OPTAIN CS-10

Moritz Shore, Csilla Farkas

2023-06-09

Contents

Introduction

The aim of page is to:

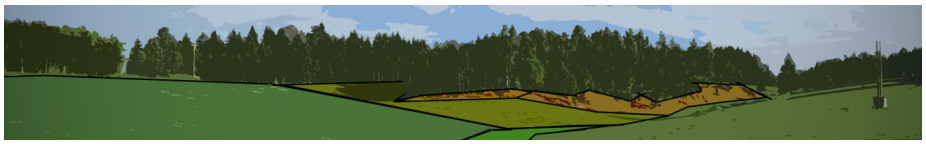
1. **Document** the process of setting up the SWAT+ and SWAP models for Kraakstad, Norway (Case study 10 / CS10)
2. **Link and execute entire workflow** in parallel with the documentation using `Rmarkdown`
3. Make the model setup process **reproducible**
4. Be a **reference** to the CS10 modelers, as well as other OPTAIN modelers.

If all goes well, the compiling of this GitBook will also compile the OPTAIN model setup in tandem, from source input files to final model setup (..and beyond!)

The model setup follows the OPTAIN modelling protocols for SWAP (**?**) and SWAT+ (**?**). We begin with SWAT+ and the input file preparation for SWATBuildR, in Section **??**.

Useful links:

- NIBIO project website
- NIBIO GitLab hosting model files (restricted access)
- UFZ GitLab hosting OPTAIN related software and data (restricted access)
- OPTAIN project website
- OPTAIN Zenodo hosting various files and documents
- SWATbuildR (restricted), SWATfarmR, SWATdoctR (restricted), SWATplusR, R-packages core to the OPTAIN workflow
- svatools, an R-package used in the OPTAIN workflow



Chapter 1

Input Data Preparation

This Section covers the input data preperation fror the SWATbuildR. The actual creation of the input files will not be covered, since these steps were performed before the documentation project, by a different team member who is no longer active in the project. These documentation gaps may be filled over time. The main content of this page will contain any **changes** we have made to the aforementioned input files.

We will need the following libraries.

```
require(raster)
require(sf)
require(ggplot2)
require(mapview)
require(dplyr)
require(sp)
require(terra)
require(gifski)

# Common ggplot theme for simple features
sf_theme <- theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank()
        )
```

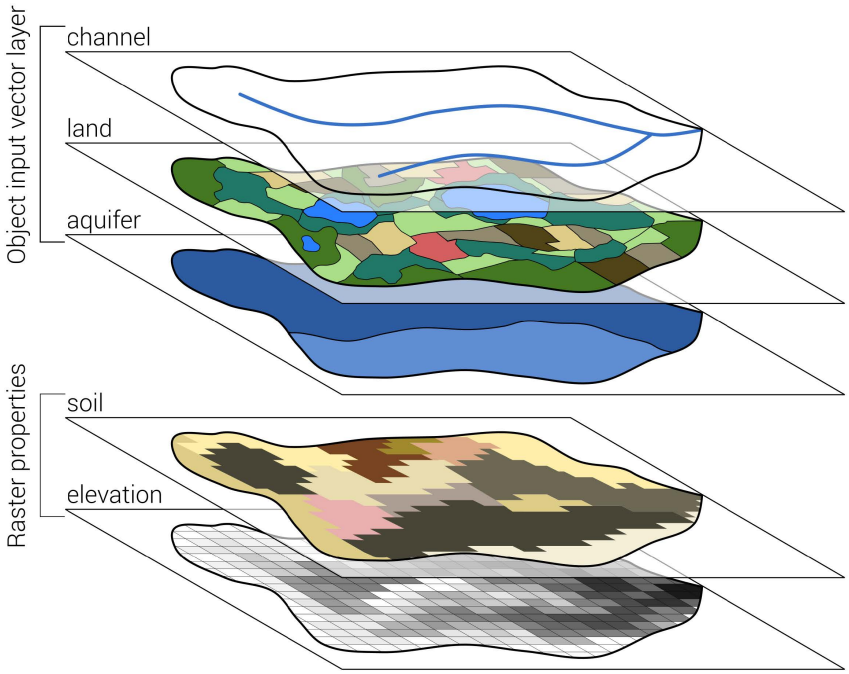


Figure 1.1: Required vector and raster inputs for a SWAT+ model setup with SWATbuildR @schürz2022

1.1 Digital Elevation Model

The DEM is sourced from hoydedata.no and has a resolution of 10m (unconfirmed). 1m Resolution was available but not used.

```
dem_path <- "model_data/input/elevation/dtm3_ns_v5.tif"
```

```
dem_rast <- raster(dem_path)
plot(dem_rast)
```

