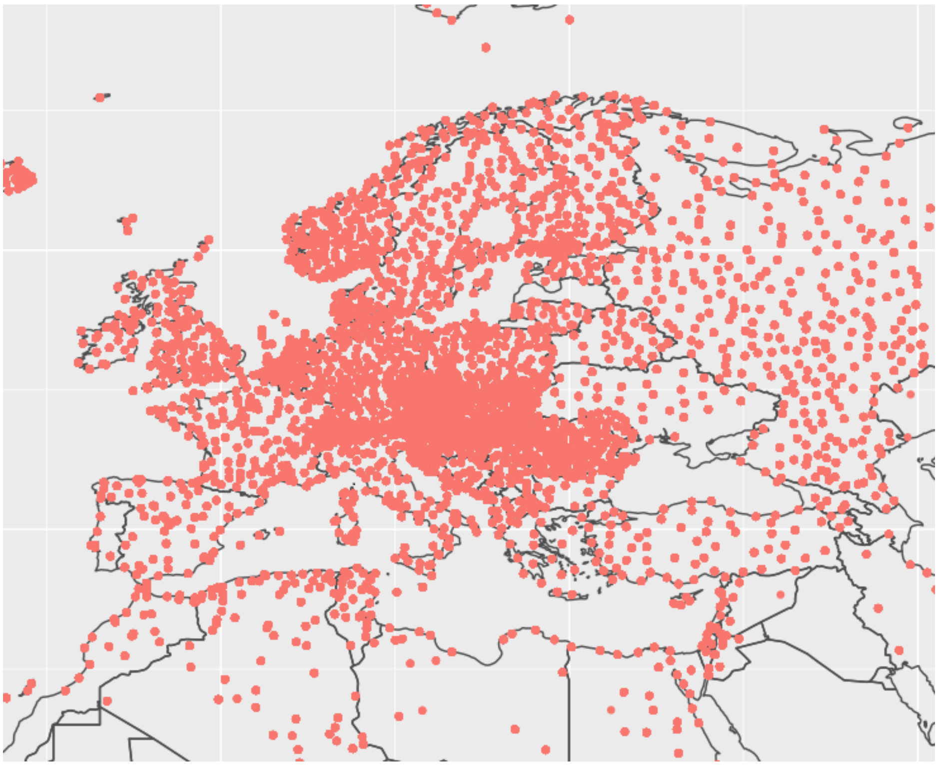
HarpIO for OBSOUL format used by OPLACE



Report from RC LACE stay in Prague, 06.03 - 31.03.2023
Martin Petraš, SHMU, martin.petras@shmu.sk
in collaboration with Alena Trojáková, CHMI

1 Introduction

Verification is an important part of research and operational numerical weather prediction (NWP). In the ACCORD consortium, the HARP software is used for verification and the aim of this work is to extend the support of input formats to the OBSOUL format used by the observation preprocessing system for LACE (OPLACE).

There are two obsoul data formats available in OPLACE system:

- obsoul_1_xxxxxx_hu_YYYYMMDDHH - GTS surface data
- obsoul_1_xxxxxy_[hu,cz,sk,si,ro,pl,at,cr]_YYYYMMDDHH - national surface data

The first version of HarpIO support for OBSOUL reading was developed by Martin Petras in 2022. This version contains read_obosul functions that can only read local (national) obsoul files separately. Moreover there is some difference between GTS data and national data. I'll describe a few.

- GTS data contains SYNOP data with station IDs having only integers. There are also SHIP data in GTS obsoul files which can have partially or fully string IDs. Since they have the same obstype = 1, some filtering is necessary to be applied.
- national data have usually less parameters (T2m,wind,RH2m...) while GTS can have more (up to 6 parameters with not predefined order)
- national data have specific station identifications with 2 character prefix for each country (CZ,HU, ...).

Since the national and GTS obsouls are usually being merged, the old **read_obsoul.R** function needs to be updated. These changes have been implemented into the code over the past few months. During ACCORD scientific visit in Prague, assisted by Alena Trojakova, updated code was tested. In order to ensure that all data are read and written from obsoul into SQlite properly. Furthermore, we carried out some experiments to compare HARP with a local verification tool VERAL used at CHMI.

2 Used configurations

Verification is performed using two verification tools, Harp and VERAL. The observation data was derived from two sources: obsoul from OPLACE and vobs from ECMWF observation database. We considered only two stations in experiments to simplify the validation approach: **PRAHA-RUZYNE** with SID number **11518**, and station **PRAHA-LIBUS**, with SID number **11520**. For forecast data, two deterministic models have been used: **ALADIN-CY43** *(2.3km)* and **ALADIN-CY46** *(2.3km)* (in graph named Dakw).

2.1 Harp

We are using the development version of HarpIO package, a version based on andrew-MET/harpIO develop version since this package contains the first version of changes that supports reading of obosul files. The installation of this version is as follows:

```
remotes::install_github("meteorolog90/harp-develop", "develop")
```

If you don't have *remotes* package installed, make sure you install it:

```
install.packages("remotes")
```

For a better user experience, it is recommended to install the *renv* package [1]. It allows you and your colleagues to work in a separate, reproducible environment without any hassle. You can find more information in Appendix A.

2.2 VERAL

The CHMI operates with the VERAL verification package. It supports point verification and elementary spatial analysis. Over the domain of the model, VERAL calculates standard deviation, mean error (bias), and root mean square error. VERAL scores were used for comparison with Harp scores. As a result of this comparison, we were able to solve some issues.

3 Second update of HarpIO for OBSOUL

3.1 Adding new parameters

A total of six variables were defined in the obsoul files, where varno represents the variable number to identify observed parameter (Table 1). There are several new parameters in the updated version (Table 2). In order for harp to read the new parameters, they must be defined in **harp_params.R**. The new parameters are following: 6h and 1h precipitation accumulation, minimum temperature (over the last 12h) reported at 06 UTC and maximum temperature (over the last 12h) to reported at 18 UTC and snow depth.

varno	name	harp name
1	mean sea level pressure	Pmsl
39	2m temperature	T2m
58	relative humidity	RH2m
7	specific humidity 2m	q2m
41	wind speed&direction	S10m,D10m
91	cloudiness	CCtot

Table 1: Variable names and corresponding numbers (*varno*) in obsoul files

varno	name	harp name
79	1h precipitation accumulation	AccPcp1h
80	6h precipitation accumulation	AccPcp6h
81	minimum temperature	Tmin
82	maximum temperature	Tmax
92	snow depth	Snow

Table 2: Added new variables with corresponding numbers

Figure 1. shows how to add new parameters to Harp interfaces. The process would involve creating a list for a specified format, in this case for obsoul, where you would define:

- **name**: for obsoul *varno*
- **units**
- **harp_name**: Tmin, Tmax, T2m etc.

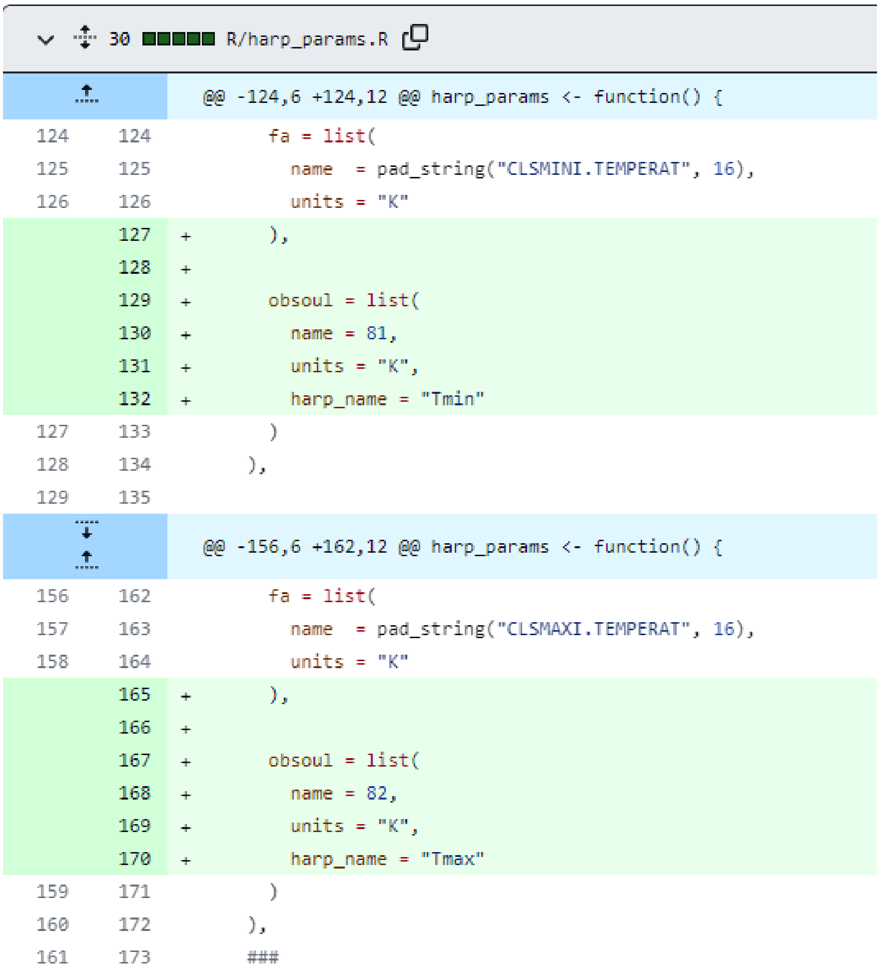


Figure 1: Adding new parameters into Harp interfaces

3.2 Code updates

To read and perform T2m correction from FA files we had to replace surface geopotential name, see Figure 2. **sfc_geo** replaced from "SURFGEOPOTENTIEL" into SPECSURFGEOPOTEN in **get_fa_param_info.R**.

