Session 2: SWAT simulation, calibration, and sensitivity analysis in R

Christoph Schürz *christoph.schuerz@boku.ac.at*

The goal of this second session is that you get familiar with the `SWATplusR` package. After the package installation we will load a demo SWAT project and explore the functionality of the `SWATplusR` package. The essential part of this session will be to learn how to utilize the package functionality and combine it with other `R` packages. In a small case study we will execute the SWAT model with different model parametrizations, extract simulation results, evaluate and visualize the simulation results (using the packages `HydroGOF` and the `tidyverse`), perform a parameter sensitivity analysis (employing packages such as `sensitivity` or `fast`), and perform a first model calibration.

Contents

Package installation

Package dependencies

Below you find a list of packages that are required to install `SWATplusR`. Among these packages you find tools that can be useful in your daily routine when analyzing data in `R` (e.g. most of the packages included in the `tidyverse`). Please install all the packages by executing the following lines of code.

```r
install.packages(c("doSNOW", "foreach", "RSQLite", "tidyverse"))
```

SWATplusR

You can install the `SWATplusR` package from my *github* repository (set to public for the course). To install from repositories with ease you can use the package `devtools`

```r
install.packages("devtools")

# use the function install_github from the devtools package to install
devtools::install_github("chrisschuerz/SWATplusR")
```

Additional packages

To perform our analyses we require further functionality provided by other `R` packages. Here I provide a list of packages you have to install. I briefly outline what each of these packages does for you.

`lhs`

This package provides different methods to draw Latin Hypercube Samples. We will sample the SWAT model parameters in the calibration example using latin hypercube sampling.

```r
install.packages("lhs")
```

`hydroGOF`

This package provides a comprehensive library of objective criteria used in hydrology (e.g. NSE, KGE, pbias, etc.) to evaluate time series of simulated discharge etc.

```r
install.packages("hydroGOF")
```

`sensitivity`

`sensitivity` provides a large variety of methods to perform Global Sensitivity Analysis (GSA, e.g. Sobol or Delsa).

```r
install.packages("sensitivity")
```

fast

The Fourier Amplitude Sensitivity Test (FAST) is a method to perform GSA with few model evaulations. This package impelements this method in `R`.

```
install.packages("fast")
```

Load required R packages

```r
library(SWATplusR)
library(tidyverse)
library(lubridate)
library(forcats)
library(lhs)
library(fast)
library(sensitivity)
library(hydroGOF)
```

Loading the SWAT demo

Load the project folder

The `SWATplusR` package provides very simple model setups of a head watershed of the Little River Experimental Watershed (LREW). Model setups can be retrievew for SWAT2012 and for SWAT+. The goal is to provide the demos for all operating systems in the future. At the moment the SWAT2012 demo is available for Windows and Linux and the SWAT+ demo is only available for Windows. For this workshop this means that you can work with SWAT+ if you use Windows as your operating system. Linux users have to work with SWAT2012 at the moment. The provided functionality is however very similar for both models.

```r
# The path where the SWAT demo project will be written
demo_path <- "C:"
# The SWAT version you want to use
swat_version <- "plus" #or "2012" on Linux
# The function writes the demo folder to the defined path and returns the final
# path of the project folder in R
proj_path <- load_demo(dataset = "project",
                       swat_version = swat_version,
                       path = demo_path)
```

Load observation data

The demo contains observation time series data for the main outlet of the demo catchment. Load the table with the observation data and assign it to a variable in your *R* work space.

```r
q_obs <- load_demo(dataset = "observation")
```

First SWAT simulations

After loading the demo project you can already perform your first SWAT simulation. To run SWAT from *R* you can simply use the functions `run_swat2012()` to run a SWAT2012 project or `run_swatplus()` to run a SWAT+ project. We will go though all the parameters these functions provide. The minimum requirement to run simulations in a project is to provide the path to the project and what simulation output the function should return to *R*. Below is a mimimum example to simulate the discharge at the main outlet.

```
q_out <- run_swatplus(project_path = proj_path,
                      output = define_output(file = "channel",
                                             variable = "flo_out",
                                             unit = 3))
```

The function returns the simulation at the catchment outlet as a table with a date column and a column with the simulated discharge.