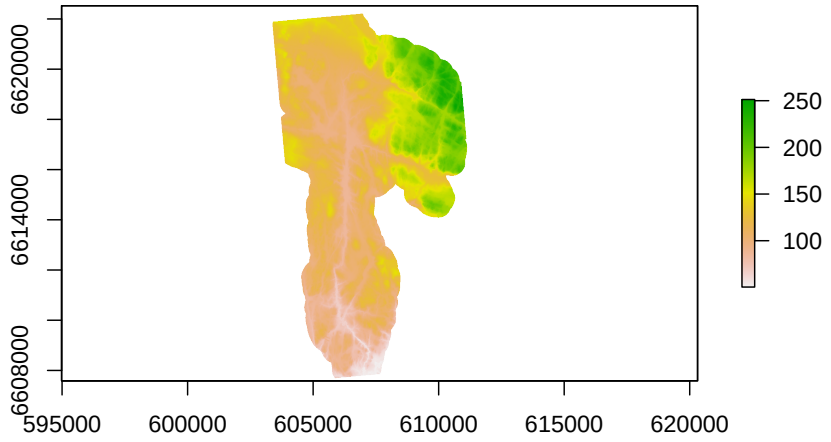


Figure 1.2: CS10 Digital elevation model (DEM).

To our knowledge, it has a 10 meter resolution. 1 meter resolution was available but there seems to have been issues with using it. It is definitely preferable to use the 1m DEM as certain important information can be lost with (max allowed resolution) of 10m.

1.2 Soil Data

Required are:

1. a GeoTiff raster layer that defines the spatial locations of the soil classes (with integer ID values).

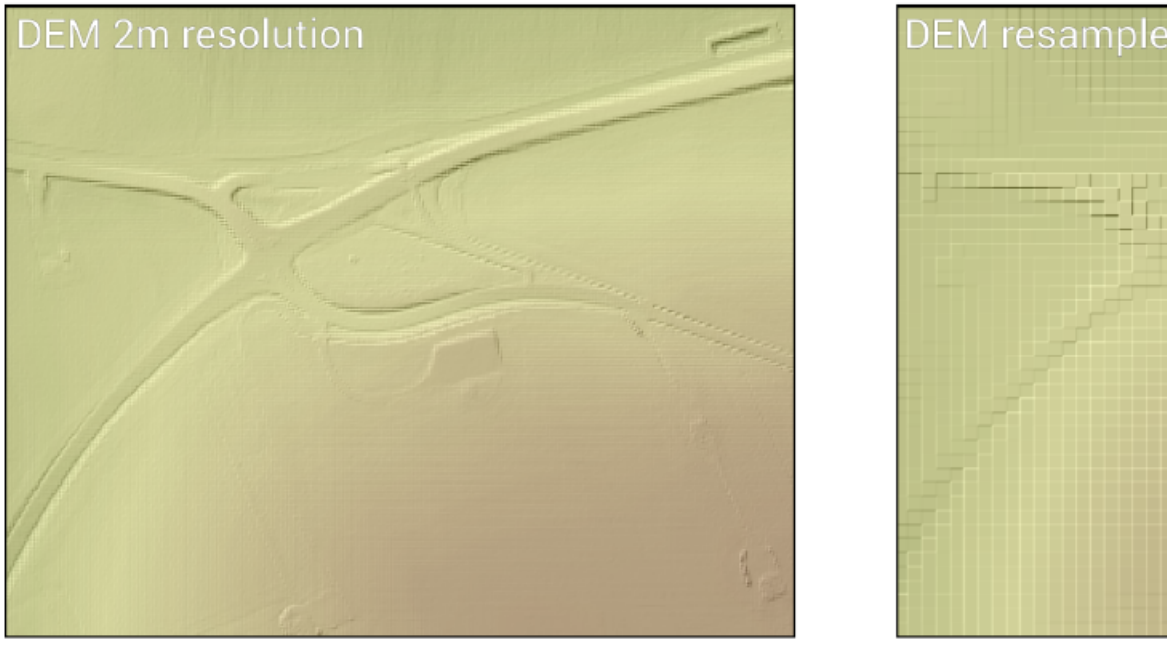


Figure 1.3: An example of a hydrologically effective landscape features being lost due to coarse DEM resolution. From @schürz2022

```
"model_data/input/soil/soil_layer.tif"
```

2. a lookup table in .csv format, which links the IDs of the soil classes with their names

```
"model_data/input/soil/soil_lookup.csv"
```