If the GTS and national obsouls are merged, then SID would contain a mixture of standard WMO numbers (11520,...) and national identifiers with a unique prefix per country (AT12345, CR12345, CZ12345, ...). Furthermore, SYNOP and SHIP have the same obstype number, so SHIP needs to be removed. Modifications to handle this have been made by adapting existing function modify_sid as illustrated on Figure 3. The country argument was deprecated and removed from the code [Figure 4].

```
5 R/get_fa_param_info.R
@@ -27,7 +27,6 @@ get_fa_param_info <- function(param, fa_type="arome", fa_vector=TRUE, rotate_win
    if (existsFunction("fa_override")) {
      if (!is.null(fa_override(param$fullname))) return(fa_override(param$fullname))
    }
-
    # generic templates (there are exceptions!)
    if (tolower(param$fullname) %in% hardcoded_fields) {
      FAname <- switch(tolower(param$fullname),
@@ -44,7 +43,7 @@ get_fa_param_info <- function(param, fa_type="arome", fa_vector=TRUE, rotate_win
                       "td2m" = c("CLSHUMI.RELATIVE", "CLSTEMPERATURE  "),
                       "z0m"  = ,
                       "sfc_geopotential" = ,
-                      "sfc_geo" = "SURFGEOPOTENTIEL",
+                      "sfc_geo" = "SPECSURFGEOPOTEN",
                       "lsm"  = "SURFIND.TERREMER",
                       "cape" = "SURFCAPE.POS.F00", # "SURFCAPE.MOD.XFU"
                       "cien" = "SURFCIEN.POS.F00",
```

Figure 2: Surface geopotential name replace

```
@@ -251,22 +261,18 @@ obsoul_param_code_to_name <- function(x, param_defs) {

  ###
  # Function to add a country indicator to site IDs
- modify_sid <- function(x, country) {
-
-   country_codes <- list(
-     at = 90,
-     cr = 91,
-     cz = 92,
-     hu = 93,
-     pl = 94,
-     ro = 95,
-     si = 96,
-     sk = 97
+ modify_sid <- function(x) {
+   x <- str_replace_all(x,
+     c("^AT" = "90",
+       "^CR" = "91",
+       "^CZ" = "92",
+       "^HU" = "93",
+       "^PL" = "94",
+       "^RO" = "95",
+       "^SI" = "96",
+       "^SK" = "97"
+     ))
    )
-
-   x <- gsub("[[:alpha:]]|", "", x)
-   x <- paste0(country_codes[[country]], x)
-
    as.numeric(x)

  }
```

Figure 3: Update modify_sid function

```
@@ -58,6 +59,8 @@ read_obsoul <- function(
    obs_df <- dplyr::mutate(
      obs_df,
      type = dplyr::case_when(
+       str_sub(.data[["xx"]],-2,-1) == 14  ~ "synop",
+       str_sub(.data[["xx"]],-2,-1) == 24  ~ "ship",
        .data[["type"]] == 1   ~ "synop",
        .data[["type"]] == 2   ~ "airep",
        .data[["type"]] == 3   ~ "satob",
@@ -81,7 +84,7 @@ read_obsoul <- function(
    # for other obs types
    if (!is.null(obs_df[["synop"]])) {
      synop <- tidy_obsoul_synop(
-       obs_df[["synop"]], param_defs, country, max_obs
+       obs_df[["synop"]], param_defs, max_obs
      )
    } else {
      synop = list(synop = NULL)
@@ -94,13 +97,13 @@ read_obsoul <- function(
  ###
  # Function to tidy synop data
  ###
- tidy_obsoul_synop <- function(synop_df, param_defs, country, max_obs) {
+ tidy_obsoul_synop <- function(synop_df, param_defs, max_obs) {

    # Modify SID depending on country, set validdate in unix time
    # and convert parameter codes to names
    synop_df <- dplyr::mutate(
      synop_df,
-     SID  = modify_sid(.data[["SID"]], country),
+     SID  = modify_sid(.data[["SID"]]),
      validdate = suppressMessages(
        str_datetime_to_unixtime(
          paste0(
```

Figure 4: Removing country argument and filtering ship from synop

The reading of obsoul files with more than 57 columns was affected by a bug. If there are 62 columns in a row, the last part is cut off into the next row. To avoid this bug, some parts of the code have been corrected [Figure 5].

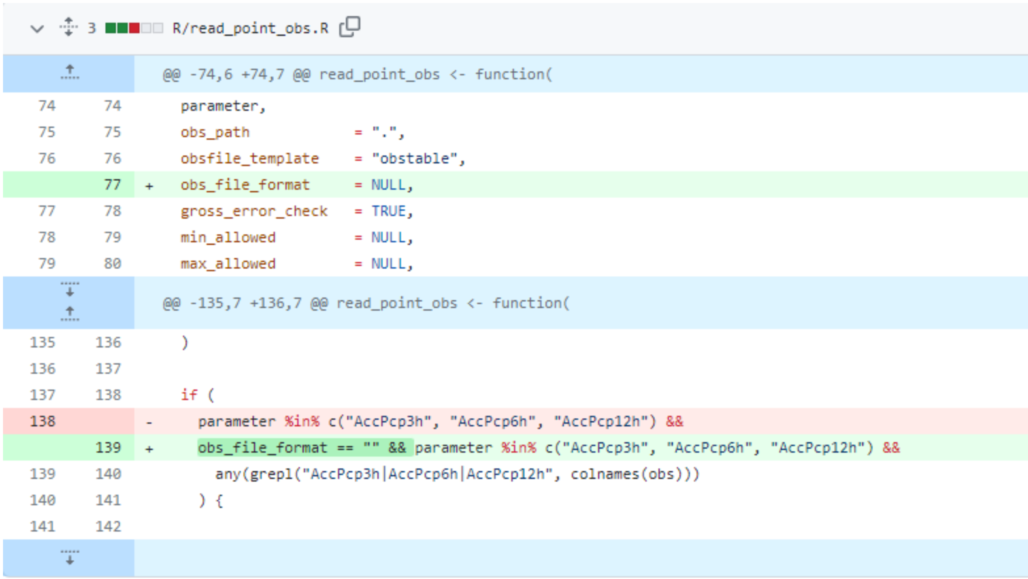
```
@@ -115,19 +118,26 @@ tidy_obsoul_synop <- function(synop_df, param_defs, country, max_obs) {
    )

    # Gather all observations sections into common columns
- synop_df <- lapply(
+ synop_df <- lapply(
      1:max_obs,
      function(x) dplyr::select(
        synop_df,
        !dplyr::matches("^obs[[:digit:]]+"),
        dplyr::starts_with(paste0("obs", x))
-     ) %>%
-       dplyr::rename_with(
-         ~gsub("^obs[[:digit:]]+", "obs", .x)
-       )
-   ) %>%
-     dplyr::bind_rows() %>%
-     dplyr::select(
+     )
+   )
+ synop_df[[1]] <- synop_df[[1]] %>% select(-starts_with("obs10"))
+
+
+ colnames = c("num_col","type","xx","lat","lon","SID","date","hms","elev","num_obs","xx1","xx2","validdate", "obs_code","obs_1","obs_2","obs_3","obs_end")
+
+ for (i in seq_along(synop_df)){
+   colnames(synop_df[[i]]) <- colnames
+ }
+
+
+ synop_df <- synop_df %>% bind_rows() %>%
+       dplyr::select(
        .data[["SID"]],
        .data[["lat"]],
        .data[["lon"]],
@@ -194,7 +204,7 @@ obsoul_cols <- function(max_obs) {
    col_names <- c(
      "num_col",
      "type",
-     "xx",
+     "xx", # (11 for a basic SYNOP, 14 for an automatic SYNOP, 21 for a basic SHIP, 24 for an automatic SHIP,...)
      "lat",
      "lon",
      "SID",
```

Figure 5: Preformed modification to read obsoul

The obsoul files contain only 6h precipitation, therefore **read_point_obs** function needs to be update as it by default tries to derive 6h accumulation from 12h ones. A new argument **obs_file_format** is added to skip the problematic part. In the default configuration, this argument is not defined. In order to read observation from obsoul, we need to define it as follows:

```
obs_file_format = "obsoul"
```

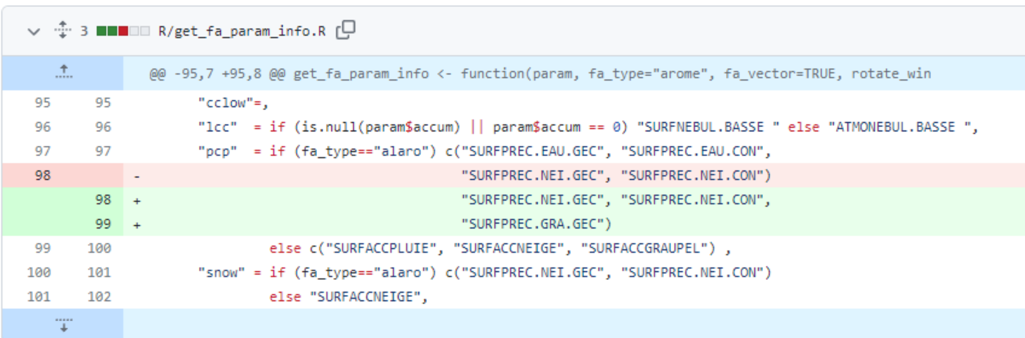


```
3 R/read_point_obs.R
@@ -74,6 +74,7 @@ read_point_obs <- function(
    parameter,
    obs_path            = ".",
    obsfile_template    = "obstable",
+   obs_file_format     = NULL,
    gross_error_check   = TRUE,
    min_allowed         = NULL,
    max_allowed         = NULL,
@@ -135,7 +136,7 @@ read_point_obs <- function(
    )

    if (
-     parameter %in% c("AccPcp3h", "AccPcp6h", "AccPcp12h") &&
+     obs_file_format == "" && parameter %in% c("AccPcp3h", "AccPcp6h", "AccPcp12h") &&
        any(grepl("AccPcp3h|AccPcp6h|AccPcp12h", colnames(obs)))
      ) {
```

Figure 6: Preformed modification to read 6h precipitation observation from obsoul

Recent versions of the ALARO model configuration contains new prognostic graupels that generate a new precipitation flux (SURFPREC.GRA.GEC) to be added in the total precipitation and total snow. The modification is needed in **get_fa_param_info.R, harp_params.R**



```
3 R/get_fa_param_info.R
@@ -95,7 +95,8 @@ get_fa_param_info <- function(param, fa_type="arome", fa_vector=TRUE, rotate_win
      "cclow"=,
      "lcc"  = if (is.null(param$accum) || param$accum == 0) "SURFNEBUL.BASSE " else "ATMONEBUL.BASSE ",
      "pcp"  = if (fa_type=="alaro") c("SURFPREC.EAU.GEC", "SURFPREC.EAU.CON",
-                                     "SURFPREC.NEI.GEC", "SURFPREC.NEI.CON")
+                                     "SURFPREC.NEI.GEC", "SURFPREC.NEI.CON",
+                                     "SURFPREC.GRA.GEC")
               else c("SURFACCPLUIE", "SURFACCNEIGE", "SURFACCGRAUPEL") ,
      "snow" = if (fa_type=="alaro") c("SURFPREC.NEI.GEC", "SURFPREC.NEI.CON")
               else "SURFACCNEIGE",
```

Figure 7: Adding SURFPREC.GRA.GEC fields into precipitation flux

4 Experiments

As the code had been changed, it was necessary to control the read and write of data from the obsoul file. We ran multiple quality checks to ensure everything worked perfectly.

