         "NMB": ["<-100.0", ">100.0"],
         "NRMSE": [">100.0"]}
```

You will also need to add the variable **remove_extreme_stations** in your configuration file, referencing the group of statistics to filter by that you defined, e.g. CAMS:

```
remove_extreme_stations = CAMS
```

Remove stations by statistics

Any absolute statistic can be set to be a bias statistic by adding _bias e.g.:

```
"p95_bias": ["<10",">20"]
```

The statistics can be general, across all components, or they can be specific per component, for example:

```
"CAMS": {"r": {"sconco3": ["<0.3"],
                "sconcno2":[<0.55]},
        "NMB": {"sconco3": ["<-100.0", ">100.0"],
                "sconcno2": ["<-20.0", ">20.0"]},
        "NRMSE": {"sconco3": [">100.0"],
                  "sconcno2":[">200.0"]}}
```

Define map bounds and colour by pollutant

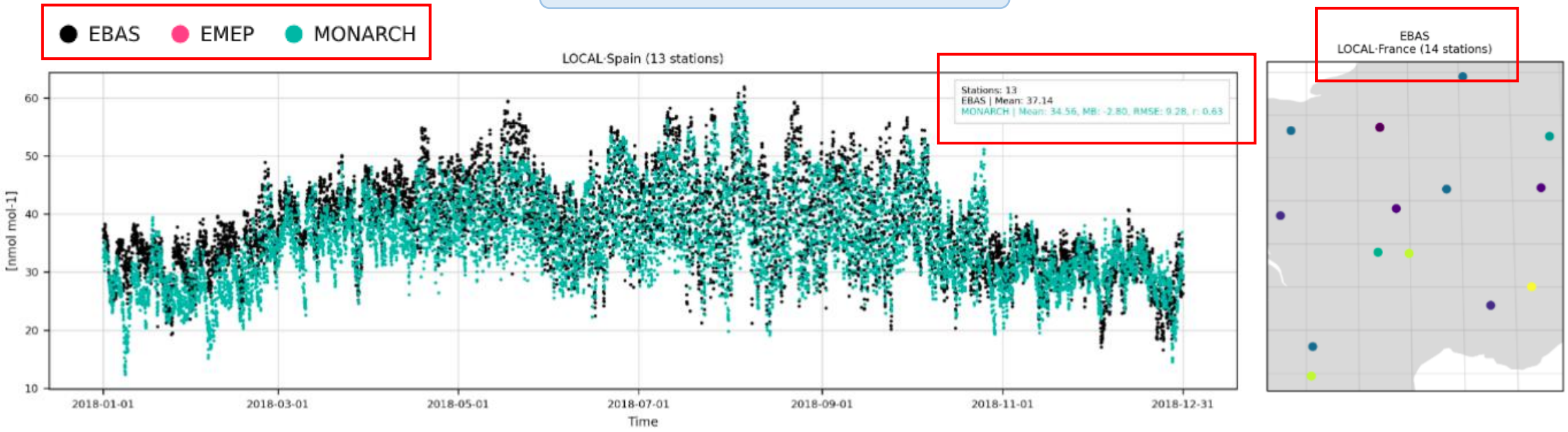
Users can define the colour and bounds of the colorbar (cmap, vmin and vmax) per species using a dictionary, with the keys being the names of the species inside *settings/basic_stats.json* and *settings/experiment_bias_stats.json*. An example can be seen in the code below:

```
"Mean": {"function": "calculate_mean",
            "order": 0,
            "label": "Mean",
            "arguments": {},
            "units": "[measurement_units]",
            "minimum_bias": [0.0],
            "vmin_absolute": {"sconco3": 0, "sconcno2": 0},
            "vmax_absolute": {"sconco3": 20, "sconcno2": 5},
            "vmin_bias": {},
            "vmax_bias": {},
            "cmap_absolute": {"sconco3": "viridis", "sconcno2": "viridis"},
            "cmap_bias": "RdYlBu_r"},
```

Set observations label

By defining **observations_data_label** in our configuration files, we can customise the name of the observations in the legend, plots and maps.

observations_data_label = EBAS



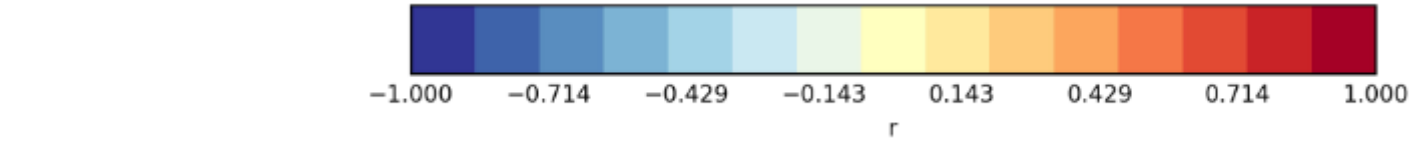
Set exceedances thresholds

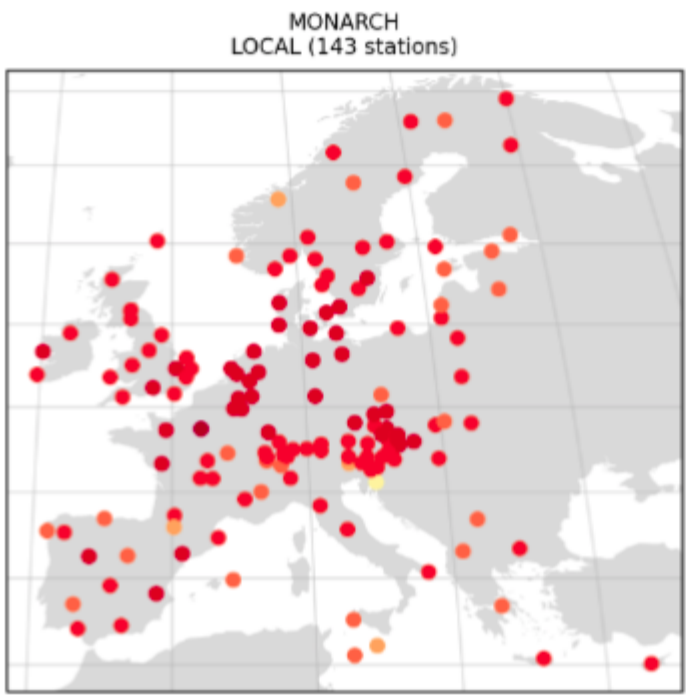
The **Exceedances** statistic gives the number of instances above a threshold. The threshold values can now be set in the file *settings/exceedances.json* per component, or network-component pair, as so:

```
{"sconco3": 90.21,
 "sconcno2": 106.38,
 "EBAS|sconco3": 109.77,
 "EBAS|sconcno2": 88.88}
```

In the case a threshold is set for a specific component, and per network-component, then the threshold for network-component is taken preferentially.

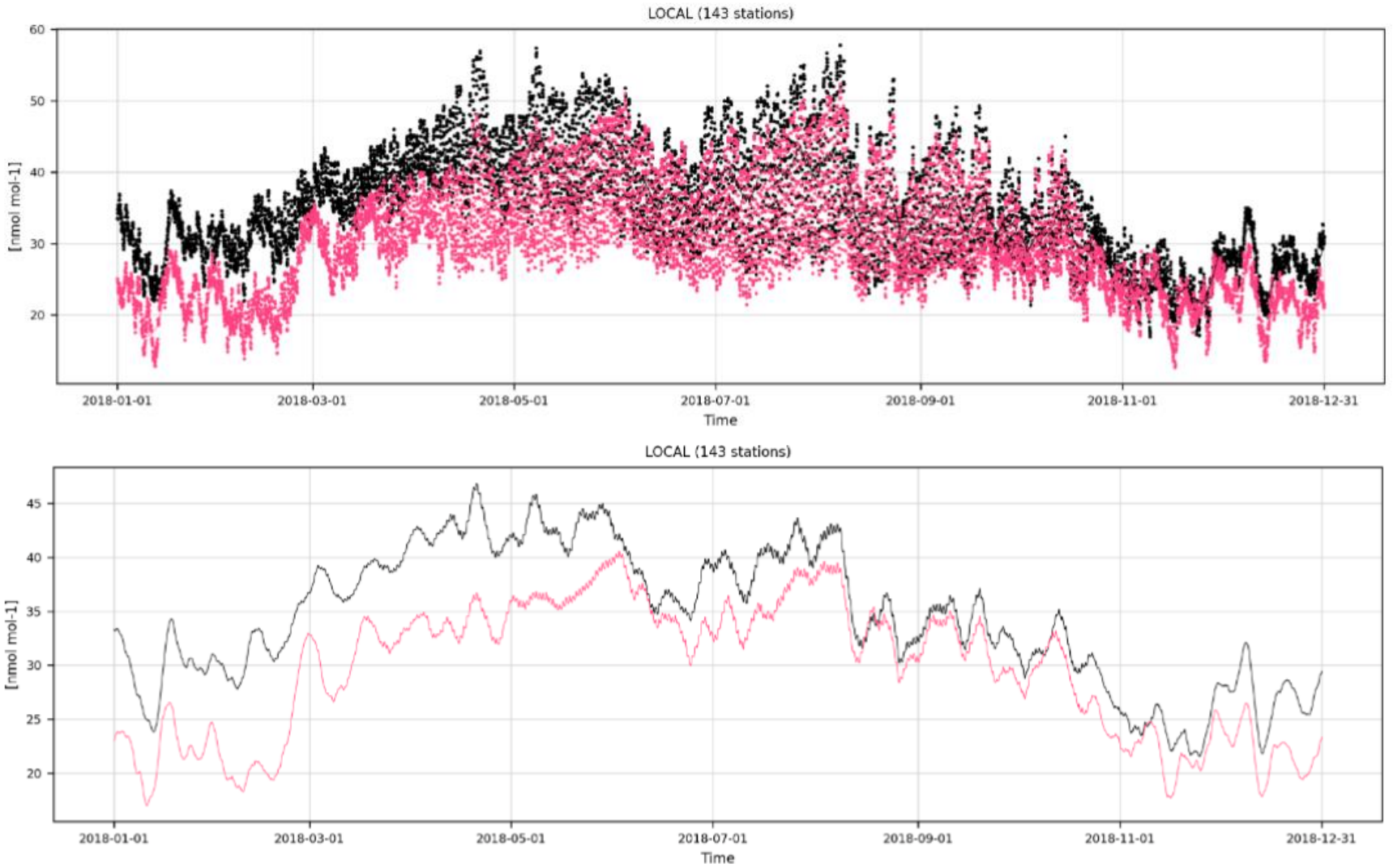
Show model domain on maps





map-r_domain

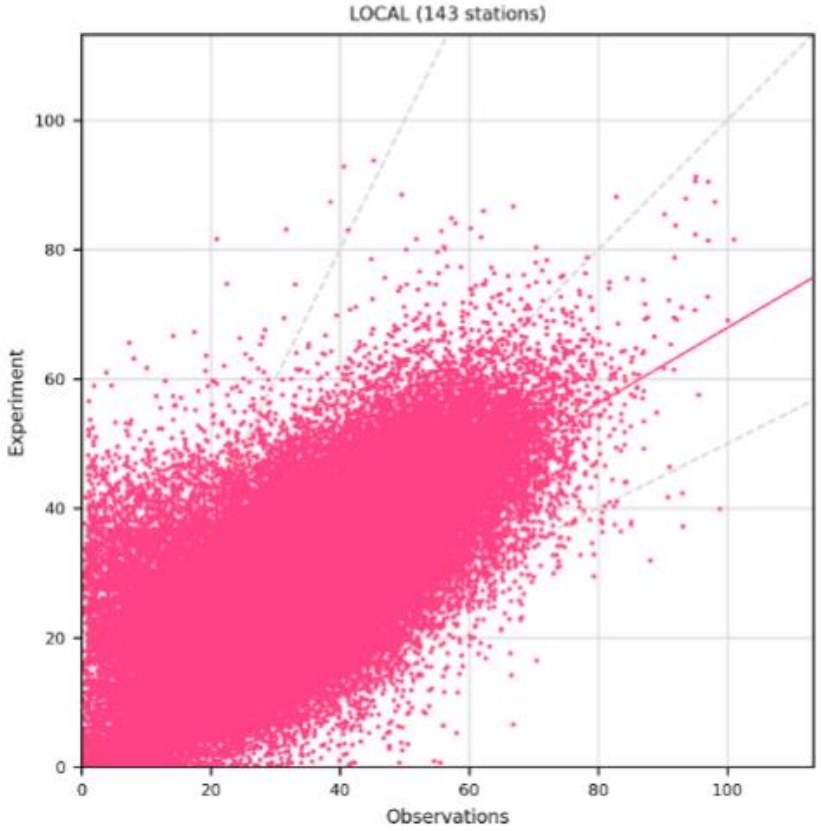
Hide points on timeseries

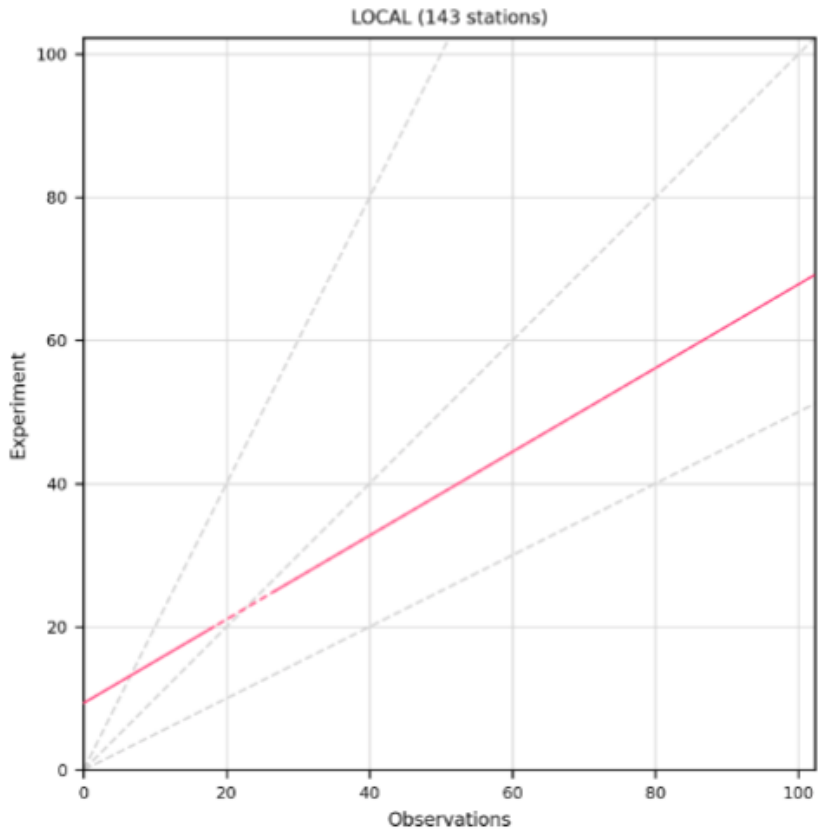


timeseries_smooth_hidedata

Hide points on scatter plot

scatter_regression_hidedata





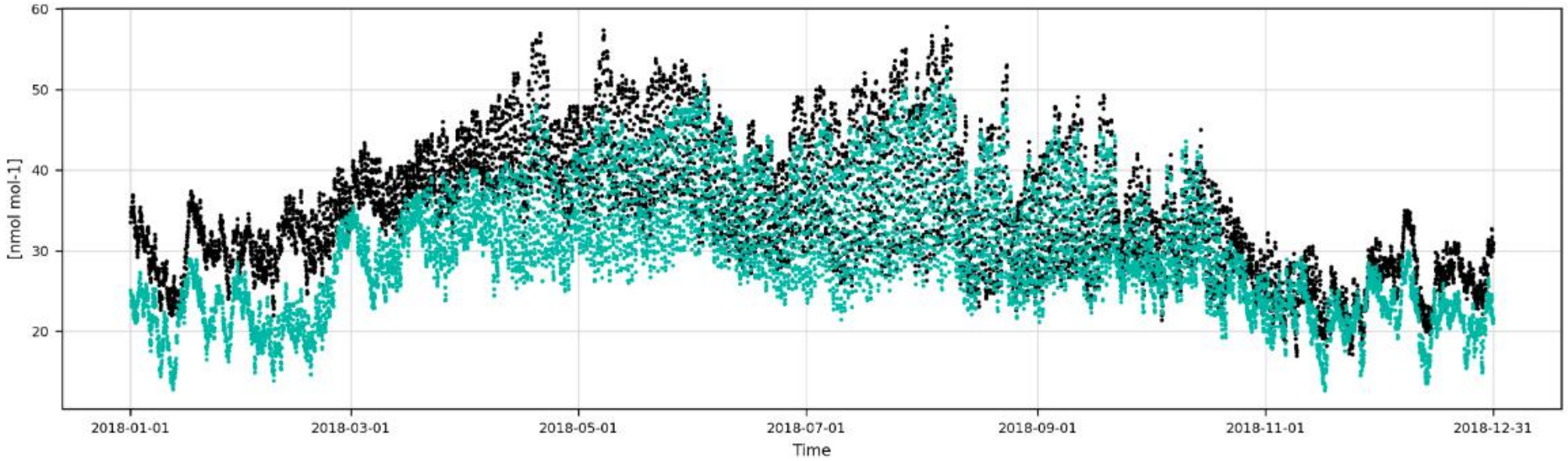