3. a usersoil table in .csv format, which provides physical and chemical parameters of all soil layers for each soil class that is defined in the raster layer and the lookup table. An example soil dataset can be acquired e.g. from the ExampleDatasets folder which comes with an installation of SWAT+.

```
"model_data/input/soil/UserSoil_Krakstad.csv"
```

A high resolution soil map is required to accurately represent the Hydrologic Response Units (HRUs) used in the OPTAIN project:

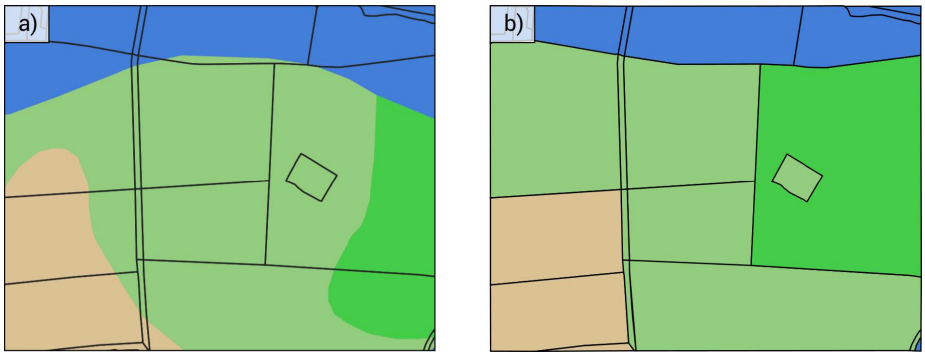


Figure 1.4: Spatially adequate representation of soil heterogeneity on field plot scale. The different colours represent different soil classes. The black border lines represent the boundaries of spatial objects (e.g. agricultural fields). a) the original soil input layer. b) the dominant soil aggregation as performed by SWATbuildR. @schürz2022

> The use of soil information on a scale which is much more detailed than the scale of the spatial objects in a model setup is however not recommended. It might be difficult to provide parameters to the usersoil table for less common soil types. Moreover, by default SWATbuildR uses the dominant soil class for each spatial object in the model setup process (see Figure 2.3b). Thus, a great share of the detailed information would get lost during the model setup process. One exception from that rule may be soil physical data which are available in a raster format

Our soil map is created from merging two separate datasets, the first being **Jordsmonn** (Viken) from NIBIO and **Loesmasser** from NGU. We don't currently know how they were combined to create the final soil map. It is located here

```
soil_path <- "model_data/input/soil/soil_map_UTM32N.shp"
```

Lets have a look:

```
soil_shp <- read_sf(soil_path)
soil_plot <- ggplot(soil_shp) + geom_sf() + sf_theme
print(soil_plot)
```

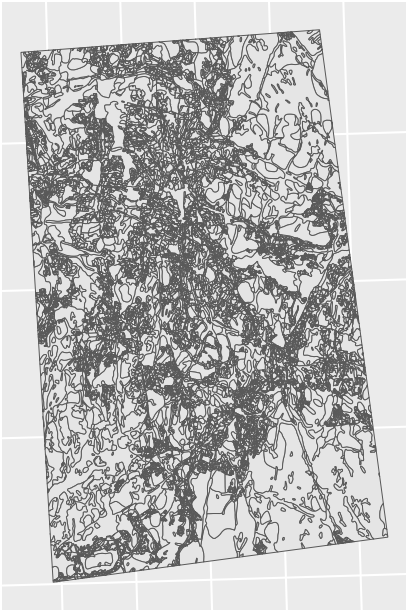


Figure 1.5: Soil map for CS10

1.3 Watershed Boundary

The watershed boundary must be a single polygon GIS vector layer. Presumed source of the data is NVE. It is located here:

```
basin_path <- "model_data/input/shape/cs10_basin.shp"
```

Preview:

```
basin_shp <- read_sf(basin_path)
basin_map <- mapview(basin_shp, alpha.region = 0, legend = FALSE)
basin_map
```

1.4 Land object input

Requirements:

1. column `id` number for each element
2. column `type` defining land use
3. column `drainage` containing the `id` of the channel recieving water from the drained field.

