Copernicus Climate Change Service

Seasonal Forecast verification tool for the C3S_512 contract

Issued by: BSC

Author: Jesús Peña-Izquierdo

Contract: Copernicus C3S_512

Date: 31/07/2020

IMPLEMENTED BY ECMWF

Contributors

BSC

Jesús Peña-Izquierdo

Table of Contents

1. Overview

A specific python-based software tool has been developed for the verification of the different seasonal forecast systems available in the Climate Data Store (CDS). This tool provides a complete set of functionalities ranging from direct downloading of selected datasets from the CDS to the computation and plotting of the specified metrics. All these functionalities are integrated in to a workflow that automatically identifies what operations are required and which ones can be skipped. It runs starting from a configuration file, which allows a high degree of parametrization, including all required dataset download parameters, computation specifications (e.g. chunks sizes on selected dimensions, number of cores, maximum allowed memory usage...), metrics parameters and plenty of plotting details. The tool has been developed in a modular fashion to allow easy debugging, modification and extension.

The software uses the xarray package to facilitate multidimensional array manipulation and it allows efficient parallel computing, by integrating the Dask library. For some of the verification metrics (fRPSS and fCRPSS), the R package SpecsVerification is used, which is a specific library for seasonal forecast verification, also used in the preoperational contract C3S_51_Lot3. This R package is wrapped using the python R2Py package.

A copy of the code runs at ECMWF virtual machines and is also available in the BSC Gitlab repository under the s-prototype branch[1].

2. Workflow and modules

In order to perform the verification of a seasonal forecast system, a reference dataset (usually from observations or a reanalysis) is also required in addition to the seasonal forecast dataset to compute the corresponding verification metrics. Thus, two datasets for the same variable, temporal period, spatial and temporal resolution are needed. Once the two datasets are preprocessed so they are completely equivalent (previous conditions are met), the computation of metrics can start and successively the generation of the corresponding figures. All these required tasks are performed by the seasonal forecast verification tool which comprises four main modules. Each module is responsible for one different task: 1. downloading, 2. transferring, 3. computing and 4. plotting. All the parameters that define these tasks are specified in the configuration file conf.yaml which includes a specific section for each of the modules. The main script, eqc.py, calls the 4 modules in consecutive order. However, the execution of any module can be manually enabled/disabled in the configuration file or automatically skipped in case executing a given task is not necessary (for example if the corresponding expected result already exists).

Each module expects specific inputs and provides specific outputs (figure 1). In this sense, the workflow is defined in a way that it tries to never stop, even in the case of missing inputs (i.e. missing files). If required files for a specific task are missing, the task will be skipped, an error log will

[1] https://earth.bsc.es/gitlab/external/c3s512-seasonal-forecast-eqc/-/tree/s-prototype

be generated and the next task/module will be launched. The same happens (without error) in case the expected results generated by a given tasks are found (because of a previous run); by default the workflow will skip this task to the next module. The tool will keep running tasks until completing all the pending tasks.

This procedure is very convenient when a set of different variables and/or systems are to be assessed. If in one case data is missing (e.g. files are missing for a specific variable), the tool will proceed with the following variable and/or system. In the same way, an uncompleted task of a given variable/system can be resumed easily launching a complete job including many tasks. Only the missing ones will be executed.

