Using the ECMWF seasonal verification suite

Dave MacLeod, University of Oxford

23 March 2016

The seasonal verification suite is at its heart a set of Fortran scripts designed to carry out analysis of ensemble seasonal hindcasts at ECMWF. It is designed in such a way to be automatic, such that all tasks from downloading data from MARS, analysing data and archiving results are done automatically.

The suite is controlled by the SMS/CDP job management system. Each verification job (which can comprise multiple experiments, variables, lead time targets etc.) is controlled by a single definition file. In normal operations this is the only file which should be modified.

The suite also has plotting routines, however these are not set up for external users. Instead I have a set of NCL scripts and functions which plot the data output by the verification suite.

All the scripts from the verification suite and the external NCL plotting routines are available online, at my gitHub: INSERT LINK HERE.

In addition there are some helper bash scripts which make efficient various steps in the workflow. This document describes to some level of detail:

- Basic workflow
- The verification suite

- Plotting scripts

We start with a getting started section, follow this to get started.

Contents

Chapter 1

Getting started

First one needs to set up CDP on ecgate if you haven't done this before. This is necessary to submit jobs. Unfortunately CDP is being retired and details for this are no longer available on the website. They are however copied into the document 'ECMWF_User_guide.pdf'. Follow this along (i.e. 'Getting started with CDP/XCDP') and get to the point where you can sucesesfully log in to cdp and submit a .def file.

Next thing is to copy all the files on gitHub in the folder **ecmwfVerificationScripts** to somewhere on ecgate (e.g. your home directory).

Once this is done, you can submit the example suite (scores_example.def). This is done with the following commands (after navigating to the ecmwfVerificationScripts directory):

```
CDP
ecgate
play scores_example.def
begin scores_example
```

Note that ecgate is my alias for logging to the server within CDP, suitename.def is the name of the definition file, and the second suitename (i.e. used with begin) is the name of the suite defined within the .def file. These can in theory be different but keep these the same for clarity - checking

you have done this is always a good first check if you get errors in job submission!

Alternatively an easier way is to use the automatic bash script on gitHub: subSuite.sh.

```
bash subSuite.sh scores_example.def
```

Which will log into CDP, submit the suite automatically and log out.

There is also resubSuite.sh which works in the same way, for a suite which is already running and you want to resubmit.

N.B. for these two bash scripts ensure the CDP alias (i.e. ecgate) is correct for you.

The example suite scores_example.def does a basic validation of 2mT for System 4. It should now be submitted, you can check the progress in XCDP (you might have to enable viewing of this specific suite by right clicking on 'ecgate', selecting scores_example.def from the list and turning it on.

Now this is done, have a look at the rest of the document for more detail.

Chapter 2

Workflow

A schematic of the workflow is shown in figure 2.1. This describes the basic function of the suite. First is the definition file, which the user should modify to run the desired verification on their hindcast. This is described in more detail below.

This def file is described in more detail, and is (in theory) the only part of the verification suite that you should need to interact with in normal operations. Once submitted to CDP the suite runs automatically in several stages (you can see these in XCDP by middle-clicking on the suite name).

Firstly definitions are made, this is quite a quick step. Then, the suite downloads the necessary monthly mean data from MARS to SCRATCH on ecgate. This can take a long time, depending on the number of variables/ensemble size.

After this, the suite calculates anomalies (cal_anom.sms).

Once this is done, various families of scripts are called, depending on which options are turned on in the .def file. Note that I have never used some of these - those described in this document should work, others probably do work but there is no guarantee.

Each family in turn will perform analysis on the monthly mean grib files, calculating the relevant scores. Once finished, these are archived in a .tar file and moved to ECFS.