Set custom plot characteristics

If you want to customise a specific plot type (e.g. timeseries_smooth), so that it has different format than the default one, you can add a new key in the dictionary *settings/plot_characteristics_{mode}.json*.

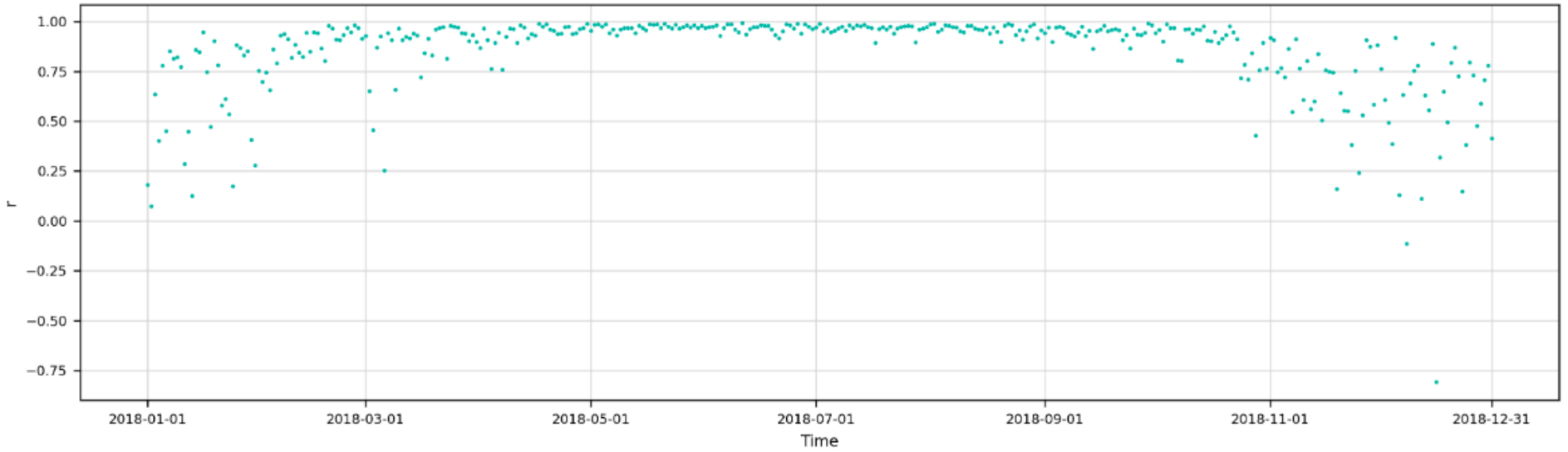
```
"timeseries": {"orientation": "landscape",
               "figure": {"ncols": 1, "nrows": 2},
               "grid": {"axis": "both", "color": "lightgrey", "alpha": 0.8},
               "page_title": {"t": "Timeseries", "fontsize": 15, "ha": "left", "x": 0.05, "y": 0.98},
               …},