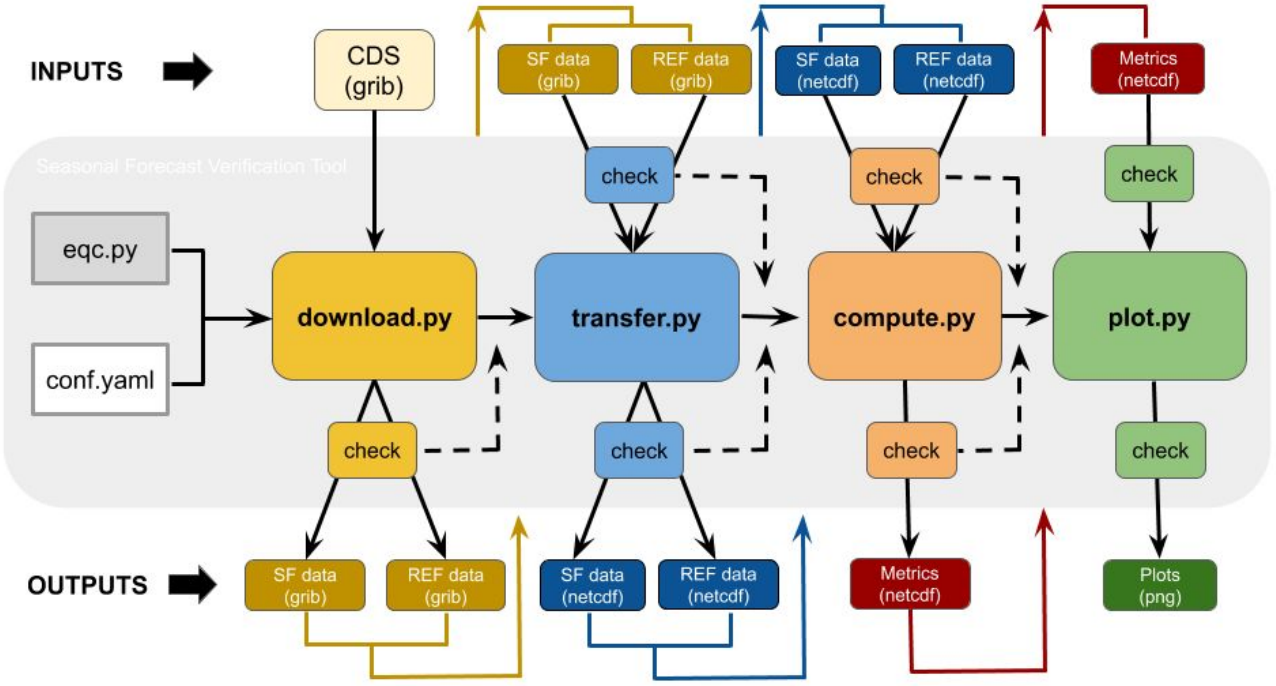

Figure 1. A schematic description of the seasonal forecast verification tool workflow. The main script (eqc.py) launches the tasks/modules specified in the configuration file (conf.yaml). The 4 different modules requires specific inputs and provides specific outputs which are the inputs of the successive tasks. The workflow is designed in a way that it tries to never stops until finish; if required files are missing or expected outputs are already available, the tool will skip the current task and launch the following module. Dashed lines represent these possible skip pathways.

An schematic description of this workflow is shown in figure 1 and briefly explain here:

- **Module download** is defined by script src/download.py. It downloads the specified datasets from the CDS in the original grib file format. Before sending the download request to the CDS, the script checks what is already downloaded and what is missing. The download task can be set so it only downloads the missing files or on the contrary re-download the dataset

even in case the corresponding files are already downloaded (and with problems for example). Check section 3.2 for details.

- **Module transfer** is defined by script src/transfer.py. It converts the original grib downloaded file to a more convenient and efficient netcdf file format. It also interpolates the reference dataset to the seasonal forecast grid. The use of netcdf greatly improves the performance allowing different types of chunking and a more efficient use of the memory. During C3S_512 contract, module download was rarely used since the datachecker had already downloaded most of the datasets. Then, the transfer module was in charge of extracting the required data from the much larger files downloaded by the datachecker in grib (located at /shared, these files include more products that the ones required for the verification) and convert them in to netcdf. Module transfer first checks that expected grib files are present in the corresponding source folder and then if the expected outputs in netcdf files are also already generated. The transfer of netcdf files can be set so if corresponding netcdf files are found, the transfer of files is skipped for those found. Check section 3.3 for more details.
- **Module compute** is defined by script src/compute.py. It computes the specified metrics (mean bias, mean correlation, fRPSS and fCRPSS) between the seasonal forecast system and the selected reference (for the moment reanalysis ERA5). Before computing the metrics, this module checks the consistency of nan's through the time series, the matching of the variable units between the seasonal forecast and the reference datasets (if there is not a match it allows a unit conversion) and finally prepare the two different multidimensional arrays to be comparable (seasonal forecast dataset has several dimensions not present in the reference: members, start months and leadtime months). For each variable and system, one only netcdf file is created including all the computed metrics. Analogously to all modules, module compute checks if input files in netcdf format are available and if so, checks if metrics are already computed by looking for the corresponding file. Check section 3.4 for more details.
- **Module plot** is defined by script src/plot.py. It plots all the previously computed metrics for all the possible combinations of metrics, start dates and leadtime months. Again, it first checks if the required file with the corresponding metrics is available, if so, it checks if expected plots are available. It can be set that only missing plots are generated or the generation of all plots should be generated no matter if they already exists. Check section 3.5 for more details.

3. Configuration file: conf.yaml

The seasonal forecast verification tool is controlled by the configuration file conf.yaml. In this section most important parameters of the configuration file will be described.

3.1 Dataset parameters

Section '**dataset_parameters'** (figure 2) of the configuration file specifies all the attributes needed to identify the corresponding datasets. This is the only section to be modified when a different

variable or system is to be evaluated. Values for most of these parameters should be taken from the CDS download form since they will be used by the cdsapi to download the data.

