Intro to LakeEnsemblR - G21.5

Tadhg Moore, Jorrit Mesman, Robert Ladwig, Johannes Feldbauer

2020-09-28

1. Introduction

LakeEnsemblR is a new R package created by GLEON-members, that facilitates running ensemble model runs of up to five different models (FLake, GLM, GOTM, Simstrat, and MyLake) in a consistent and standardised manner.

During this workshop, you will learn how to set-up and run LakeEnsemblR for an example set-up, how to calibrate the models, and how to use several post-processing functions that are included in the package. Once you understand how the package works, you can also try to change model parameters, or apply it to your own lake data.

The code is available at https://github.com/aemon-j/LakeEnsemblR, and several example set-ups can be found on https://github.com/aemon-j/LER_examples. There is also a Wiki page that hosts information and some Frequently Asked Questions (FAQ): https://github.com/aemon-j/LakeEnsemblR/wiki

2. Installation and loading of packages

Install LakeEnsemblR from Github and if you haven't done so already, `install.packages("devtools")` first.

We will be running five different models, each of which has a corresponding R package that contains the model executables. We will also need `gotmtools` and `glmtools`, which help in reading and writing model-specific files. The GLEON package `rLakeAnalyzer` is also used for running analysis and formatting of data allows for compatibility with this package. Install these packages first (unless you've already done this), followed by `LakeEnsemblR` itself.

WARNING: Installing all these packages can potentially take a long time so it is **RECOMMENDED** that you do this **PRIOR** to attending the workshop.

```
#install.packages("devtools")
devtools::install_github("GLEON/rLakeAnalyzer")
devtools::install_github("USGS-R/glmtools", ref = "ggplot_overhaul")
devtools::install_github("GLEON/GLM3r", ref = "GLMv.3.1.0a3")
devtools::install_github("aemon-j/FLakeR", ref = "inflow")
devtools::install_github("aemon-j/GOTMr")
devtools::install_github("aemon-j/gotmtools")
devtools::install_github("aemon-j/SimstratR")
devtools::install_github("aemon-j/MyLakeR")
```

Once you have the packages installed you can install `LakeEnsemblR`. An aside note, if you only wish to use one or two of the lake models with `LakeEnsemblR` then you only need to install those packages and all the `LakeEnsemblR` functions will work with those models.

```
devtools::install_github("aemon-j/LakeEnsemblR")
```

3. Load example set-up

An example set-up for Lough Feeagh (Ireland) has been included in the package. Load it with the following lines of code:

First, set your working directory. Since we will be copying the Lough Feeagh set-up in here, this can be any folder on your PC (in R this can be done by `setwd()`).

Now we copy Lough Feeagh example folder into the working directory and change our working directory to the newly created "feeagh" folder

```
template_folder <- system.file("extdata/feeagh", package= "LakeEnsemblR")
file.copy(from = template_folder, to = ".", recursive = TRUE)
```

Set working directory to the new folder (again, in a regular script by using `setwd()`).

```
setwd("./feeagh")
```

Load libraries

```
library(gotmtools)
library(LakeEnsemblR)
```

Have a look at the feeagh folder. There will be six files

```
list.files()
```

```
## [1] "LakeEnsemblR.yaml"
## [2] "LakeEnsemblR_bathymetry_standard.csv"
## [3] "LakeEnsemblR_ice-height_standard.csv"
## [4] "LakeEnsemblR_inflow_standard.csv"
## [5] "LakeEnsemblR_meteo_standard.csv"
## [6] "LakeEnsemblR_wtemp_profile_standard.csv"
```

"LakeEnsemblR.yaml" is the configuration file for LakeEnsemblR. It contains all information needed to run the run the models; start and end of simulation, time step, names of input files, output settings, etc. You can open the file directly in RStudio by holding "Ctrl" and left-clicking the filename ("LakeEnsemblR.yaml") in the R editor. Alternatively you can use a text editor (e.g. Notepad++).

Have a look at the LakeEnsemblR.yaml file and see if you understand what each section does. You will see that in this file, there are the names of the other files in the directory, e.g. "LakeEnsemblR_meteo_standard.csv", for meteorological input. Open the meteo file in a text editor. The column names are important; this is how LakeEnsemblR knows what column is what variable. The required headers for any variable can be found in the dictionary which comes with the package

```
print(met_var_dic)
```

```
##                                       standard_name short_name
## 1                                          datetime       time
## 2   Longwave_Radiation_Downwelling_wattPerMeterSquared        lwr
## 3  Shortwave_Radiation_Downwelling_wattPerMeterSquared        swr
## 4                          Cloud_Cover_decimalFraction         cc
## 5                              Air_Temperature_celsius       airt
```

```
## 6                          Relative_Humidity_percent       relh
## 7                      Dewpoint_Temperature_celsius       dewt
## 8     Ten_Meter_Elevation_Wind_Speed_meterPerSecond wind_speed
## 9         Ten_Meter_Elevation_Wind_Direction_degree   wind_dir
## 10            Ten_Meter_Uwind_vector_meterPerSecond        u10
## 11            Ten_Meter_Vwind_vector_meterPerSecond        v10
## 12                  Precipitation_millimeterPerDay     precip
## 13                 Precipitation_millimeterPerHour   precip_h
## 14                       Rainfall_millimeterPerDay       rain
## 15                      Rainfall_millimeterPerHour     rain_h
## 16                       Snowfall_millimeterPerDay       snow
## 17                      Snowfall_millimeterPerHour     snow_h
## 18            Sea_Level_Barometric_Pressure_pascal      p_sea
## 19         Surface_Level_Barometric_Pressure_pascal     p_surf
## 20                         Vapour_Pressure_milliBar      vap_p
## 21                  Extinction_Coefficient_perMeter   ext_coef
```

Or alternatively, this is further described on our Wiki page: https://github.com/aemon-j/LakeEnsemblR/wiki/2.-Setting-up-LakeEnsemblR which gives further details on which variables are required and which can be internally calculated if not present.

(From this point on, we'll refer to LakeEnsemblR as "LER")

4. Export configuration files

Each of the five models requires its own set of configuration and input files, preferably in different file formats and units. But, LER takes care of this for you! Using the LER config file and your input files, the `export_config()` function will create separate folders for each model, with the right set-up for each model.

```
export_config("LakeEnsemblR.yaml", model = c("FLake", "GLM", "GOTM",
                                             "Simstrat", "MyLake"))
```

Now have a look at the folder again. Folders for each model have been created. Because our folder was empty, LER used templates stored in the package and the information in the input and config files to create these folders (if the files in the "config_files" section already exist, it will make changes in these files instead).

Each model can now be run in its respective folder.

Something we won't treat further now, but which may be interesting for you when you set up your own simulations; `export_config()` only exports the information included in the LER config file. Other settings in the model-specific config files are kept in their defaults. Should you wish to change model-specific parameters, you can add them to the "model_parameters" section in the LER config file. In our example, this has been done for several parameters in FLake, GLM, GOTM, and MyLake. Should you wish to change settings that are more difficult to access than by just changing a parameter (e.g. inflow-depth in Simstrat), you can do that in between running `export_config()` and `run_ensemble()`

5. Running the model ensemble

As mentioned at the start of this script, the models are actually run by other packages; FLakeR, GLM3r, GOTMr, SimstratR, and MyLakeR. These packages contain executables for Windows, MacOS, and Linux (except for MyLakeR, which instead contains the MyLake code in R). LER calls these packages to run the models in the folders we've just created.

```
run_ensemble("LakeEnsemblR.yaml", model = c("FLake", "GLM", "GOTM",
                                            "Simstrat", "MyLake"))
```

Each model folder will now have an "output" folder which contains the model-specific output. However, LER has also compiled these results in a standardised netcdf format, in an "output" folder in our main directory. NetCDF files can be easily accessed by PyNcView https://sourceforge.net/projects/pyncview/. (If you wish to find out what a NetCDF file is and why we use it, check out our Wiki: https://github.com/aemon-j/LakeEnsemblR/wiki/What-is-a-NetCDF-file%3F)

If you prefer text output, you can also opt for that type of output in the LER config file. The output generated is in the same format that is used in `rLakeAnalyzer` so will allow you to use functions from that package, which you may be familiar with, for loading and plotting the data. Currently the post-processing functions in LER only work with the netCDF format.