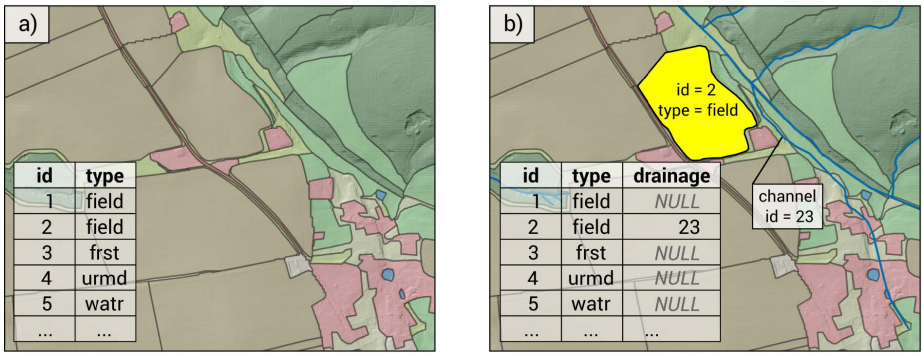


Figure 1.6: Land object polygons with attribute table. a) shows a land layer without tile drainage implemented. b) shows a case where the agricultural field with the land id = 2 has tile drainage activated and drains the tile flow into the channel id = 23 @schürz2022

The creation of the land object map was not documented. It is located here:

```
lo_path <- "model_data/input/land/CS10_LU.shp"
```

A preview:

```
lo_shp <- read_sf(lo_path)
lo_plot <- ggplot(lo_shp) + geom_sf(mapping = aes(fill = drainage)) + sf_theme +
  ggtitle("CS10 Landuse map", "Tile drainage indicated by ID of recieving channel")
print(lo_plot)
```

1.4.1 Data sources

Only partial documentation is available:

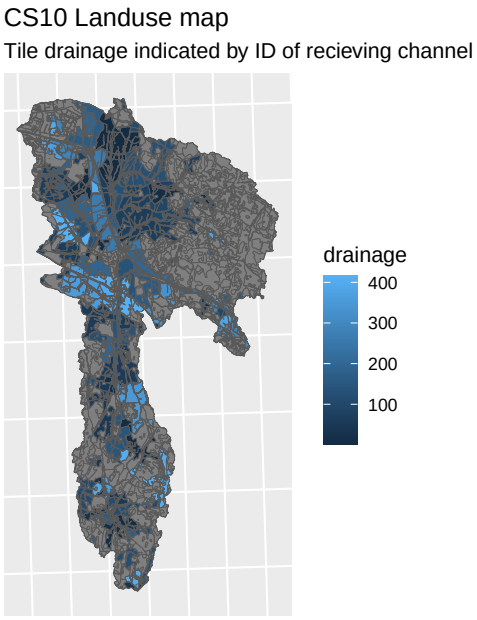


Figure 1.7: CS10 Landuse map

1. For Landuse, the Corine landcover 2018 (CLC) sourced from NIBIOs map service kart8 (details unknown)

2. Field boundaries were sourced from the norwegian property register, hosted on kartverket (details unknown)

1.4.2 Delineation of land objects

Currently not documented

1.4.3 Standing water bodies and the landuse type = 'watr'

Currently not documented

1.4.4 Tile drainage option

To determine which of our agricultural fields need drainage, we are going to use the Jordsmonn Map, specifically the Natural drainage capacity (Naturlige dreneringsforhold) described in column "JR_DREN". Documentation for this

dataset can be found here. The datasets have been clipped to the catchment area before this analysis (to reduce file sizes and computation time).

Discussion of this topic can be found in Issue #65

```
jordmonn <- "model_data/input/soil/jordsmonn.shp"
jm_shp <- read_sf(jordmonn)
```

There is a lot of junk in this file (89 columns of data), we only need `JR_DREN`

```
jm_shp_filt <- jm_shp %>% select(geometry, JR_DREN)
```

Classes 1,2 and 3 have the need for tile drains, whereas class 4 has enough natural drainage capacity to forgo tile drainage.

```
jm_drained <- jm_shp_filt %>% filter(JR_DREN != "4")
jm_not_drained <- jm_shp_filt %>% filter(JR_DREN == "4")
```

A comparison:

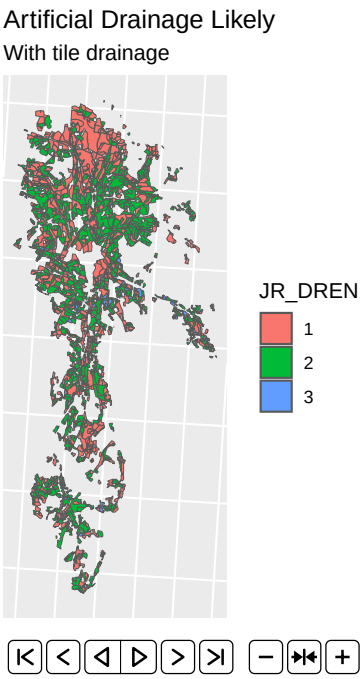


Figure 1.8: Tile drainage Classes

The drainage column of our land use map **has already been defined**, however every field regardless of its natural drainage class, has been assigned tile drains. We will now remove the tile drains which are located on class 4 soils.