- **'dataset_sf'**: Dataset name of the seasonal forecast. Corresponds to the cdsapi field 'dataset'.
- **'variable'**: Name of the variable. Corresponds to the cdsapi field 'variable'.
- **'originating_centre'**: Institution of the seasonal forecast model system. Corresponds to the cdsapi field 'originating_centre'.
- **'system'**: Version of the seasonal forecast model system. Corresponds to the cdsapi field 'system'.
- **'years'**: A list of two elements defining the temporal verification period, the initial and final years, both included; e.g. [1993, 2016] or for one sole year [1993, 1993].
- **'start_months'**: A list defining the start months including the initial and final start months; e.g. [1, 12] for the all the months.
- **'leadtime_months'**: A list defining the leadtime months now including all the required elements; e.g. [1, 2, 3, 4, 5, 6]
- **'leadtime_periods'**: A list of lists defining forecast periods as a combination of several leadtime months. Each list within the list corresponds to a period. All the required leadtime months should be specified; e.g. [[2, 3, 4], [4, 5, 6]].
- **'levels'**: In case a dataset with pressure levels is indicated, a list of the different levels. It corresponds to the cdsapi field 'pressure_level'. This parameter is only used if the dataset has pressure levels. It is neglected otherwise.
- **'product_type_sf'**: It specifies the type of seasonal product (mean, maximum, minimum…). It corresponds to the cdsapi 'product_type'. The value by default is 'monthly_mean'.
- **'dataset_ref'**: Dataset name of the reference, for the moment ERA5. Corresponds to the cdsapi field 'dataset'. Note that 'dataset_sf' should agree with the one specified in 'dataset_ref', for example if the first one corresponds to a pressure level dataset, the second should be a pressure level dataset as well.
- **'variable_ref'**: Name of the variable in the reference dataset. Corresponds to the cdsapi field 'variable'. This parameter is optional. By default tt should be left empty what implies that the names of the variable in the seasonal forecast and in the reanalysis datasets are exactly the same. Otherwise, the variable name of the reference dataset should be specified here.
- **'reference_name'**: This parameter refers to the name of the reference. It is used as an identifier in some of the generated files or plots.
- **'product_type_ref'**: It specifies the type of reanalysis product. It corresponds to the cdsapi 'product_type'. For the case of ERA5 the default value is 'monthly_averaged_reanalysis'.

```
# Configuration file for Seasonal Forecast EQC

# Dataset parameters
#.....................
dataset_parameters:
  dataset_sf: seasonal-monthly-single-levels #seasonal-monthly-single-levels #seasonal-monthly-pressure-levels
  variable: 2m_temperature #temperature #u_component_of_wind #total_precipitation #10m_u_component_of_wind #2m_temperature
  originating_centre:  ecmwf # dwd #dwd #meteo_france #ecmwf #cmcc
  system: 5 #5 #7 #2 #3
  years: [1993, 2016] #[1993, 2016] # List (i.e. [1993, 2016] for the period 1993 to 2016 both included.
  start_months: [1, 12] #[1, 12] #Has to be a list (i.e. [1,12] for the whole year or [3,3] for just month 3)
  leadtime_months: [1, 2, 3, 4, 5, 6]
  leadtime_periods: [[2,3,4], [4,5,6]]
  levels: [850, 500, 200, 50] # Pressure levels to be downloaded. If single-level dataset is specified this parameters is

  product_type_sf: monthly_mean # Product type refers to individual members monthly means, monthly max, monthly min, month

  dataset_ref: reanalysis-era5-single-levels-monthly-means # reanalysis-era5-pressure-levels-monthly-means
  variable_ref: #mean_top_net_short_wave_radiation_flux # Leave empty to use same variable than for seasonal dataset. Othe
  reference_name: ERA5
  product_type_ref: monthly_averaged_reanalysis
```

Figure 2. Example of the 'dataset_parameters' section of the configuration file conf.yaml

3.2 Download parameters

Module **'download_parameters'** includes all the required parameters related with the download of the data from the CDS.