Let's generate some text output, and also make this change in the LER config file from within R, using LER's input_yaml_multiple function.

```
input_yaml_multiple("LakeEnsemblR.yaml", value = "text", key1 = "output", key2 = "format")
run_ensemble("LakeEnsemblR.yaml", model = c("FLake", "GLM", "GOTM",
                                            "Simstrat", "MyLake"))
```

Now text output has been generated in the "output" folder. But let's set the output format back to netcdf, as we'll need that later.

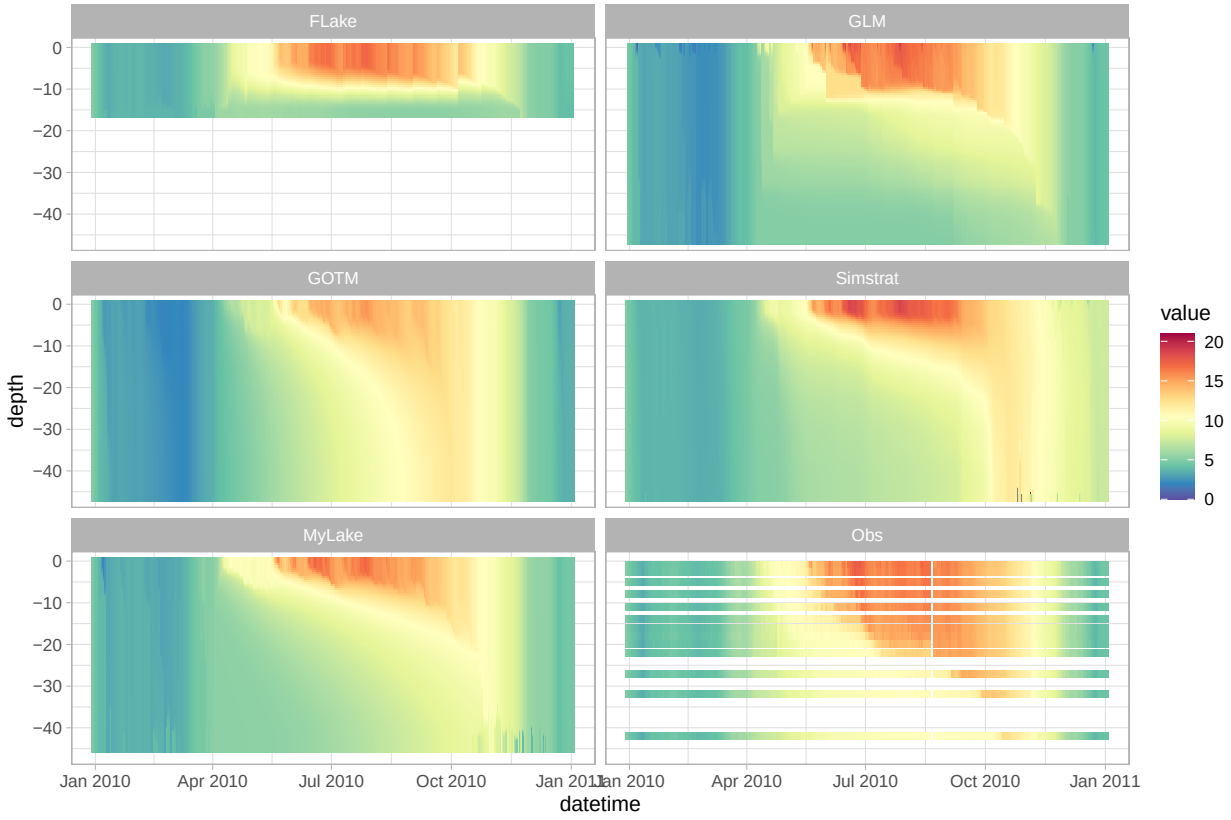
```
input_yaml_multiple("LakeEnsemblR.yaml", value =  "netcdf", key1 = "output",
                    key2 = "format")
```

Exploring the netCDF file

Have a look at the netcdf output by opening it in PyNcView. In the left menu, select "z,time,model,member,lat,lon" and click "temp". Then in the right-hand menu, tick the boxes "member" and "model". You can increase the value of "model" by increments. The order of the models is the same as in our function call (i.e. "FLake", "GLM", "GOTM", "Simstrat", "MyLake"), followed by observations. We'll forget about "member" right now, and treat that later. Under "time,model,member,lat,lon", you can also see plots of ice thickness, but that's not too exciting as Lough Feeagh has no ice cover...

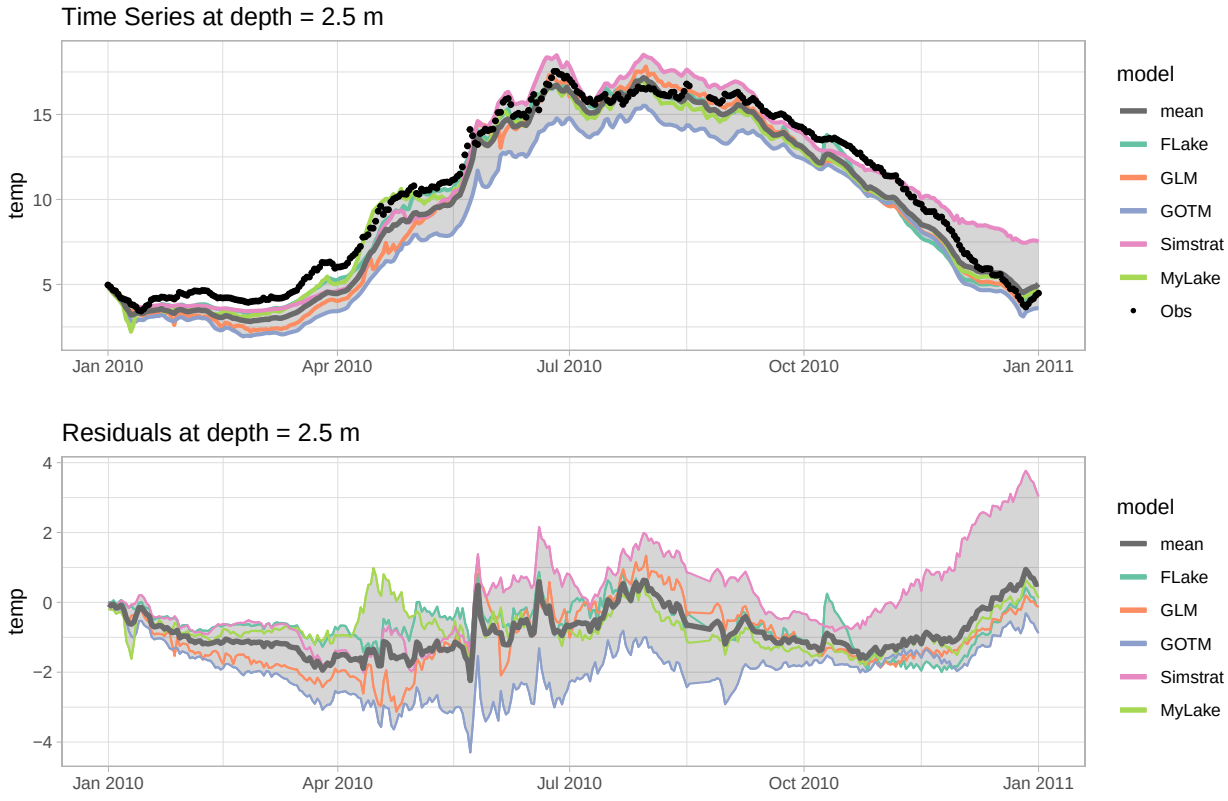
You can also plot the output with the LER `plot_heatmap()` function. It returns a ggplot object, that you can modify further if needed (as done here with `scale_colour_gradientn()` and `theme_light()`).

```
library(ggplot2)
plot_heatmap("output/ensemble_output.nc") +
  scale_colour_gradientn(limits = c(0, 21),
                         colours = rev(RColorBrewer::brewer.pal(11, "Spectral")))+
  theme_light()
```