4.1 Stations list

The read functions in Harp enable you to transform data as it is read in. In other words, it can interpolate gridded data to geographic points, regrid and reproject gridded data. In order to do so, you need to provide a list of stations that you want to interpolate to. The station data structures must include columns like 'SID', 'lat', and 'lon' that indicate the position of the station. As a default, Harp has implemented a default station list if a station list is not provided. Typing command "station_list" will show us the first ten station:

```
> station_list

    SID   lat    lon    elev      name

   1001  70.9   -8.67   9.4     JAN MAYEN
   1002  80.1   16.2    8       VERLEGENHUKEN
   1003  77     15.5    11.1    HORNSUND
   1004  78.9   11.9    8       NY-ALESUND II
   1006  78.3   22.8    14      EDGEOYA
   1007  78.9   11.9    7.7     NY-ALESUND
   1008  78.2   15.5    26.8    SVALBARD AP
   1009  80.7   25.0    5       KARL XII OYA
   1010  69.3   16.1    13.1    ANDOYA
   1011  80.1   31.5    10      KVITOYA
```

The latitude and longitude of this station list are specified to four decimal places. To demonstrate the importance of accurately describing the station's position, and considering creating your own station list, let's perform some experiments.
The first step will be to create two lists in which two stations will be listed. The first station list gives lat and lon in short form, and second station list includes long descriptions of lat and lon. Station list with short formatting:

```
    SID     lat     lon     elev       name

   11518    50.1    14.26   365.0    PRAHA-RUZYNE
   11520    50.0    14.45   304.0    PRAHA-LIBUS
```

and station list with long formatting:

```
    SID      lat       lon      elev        name

   11518   50.10028  14.25556  365.349    PRAHA-RUZYNE
   11520   50.00778  14.44694  302.00     PRAHA-LIBUS
```

As can be seen, the long formatting includes detailed lat,lon information. Taking these into account, let's see how they impacted verification scores. The following setup is run: verification period is from 2023-03-01 to 2023-03-17, with lead-time of 24 hours that were starting at 00 UTC. Verification step was 6 hours, model output file were in the FA format. Observation values was taken from obsoul and vobs files. Largest differences were found for 10m wind [Figure 8] and smaller ones for other parameters [Figure 9] where

- blue line-doted line represent scores from vobs files and long formatted station position
- black line-doted line represent scores from vobs files and short formatted station position
- gray line represent scores form obsoul files and long formatted station position
- red line represent scores from obsoul files and short formatted station position

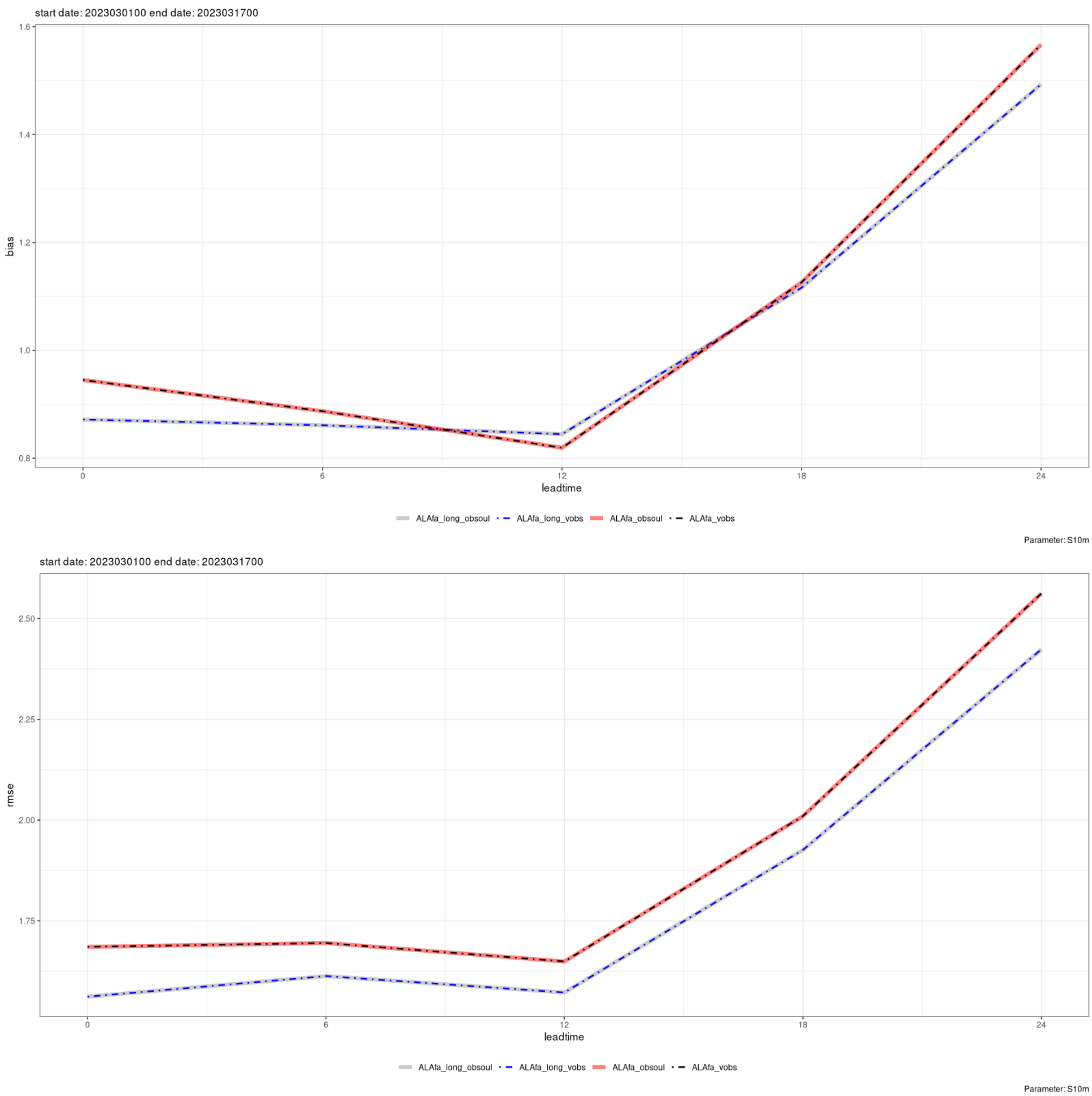


Figure 8: Impact of point position precision on BIAS (top) and STD (bottom) for parameter S10m

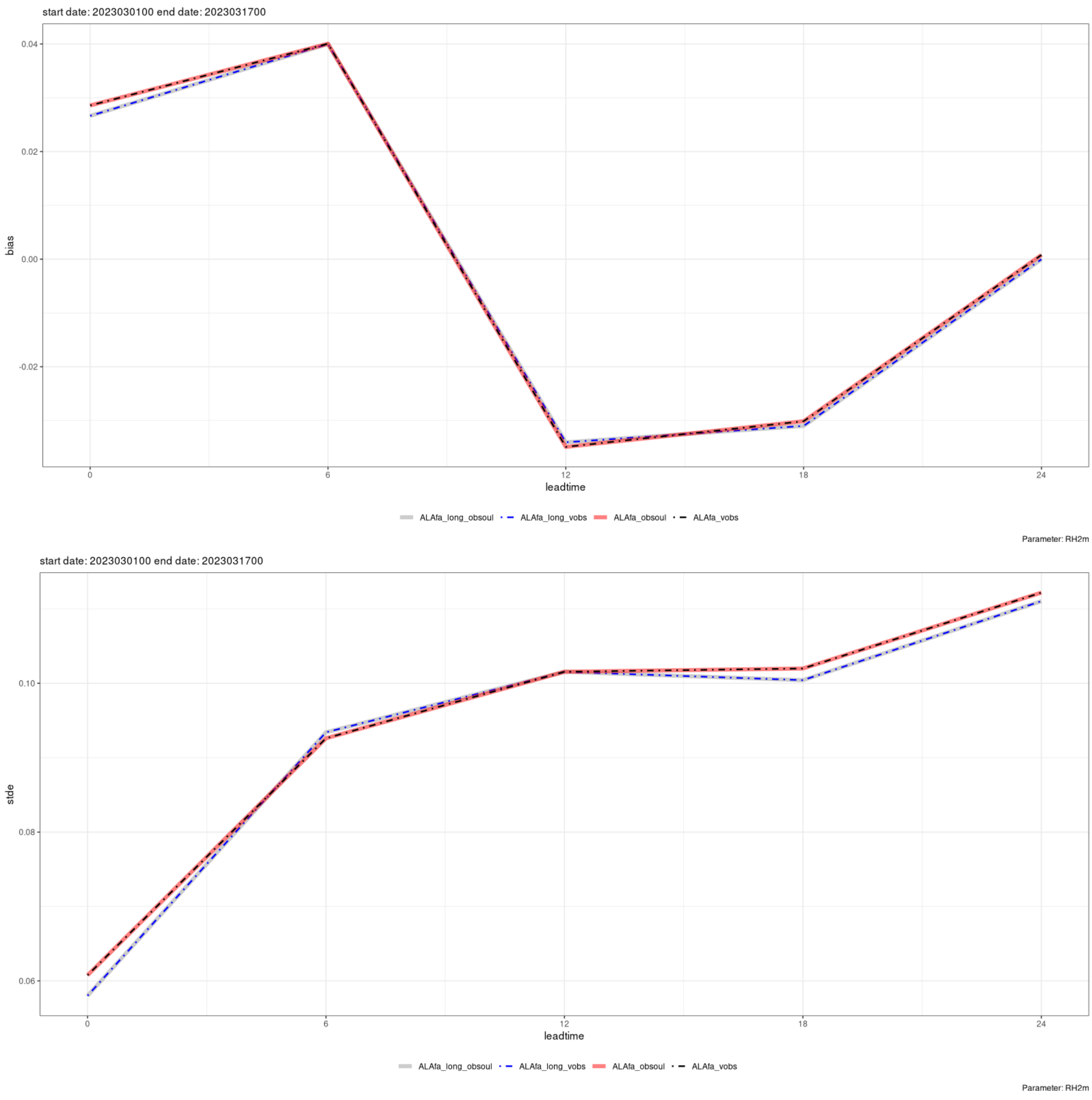


Figure 9: Impact of point position precision on BIAS (top) and STD (bottom) for parameter RH2m

4.2 Comparison of obsoul and vobs scores

In order to check that data from the obsoul file is properly handled we decided to take observations from a single station, Praha-Libus (11520), and compare it with reference observation source in vobs format. Observations were read separately from vobs and obsoul for T2m, RH2m, S10 and D10m. The forecast was read from FA files, for two model experiments ALADIN-CY43 and ALADIN-CY46, named ALAD and Dakw respectively.