- **'execute'**: 'True' or 'False'. This parameter enables or disables the download of files from the CDS
- **'force_download'**: 'True' or 'False'. This parameter enables or disables the re-download of files. Value 'True' removes already downloaded files and re-download them.
- **'retry_download'**: 'True' or 'False'. This parameters enables or disables a download loop to avoid download timeouts. If a download fails, a new request is made until the download is successful. Recommended value is 'False' to avoid an infinite loop in case of an incorrect download.
- **'download_aggregation_sf'**: 'monthly' or 'start_month'. This parameter specifies the temporal aggregation of the downloaded seasonal forecast files. 'monthly_files' refers to 1 file per start month and year (recommended), 'start_month' refers to 1 file per start month for all years (not recommended.
- **'file_format_sf'**: 'grib' or 'netcdf'. This parameter specifies the file format of the downloaded files. 'grib' is recommended to avoid corrupted conversion from the CDS.
- **'download_aggregation_ref'**: 'monthly' or 'start_month'. This parameter specifies the temporal aggregation of the downloaded files of the reference. 'monthly_files' refers to 1 file per start month and year (recommended), 'start_month' refers to 1 file per start month for all years (not recommended.

```
# Downloading parameters
#.....................
download_parameters:
  execute: True
  force_download: False # Force the download even if data is already downloaded. False by default
  retry_download: False # Enable or disable downloading even if data is not able. This should be False by default
  download_aggregation_sf: monthly #monthly or start_month: 'monthly' for downloading monthly files (recommended). Otherwi
  file_format_sf: grib #grib or netcdf. This should be preferably grib
  download_aggregation_ref: monthly #monthly or single_file: 'monthly' for downloading monthly files (recommended). Otherw
  file_format_ref: grib #grib or netcdf. This should be preferably grib
```

Figure 3. Example of the 'download_parameters' section of the configuration file conf.yaml

3.3. Transfer parameters

Module **'transfer_parameters'** includes all the required parameters related with the extraction, conversion and transfer of files from the CDS downloaded files or from previously archived files. In general, this task is done from grib files to netcdf files.

It is also important to note that reference dataset will not be in general in the grid used by the seasonal forecast dataset. Thus, the reference fields should be interpolated from their original grid to the one of the seasonal forecast. This procedure is also done by this module. Furthermore, different seasonal forecast system may have the same spatial resolution but different grids. For this reason, for each of these seasonal forecast grid there should be an interpolated field of the reference dataset.

- **'execute'**: 'True' or 'False'. This parameter enables or disables the transfer of files from downloaded folder or previously downloaded archive.
- **'force_transfer'**: 'True' or 'False'. This parameter enables or disables the repetition of transfer of files. Value 'True' removes already transferred files and transfer them again.
- **'downloaded'**: 'True' or 'False'. This parameter specific if archive files are obtained through the download module ('True') or they were previously downloaded ('False') with a specific folder tree and file name convention. In case of False, transfer.py should be updated to include the corresponding structure of folders and file name convention. By default, value is 'False' and the convention specified corresponds to the one used in the datachecker archive.
- **'compression'**: 'True' or 'False'. This parameter indicates if netcdf transferred files are compressed. Compressed files are smaller but performance is low. Recommended value is 'False'.
- **'file_format_read'**: 'grib' or 'netcdf'. This parameters indicates the file format of the downloaded or archived files. Default is 'grib'.
- **'file_format_used'**: 'grib' or 'netcdf'. This parameters indicates the file format generated by the transfer. Default is 'netcdf'.
- **'data_aggregation_sf'**: 'monthly' or 'start_month'. It specifies how the seasonal forecast files are or will be stored when transferred; 'monthly' refers to 1 file per start month and year (recommended), 'start_month' refers to 1 file per start month for all years.

- **'data_aggregation_ref'**: 'monthly' or 'start_month'. It specifies how the files of the reference dataset are or will be stored when transferred; 'monthly' refers to 1 file per start month and year (recommended), 'start_month' refers to 1 file per start month for all years.

```
# Transfering parameters
#.....................
transfer_parameters:
  execute: True
  force_transfer: False # Force the data transfer from the /shared
  downloaded: False # Specify if transfer is done from downloaded files with download.py or from an alternative archive. If later case, 2 eas
  compression: False # If transferred data is to be compressed or not. For single_file data_aggregation it takes a lot to generate
  file_format_read: grib # File format of the file in the datachecker archive ['grib' / 'netcdf']
  file_format_used: netcdf # Specificy of using grib ['grib'] from the download or netcdf ['netcdf'] from the transfer
  data_aggregation_sf: monthly #monthly or start_month: 'monthly' for computing from multiple files. start_month for computing from one file
  data_aggregation_ref: monthly #monthly or single_file: 'monthly' for computing from multiple files. single_file for computing from one sing
```

Figure 4. Example of the 'transfer_parameters' section of the configuration file conf.yaml

3.4. Compute parameters

Module **'compute_parameters'** includes all the required parameters related with the computation of metrics. Currently 4 different metrics are computed; mean bias, mean correlation, fRPSS and fCRPSS. These metrics are computed for each start month (12) and each leadtime (6 +2). The workflow is such that metrics are successively computed for each start month including all leadtimes. Alll fields computed corresponding to a given start month are stored in one specific file. Thus, for a given variable 12 different files are generated, one per start month, including the 4 metrics with all the corresponding leadtimes.