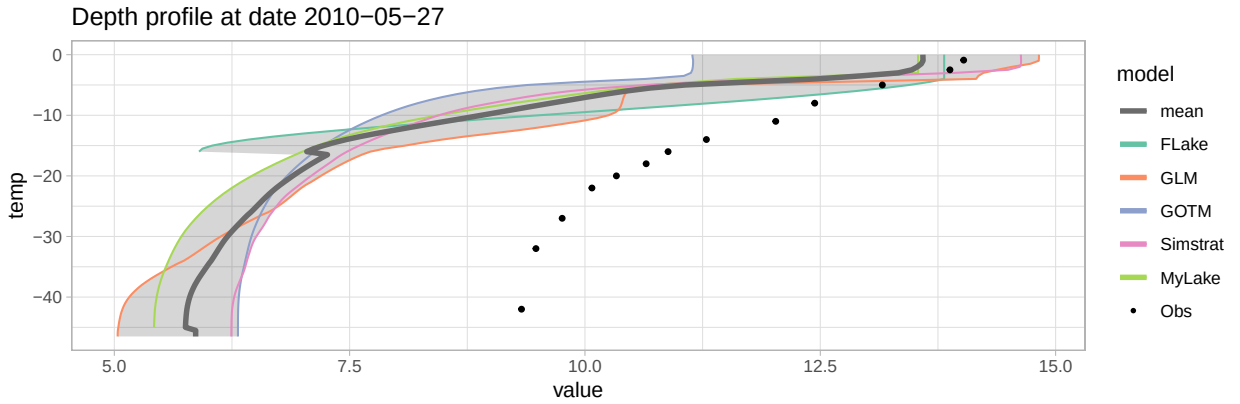
Using the `plot_ensemble()` function plots of time of the ensemble can easily be created. If observed data is available the funciton also plots the residuals can be created:

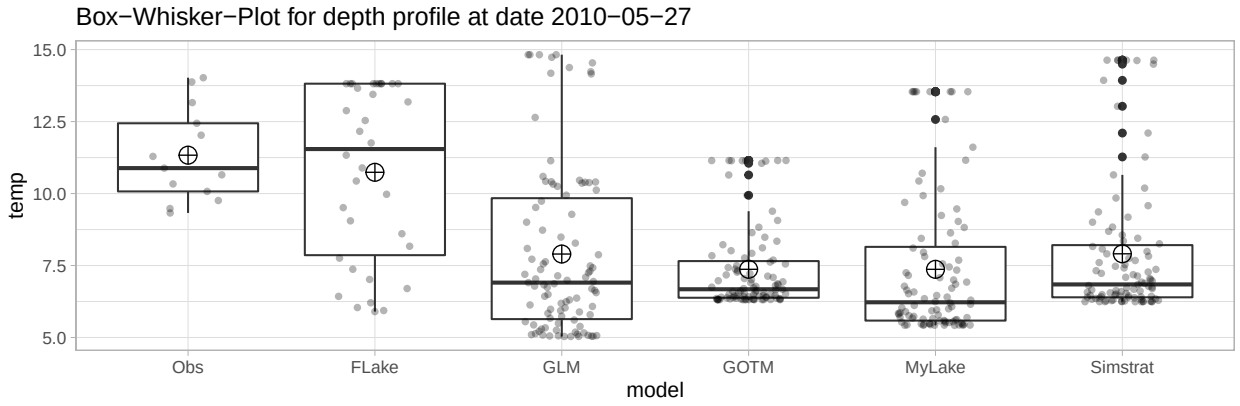
```
library(ggpubr)
p1 <- plot_ensemble("output/ensemble_output.nc", model = c("FLake", "GLM",
                                                           "GOTM", "Simstrat",
                                                           "MyLake"),
                    var = "temp", depth = 2.5,
                    residuals = TRUE)

ggarrange(p1[[1]] + theme_light(),
          p1[[2]] + theme_light(),ncol = 1, nrow = 2)
```



The function can also be used to plot depth profiles at selected dates and additionally box plots of the variable can be created:

```
p2 <- plot_ensemble("output/ensemble_output.nc", model = c("FLake", "GLM",
                                                           "GOTM", "Simstrat",
                                                           "MyLake"),
                    var = "temp", date = "2010-05-27 00:00:00",
                    boxwhisker = TRUE, residuals = FALSE)

ggarrange(p2[[1]] + theme_light(),
          p2[[2]] + theme_light(), ncol = 1, nrow = 2)
```



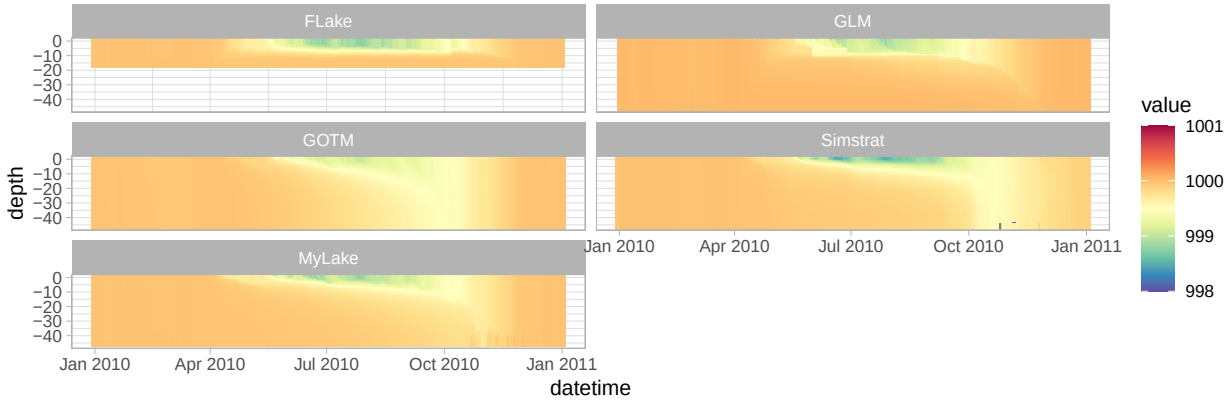


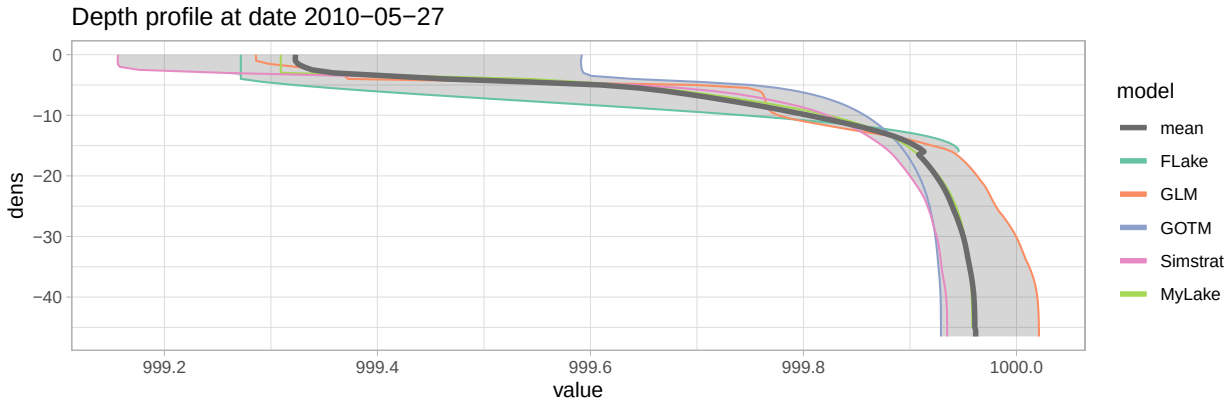
LakeEnsmblr can also gather other output variables from the model, such as density or salinity, in the created netcdf output file. To do so you just need to add them to the LakeEnsemblR.yaml file in the output section

```
input_yaml_multiple("LakeEnsemblR.yaml", value = c("temp", "ice_height",
                                                   "dens"),
                    key1 = "output", key2 = "variables")

run_ensemble("LakeEnsemblR.yaml",
             model = c("FLake", "GLM", "GOTM", "Simstrat", "MyLake"),
             parallel = TRUE,
             add = FALSE)
```