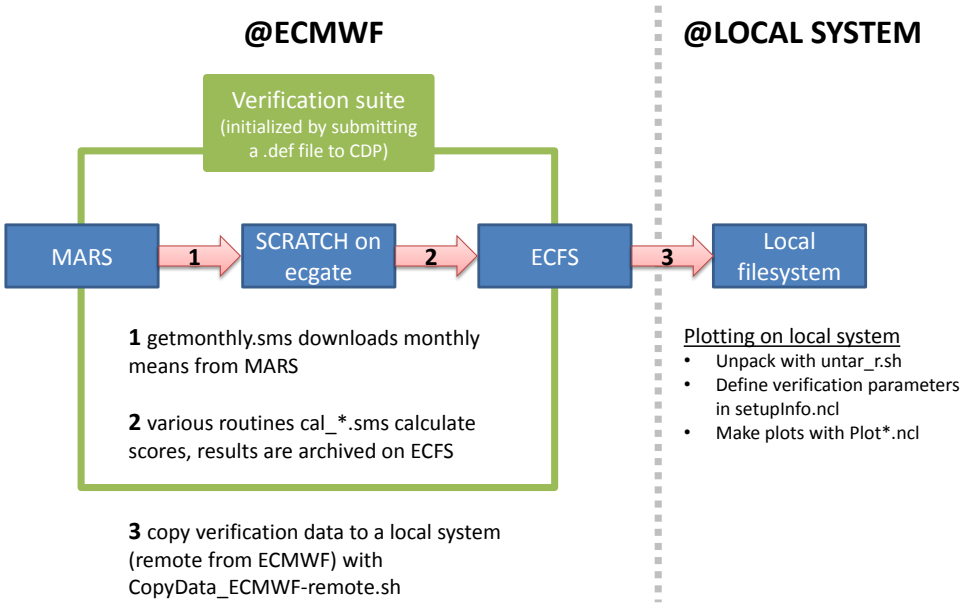

Figure 2.1: Data flow in the verification suite and associated plotting routines.

Once the suite is done with one set of scores, it moves on to the next automatically. Once finished, it pauses at the final cleaning task. If you are fully done with the suite, then you can run this task to clean up the working directory etc (N.B. The clean task does not affect the scores moved to ECFS; these are safe).

Following this, data can be downloaded to a remote system from ECMWF, where it can be processed. Processing and plotting scripts are available on gitHub inside /remotePlottingScripts and described in the relevant section below.

Chapter 3

The verification suite

There is a lot going on in the suite and all won't be described here - only the relevant parts.

3.1 The definition file

This is the key file to modify, thankfully it has lots of nice comments, so below are some pointers.

Paths

At the start are various paths and general definitions. Those which need to be changed are:

- SUITE_NAME, change this to the same thing as the .def file name (without the extension)
- VSUITE_LOC, change this to the location of the verification Suite

Aside from this you shouldn't need to modify anything in this first section

Experiment definition

This contains details about the experiments you want to look at. These are identified primarily with their 4 letter experiment id, and the type.

Note that if you want to analyse multiple experiments the arguments must be contained within quotation marks for each, e.g.:

```
edit EXPVER         "0001 b16r"
edit CLASS         "od uk"
```

This will analyse two experiments, 0001/od (System 4) and then the experiment b16r which was created by a uk user.

Note that all variables in this section follow the quotation mark syntax, except for MULTI and FCLENMON which have a single unquoted value for all.

Dates

Self-explantory; START_MONTH/DAY/TIME can have multiple values contained within the quotations to look at multiple start dates separately, e.g.:

```
edit START\_MONTH       "05 11"
edit START\_DAY         "01 01"
edit START\_TIME        "00 00"
```

will analyse experiments initialized 00H on 1 May and November.

Variables

Here you choose which variables to look at. One argument in each quoted array for each variable. Note that TLST and FLST are always 128 and 0

respectively, except for precipitation (PLST=228), which should take 172 and 1.

Parameters for diagnostics

This short section tells the suite which targets to calculate scores for (i.e. month 1, month 1-3 average, month 2-4 average). It uses 1-indexing, such that

```
edit SDIA     "1 2"
edit EDIA          "1 4"
```

will look at the first month and month 2-4. Following the definitions above in "Dates", this corresponds to November/DJF and May/JJA separately.

Regions available

The important parameter here is RDIA, which is a list of the regions to study, these are defined in the config.h file. Note that any which end with a G correspond to the Giorgi & Francisco regions.

These regional scores are active in cal_rel_reg & cal_acc_reg.

Suite definition

Following this setup comes the list of switches which control which scores you want. This is nicely commented, so look in the example def file for more details.

Finally comes the beginning of the suite, which is written in the SMS language, defining which tasks run and with what dependencies. This structure corresponds to what you see in XCDP when the suite is running.