Before the start of the computation, different procedures are performed to guarantee seasonal forecast and the reference dataset are comparable:

- Reference data do not present ensemble members, start months either leadtime months dimensions. During these preprocessing, reference fields are manipulated in order to have the corresponding and expected dimensions of the seasonal forecast.
- Units of the two datasets are checked; If units do not match an ERROR message appears and the computation cannot be done. If a unit conversion is possible, it should be indicated in an additional configuration file, src/units_conversion.yaml.
- Nan's are also checked. The numbers of nan's are expected to be constant for all start months, leadtime months and ensemble members. If not, a WARNING or an ERROR (depending on the case) will appear. Nan's can cause problems during the computation of different metrics, for this reason during these preprocessing nan's are filled with dummy values that are later masked with nan's in the last step when storing the results.

The following parameters can be adjusted in the 'compute_module'.

- **'execute'**: 'True' or 'False'. This parameter enables or disables the computation of metrics.
- **'recompute'**: 'True' or 'False'. This parameter enables or disables the recomputation of metrics. Value 'True' removes existing files with computed metrics, compute them again and save them.

- **'stop_different_number_nans'**: 'True' or 'False'. This parameter enables or disables the break of the execution in case a different number of nan's are present at different time steps or members or leadtimes. Default is 'False'.
- **'metrics'**:
    - **'bias'**: 'True' or 'False': This parameter enables or disables the computation of the metric mean bias.
    - **'correlation'**: 'True' or 'False': This parameter enables or disables the computation of the metric mean correlation.
    - **frpss**: 'True' or 'False': This parameter enables or disables the computation of the metric fRPSS.
    - **fcrpss**: 'True' or 'False': This parameter enables or disables the computation of the metric fcrpss.
- **'xarray_parameters'**:
    - **'chunks'**:
        - **'latitude'**: A number. This parameter indicates size of the chunks in the latitude dimension. This is a very important parameter since it greatly affects the performance; a large number leads to a higher performance but also to a greater use of the memory that can lead to exceed the available memory. Recommended value 100.
        - **'longitude'**: A number. This parameter indicates size of the chunks in the longitude dimension. This is a very important parameter since it greatly affects the performance; a large number leads to a higher performance but also to a greater use of the memory that can lead to exceed the available memory. Recommended value 200.
    - **'dask_distribution'**: 'True' or 'False'. This parameter specified how dask works under the hood of xarray. 'True' starts a dask.distributed server and 'False' a single machine scheduler. This may have a large impact on performance when using tens of cores. But configuration is not easy. Single machine scheduler works fine enough, recommendation is 'False'.
    - **'n_workers'**: A number. This is only used when 'dask_distributed' is 'True'.
    - **'thread_per_worker'**. A number. This is only used when 'dask_distributed' is 'True'.
    - **'n_cores'**: A number. This is only used when 'dask_distributed' is 'True'.
    - **'split_factor'**: A number. It is used in multiApply when computing fRPSS and fCRPSS to indirectly indicate the number of chunks. It does not seem to work properly. Recommended value is 1.
    - **'n_leadtimes'**: A number. It indicates the number of leadtimes used at the same time when computing fRPSS and fCRPSS. It does not improve performance since multiApply does not seem vectorize here. Recommended values is 1
    - **'memory_limit'**: A number. This parameter indicates the maximum number of RAM GB are available. A larger number than reality allows avoid constant warning. Recommended '20GB'.
    - **'processes'**: 'True' or 'False'. It indicates to dask what type of process is to be done. Recommended for array computations is 'False'.
    - **'silence_logs'**: 'True' or 'False'. It indicates the level of message the dask scheduler shows. It does not seem to work properly.