As can be seen [Figure 10] the verification scores computed from two different observation sources and models provide the same results. The scores for ALAfa (ALADIN-CY43) with obsoul observations (in gray) and ALAfa_vobs with vobs observations (in blue) cover each other. Similarly for experiment Dakw where the black dashed lines (vobs) and the red lines (obsoul) overlap. The results confirm that the new HarpIO for obsoul handle data correctly.

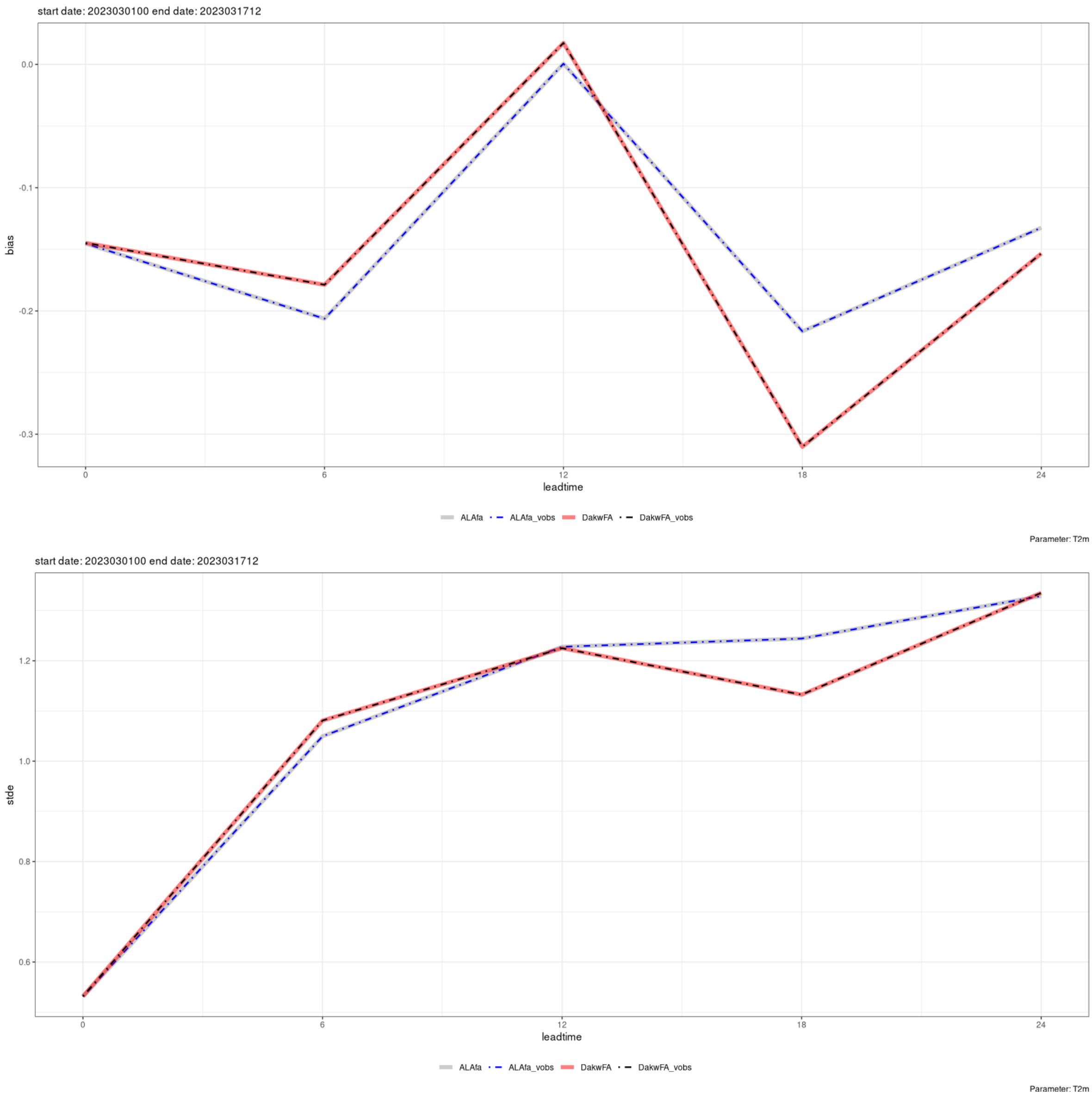
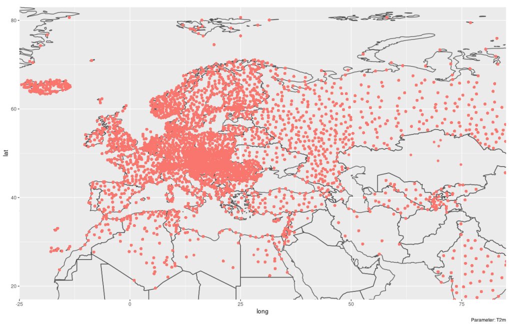


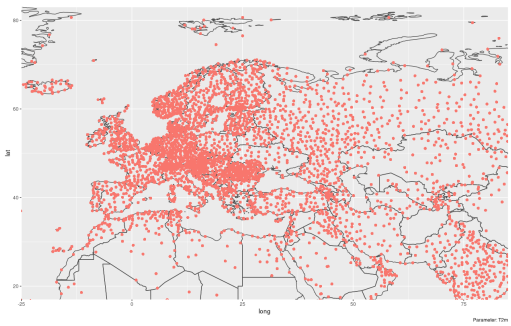
Figure 10: Comparison of T2m BIAS (top) and STD (bottom) using observations from different sources (obsoul and vobs)

4.3 Coverage of obsoul and vobs data

A more extensive comparison of obsoul and vobs data is problematic because of differences in geographic coverage of stations. The harpIO interfaces was used to check the coverage for different parameters. Figures 11-18 show differences in station numbers. In some regions obsoul files have more data while in others vobs files have a better coverage. It would be desirable to improve data coverage in both data sources. Furthermore, precipitation offers a lot of room for improving coverage. According to [Figure 18], the data availability is poor for 6h/1h precipitation accumulations. It's not because of poor measurement coverage, but because of technical issues that needs to be resolved.

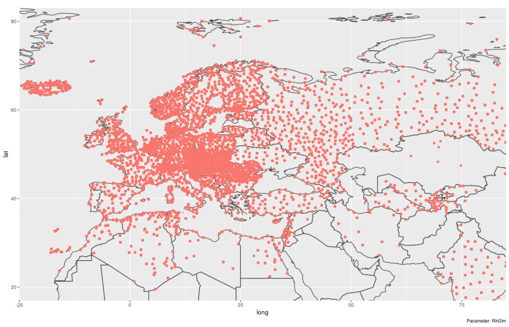


(a) obsoul

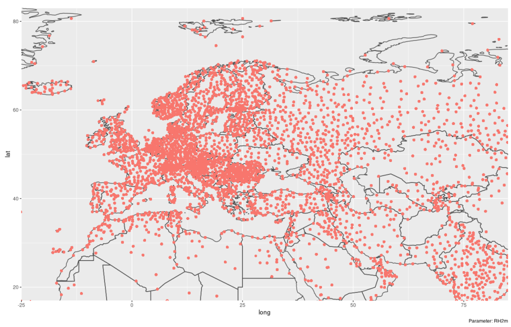


(b) vobs

Figure 11: Geographical coverage of T2m parameters over Europe.

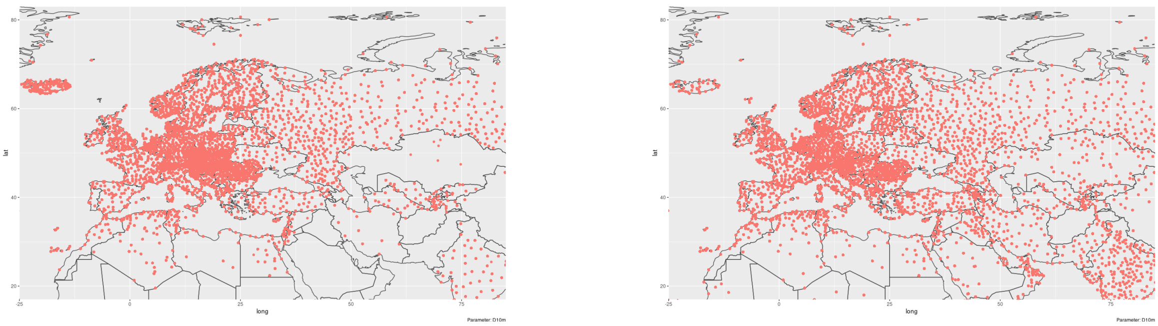


(a) obsoul



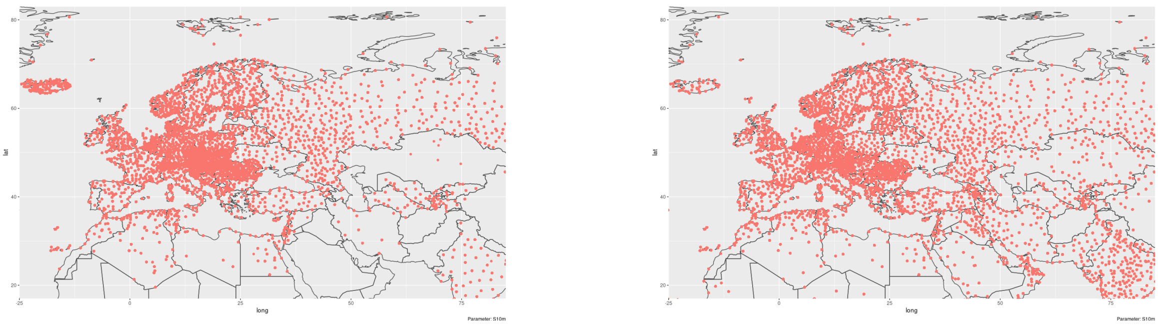
(b) vobs

Figure 12: Geographical coverage of RH2m parameters over Europe.