First we will drop any unneeded data in our cluttered land object map.

```
lu_map <- lo_shp %>% select(geometry, id, type, drainage)
```

And convert our drainage map to the same projection.

```
jm_shp_filt <- st_transform(jm_shp_filt, st_crs(lu_map))
```

Then we will join the drainage class attribute spatially.

```
intersect_pct_4 <- st_intersection(lu_map, jm_shp_filt %>% filter(JR_DREN == "4")) %>%
   mutate(intersect_4 = st_area(.)) %>%   # create new column with shape area
   dplyr::select(id, intersect_4) %>%   # only select columns needed to merge
   st_drop_geometry()
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

```
intersect_pct_not_4 <- st_intersection(lu_map, jm_shp_filt %>% filter(JR_DREN != "4")) %>%
   mutate(not_4 = st_area(.)) %>%   # create new column with shape area
   dplyr::select(id, not_4) %>%   # only select columns needed to merge
   st_drop_geometry()
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

Total the individual intersectoins

```
class_4 <- intersect_pct_4 %>% group_by(id) %>% summarise(class_4 = sum(intersect_4))
not_class_4 <- intersect_pct_not_4 %>% group_by(id) %>% summarise(not_class_4 = sum(not_4))
```

Left join them to our land use map by their ID

```
lu_drains <- left_join(lu_map, class_4, by = "id")
lu_drains <- left_join(lu_drains, not_class_4, by = "id")
```

Set NA values to 0

```
lu_drains$class_4[which(lu_drains$class_4 %>% is.na())] = 0
lu_drains$not_class_4[which(lu_drains$not_class_4 %>% is.na())] = 0
```

Predefining our natural drainage attribute as false and defining our generic land use classes which are to be excluded from the manipulation.

```
lu_drains$natural_drain = FALSE
generics <- c("frst","rngb","urml","utrn","past", "watr", "wetf")
```

```
lu_drains$natural_drain[which((lu_drains$class_4 >= lu_drains$not_class_4) &
                              (lu_drains$type %in% generics == FALSE))] = TRUE
```

How many drains are we removing?

```
removed_len <- which((lu_drains$class_4 >= lu_drains$not_class_4) &
                       (lu_drains$type %in% generics == FALSE)) %>% length()
removed_len
```

```
## [1] 238
```

Seems like a lot, but how many drained HRUs are there in total?

```
total_drains <- lu_map %>% filter(drainage > 0) %>% pull(drainage) %>% length()
total_drains
```

```
## [1] 1947
```

```
paste(((removed_len / total_drains) * 100) %>% round(1),
      "% of drained agricultural HRUs will have their drains removed.")
```

```
## [1] "12.2 % of drained agricultural HRUs will have their drains removed."
```

Time to remove:

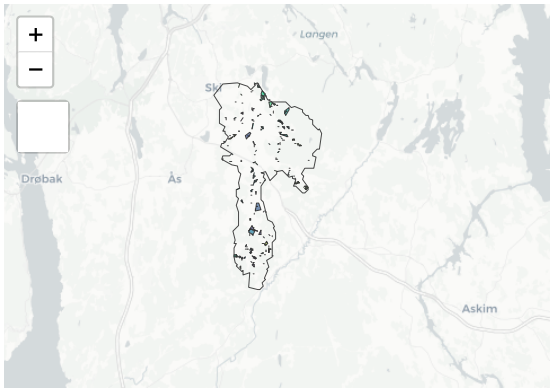
```
lu_drains$drainage[which(lu_drains$natural_drain == TRUE)] = NA
```

And drop the columns we don't need

```
lu_map_v2 <- lu_drains %>% select(geometry, id, type, drainage)
```

Which fields are naturally drained?

```
basin_map + mapview(lu_drains %>% filter(natural_drain), zcol = "type", legend = FALSE)
```



Our land use map now looks like this. The Number value points to the channel being drained to.

```
lu_v2 <-
  ggplot(lu_map_v2) + geom_sf(mapping = aes(fill = drainage)) + sf_theme +
  ggtitle("CS10 Landuse map",
          "Tile drainage indicated by ID of recieving channel")
```

```
print(lo_plot)
```

```
print(lu_v2)
```