- **‘dashboard_address’**: A port number for the dask dashbord. Set to NONE to deactivate since it does not seem to work properly.
- **‘time_dimname’**: Name of the temporal dimension: ‘time’.
- **‘starttime_dimname’**: Name of the start month dimension: ‘start_time’.
- **‘leadtime_dimname’**: Name of the leadtime dimension: ‘step’.
- **‘member_dimname’**: Name of the ensemble members dimension: ‘number’.
- **‘lon_dimname’**: Name of the longitude dimension: ‘longitude’.
- **‘lat_dimname’**: Name of the latitude dimension: ‘latitude’.
- **‘pressure_dimname’**: Name of the pressure levels dimension: ‘isobaricInhPa’.

```
# Computing parameters
#.....................
compute_parameters:
  execute: True # Enable disable computing metrics. If previously computed metrics exist, computation is disabled.
  recompute: False # Allow recomputing metrics even if file with metrics is found
  stop_different_number_nans: False # Stop if different number of nan's are found in different files. If false, keep running and masks with r

  #Metrics
  metrics:
    bias: True
    correlation: True
    frpss: True
    fcrpss: True

  #Xarray parameters
  xarray_parameters:
    chunks:
       latitude: 100
       longitude: 200
    dask_distributed: False # Specificy whether to start a dask.distributed or the default single machine scheduler
    n_workers: 5
    threads_per_worker: 2
    n_cores: 10 # Number of cores used. This is parameter is shared by multiApply / Rpy2 to compute frpss and fcrpss
    split_factor: 1 # Coefficient for chunking in multiApply. Default and minimum: 1. Maximum: 'greatest' (works very bad). In theory the lar
    n_leadtimes: 1 # Specify how leadtimes are computed in each loop
    memory_limit: 20GB # Set this unrealistically large to avoid useless dask warnings
    processes: False # Threads activated. Recommended for array-like computations
    silence_logs: False # Whether to print dask logs. NOT WORKING
    dashboard_address: None # port for the dask dashboard. Set to None to deactivate it. NOT WORKING
    time_dimname: time
    starttime_dimname: start_time
    leadtime_dimname: step
    member_dimname: number
    lon_dimname: longitude
    lat_dimname: latitude
    pressure_dimname: isobaricInhPa
```

Figure 5. Example of the ‘compute_parameters’ section of the configuration file conf.yaml

3.5. Plot parameters

Module **‘plot_parameters’** includes all the required parameters related with the plotting of figures. For each variable, in general, 4 different metrics, 12 start months and 6 + 2 leadtimes leads to a total of 384 figures. All these figures are plot maps. The only difference between each figure is the field plotted. The following parameters can be set in the ‘plot_module’:

- **‘execute’**: ‘True’ or ‘False’. This parameter enables or disables the plotting of figures.
- **‘replot’**: ‘True’ or ‘False’. This parameter enables or disables the re-plotting of existing figures. Value ‘True’ leads to the removal of existing figures files and generation of new ones.

- **'bias' / 'correlation' / 'frpss' / 'fcrpss'**:
  - **'plot_figure'**: 'True' or 'False'. This parameter enables or disables the generation of plots for the corresponding metric
  - **'metric_longname'**: This is the name of the metric used in the title of the corresponding figure. Should be formatted to be read properly.
  - **'metric_shortname'**: This is the name of the metric to be used in the corresponding figure file name. This should be short and without containing spaces.
  - **'cbar_nlevels'**: Number of levels in the colorbar/colormap. Recommended is 11.
  - ' **cbar_minmax'**: This list with two numbers indicates the minimum and maximum of the colorbar [min, max]. If a number is specified it will appear in the corresponding figure.If None is specified it will use the percentile indicated in parameter **'cbar_pminmax'**. Depending on the metric, different values should be used. bias => [None, None], correlation => [-1, -1], frpss => [0, 1], fcrpss => [0, 1].
  - **'cbar_pminmax'**: A list with two numbers (0<numbers<1) specifying the percentiles used in the minimum and maximum of the colorbar [pmin, pmax]. These percentiles are only used when None is correspondingly indicated in 'cbar_minmax'. For the bias, recommended is [0.02, 0.98]. For the other metrics is [None, None]
  - **'cmap_type'**: 'sequential' or 'diverging'. This parameter specifies if the colormap used in the corresponding metric should be 'sequential' or 'diverging'.
  - **'cmap_double_white'**: 'True' or 'False'. This parameters specifies if the a diverging colormap uses 2 white intervals (around 0) or 1 white interval (centred in 0). True recommended for correlation. False recommended for bias
  - **'cbar_ndigits'**: A number indicating the number of digits in the colorbar axis. Recommended 2.
  - **'cbar_extend'**: 'neither', 'min', 'max' or 'both'. This parameters specifies if the colorbar should indicate smaller/larger values exist show a triangle in the lower/upper edge of the colorbar. For 'neither' a flat edge is shown in both sides. Recommended is bias => 'both', correlation => 'neither', frpss => 'min', fcrpss => 'min'.
  - **'conf_level'**: A number (0<number<1) indicating the confidence interval used in the statistical significance test (t-test). It is only used for the correlation plots. Recommended is 0.05 (i.e. 95% confidence interval).
  - **'hatches'**: A list with a number of punctuation marks used for plotting the statistical significant field. This is only used for the correlation plots. Recommended ['....']. Any punctuation mark can be used. More elements implies smaller/more detailed fields.
  - **'lines_colors'**: A list of strings or list of RGB numbers indicating the color of the lines in plots with latitudinal zonal means. Recommended ['k', 'b', 'c', 'y', [1,0.5,0], [1,0.5,0.7],'r']
  - **'line_style'**: A string indicating the style of the lines in time series plots. Recommended '-'
  - **'lines_widths'**: A number indicating the width of the lines for zonal means of latitudinal regions in time series plots. Recommended 2
  - **'line_width'**: A number indicating the width for line of global mean in time series plots. Recommended 4

- ‘cmap_double_white’: ‘True’ or ‘False’. This parameters indicates if a double white interval around 0 is required in the colorbar/colormap. Recommended ‘False’.