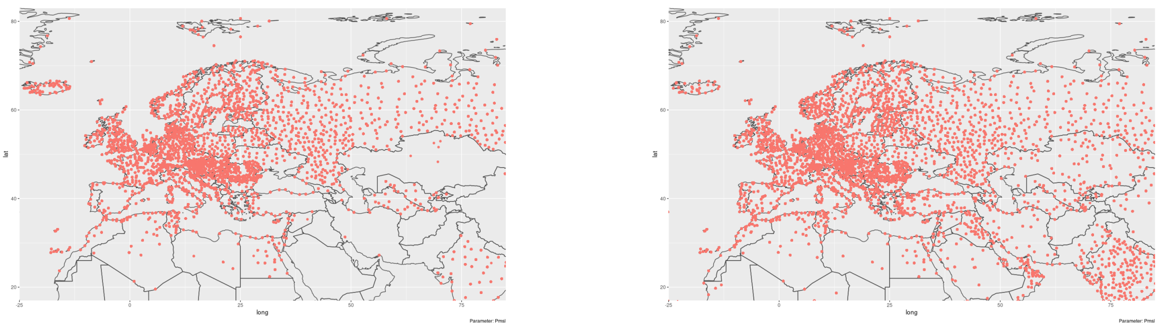
(a) obsoul (b) vobs

Figure 13: Geographical coverage of D10m parameters over Europe.



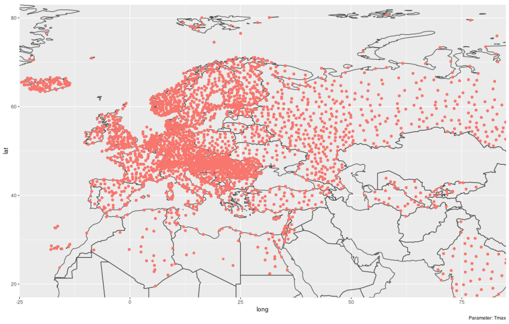
(a) obsoul (b) vobs

Figure 14: Geographical coverage of S10m parameters over Europe.

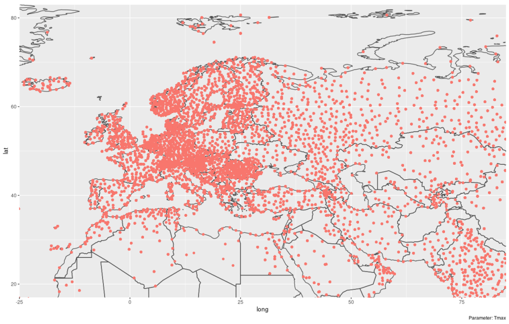


(a) obsoul (b) vobs

Figure 15: Geographical coverage of Pmsl parameters over Europe.

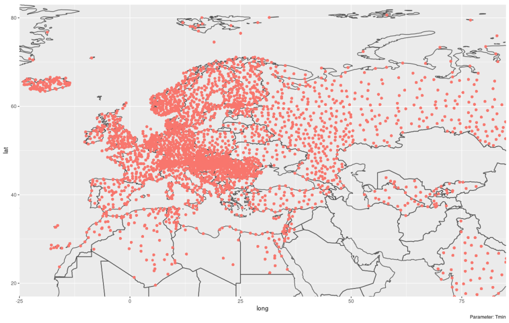


(a) obsoul

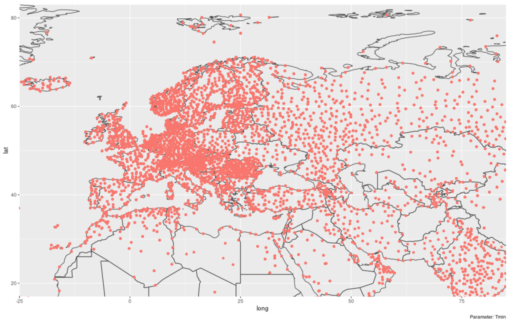


(b) vobs

Figure 16: Geographical coverage of Tmax parameters over Europe.

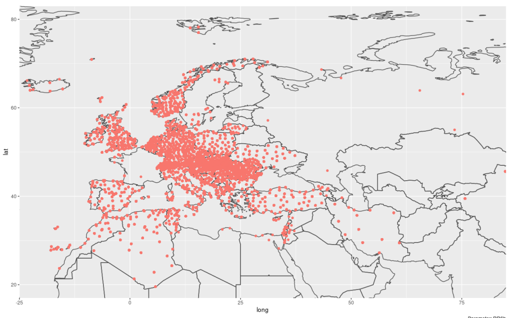


(a) obsoul

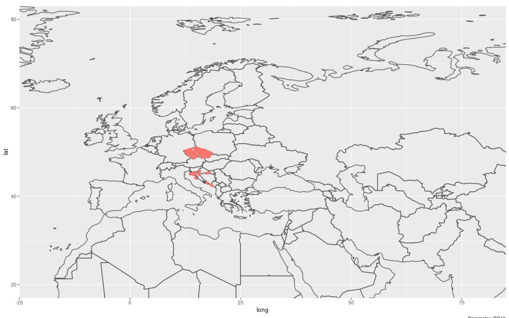


(b) vobs

Figure 17: Geographical coverage of Tmin parameters over Europe.



(a) 6h precipitation obsoul.



(b) 1h precipitation obosul

Figure 18: Geographical coverage of precipitation parameters over Europe.

4.4 Comparison of HARP and VERAL scores

In the final assessment, we compared the Harp scores with respect to the local verification tool VERAL (both using the same observations in obsoul format). To accomplish this, one station Praha-Ruzyne was selected for a one-day, date **2023-03-05** forecast starting from **00**UTC. The height correction was not performed on the T2m parameter [Figure 19]. The blue dotted line represents the verification score by VERAL and the grey line is Harp's verification score.

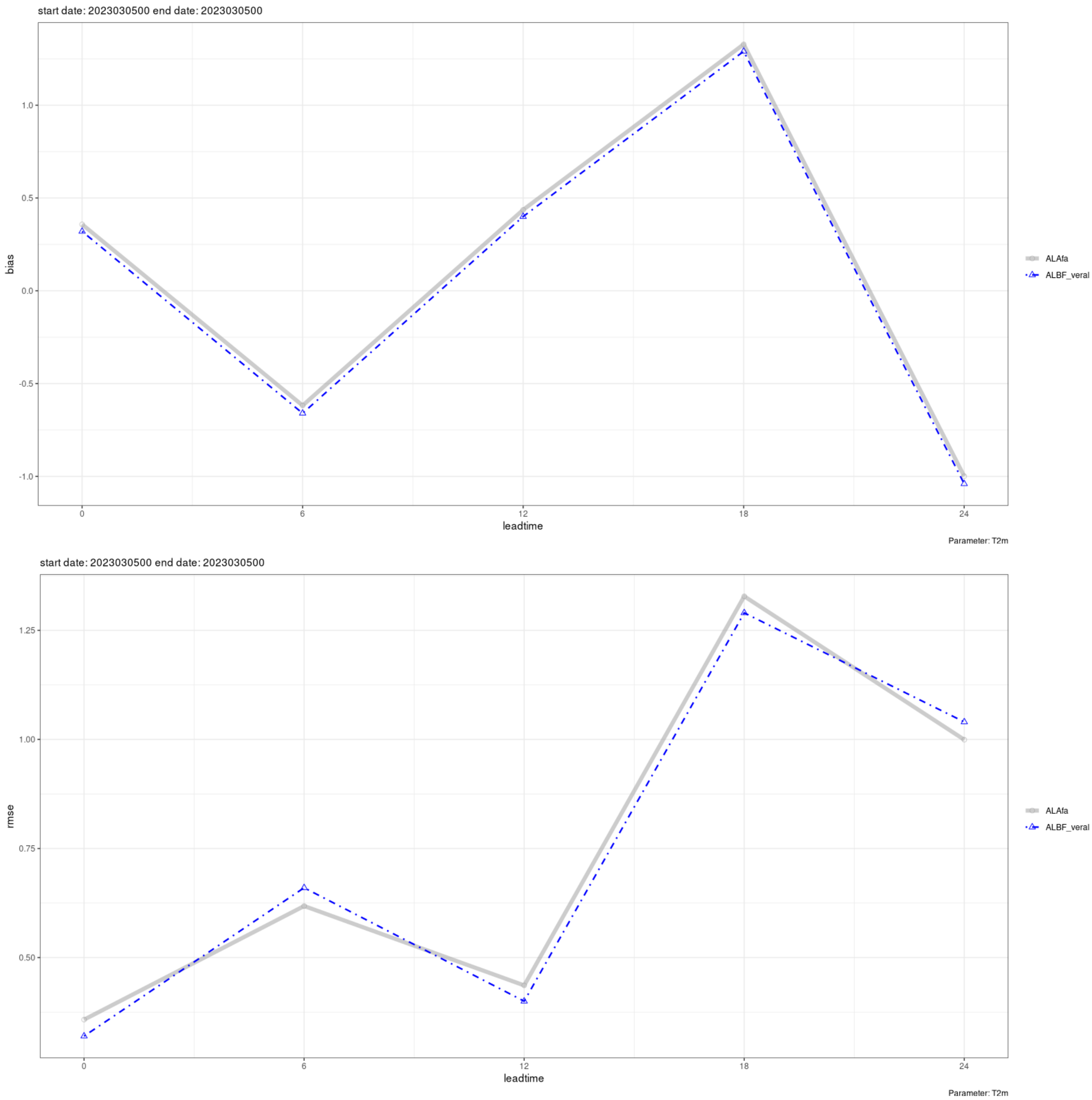


Figure 19: BIAS (top) and RMSE (bottom) for T2m non-corrected using Harp (gray) and VERAL (blue-dotted)

This comparison helped to identify several issues. The attention must be paid to the units of the individual parameters in both observation and forecast, e.g. cloudiness can be in fraction or oktas. For inter-comparison of Harp and VERAL the cloudiness fractions were converted into oktas scale using the "step scaling" function according to Svabik [2].

In the case of observed precipitation, the Harp can derive precipitation from other network times. For example, missing 6h precipitation at 06 UTC and 18 UTC can be derived from 12h ones reported at 06 UTC and 18 UTC minus 6h accumulations reported at 00 UTC and 12 UTC. But obsoul already contains 6h precipitation for 00 UTC, 06 UTC, 12 UTC and 18 UTC, so this derivation part needs to be skipped. An new argument **obs_file_format** was added into **read_point_obs** function to accomplish this. The user needs to specify if he is using obsoul data format. Section 3.2 provide more details about this changes.

Furthermore we need to add another feature to accumulate all precipitation fluxes. Harp counts only four precipitation fluxes (SURFPREC.EAU.CON, SURFPREC.EAU.GEC, SURFPREC.NEI.CON, SURFPREC.NEI.GEC), while VERAL consider five of them (in addition, there is SURFPREC.GRA.GEC - the precipitation flux of new prognostic graupel. Precipitation or any other cumulative parameter is a rather special case. Accumulated fields must be named AccXXXnnh, with XXX being the parameter name (e.g. Pcp) and nn being the accumulation period in hours. It is necessary to include the trailing "h" (otherwise it is treated as a pressure level). For example, AccPcp6h. Due to the fact that different accumulation periods may be required for verification (e.g. 1h, 6h, 24h), the extraction routine does not de-cumulate the data. Therefore, it would be better to extract only the Pcp table. The verification routine will then read the different data entries and decumulate. For example, data extraction is run with:

```
param = Pcp
```

and the verification for 6h precipitation, is run with the:

```
param = AccPcp6h
```

After this, there is still a small difference in the precipitation scores, which could not be explained.