Changing parameters in a simulation

A single parameter set

To modify the model parameters in a simulation you simply provide a vector with the parameter values to the `run_swat()` function. A single parameter set can be provided as a vector. To provide an entire set of parameter combinations I recommend to provide them as a tibble.

For the provided parameter set the naming is very essential and has to follow some rules in order to be interpreted by the function in the correct way. A parameter name can consist of several parts. Some are required, some are optional. The minimum requirement for a parameter name is the actual name in the model an the type of change. Here is an example

```
par_name <- "cn2|change = abschg"
```

The sytax means that we modify the parameter CN2 and we change it by adding an absolute value. If you want to assign an individual name to the parameter in oyur *R* project you can do this as follows:

```
par_name <- "my_name::cn2|change = abschg"
```

Below we define a single parameter set where we reduce all CN2 values by 5 and set the alpha value to 0.5.

```
par_single <- c("cn2.hru|change = abschg" = -5,
                "alpha.gw|change = absval" = 0.5)
```

With this parameter set we can again run our model

```
q_out <- run_swatplus(project_path = proj_path,
                      output = define_output(file = "channel",
                                             variable = "flo_out",
                                             unit = 3),
                      parameter = par_single)
```

Several parameter combinations

The definition of multiple parameter combinations works in a similar way. Now we have to define the parameters with a tibble.

```
par_set <- tibble("cn2.hru|change = abschg" = runif(8,-15,10),
                  "alpha.gw|change = absval" = runif(8, 0, 1))
```

To run SWAT simulations with many parameter sets the `SWATplusR` package provides the option to run the simulations in parallel. To perform parallel simulations you simply define the number of threads as demonstrated below.

```
q_out <- run_swatplus(project_path = proj_path,
                      output = define_output(file = "channel",
                                             variable = "flo_out",
                                             unit = 3),
                      parameter = par_set,
                      n_thread = 4)
```

Now instead of only providing a table with the simulations, the function returns by default a list that stores the parameter set that was used, a table that shows details of the parameter definition and the simulation results. As more than one simulation was performed, the columns for the simulation results are now named run_1 to run_8.

A simple case study

Sensitivity analysis with FAST

Sample parameters and perform simulations

In this example we apply the `fast` for sensitivity analysis. The Fourier Amplitude Sensitivity Test (FAST) is a method to perform GSA with few model evaulations. It only requires a few simulations when the number of parameters is low and sharply increases to tenth of thousands for more than 20 parameters. Therefore for our example we select only 7 parameters that are relevant in many model SWAT model applications.

The FAST method requires a specific parameter sampling design. Fortunately, the `fast` package provides a function to sample the model parameters.

```
par_names <- c("cn2.hru | change = abschg",
               "lat_ttime.hru | change = absval",
               "lat_len.hru | change = absval",
               "k.sol | change = pctchg",
               "awc.sol | change = pctchg",
               "esco.hru | change = absval",
               "canmx.hru | change = absval")


par_fast <- fast_parameters(minimum = c(-15,  0, 10, -50, -50, 0,  0),
                            maximum = c( 10, 10, 75,  50,  50, 1, 25),
                            names = par_names)
```

To perform a FAST analysis for 7 parameters 167 model evaluations are required. We run the SWAT model with the parameter set sampled with `fast` and evaluate the simulations. In our example we define a simulation period from 1993 to 2005 and skip the first three years of simulations for writing the outputs.

```r
q_fast <- run_swatplus(project_path = proj_path,
                       output = list(q_out = define_output(file = "channel",
                                                           variable = "flo_out",
                                                           unit = 3)),
                       parameter = par_fast,
                       start_date = "1993-01-01",
                       end_date = "2005-12-31",
                       years_skip = 3,
                       n_thread = 4)
```

Evaluate the simulations

For this simple example we use the NSE as objective criterion. The NSE function is available from the `hydroGOF` package. To evaualte the simulations we reduce the observation to the same time period.

```r
q_obs <- filter(q_obs, date >= ymd("1996-01-01"), date <= "2005-12-31")

nse_fast <- q_fast$simulation$q_out %>%
  select(-date) %>%
  map_dbl(., ~NSE(.x, q_obs$q_out))

sens_fast <- sensitivity(nse_fast, 7)
```