Now we can plot this new variable in the netcdf file using e.g. the `plot_heatmap()` or the `plot_ensemble()` function:

```
p3 <- plot_heatmap("output/ensemble_output.nc", var = "dens") +
  theme_light() + scale_colour_gradientn(limits = c(998, 1001),
                                         colours = rev(RColorBrewer::brewer.pal(11, "Spectral")))
p4 <- plot_ensemble("output/ensemble_output.nc", model = c("FLake", "GLM",
                                                           "GOTM", "Simstrat",
                                                           "MyLake"),
                    var = "dens", date = "2010-05-27 00:00:00") +
  theme_light()

ggarrange(p3, p4, ncol = 1, nrow = 2)
```

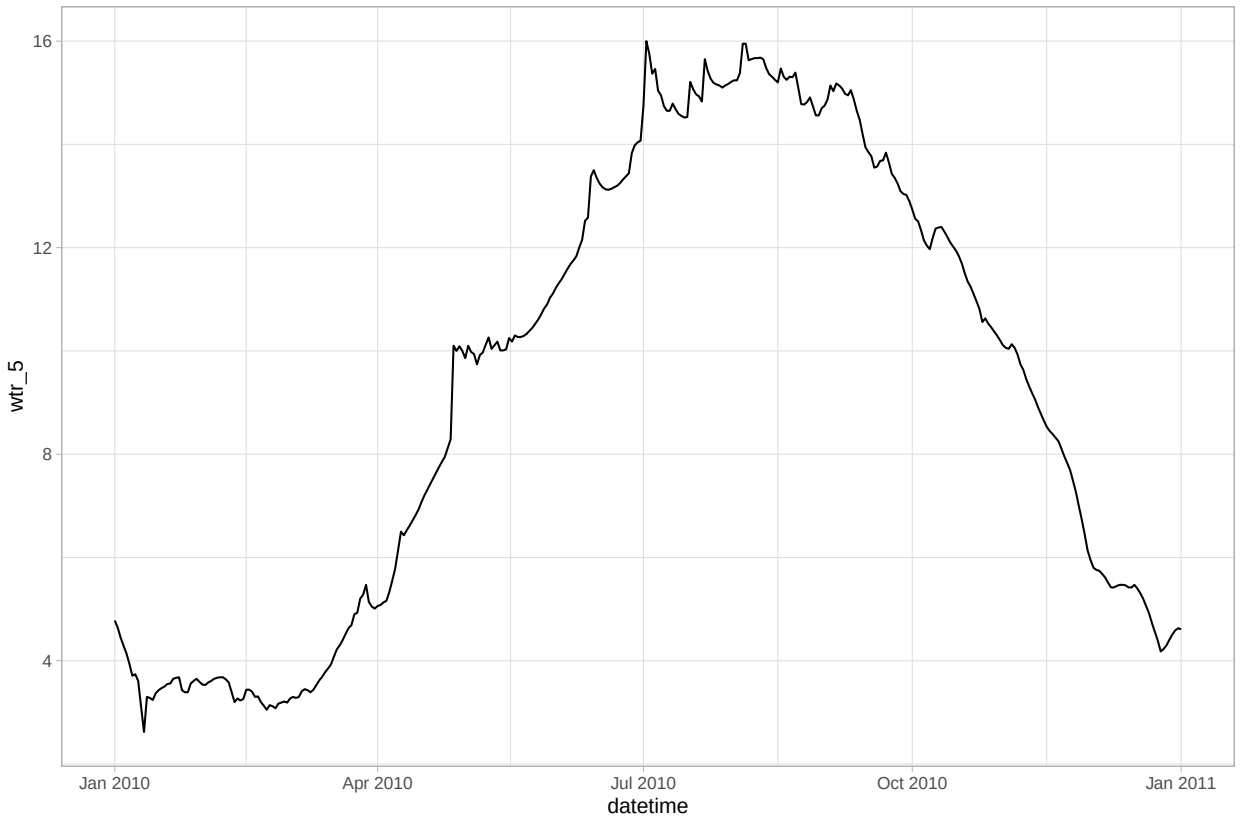




Plotting text output

There are no functions in LER to plot the text output, but you could use this code, for example:

```
plot_model <- "MyLake" # Model names are case-sensitive
plot_depth <- 5 # In our example, output is given every 0.5 m

df <- read.csv(paste0("./output/Feeagh_", plot_model, "_temp.csv"))
df$datetime <- as.POSIXct(df$datetime)

ggplot(df)+
  geom_line(aes_string(x = "datetime", y = paste0("wtr_", plot_depth)))+
  theme_light()
```



Both the netcdf output and the text output are structured in a standardised format, which makes further analysis or plotting easy. Only water temperature and ice thickness are currently generated in the "output" folder. Some models have more output variables (e.g. energy dissipation rate in GOTM). This output is still available in the model-specific folders, and can be accessed if you'd like.

6. Calibration

In the previous section we ran the models using default or pre-set (in the "model_parameters" section) parameters, and with observed meteorological forcing. However, often you will want to apply some form of calibration to your model.

The LER config file also has a "calibration" setting, which we will use here. Let's use the settings that are present in this file.

Essentially, the calibration section has two parts: "met" and "model-specific". In the "met" section, you can scale wind speed, shortwave radiation and longwave radiation, which will be calibrated separately for each model. In the other sections, you can provide the name of each model and calibrate model-specific parameters.

Have a look at the calibration section to see which parameters we're calibrating and between what ranges.

We call the calibration almost the same way as the normal ensemble run, but with the `cali_ensemble()` function. We add the "parallel = TRUE" statement, which will calibrate each model on a separate core, speeding up the process. We're using the "MCMC" (Markov Chain Monte Carlo) method here. To save some time, we will only do 10 iterations for each model (this is way too few for a proper calibration, in which case several thousands of iterations are more appropriate). The speed of running each model differs quite a lot, with FLake being the fastest and MyLake being the slowest. You can follow progress by looking at the csv files in the "cali" folder that are being created during the calibration.

PS: you need to run `export_config()` before starting the calibration. You have done that earlier in this script.

Depending on your PC, this function could take a few minutes to run. If you want to speed it up, you can remove MyLake from the model argument.

```
  cali_result <- cali_ensemble("LakeEnsemblR.yaml",
                               model = c("FLake", "GLM", "GOTM", "Simstrat", "MyLake"),
                               num = 10,
                               cmethod = "MCMC",
                               parallel = TRUE)
```

You can access the best parameters for each model in cali_result, e.g.for GLM

```
  cali_result[["GLM"]][["bestpar"]]
```

```
##          wind_speed                 swr mixing/coef_mix_hyp
##            1.397402            1.083248            1.061905
```

For cmethod = "LHC", the different parameter sets and goodness-of-fit metrics will instead be put in tables in the "cali" folder. The reason for this is that sometimes one parameter set may have the lowest RMSE, and another the highest Pearson's r.