Overall the scores from both tools are fairly similar [Figures 19-24]. For more information about scripts for data extracting and reading observation check Appendix B.

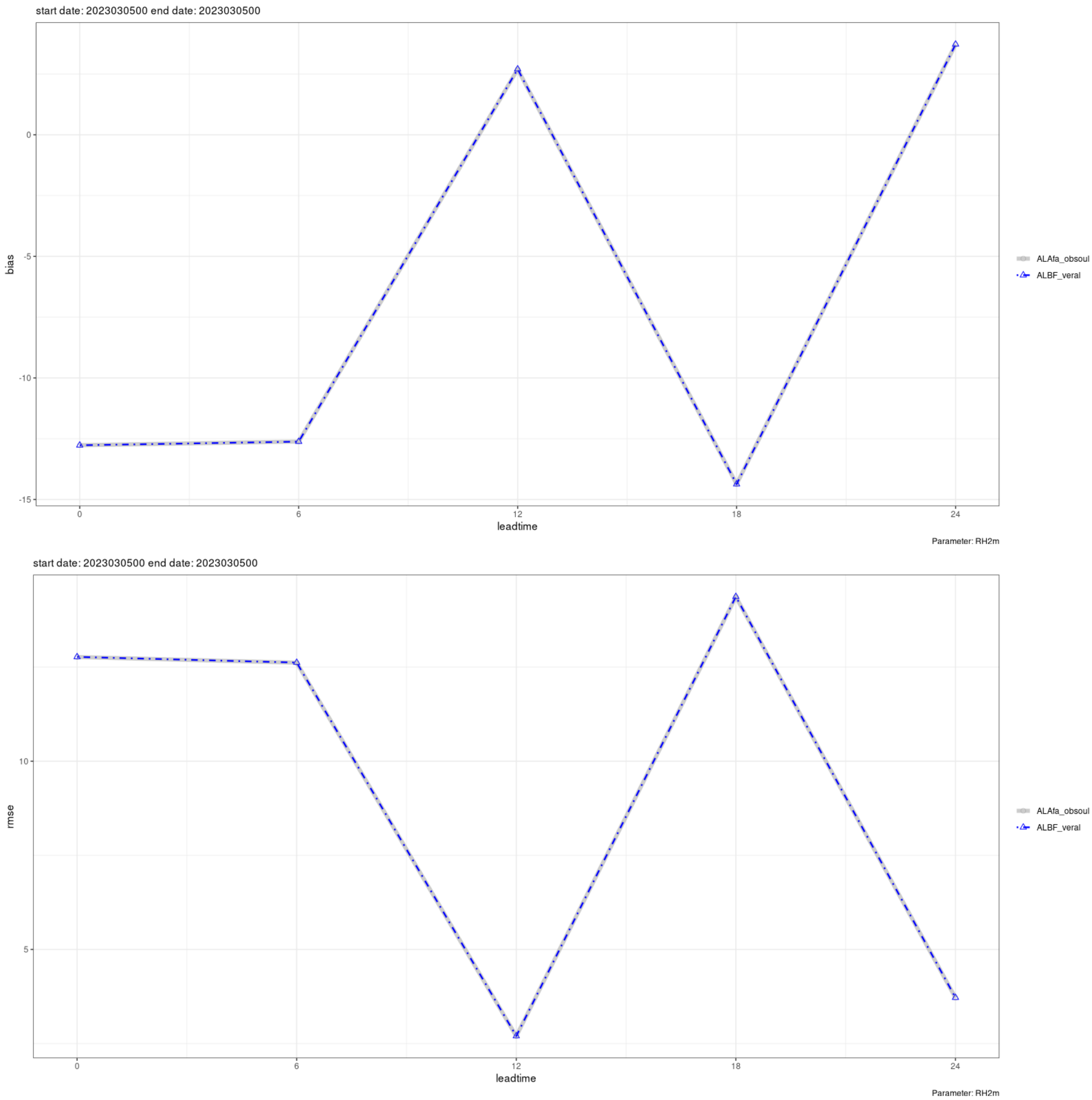


Figure 20: BIAS (top) and RMSE (bottom) for RH2m using Harp (gray) and VERAL (blue-dotted)

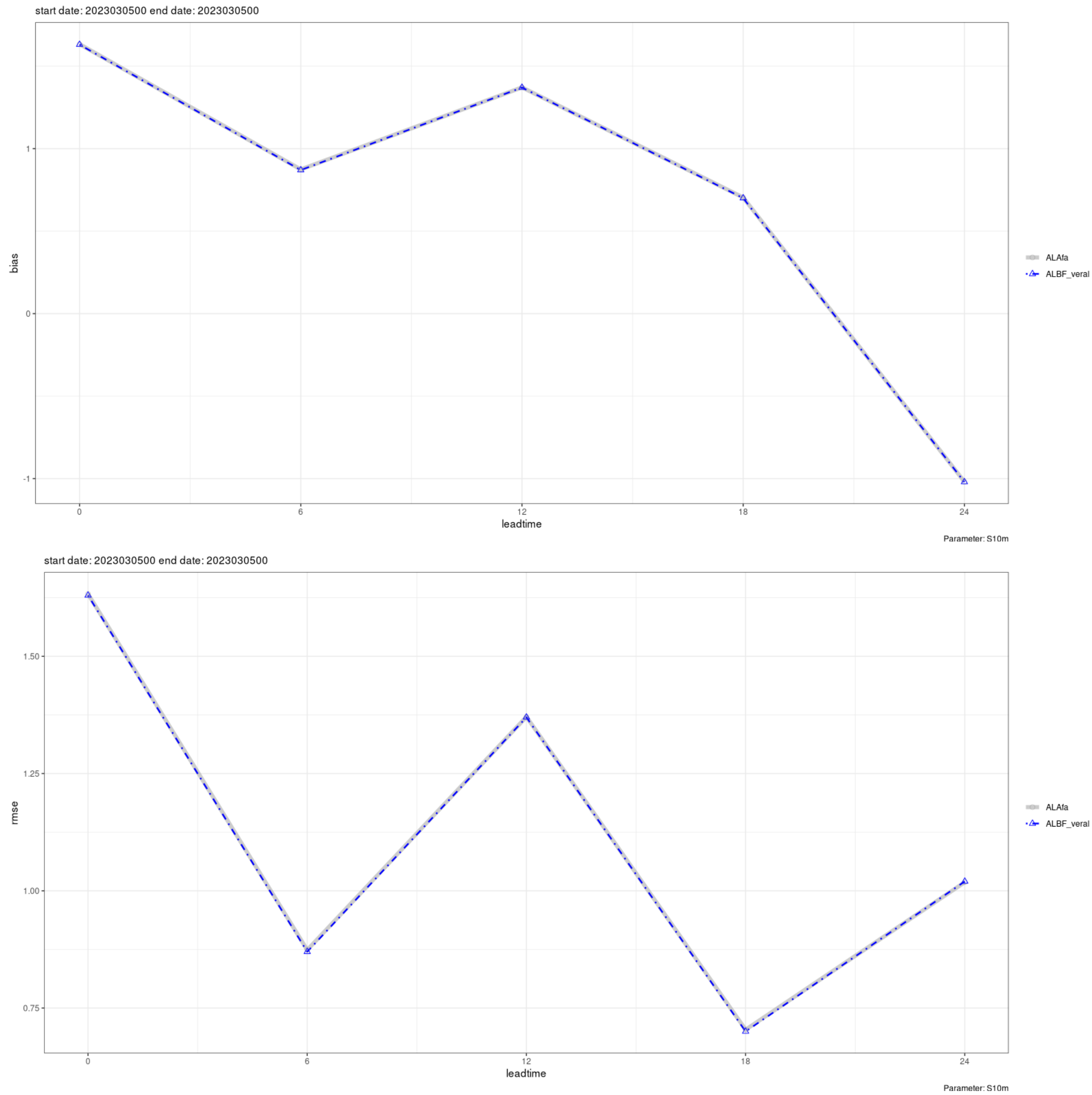


Figure 21: BIAS (top) and RMSE (bottom) for S10m using Harp (gray) and VERAL (blue-dotted)