3.2 Suite internals

Hopefully it isn't necessary to ever modify the internals of the suite, but a few comments here just to orientate you. These are all contained in verificationSuite/scores.

Various definitions are given in /include/config.h. Included here are the various regional definitions, these can be modified if you like. Also in /include are *get_exp* and *get_ref*; these download the data from MARS to the work directory and are called by *getmonthly.sms*.

Once data is downloaded, scores are run separately by various scripts. The format here is that the main CDP script is in the root scores folder with an extension .sms, which copies and compiles the Fortran file from /scores/bin to the work directory on SCRATCH (e.g. *cal_bias.sms* runs and calls and compiles */bin/cal_bias.f* in the work directory).

Once each task is complete, it archives the data (into a .tar file) and copies to ECFS. The final path is set by the variables in the .def file, and is: ${ECDISK}/data/${ORIGIN}/${CLASS}/${EXPVER}/s${SYSTEM}/m${METHOD}

A selection of tasks are listed below, along with the directory within the ECFS directory the data is stored. Note that other tasks are available, but these have not been tested for external users.

- cal_bias: calculates bias, climatology, interannual variation and ratio of model to reference standard deviation (output in /clim)
- cal_anom.sms: calculates anomalies for the model and reference relative to the mean (output in /anom)
- cal_acc_map: calculates maps of anomaly correlation, mean square skill score, ratio of spread to error and normalised spread (output in /acc)
- cal_acc_reg: as cal_acc_map, but calculates the value across the regions defined in RDIA (output in /acc)
- cal_rel_map: calculates maps of brier score, reliability skill score, roc, sharpness and ignorance (output in /rel)

- cal_rel_reg: as cal_rel_map. Also calculates statistics to produce reliability diagrams
- cal_ts.sms: calculate timeseries for predefined metrics (details in .def comments and config.h)

Chapter 4

Plotting

Now the suite is hopefully finished, you can download the data to your local location and plot it.

4.1 Downloading and unpacking the data

There is a script which will download the data for an experiment. The script is on gitHub as ecmwfVerificationScripts/copyData_ECFS-remote.sh. Some parameters will need to be changed at the top.

This is set up to replicate the ECFS verification directory structure on copy to SCRATCH, before copying to a remote location. There is nothing tricky going on with the script - it exists because there is no remote interaction directly with ECFS, everything must be copied first to SCRATCH.

Once everything is copied to your local location you first need to unpack it before doing anything. This can be done easily with the script /remotePlottingScripts/untar_r.sh.

Again nothing special, it just unpacks tar files recursively in the verification directory. To use this you will need to copy it to $HOME/bin & set permissions so it is open to execution. Then you just need to go to the downloaded directory and run

`untar_r.sh`

4.2 Setting up plotting

Now plotting can begin. There is a lot to plot, so it is set up in such a way that all the general information is in setupInfo.ncl, and the plotting scripts named *Plot** plot individual scores. Only setupInfo.ncl necessarily needs to be modified to fit your experiment, with details of targets, variables etc. Also need to change the directories!

In individual plotting scripts the loops can be modified to only output certain plots, or change resources etc. if you don't like the look of the plots.

4.3 Making the plots

Functions of the plotting scripts are listed below:

- PlotClimatologyMaps.ncl : plot maps created by cal_bias
- PlotClimatologyMapComparison.ncl : plots cal_bias output with reference to a control experiment
- PlotDeterministicMaps.ncl : plot maps created by cal_acc_map
- PlotDeterministicMapComparison.ncl : plots cal_acc_map output with reference to a control experiment
- PlotDeterministicScoresCollated.ncl : plots cal_acc_reg output
- PlotProbabilisticMaps.ncl : plot maps created by cal_rel_map
- PlotProbabilisticMapComparison.ncl : plots cal_rel_map output with reference to a control experiment
- PlotProbabilisticScoresCollated.ncl : plots cal_rel_reg output
- PlotReliabilityDiagrams.ncl : plots reliability diagrams, created by cal_rel_reg

- PlotReliabilityMap.ncl : plots reliability category maps based on Giorgi regions. Note that this relies on you running the verification for all the Giorgi regions.

These plotting scripts rely to some extent on user-created functions, contained in /plotScores/functions. These are relatively well commented/simple and shouldn't need modification.