Figure 6a. Example of the ‘plot_parameters’ section of the configuration file conf.yaml. It continues in fig. 6b

```yaml
frpss:
  plot_figure: True
  metric_longname: fRPSS
  metric_shortname: frpss
  cbar_nlevels: 11
  cbar_minmax: [0, 1]
  cbar_pminmax: [None, None] # vmin and vmax as a percentile. None will use the one define in cbar_minmax
  cmap_type: 'sequential' # Sequential or diverging (0 centred) colorbar
  cbar_ndigits: 2 # number of non-zero digits in the vmin, vmax
  cbar_extend: min
fcrpss:
  plot_figure: True
  metric_longname: fCRPSS
  metric_shortname: fcrpss
  cbar_nlevels: 11
  cbar_minmax: [0, 1]
  cbar_pminmax: [None, None] # vmin and vmax as a percentile. None will use the one define in cbar_minmax
  cmap_type: 'sequential' # Sequential or diverging (0 centred) colorbar
  cbar_ndigits: 2 # number of non-zero digits in the vmin, vmax
  cbar_extend: min

# General plot parameters
projection: robinson # rectangular or robinson
colorbar_fraction: 0.024
colorbar_pad: 0.02
figure_size: [12, 12]
figure_dpi: 300
file_format: png
cmap:
  sequential: OrRd
  diverging: PuOr_r #RdBu_r
axis_fontsize: 13
title_fontsize: 15
text_fontsize_F: 13 # For the footnote
text_fontsize_D: 10 # For the generation date
xticks: [-150, 150, 30] # Parameters to create xticks using np.arange(xI,xF,dx)
yticks: [-60, 60, 30] # Parameters to create xticks using np.arange(xI,xF,dx)
```

Figure 6b. Example of the ‘plot_parameters’ section of the configuration file conf.yaml.

- **‘projection’**: ‘rectangular’ or ‘robinson’. This parameter specifies the map projection used in the plots. Recommended is ‘robinson’.
- **‘colorbar_fraction’**: A number (0<number<1). This parameter indicates the width of the colorbar. Recommended 0.024.
- **‘colorbar_pad’**: A number (0<number<1). This parameter indicates the distance between the colorbar and the axes. Recommended 0.02.
- **‘figure_size’**: A list with two numbers [dx, dy]. This parameter indicates the size of the figure. Recommended [12, 12].
- **‘figure_dpi’**: A number. This parameter indicates the resolution of printed/saved figure in dots per inch (dpi). Recommended 300.

- **'file_format'**: A string indicating the file format of the generated figures. Recommended is 'png'.
- **'cmap'**:
  - **'sequential'**: A string indicating the colormap used for sequential fields. Recommended is 'OrRd'.
  - **'diverging'**: A string indicating the colormap used for diverging fields. Recommended is 'PuOr_r'.
- **'nans_color'**: A string. This indicates the color used for nan's. Recommended 'lightgrey'. If empty color is white.
- **'axis_fontsize'**: A number indicating the size of the axis fonts
- **'title_fontsize'**: A number indicating the size of the title font.
- **'text_fontsize_F'**: A number indicating the size of the footnote font.
- **'text_fontsize_D'**: A number indicating the size of the generation date font.
- **'xticks'**: A list with three numbers defining the location of the ticks in the x-axis (longitude). Recommended is [-150, 150, 30] => 150ºW, 120ºW,...0º,...120ºE, 150ºE
- **'yticks'**: A list with three numbers defining the location of the ticks in the y-axis (latitude). Recommended is [-60, 60, 30] => 60ºS, 30ºS, 0º, 30ºN, 60ºN.

3.6. Folders parameters

Module **'folder_parameters'** specifies all the required folder paths. The following parameters can be set in the 'plot_module':