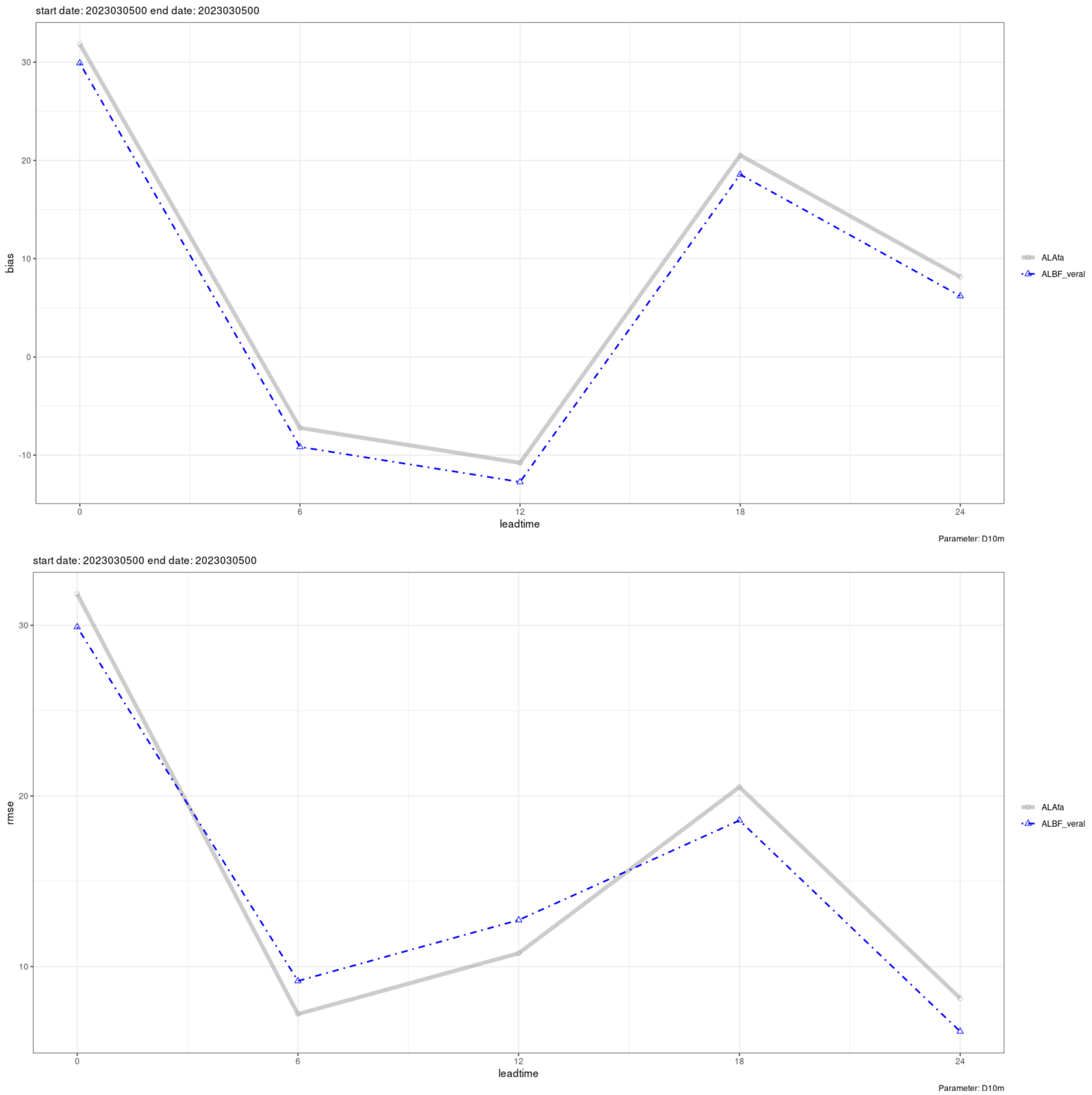


Figure 22: BIAS (top) and RMSE (bottom) for D10m using Harp (gray) and VERAL (blue-dotted)

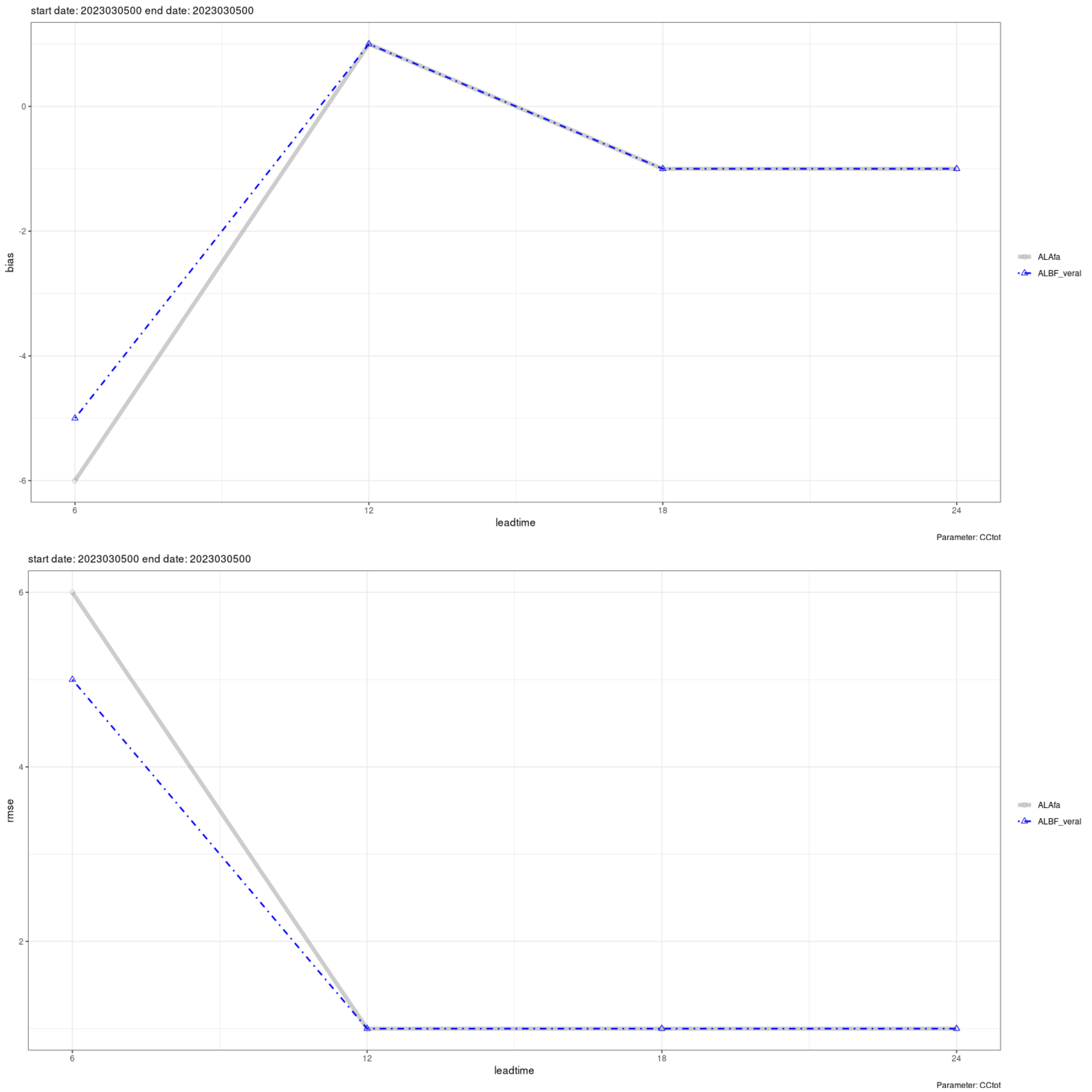


Figure 23: BIAS (top) and RMSE (bottom) for CCtot using Harp (gray) and VERAL (blue-dotted)

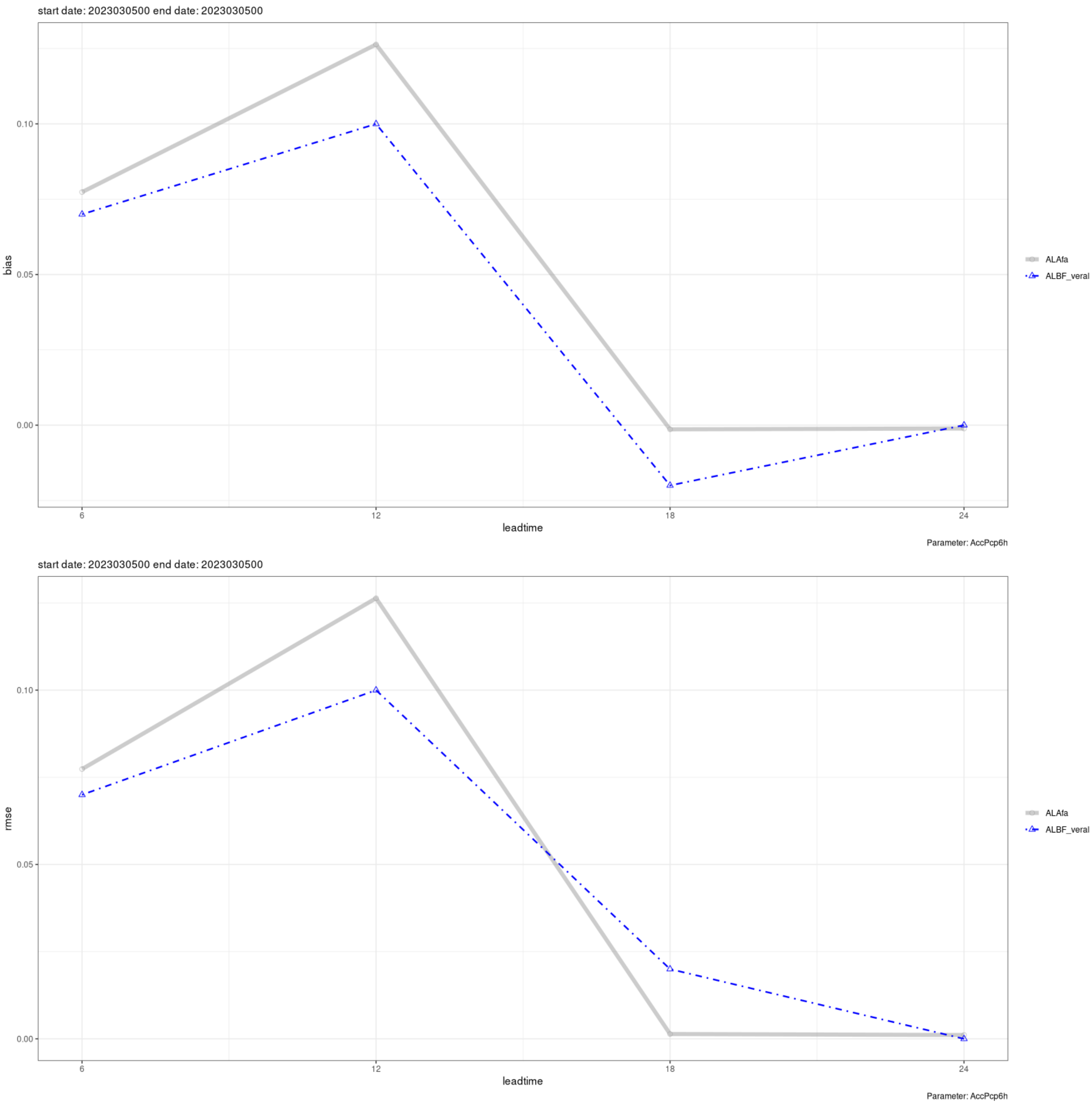


Figure 24: BIAS (top) and RMSE (bottom) for AccPcp6h using Harp (gray) and VERAL (blue-dotted)

5 Conclusion

The purpose of this report was to test the updated **read_obsoul** function, which is capable of reading obsoul format. In order to ensure everything works as it should, we perform some comparison tests. We can conclude based on the results that the updated changes have no effect on reading and verifying. It is backed up by a comparison to VERAL.

Through testing, we gain a better understanding of the source code, opening up new possibilities for future implementations. The report shows we have already begun this process by adding some new parameters. There was also a slight update to the FA files. It was discovered that there was a problem with reading geopotential fields from FA files. The problem has been addressed and will be solved.

Currently, this updated version is not included in the official Harp repository. This is due to the fact that the newest versions have a number of significant changes. Our plan is to incorporate this repository into the official release after it has been released.