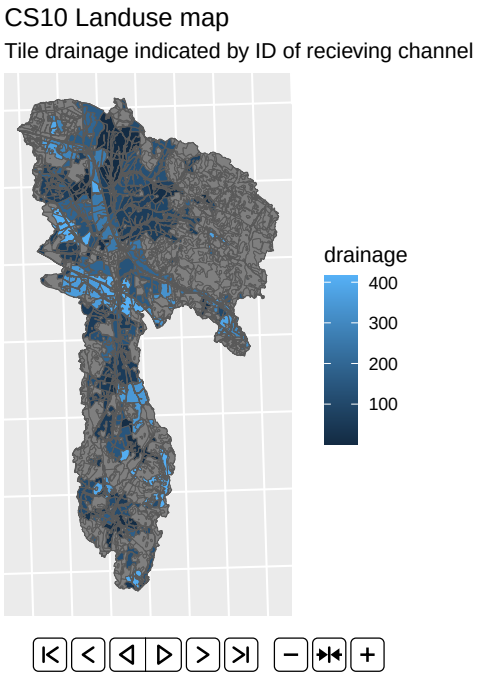


Figure 1.9: Tile drainage which was removed

Time to save the changes:

```
write_sf(lu_map_v2, "model_data/input/land/CS10_LU_V2.shp")
```

And cleanup

```
rm(list = ls())
```

1.4.5 Existing and potential structures to retain water and nutrients

Currently not documented

1.5 Channel object input

Limited documentation available:

Stream-Reach vector was created using QSWAT

1.5.1 Data sources and data processing

Currently not documented

1.6 Point sources locations

Currently not documented

1.7 Aquifer Objects

Currently not documented

Chapter 2

Model setup with SWATBuildR

```
# run the buildR?
run_buildr = FALSE
if (run_buildr) {
  print(paste("Code executed @", Sys.time()))
} else{
  datemod <-
    file.info("model_data/cs10_setup/optain-cs10/optain-cs10.sqlite")
  print(paste("BuildR last run @", datemod$mtime))
}
```

```
## [1] "BuildR last run @ 2023-06-09 11:34:43.301388"
```

The OPTAIN project is using the COCOA approach (**?**) and as such, needs to use the `SWATBuildR` package to build the SWAT+ model setup and calculate the connectivity between HRUs in the catchment. This chapter covers this process.

2.1 BuildR Setup

We will need the following packages for this chapter:

```
# not loading most packages because they interfere with the `install_load`
# functions of BuildR. this means I need to prefix everything with :: until
# Christoph turns BuildR into a proper package with proper scoping.
```

```
# require(dplyr)
# require(ggplot2)
# require(mapview)
# require(sf)
# require(raster)
# require(terra)
# require(raster)

# Common ggplot theme for simple features
sf_theme <- ggplot2::theme(axis.text.x=ggplot2::element_blank(),
        axis.ticks.x=ggplot2::element_blank(),
        axis.text.y=ggplot2::element_blank(),
        axis.ticks.y=ggplot2::element_blank()
        )
```

We will define the location of our project here, and give it a name:

```
project_path <- 'model_data/cs10_setup'
project_name <- 'optain-cs10'

# load the BuildR
source('model_data/swat_buildR/init.R')
# Note the many "maskings" occurring here, dangerous!
```

2.2 Processing input data

SWATBuildR reads input data, performs some checks on it, and saves it in a compatible format for subsequent calculations. Little documentation exists on the creation of these data sets. IF you have any information to add, feel free to do so!

2.2.1 High-Resolution Digital Elevation Model (DEM)

The high-resolution DEM is the basis for calculation of water connectivity, among other things. Most of the documentation of the creation of the DEM for CS10 has been lost to the sands of time. All we know is that it is located here:

```
dem_path <- "model_data/input/elevation/dtm3_ns_v5.tif"
```

2.2.2 Processing the Basin Boundary

The basin boundary has presumably been created using the defined outlet point of the catchment and the DEM. No more is currently known about this file other than that it is located here:

```
bound_path <- "model_data/input/shape/cs10_basin.shp"
```

```
bound_sf <- sf::read_sf(bound_path)
basin <- ggplot2::ggplot(bound_sf) + ggplot2::geom_sf() + sf_theme
print(basin)
```

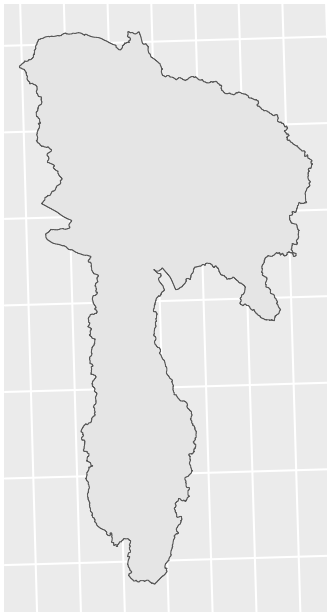


Figure 2.1: CS10 Basin, caclulated from the DEM.