- **'base_dataOUT_path'**: This parameter specifies the base path of the folder that contains the data generated by the transfer module. Currently it is '/data/cds_downloads'.
- **'base_dataIN_path'**: This parameter specifies the base path of the folder that contains the data downloaded by the download module or previously downloaded and archived. Currently it is '/shared/cds_downloads/' for the datachecker downloads or '/data/cds_downloads_grib/' for the download module.
- **'logs_path'**: This parameter indicates where the error logs are stored. They are currently stored in '/data/suso/LOGS'. This should be modified by each user.
- **'figures_path'**: This parameter specifies where the generated plots are stored. They are currently stored at '/shared/plots_scientific_assessment'
- **'metrics_data_path'**: This parameter specifies where files with computed metrics are stored. They are currently stored at '/shared/netcdfs_scientific_assessment'.

```
# Working folders
#.................
folders:
  base_dataOUT_path: /data/cds_downloads/ #/data/cds_downloads_nc/ #Folder where the data will be stored in netcdf after Data Transfer.
  base_dataIN_path: /data/cds_downloads_grib/ #/shared/cds_downloads/ #/data/cds_downloads_grib/ #Folder where to download from the CDS or wh
  logs_path: /data/suso/LOGS
  figures_path: /shared/plots_scientific_assessment/seasonal
  metrics_data_path: /shared/netcdfs_scientific_assessment/seasonal
```

Figure 7. Example of the 'folders_parameters' section of the configuration file conf.yaml.

4. Additional tool files

In addition to the main module scripts described above, there are other several scripts containing relevant functions or necessary information. A short description is explained below:

- src/tools.py: This script includes several generic functions used in the different main modules.
- src/grib_conventions.yaml: This configuration file contains different grib file key required for different types of variables
- src/unit_conversions.yaml: This configuration file contains different unit conversions used to convert the reference dataset to the seasonal forecast dataset units.
- src/naming_convention.yaml: This configuration file contains different name conventions used in the plots and file names.
- src/plot_element_locations.yaml: This configuration file contains different parameters for correctly location text and footnote in the figures.
- src/varsys2assess.yaml: This configuration file allows to run the verification tool for more than one variable. Lists of different 'variable', 'originating_centre' and 'system' can be provided so the execution of the tool is looped including all the possible combinations.
- src/packages_list.txt: This text file includes all the python packages versions required to replicate a working conda environment simply following the instruction explained in section 6.

5. How to run the tool.

The main script of the verification tool is eqc.py. This script can be converted to executable running the following command.

```
$ chmod +x eqc.py
```

Once it is executable, the tool can be run simply with:

```
$ ./eqc.py
```

The main script will automatically load the configuration file conf.yaml and all the corresponding parameters. The execution of the tool will provide a detailed log of the ongoing tasks. Also, if any unexpected error appears, an error log will be generated with an unequivocal file name and stored at the path specified in the 'logs_path' parameter of the configuration file.

Additionally, as briefly explained in section 4., the additional file varsys2assess.yaml can be passed to eqc.py in order to allow the computation of multiple combinations of 'variable' + 'originating_centre' + 'system'. Different values for 'variable', 'originating_centre' and 'system' can

be provided as lists so the execution of the tool is looped including all the possible combinations. This is run executing:

```
$ ./eqc.py varsys2assess.yaml
```

Finally, to run the tool in the background when running the tool in a remote machine (allowing disconnection from the server) command nohup can be used and a log file will be stored in the specified log file. For example:

```
$ nohup ./eqc.py &>/data/suso/LOGS/log_ecmwfs5_2m_temp.log &
```

or

```
$ nohup ./eqc.py varsys2assess.yaml &>/data/suso/LOGS/log_ecmwfs5_dwd2_wind_vars.log &
```

6. Installation

The seasonal forecast verification tool can be installed by cloning the repository from the BSC gitlab.

```
$ git clone https://earth.bsc.es/gitlab/external/c3s512-seasonal-forecast-eqc.git
```

A complete list of the required packages can be found at src/packages_list.txt. In order to install all these python packages using anaconda create a new python 3 environment (called for example 'eqc') running the following command.

```
$ conda create --name eqc --file ./src/packages_list.txt
```

Additional packages may require the installation via pip. For example the cdsapi and the cds downloader version developed for the C3S_512 (which allows monitoring the download requests statistics).

```
$ pip install cdsapi
```

and

```
$ pip install C3S512
```

R software also needs to be installed together with two specific R packages; SpecsVerification and multiApply. To install these packages, open R in a terminal and simply run:

```
> install.packages("SpecsVerification")
```

and

> *install*.packages("*multiApply*")

7. Possible future improvements

Some possible future improvements are listed here:

- The module donwload.py works fine, however the integration with the transfer.py has not been tested intensively since until now data used has been from the datachecker archive, so download.py has been very little used. A workflow including download + transfer modules may easily require some debugging.
- Dask library is a very powerful package for multi-core parallel computation. The current working implementation is using a single-machine (multi-core) scheduler which works fine. However Dask allows a cluster scheduler for supercomputer architectures. This more complicated type of process are already implemented in the verification tool but it required a profound investigation to work properly.
- In order to improve performance, the use of zarr file format instead of netcdf appears to provide very significant benefits in terms of computing performance (between x2 and x5 increase) and in terms of storing space (a factor of 3 reduction).