Visualize the sensitivity analysis result

```r
result_fast <- tibble(parameter = q_fast$parameter$definition$par_name,
                      fast = sens_fast) %>%
  mutate(parameter = factor(parameter) %>% fct_reorder(., fast))
ggplot(data = result_fast) +
  geom_bar(aes(x = parameter, y = fast), stat = "identity") +
  xlab("Parameter") +
  ylab("Sensitivity") +
  coord_flip() +
  theme_bw()
```

It is clear from the sensitivity analysis, that the by far most dominant parameter is the lateral travel time *lat_ttime*. Other parameters that are sensitivite are lat_len, K, CN2, and ESCO. In the model calibration we will focus on these four parameters.

Dotty plots

Dotty plots can be helpful to show parameter ranges where the model showed a better performance. We can utilize the performed simulations to plot dotty plots for the 7 analyzed parameters. You can use this short code snippet for other applications in future when you think that dotty plots are applicable.

```r
dotty_data <- par_fast %>%
  mutate(nse = nse_fast) %>%
  gather(key = "parameter", value = "value", -nse)

ggplot(data = dotty_data) +
  geom_point(aes(x = value, y = nse)) +
  theme_bw() +
  ylim(c(-5, 1)) +
  facet_wrap(parameter ~ ., scales = "free_x" )
```

The dotty plots for our example show that

Model calibration

For demonstration we will perform the model calibration with only a low number of model evaluations (So do not expect very good results :)). We sample the selected influential parameter using Latin Hypercube Sampling available from the `lhs` package. To keep the computation time low, we will only perform 500 simulations. ### Parameter sampling The dotty plots indicate that good model results can only be achieved with low lat_ttime values. Therefore we will constrain its parameter range in the calibration. All other parameters do not show a very clear pattern. Therefore we will keep their ranges unchanged.

```r
par_bound <- tibble("cn2.hru | change = abschg" = c(-15, 10),
                    "lat_ttime.hru | change = absval" = c(0, 3),
                    "lat_len.hru | change = absval" = c(10, 50),
                    "k.sol | change = pctchg" = c(-50, 50),
                    "esco.hru | change = absval" = c(0, 1))

lhs_samp <- randomLHS(n = 500, k = 5) %>% as_tibble(.)

par_cal <- map2_dfc(par_bound, lhs_samp, ~ (.y * (.x[2] - .x[1]) + .x[1]))
```

Model simulation and evaluation

```r
q_cal <- run_swatplus(project_path = proj_path,
                      output = list(q_out = define_output(file = "channel",
                                                          variable = "flo_out",
                                                          unit = 3)),
                      parameter = par_cal,
                      start_date = "1993-01-01",
                      end_date = "2005-12-31",
                      years_skip = 3,
                      n_thread = 4)
```

```r
nse_cal <- q_cal$simulation$q_out %>%
  select(-date) %>%
  map_dbl(., ~NSE(.x, q_obs$q_out))
```

Visualization - Dotty plots

```r
dotty_data <- par_cal %>%
  mutate(nse = nse_cal) %>%
  gather(key = "parameter", value = "value", -nse)

ggplot(data = dotty_data) +
  geom_point(aes(x = value, y = nse)) +
  theme_bw() +
  ylim(c(-5, 1)) +
  facet_wrap(parameter ~ ., scales = "free_x" )
```

Visualization - Simulated time series

```r
run_select <- names(nse_cal[nse_cal > -0.2])

cal_select <- q_cal$simulation$q_out %>%
  select(one_of(run_select)) %>%
  transmute(q_max = pmap_dbl(., max),
            q_min = pmap_dbl(., min),
            q_sim = rowMeans(.))

cal_plot <- bind_cols(q_obs, cal_select) %>%
  gather(key = "variable", value = "discharge", -date, -q_min, - q_max)


ggplot(data = cal_plot) +
  geom_ribbon(aes(x = date, ymin = q_min, ymax = q_max), alpha = 0.5) +
  geom_line(aes(x = date, y = discharge, col = variable)) +
  xlab("Date / yyyy") +
  ylab("Discharge / m ^3 s^-1") +
  theme_bw()
```