The following code and commentary is from BuildR Script version `1.5.14`, written by Christoph Schuerz.

BuildR recommends all layers to be in the same CRS, if we set `project_layer` to `FALSE`, it will throw an error when this is not the case:

> The input layers might be in different coordinate reference systems (CRS). It is recommended to project all layers to the same CRS and check them before using them as model inputs. The model setup process checks if the layer CRSs differ from the one of the basin

boundary. By setting 'proj_layer <- TRUE' the layer is projected if the CRS is different. If FALSE different CRS trigger an error.

```
project_layer <- TRUE
```

We read in and check the basin boundary and run some checks

```
bound <- read_sf(bound_path) %>% select()
set_proj_crs(bound, data_path)
check_polygon_topology(layer = bound, data_path =  data_path, label = 'basin',
                       n_feat = 1, checks = c(F,T,T,T,F,F,F,F))
```

2.2.3 Processing the Land layer

Our land layer is located here:

```
land_path <- "model_data/input/land/CS10_LU_V2.shp"
```

Documentation on its creation does not exist.

```
lu_shp <- sf::read_sf(land_path)
lu_map <- ggplot2::ggplot(lu_shp) + ggplot2::geom_sf(ggplot2::aes(fill = type)) + sf_t
  ggplot2::theme(legend.position = "none")
print(lu_map)
```

We had an issue with the classification of the land uses. For the OPTAIN project, all agricultural fields must have a unique ID, and our land uses only had the ID of the given farm `KGB` (which had many different fields). To remedy this, new IDs were generated with the format `a_###f_#` where `a_` represents the farm, and `f_` represents the respective field of that farm. The farm names needed to be shortened because the SWAT+ model often cannot handle long ID names (longer than 16 characters)

Note, the potential measures polygons were not counted as individual fields, which is why `SP_ID` does not match up with `field_ID` – This is by design.

```
readr::read_csv("model_data/farm_id/a_f_id.csv", show_col_types = F) %>% head()
```

```
## # A tibble: 6 x 4
##   KGB      type_fr sp_id field_ID
##   <chr>    <chr>   <dbl> <chr>
## 1 213/8/1  a_084       1 a_084f_1
## 2 213/7/6  a_083       2 a_083f_1
```

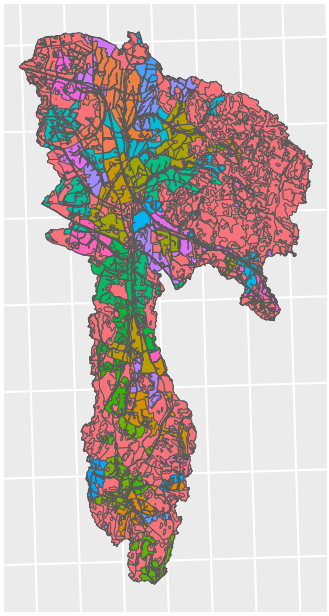


Figure 2.2: Land use map of CS10 by Farm ID

```
## 3 213/7/6  a_083       3 a_083f_2
## 4 213/7/4  a_082       4 a_082f_1
## 5 213/65/1 a_081       5 a_081f_1
## 6 213/65/1 a_081       6 a_081f_2
```

This was done in a simple QGIS workflow of dissolving by farm, splitting from single part to multipart, and then adding an iterating ID per farm field. This workflow could be replicated in R, and then shown here. It is under consideration…

BuildR will now run some checks on our land layer.

```
land <- read_sf(land_path) %>%
  check_layer_attributes(., type_to_lower = FALSE) %>%
  check_project_crs(layer = ., data_path =  data_path, proj_layer = project_layer,
                    label = 'land', type = 'vector')

check_polygon_topology(layer = land, data_path =  data_path, label = 'land',
                       area_fct =  0.00, cvrg_frc = 99.9,
                       checks = c(T,F,T,T,T,T,T,T))
```

BuildR splits the land layer into HRU (land) and reservoir (water) objects

```
split_land_layer(data_path)
```

2.2.4 Processing the Channels

No documentation exists on the creation of the channels layer, all we know is that it is located here:

```
channel_path <- 'model_data/input/line/cs10_channels.shp'
channels <- read_sf(channel_path) %>% dplyr::select("type")
channel_map <- ggplot2::ggplot() + ggplot2::geom_sf(data = bound_sf) +
  ggplot2::geom_sf(data = channels, mapping = ggplot2::aes(color = type)) + sf_theme
channel_map
```