"timeseries_smooth": {"orientation": "landscape",
                      "figure": {"ncols": 1, "nrows": 2},
                      "grid": {"axis": "both", "color": "lightgrey", "alpha": 0.8},
                      "page_title": {"t": "Smoothed timeseries", "fontsize": 15, "ha": "left", "x": 0.05, "y": 0.98},
                      …}
```

Statistical timeseries

timeseries



timeseries-r-daily



Statistical timeseries

Now we can create statistical timeseries by:

- **In the dashboard**: Select the chunking statistic and temporal resolution from the comboboxes in the timeseries plot.
- **In the offline reports and interactive mode**: Define your plot type by adding a dash after the timeseries word, then the statistic name followed by another dash and the temporal resolution (e.g. "timeseries-Mean-daily", "timeseries-r-monthly", "timeseries-RMSE-annual").


Q&A

Interactive mode

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Interactive mode

In the interactive mode, Providentia can be used a Python module and access its **backend functions to read and filter the data and make plots**.

If we add --interactive as a launch option in the command line, a Jupyter session will start:

```
./bin/providentia --interactive
```

This will create one file (*interactive.out*) containing the command to create an SSH tunnel on your local machine, which should be pasted into the terminal, and the path to open Jupyter in your browser.

Launching it from your own scripts

To use Providentia in your own scripts, you will need to append the path to your system's paths by:

```
import sys
sys.path.append("path/to/your/Providentia/directory")
```

To use the functions, you need to import Providentia as:

```
from providentia import Interactive
```

If you submit your job to slurm in order to use Providentia in parallel, you will need to set

--cpus-per-task = N, where N equals number of cores.

Reading from a configuration file

A class instance is created by the following line:

```
provi = Interactive(conf="interactive_template.conf")
```

where *provi* is the class instance, which can access all of the class variables and methods.

Only one section-subsection pair can be read and filtered at once. Where there is more than one, the specific pair wished to be read can be set:

```
provi = Interactive(conf="interactive_template.conf",
section="SECTIONNAME", subsection="SECTIONAME·SUBSECTIONNAME")
```

Printing configuration files

- Printing currently active configuration file:

```
provi.print_config()
```

- Printing any other existing configuration file using *conf* / *config:*

```
provi.print_config(conf="important.conf")
```

Overwriting configuration parameters

If wanting to overwrite any overwrite any arguments in the .conf file, directly in the script, each argument can be simply passed when initiating the class instance, e.g.:

```
provi = Interactive(conf="interactive_template.conf", network="EANET")
```

Accessing data in memory

- Accessing all variables in memory:

```
data = provi.get_data(format="xr")
```

Available formats: nc (netCDF), np (numpy) or xr (xarray)

- Accessing specific variables in memory:

```
var_data = provi.get_var(var="myvar")
```

Available formats: np (numpy) → Not passed as an argument

Note: If you have a unique network-component pair, you can just pass the variable name in var without specifying the network and species.

Applying filters

- To apply a filter not set in the configuration file:

```
provi.apply_filter(field, ...)
```

- To select the data for a specific station or multiple stations:

```
provi.select_station(station)
```

- To reset all filters (including those set in the configuration file):

```
provi.reset_filter()
```

- To keep configuration file filters and reset the ones added later:

```
provi.reset_filter(initialise=True)
```

Applying filters

The fields to filter by can be representativity fields, period fields or metadata fields.

- If the field is numeric, use *lower* and/or *upper*, e.g.:

```
provi.apply_filter(field, lower=28, upper=31)
```

- If the field is textual, use *keep* and/or *remove*, e.g.:

```
provi.apply_filter("country", keep="Spain")
```

```
provi.apply_filter("country", remove=["Spain", "France"])
```

- If the field is a representativity field (special case), use *limit*:

```
provi.apply_filter(rep_field, limit=20)
```

Calculating statistics

To calculate statistics, you should use the function:

```
stat_calc = provi.calculate_stat(stat, labela="OBS")
```

where *stat* is the statistic wished to be calculated, and *labela* is the name of the observations/experiment data. If wanting to calculate a bias statistic, you need to set *labelb:*

```
stat_calc = provi.calculate_stat(stat, labela="OBS", labelb="EMEP")
```

For bias statistics for which a subtraction is involved, it is always done as datasetb - dataseta.

If wanting to calculate statistics at each individual station you can by:

```
stat_calc = provi.calculate_stat(stat, labela="OBS", labelb="EMEP", per_station=True)
```

Plotting

You can use the plotting functions of Providentia with the function:

```
provi.make_plot(plot_type)
```

- To hide the legend:

```
provi.make_plot(plot_type, legend=False)
```

- To hide the observations label from the legend:

```
provi.make_plot(plot_type, set_obs_legend=False)
```

- To hide the colorbar:

```
provi.make_plot(plot_type, cb=False)
```

Plotting

- To limit which data is plotted:

```
provi.make_plot(plot_type, data_labels=["EMEP", "MONARCH"])
```

Note: These labels should be the dataset aliases, if set.

- To set the title, x or y labels:

```
provi.make_plot(plot_type, title="My custom title", xlabel="My custom xlabel", ylabel="My custom ylabel")
```

- To set the format:

```
provi.make_plot(plot_type, format={"figsize": [14,7], "xtick_params": {"labelsize": 22}})
```

Note: You can also set the format in *settings/plot_characteristics_interactive.json*

Plotting

- To set plot options:
  - Put them in a list:

```
provi.make_plot(plot_type, plot_options=['annotate', 'bias’])
```

  - Pass each of them as an individual argument:

```
provi.make_plot(plot_type, annotate=True, bias=True)
```

Plot options

	-[stat]	_bias	_annotate	_regression	_multispecies	_logx	_logy	_smooth	_hidedata	_domain
map	■	■	■							■
timeseries		■	■				■	■	■	
periodic	■	■	■				■			
periodic-violin			■				■			
distribution		■	■			■	■			
scatter			■	■		■	■		■	
heatmap	■	■	■		■					
table	■	■			■					
boxplot			■		■		■			
statsummary		■			■					
metadata										

■ Available

Unavailable

Saving plots and data

- To return the plot in memory instead of showing it:

```
fig = provi.make_plot(plot_type, return_plot=True)
```

- To save the plot:

```
provi.make_plot(plot_type, save="filename")
```

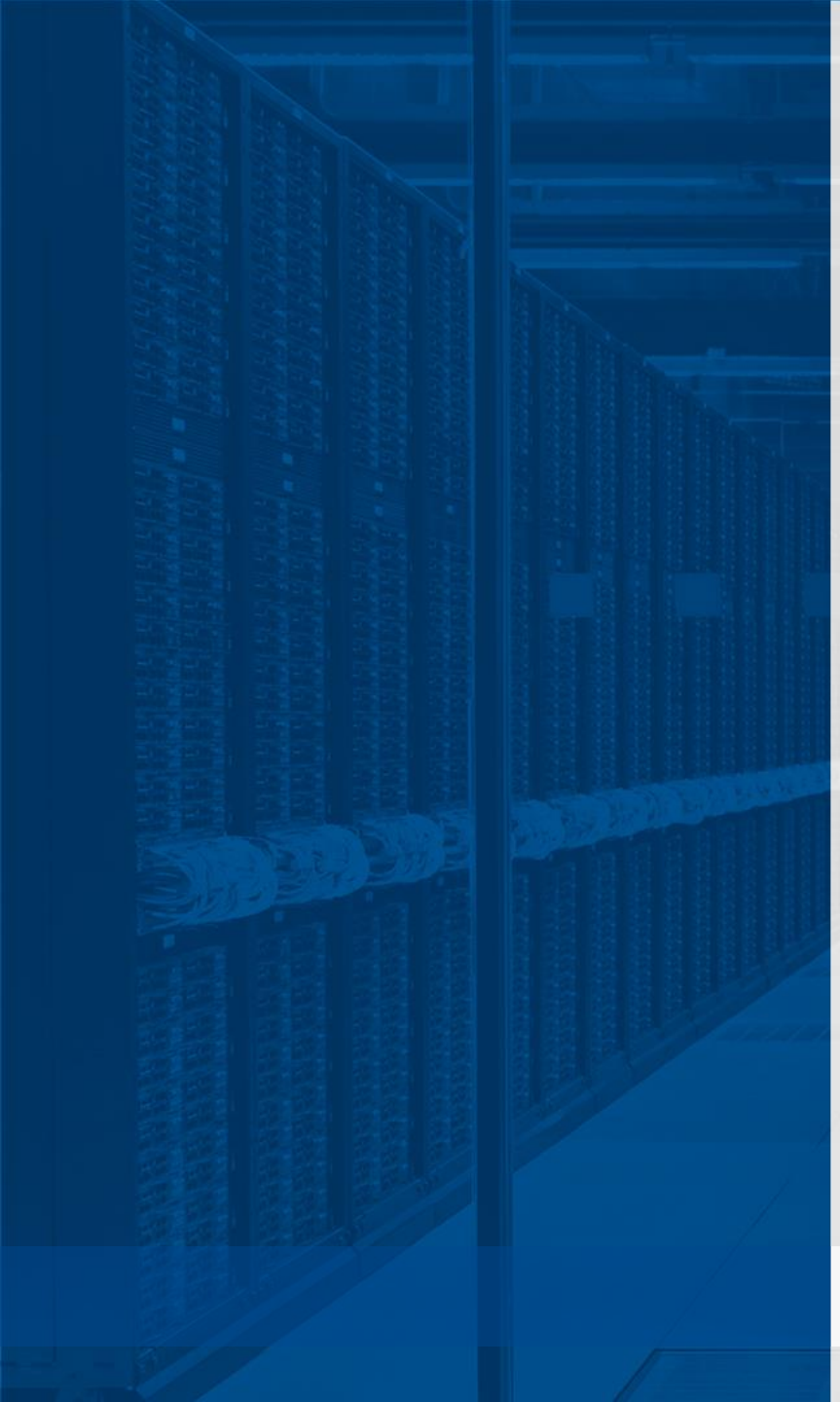
  Note: The plots will be saved inside the *plots* folder

- To save the data (with filters if any):

```
provi.save(format="nc", fname="path/to/save/data/filename")
```

- Available formats: nc (netCDF), np (numpy) or conf (Providentia .conf file)
- Note: If *fname* is not specified, the data will be saved in the *saved_data* folder


Q&A



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thank you for your attention!

More information at:
https://earth.bsc.es/gitlab/ac/Providentia

Join the #providentia Slack channel!

alba.vilanova@bsc.es | dene.bowdalo@bsc.es