6 Appendix A: Introduction to renv

The *renv* package manager helps us manage dependencies between R packages. You can use this tool to manage library paths and other project-specific information, to isolate your project's R dependencies, and to manage R packages using existing tools like install.packages() and remove.packages(). The following benefits can be achieved by using renv:

- Isolated: New or updated packages for one project will not break your other projects, and vice versa. This is because renv provides a private package library for each project.
- Portable: You can easily move your projects from one computer to another, even across different platforms. Your project's dependencies can be installed easily with renv.
- Reproducible: renv keeps track of the exact package versions you need, and allows you to install those exact versions everywhere.

You can use renv interfaces in two ways. Manage your project from the command line (terminal) or from the Rstudio IDE. The Rstudio approach is more user friendly, but sometimes we have to execute from the command line, so here i will show you both. First i will show you terminal approach. In the terminal run R-language typing *R*. The interactive interface for the R language will run. If renv is not installed on your system, you can install it from CRAN by typing this command:

```
install.packages("renv")
```

Before we create project enviroment, type:

```
.libPaths()
```

This command shows places where R will search and install upcoming packages. In my case it looks like this:

```
> .libPaths()
 [1] "/users/p6095/R/x86_64-redhat-linux-gnu-library/3.6"
 [2] "/usr/lib64/R/library"
 [3] "/usr/share/R/library"
```

As default paths, these three can be called central R libraries. But, I don't want to install in these paths, I wont to separate project from them. To accomplish this I will use R renv package. R renv will create project separated library, which will not affect the default paths. To create new virtual environment, type next command in to command line:

```
renv::init("/path/to/renv_name/")
```

Example, if i want to create new environment in /**home** directory, i will type next:

```
renv::init("/home/test/")
```

and virtual environment named **test** will be crated in /**home** directory. For example, i created environment in **"/users/p6095/R/obsoul_harp_isolate/"** directory. Ruining R and command:

```
renv::activate("~/R/obsoul_harp_isolate/")
```

environment named **obsoul_harp_isolate** will be activated. Now my **libPaths** will look like this:

```
> .libPaths()
 [1] "/users/p6095/R/obsoul_harp_isolate/renv/library/R-3.6/x86_64-redhat-linux-gnu"
 [2] "/tmp/Rtmp89l83i/renv-system-library"
```

This will be easier if you use Rstudio. When creating new project, just tick the checkbox *"Use renv with this project"*. The commands **renv::activate**, **libPaths** can be run from the Rstudio

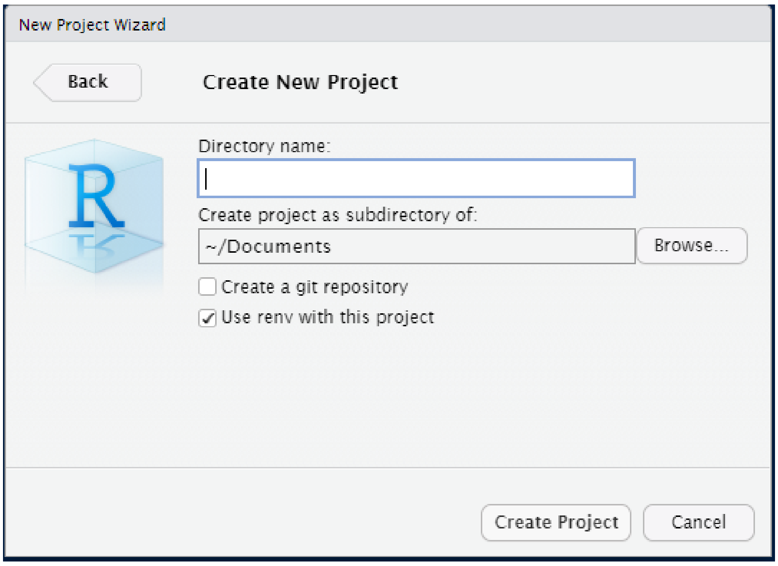


Figure 25: Create renv enviroment in Rstudio

console, as shown in the first instance (running from terminal).

7 Appendix B: Code used to read files into SQLite

As for T2m, we read from FA files with height correction:

`correct\_t2m = TRUE`

and without height correction:

`correct\_t2m = FALSE`

The same template was used for all FA files. The list of stations was created.

```
template <- "{YYYY}/{MM}/{DD}/{HH}/ICMSHALAD+00{LDT2}"
station <- read.csv("/home/mma266/app/harpData/etc
                   /stationLACE.txt")
```

Code was used to read FA files

```
ALA_FA <- read_forecast(

  start_date     = 2023030100,
  end_date       = 2023031712,
  fcst_model     = "ALAfa",
  by             = "12h",
  parameter      = "T2m",
  lead_time      = seq(0, 24, 6),
  file_path      = "/work/mma266/data",
  file_template  = template,
  transformation = "interpolate",
  transformation_opts = interpolate_opts(
                        stations         = station,
                        method           = "bilinear",
                        correct_t2m      = TRUE,
                        clim_file_format = "fa",
  ),

  output_file_opts = sqlite_opts(path = "/work/mma266/data"),
  return_data    = TRUE
)
```

In the case of other parameters, the code was the same, only the wind was different. In order to read wind direction and speed, we add the **fa_opts** function. As of right now, the installation code does not implement this function, not exported, so it must be imported manually with next command:

`harpIO:::fa_opts`

```
ALA_FA <- read_forecast(
   start_date       = 2023030100,
   end_date         = 2023031700,
   fcst_model       = "ALAfa",
   by               = "24h",
   parameter        = "D10m",
   lead_time        = seq(0, 24, 6),
   file_format      = "fa",
   file_path        = "/work/mma266/data/",
   file_template = template,
   file_format_opt  = harpIO:::fa_opts(fa_type = "alaro",
                                        rotate_wind = TRUE



  ),
   transformation = "interpolate",
   transformation_opts = interpolate_opts(
                              stations        = station,
                              method          = "bilinear",
                              clim_file_format = "fa",
                              correct_t2m     = TRUE,
                              keep_model_t2m = FALSE
   ),

   output_file_opts = sqlite_opts(path = "/work/mma266/data"),
   return_data      = TRUE
)
```

The following code was used to read obsoul files:

```
obs <- read_obs(
    seq_dates(2023030100, 2023031723),
    file_path = "/work/mma266/obs",
    by="1h",
    file_template =  "{YYYY}/{MM}/obsoul_1_all_{YYYY}{MM}{DD}{HH}",
    output_format_opts = obstable_opts(path = "/work/mma266/obsoul/"),
    return_data = TRUE )
```

Saving precipitation forecasts into SQLite:

```
ALA_FA <- read_forecast(
   start_date       = 2023030100,
   end_date         = 2023031700,
   fcst_model       = "ALAfa",
   by               = "24h",
   parameter        = "Pcp",
   lead_time        = seq(0, 24, 1),
   file_format      = "fa",
   file_path        = "/work/mma266/data/",
   file_template = template,
   file_format_opt  = harpIO:::fa_opts(fa_type = "alaro"),
   transformation = "interpolate",
   transformation_opts = interpolate_opts(
                           stations          = station,
                           method            = "bilinear",
                           clim_file_format = "fa",
                           correct_t2m       = TRUE,
                           keep_model_t2m = FALSE
   ),

   output_file_opts = sqlite_opts(path = "/work/mma266/data"),
   return_data      = TRUE
)
```

From SQLite, forecast precipitation is read as follows:

```
forecast <- read_point_forecast(

start_date = 2023030500,
end_date   = 2023030500,
fcst_model = "ALAfa",
fcst_type  = "det",
lead_time  = seq(0,24,6),
parameter  = "AccPcp6h",
by         = "24h",
file_path  = "/work/mma266/data/"
)
forecast <- set_units(forecast,"mm")
```

As can be seen, **set_units** functions were used to convert forecast units fields to **[mm]**. Make sure that forecasts and observations are in the same units. The same procedure was followed when it came to observation.

Export precipitation from SQLite:

```
obs <- read_point_obs(

first_validdate(forecast),
last_validdate(forecast),
parameter = "AccPcp6h",
obs_path  = "/work/mma266/obsoul",
obs_file_format = "obsoul" )

obs <- set_units(obs,"mm")
```

The following code was used to read CCtot from SQLite.
Forecast:

```
forecast <- read_point_forecast(

start_date  = 2023030500,
end_date    = 2023030500,
fcst_model  = "ALAfa",
fcst_type   = "det",
lead_time   = seq(6,24,6),
parameter   = "CCtot",
by          = "24h",
file_path   = "/work/mma266/data/"

)
```

and converts values into oktas:

```
forecast <- forecast %>% mutate(ALAfa_det = case_when (

ALAfa_det >=0. & ALAfa_det <0.0625 ~ 0,
ALAfa_det >=0.0625 & ALAfa_det <=0.1875 ~ 1.,
ALAfa_det >=0.1875 & ALAfa_det <=0.3125 ~ 2.,
ALAfa_det >=0.3125 & ALAfa_det <=0.4375 ~ 3.,
ALAfa_det >=0.4375 & ALAfa_det <=0.5625 ~ 4.,
ALAfa_det >=0.5625 & ALAfa_det <=0.6875 ~ 5.,
ALAfa_det >=0.6875 & ALAfa_det <=0.8125 ~ 6.,
ALAfa_det >=0.8125 & ALAfa_det <=0.9375 ~ 7.,
ALAfa_det >=0.9375 & ALAfa_det <=1 ~ 8.,
TRUE ~ 8.))

forecast <- set_units(forecast,"oktas")
```

Observation:

```
obs <- read_point_obs(

first_validdate(forecast),
last_validdate(forecast),
parameter = "CCtot",
obs_path  = "/work/mma266/obsoul",
station   = c('11518')
) %>%
    set_units(.,"oktas") %>%
    scale_point_obs(., 'CCtot',
                          scale_factor = 0.01,
                          multiplicative = TRUE)

obs <- obs %>% mutate(CCtot = case_when (

CCtot >=0. & CCtot <0.0625 ~ 0,
CCtot >=0.0625 & CCtot <=0.1875 ~ 1.,
CCtot >=0.1875 & CCtot <=0.3125 ~ 2.,
CCtot >=0.3125 & CCtot <=0.4375 ~ 3.,
CCtot >=0.4375 & CCtot <=0.5625 ~ 4.,
CCtot >=0.5625 & CCtot <=0.6875 ~ 5.,
CCtot >=0.6875 & CCtot <=0.8125 ~ 6.,
CCtot >=0.8125 & CCtot <=0.9375 ~ 7.,
CCtot >=0.9375 & CCtot <=1 ~ 8.,
TRUE ~ 8.))
```

References

[1] *renv package manager*. URL: https://rstudio.github.io/renv/.

[2] Filip Svabik. *Validation of harp Ecosystem: Point Verification of Surface Parameters*. Tech. rep. CHMI, 2019.