A third possible calibration method is cmethod = "modFit", this method uses the `modFit()` function of the FME package which provides different "classical" algorithms for constrained fitting of a model to data. For details see the `modFit()` help.

There is not yet a way of automatically enter the optimal fit of a calibration in the LER config file. Find the optimal values for each model in cali_result and enter them in the config file (in the "scaling_factors" and "model_parameters" sections. Then run export_config and run_ensemble again, to get a calibrated model run in "output/ensemble_output.nc".

First manually enter calibration values into "LakeEnsemblR.yaml", then run:

```
export_config("LakeEnsemblR.yaml", model = c("FLake", "GLM", "GOTM",
                                             "Simstrat", "MyLake"))
run_ensemble("LakeEnsemblR.yaml", model = c("FLake", "GLM", "GOTM",
                                            "Simstrat", "MyLake"))
```

7. Adding ensemble members to the NetCDF

Running multiple models is one way of creating an ensemble. Another option is to run the same model multiple times, but with different forcings, initial conditions, and/or parameter values. And a combination of both or all is possible as well, of course.

LakeEnsemblR allows multiple runs of the same models to be stored in the output netCDF file, by the "add" argument in the `run_ensemble()` function. Let's try it out by increasing the light extinction coefficient.

```
  input_yaml_multiple("LakeEnsemblR.yaml", value = 2.0,
                      key1 = "input", key2 = "light", key3 = "Kw")

  # Now run export_config and run_ensemble again, but add "add = TRUE"
  export_config("LakeEnsemblR.yaml", model = c("FLake", "GLM", "GOTM",
                                               "Simstrat", "MyLake"))
  run_ensemble("LakeEnsemblR.yaml",
               model = c("FLake", "GLM", "GOTM", "Simstrat", "MyLake"),
```

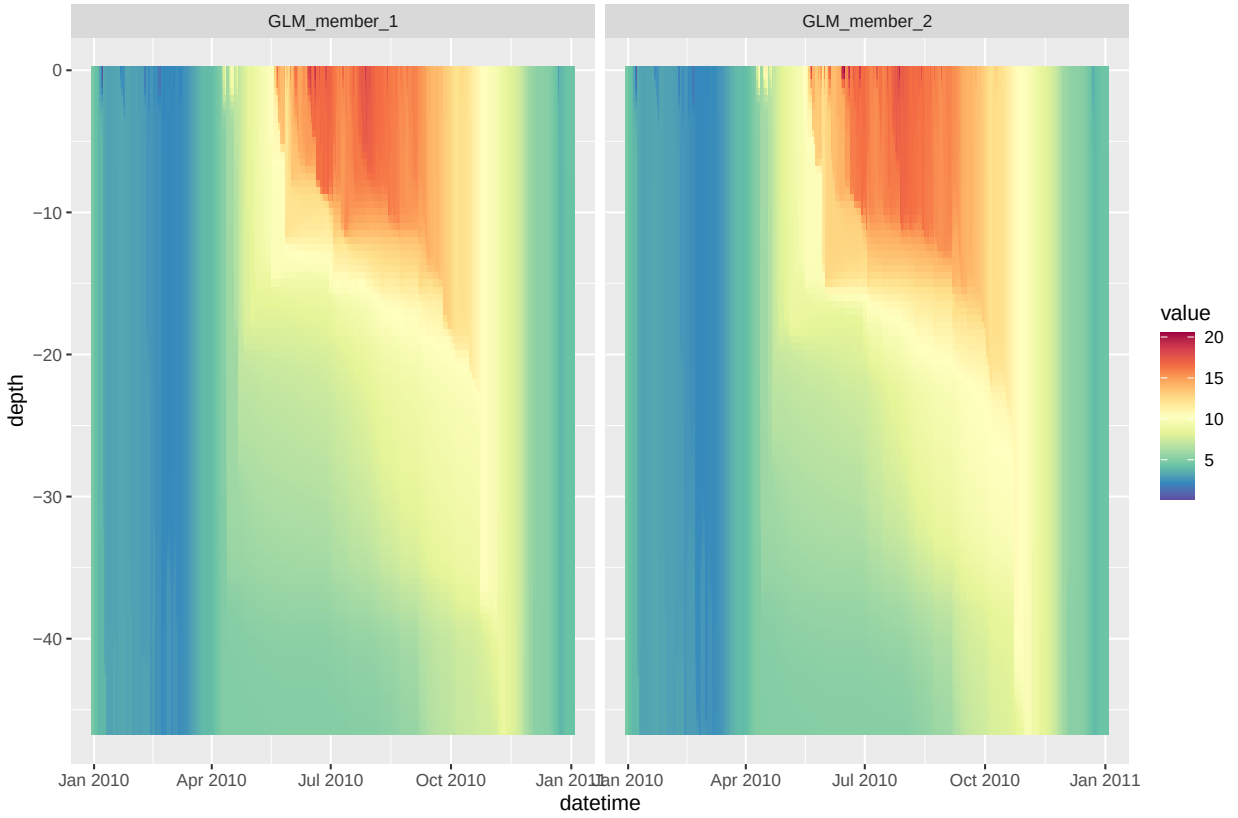
```
                parallel = TRUE,
                add = TRUE)
```

In PyNcView you can now also change the "member" counter, to view multiple ensemble members of the same model

You can plot multiple members of the same model by using the "dim" argument. Change dim_index to plot different models (1 = FLake, 2 = GLM, etc.)

```
plot_heatmap("output/ensemble_output.nc", dim = "member", dim_index = 2)
```

```
## Extracted temp from output/ensemble_output.nc
```



This also works with the `plot_ensemble()` funciton or the `calc_fit()` funciton.

8. Post-processing with LakeEnsemblR

There are a variety of functions that allow you to quickly analyse the standardised output from running LER. There are functions for calculating model performance metrics and key indices such as stratification timing, ice on/off dates and max/min surface and bottom temperatures.

```
out_res <- analyse_ncdf(ncdf = "output/ensemble_output.nc",
                        model = c("FLake", "GLM", "GOTM","Simstrat", "MyLake"))
```

This will return a list with the following entries:

```
names(out_res)
```

```
## [1] "out_df"   "stats"    "strat"    "obs_temp"
```

```
print(out_res[["stats"]])
```

```
##      model Pearson_r Variance_obs Variance_mod   SD_obs   SD_mod Covariance
## 1    FLake 0.8182917     20.90594     17.42231 4.573214 4.174838   15.61694
## 2      GLM 0.9413165     17.35083     15.86935 4.165881 3.984064   15.61980
## 3     GOTM 0.9624858     17.35899     14.57863 4.166858 3.818606   15.31140
## 4 Simstrat 0.8521864     17.35899     13.51991 4.166858 3.677337   13.05520
## 5   MyLake 0.9206529     17.35899     11.86000 4.166858 3.444204   13.20993
##        Bias      MAE     RMSE       NSE lnlikelihood
## 1 -2.050909 2.093012 3.361637 0.4594547    -16462.52
## 2 -1.821137 1.853053 2.301548 0.6947049    -16603.15
## 3 -2.085391 2.110394 2.379848 0.6737323    -17504.88
## 4 -1.243752 1.977627 2.513048 0.6361880    -19025.69
## 5 -1.922304 1.958271 2.548408 0.6258777    -19443.33
```

```
print(out_res[["strat"]])
```

```
##       model year     TsMax TsMaxDay    TsMin TsMinDay       TbMax TbMaxDay
## 1       obs 2010 17.678333      179 3.485000       13   12.760635      292
## 2       obs 2011  4.526667        0 4.526667        0    4.166299        0
## 3     FLake 2010 17.169500      210 3.303560       56    6.450840      330
## 4       GLM 2010 18.943432      172 0.224534        9    8.881755      315
## 5       GLM 2011  4.416234        0 4.416234        0    4.341768        0
## 6      GOTM 2010 15.545344      210 1.910718       53   12.105916      276
## 7      GOTM 2011  3.604110        0 3.604110        0    3.601177        0
## 8  Simstrat 2010 18.716999      210 3.418900       54 6165.299805      300
## 9  Simstrat 2011  7.509500        0 7.509500        0    7.362700        0
## 10   MyLake 2010 17.312641      209 1.932706        8   10.061987      305
## 11   MyLake 2011  4.609805        0 4.609805        0    4.609805        0
##        TbMin TbMinDay MaxStratDur MeanStratDur TotStratDur StratStart StratEnd
## 1   3.463672       13         184       184.00         184        105      289
## 2   4.166299        0          NA           NA          NA         NA       NA
## 3   3.457870       12         211       211.00         211        107      318
## 4   2.341928       57         195        50.50         202        117      312
## 5   4.341768        0          NA           NA          NA         NA       NA
## 6   2.147178       70         133        34.75         139        136      269
## 7   3.601177        0          NA           NA          NA         NA       NA
## 8  -2.025500      303         185        47.25         189        106      291
## 9   7.362700        0          NA           NA          NA         NA       NA
## 10  3.169321       56         205        53.75         215        100      305
## 11  4.609805        0          NA           NA          NA         NA       NA
##    StratFirst StratLast MeanIceDur MaxIceDur TotIceDur IceOn IceOff FirstFreeze
## 1         105       289         NA        NA        NA    NA     NA          NA
## 2          NA        NA         NA        NA        NA    NA     NA          NA
## 3         107       318         NA        NA        NA    NA     NA          NA
## 4         102       312         NA        NA        NA    NA     NA          NA
## 5          NA        NA         NA        NA        NA    NA     NA          NA
## 6         116       269         NA        NA        NA    NA     NA          NA
## 7          NA        NA         NA        NA        NA    NA     NA          NA
## 8         106       313         NA        NA        NA    NA     NA          NA
## 9          NA        NA         NA        NA        NA    NA     NA          NA
## 10        100       327         NA        NA        NA    NA     NA          NA
## 11         NA        NA         NA        NA        NA    NA     NA          NA
##    LastThaw HiceMax HiceMaxDay
```

```
## 1        NA       0         NA
## 2        NA       0         NA
## 3        NA       0         NA
## 4        NA       0         NA
## 5        NA       0         NA
## 6        NA       0         NA
## 7        NA       0         NA
## 8        NA       0         NA
## 9        NA       0         NA
## 10       NA       0         NA
## 11       NA       0         NA
```

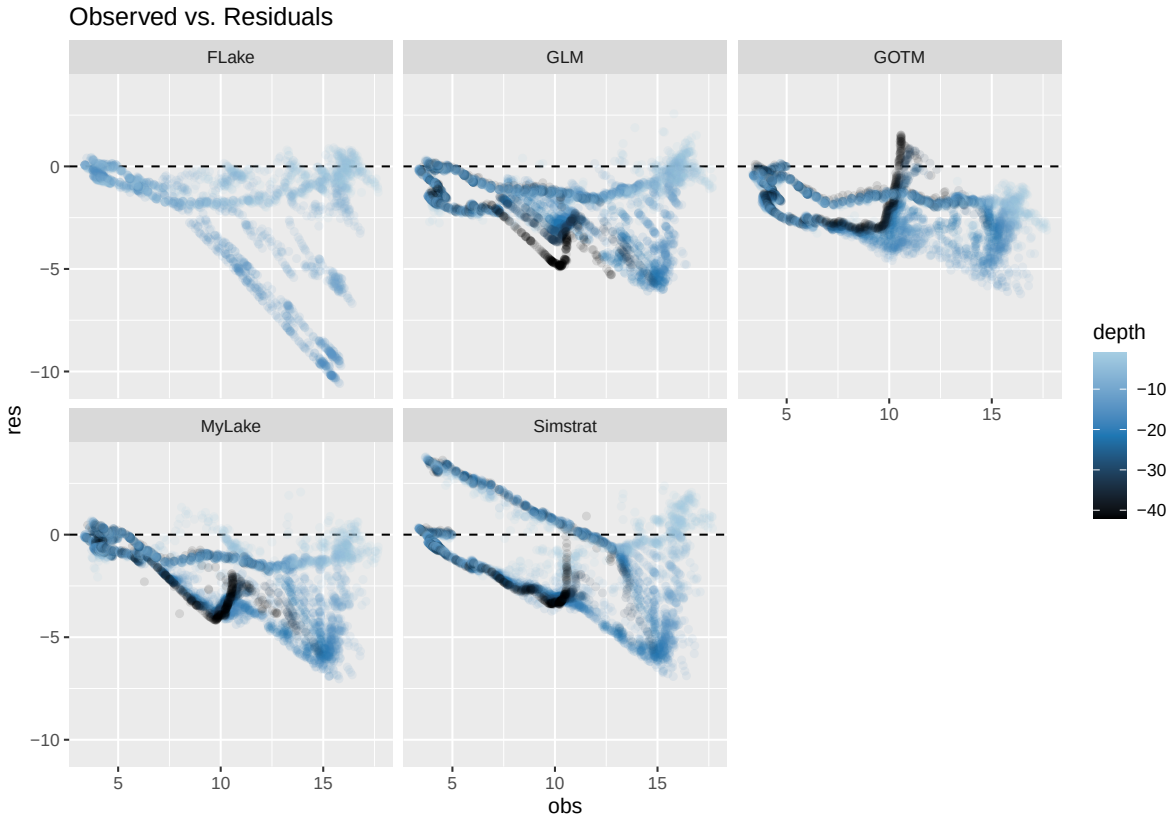
Calculate model performance statistics, e.g. RMSE (Root Mean Square Error) or Pearson's r

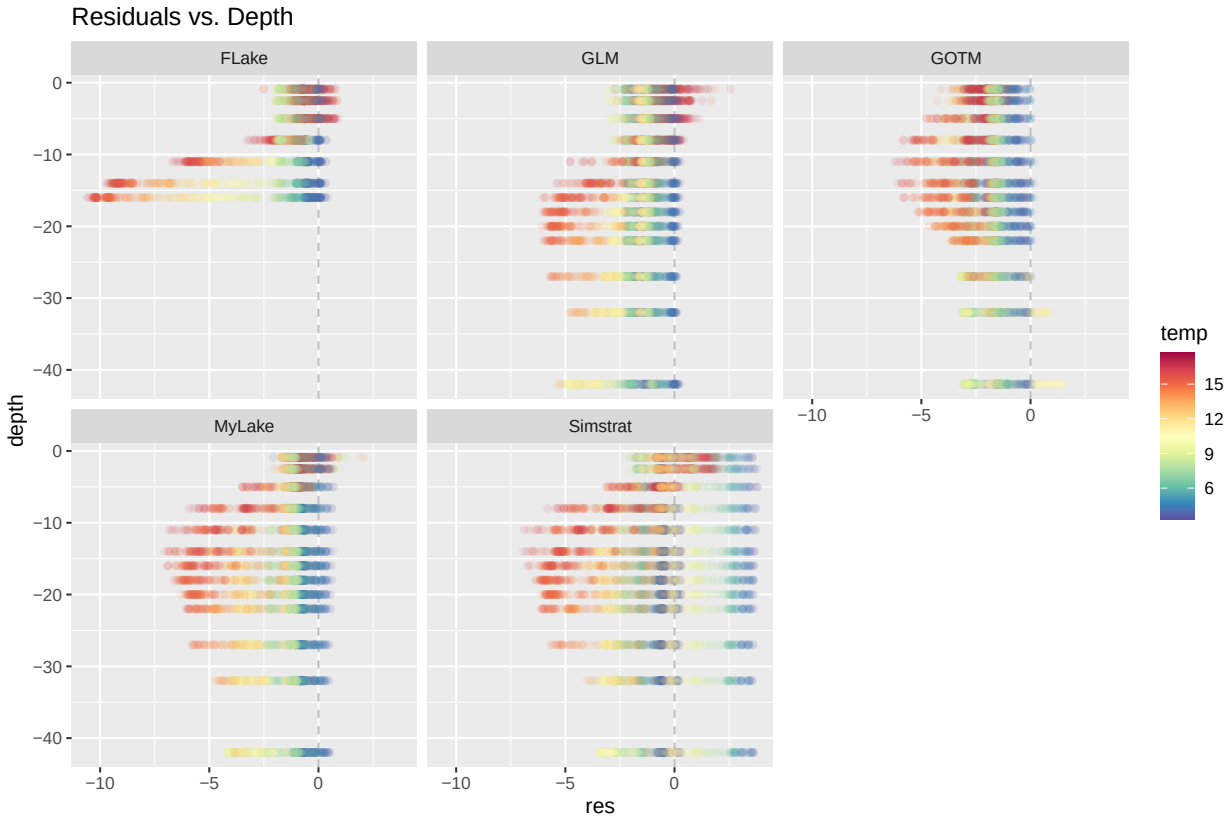
```
calc_fit(ncdf = "output/ensemble_output.nc", model = c("FLake", "GLM", "GOTM",
                                                       "Simstrat", "MyLake"))
```

```
## $FLake
##       rmse       nse         r      bias      mae     nmae
## 1 3.361637 0.6504408 0.6439385 -2.050909 2.093012 0.186658
##
## $GLM
##       rmse       nse         r      bias      mae      nmae
## 1 2.301548 0.6956982 0.9235715 -1.821137 1.853053 0.2115899
##
## $GOTM
##       rmse       nse         r      bias      mae      nmae
## 1 2.379848 0.6737323 0.9464245 -2.085391 2.110394 0.2464098
##
## $Simstrat
##       rmse      nse         r      bias      mae      nmae
## 1 2.513048 0.636188 0.8391085 -1.243752 1.977627 0.2215394
##
## $MyLake
##       rmse       nse         r      bias      mae      nmae
## 1 2.548408 0.6258777 0.9059941 -1.922304 1.958271 0.1946865
##
## $ensemble_mean
##       rmse       nse         r      bias      mae      nmae
## 1 2.363191 0.6782835 0.9201633 -1.820601 1.890548 0.2014996
```

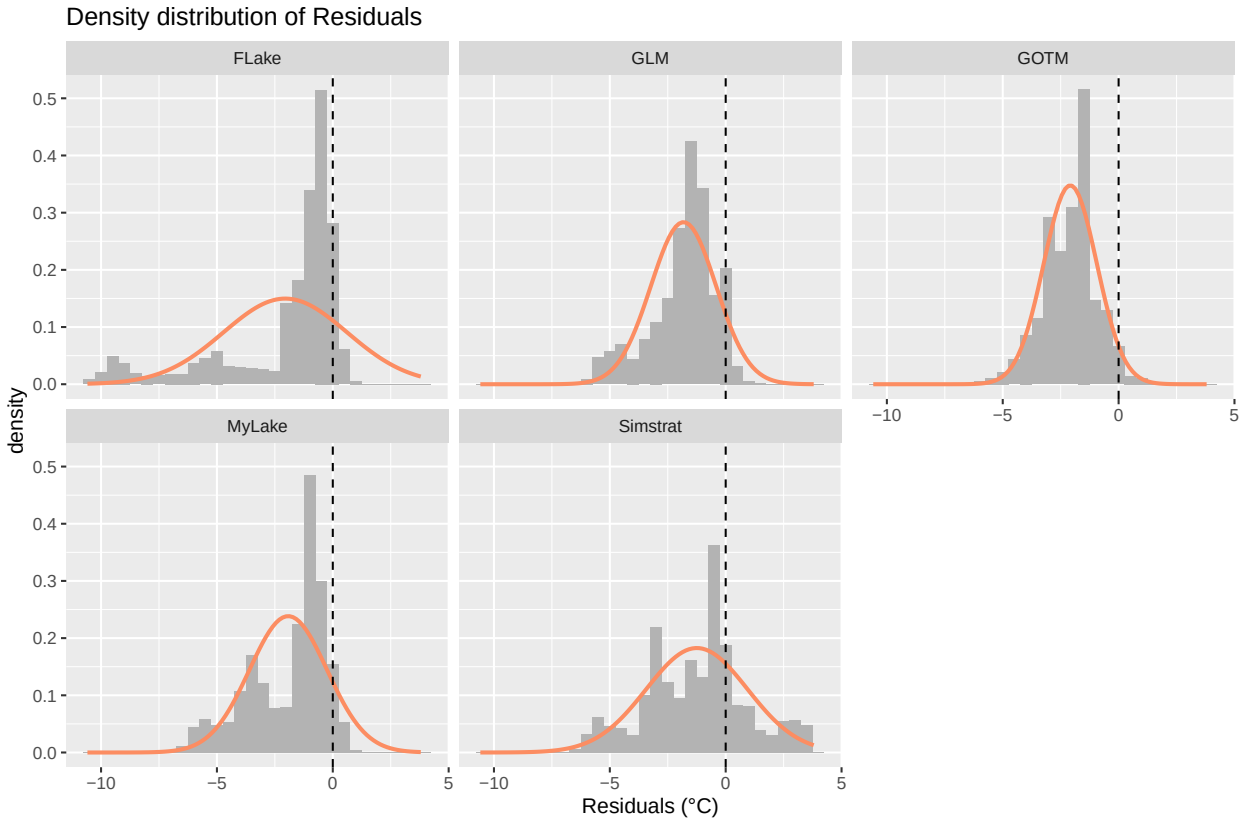
The `plot_resid()` function shows multiple plots that are helpful to discover where (time and space) the models perform well or poorly.

```
plot_resid(ncdf = "output/ensemble_output.nc", var = "temp")
```


Observed vs. Residuals
FLake
GLM
GOTM
MyLake
Simstrat
res
obs
depth
0
−5
−10
5
10
15
−10
−20
−30
−40


Residuals vs. Depth
FLake
GLM
GOTM
MyLake
Simstrat
depth
res
temp
0
−10
−20
−30
−40
−5
15
12
9
6


Year day vs. Residuals
FLake
GLM
GOTM
MyLake
Simstrat
res
yday
depth
−10
−20
−30
−40


Density distribution of Residuals
FLake
GLM
GOTM
MyLake
Simstrat
density
Residuals (°C)

Loading data into R

For further analysis the data can be loaded into R. The default format is a list format, with each model being its own object in a list. This format is the same as is used in `rLakeAnalyzer` so allows for application of such functions to the data.

Calculating Schmidt Stability

So we can easily calculate and plot the Schmidt stability for every model:

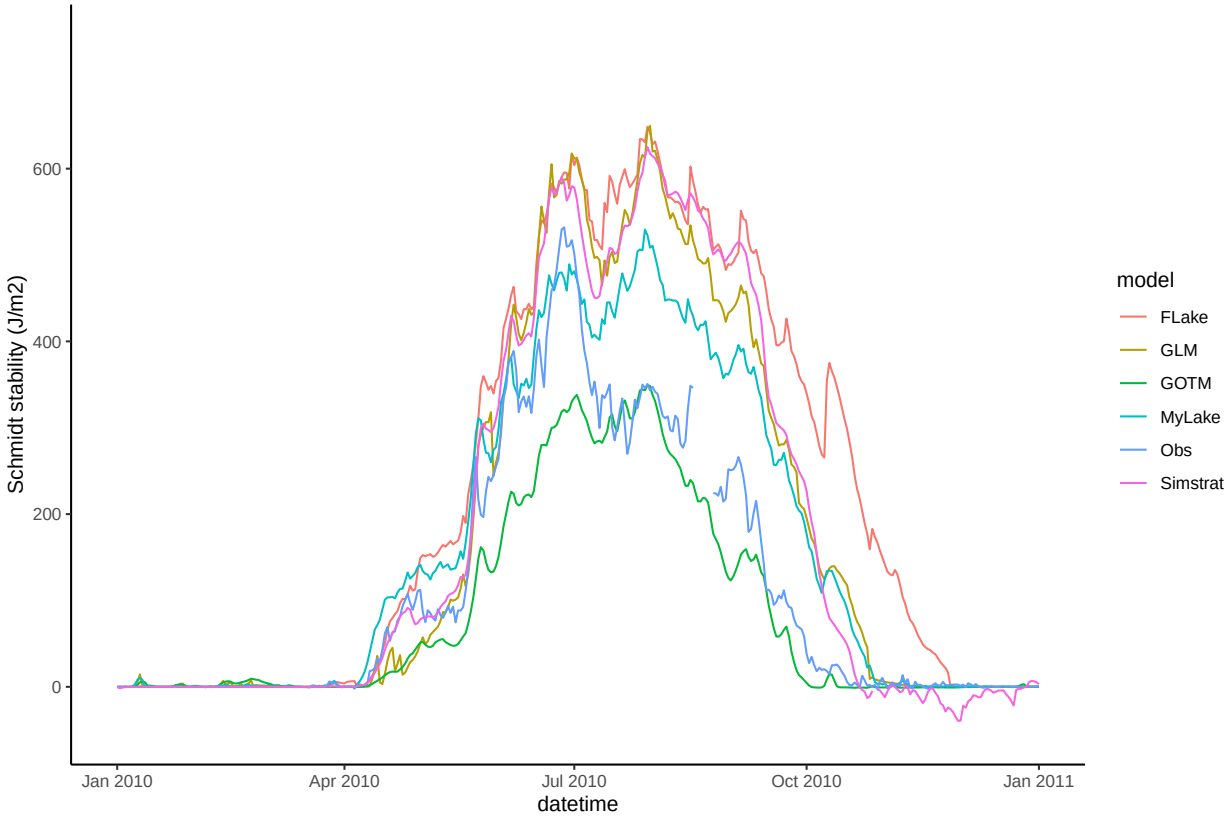
```
library(rLakeAnalyzer)
out <- load_var(ncdf = "output/ensemble_output.nc", var = "temp")
bathy <- read.csv('LakeEnsemblR_bathymetry_standard.csv')
colnames(bathy) <- c("depths", "areas")
ts.sch <- lapply(out, function(x) {
  ts.schmidt.stability(x, bathy = bathy, na.rm = TRUE)
})
```

Convert from a list object to a dataframe for plotting in ggplot

```
library(reshape)
df <- melt(ts.sch, id.vars = 1)
colnames(df)[4] <- "model"
```

Plot with ggplot

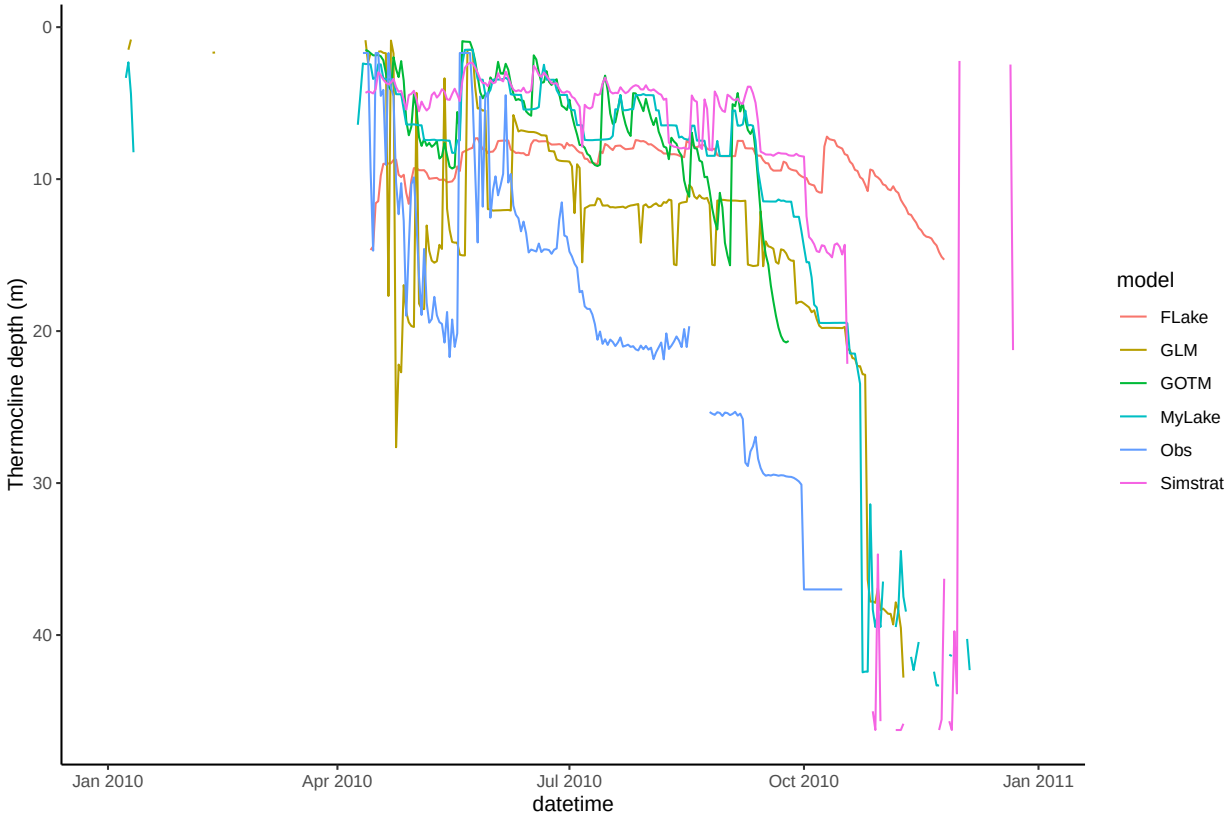
```
ggplot(df, aes(datetime, value, colour = model)) +
  geom_line() +
  labs(y = "Schmidt stability (J/m2)") +
  theme_classic() + ylim(-50, 750)
```



Calculating thermocline depth

In the same manner, we can also calculate the thermocline depth:

```
ts.td <- lapply(out, function(x) {
  ts.thermo.depth(x, Smin = 0.1, na.rm = TRUE)
})

df <- melt(ts.td, id.vars = 1)
colnames(df)[4] <- "model"

ggplot(df, aes(datetime, value, colour = model)) +
  geom_line() +
  labs(y = "Thermocline depth (m)") +
  scale_y_continuous(trans = "reverse") +
  theme_classic()
```



This workflow can be used with each of the functions within `rLakeAnalyzer`.

9. Apply LakeEnsemblR to your own data, or play around with the settings

In the next part of the workshop, you are encouraged to either apply LER to your own lake data or to one of the examples given on https://github.com/aemon-j/LER_examples. You can also try playing around with the settings for the Lough Feeagh test case (e.g. add varying light extinction, or try different calibration methods). The workshop leaders will be around to help with issues if needed. The LakeEnsemblR package provides template files for all files necessary to simulate your lake, they can be accessed by using the `get_template()` function, which will copy the template to your current working directory e.g.

```
get_template("Initial temperature profile")
```

The names of the available template files can be obtained by executing the `get_template()` function with no argument:

```
get_template()
```

```
## You need to specify what you want a template for.
## Options are:
## all
## LakeEnsemblR_config
## FLake_config
## GLM_config
## GOTM_config
## Simstrat_config
```

```
## MyLake_config
## Initial temperature profile
## Light extinction
## Inflow
## Hypsograph
## Ice height
## Meteo
## Temperature observations
```

At the end of the workshop some time is reserved to brainstorm potential uses for this package.