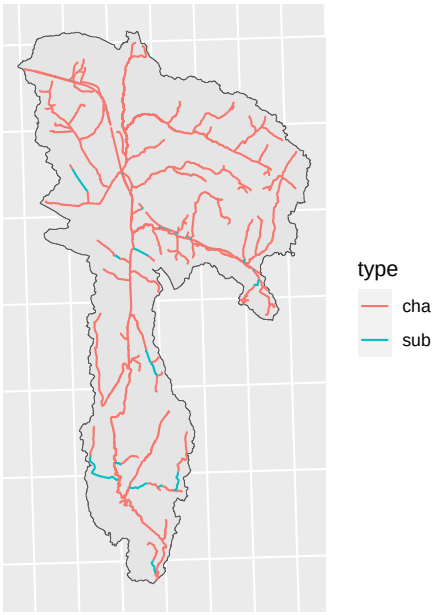


Figure 2.3: CS10 Channels including surface and subsurface channels

BuildR runs some checks:

```
channel <- read_sf(channel_path) %>%
   check_layer_attributes(., type_to_lower = TRUE) %>%
   check_project_crs(layer = ., data_path =  data_path,
                     proj_layer = project_layer,
                     label = 'channel', type = 'vector')
```

```
check_line_topology(layer = channel, data_path = data_path,
                    label = 'channel', length_fct = 0, can_cross = FALSE)
```

BuildR then checks the connectivity between the channels and reservoirs. For this we need to define our `id_cha_out` and `id_res_out`.

"Variable `id_cha_out` sets the outlet point of the catchment. Either define a channel OR a reservoir as the final outlet. If channel then assign `id_cha_out` with the respective id from the channel layer. If reservoir then assign the respective id from the land layer to id_res_out, otherwise leave as `NULL`"

```
id_cha_out <- 37
id_res_out <- NULL
```

Running connectivity checks between channels and reservoirs:

```
check_cha_res_connectivity(data_path, id_cha_out, id_res_out)
```

Checking if any defined channel ids for drainage from land objects do not exist

```
check_land_drain_ids(data_path)
```

2.2.5 Processing the DEM

BuildR loads and checks the DEM, and saves it.

```
dem <- rast(dem_path) %>%
  check_project_crs(layer = ., data_path =  data_path, proj_layer = project_layer,
                    label = 'dem', type = 'raster')
check_raster_coverage(rst = dem, vct_layer = 'land', data_path = data_path,
                      label = 'dem', cov_frc = 0.95)
save_dem_slope_raster(dem = dem, data_path = data_path)
```

Here is the result:

```
map <- raster::raster("model_data/cs10_setup/optain-cs10/data/raster/dem.tif")
raster::plot(map)
```

2.2.6 Processing soil data

Our soil map is located here:

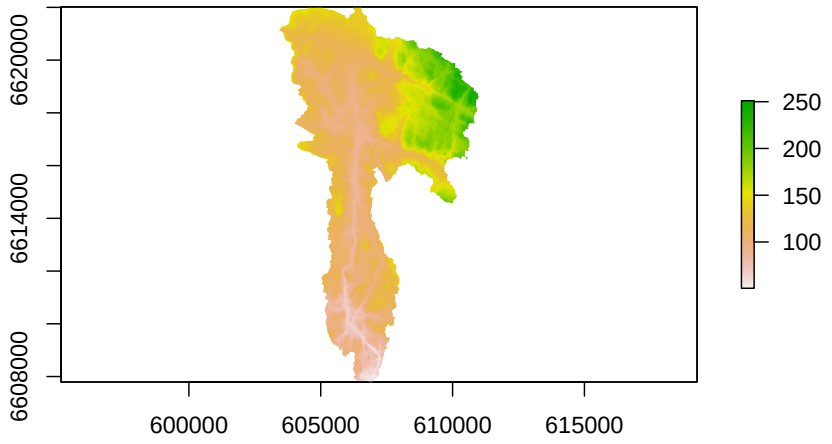


Figure 2.4: CS10 DEM cropped to basin

```
soil_layer_path  <- 'model_data/input/soil/soil_layer.tif'
```

No documentation exists on its creation.

```
map <- raster::raster(soil_layer_path)
raster::plot(map)
```

The soil data and look-up path are located here:

```
soil_lookup_path <- 'model_data/input/soil/soil_lookup.csv'
soil_data_path   <- 'model_data/input/soil/UserSoil_Krakstad.csv'
```

Not much documentation exists here either. Some can be found in the excel sheet:

`~/model_data/input/soil/swatsoil2.xlsx`

BuildR reads in the soil data, performs checks, processes, and saves.

```
soil <- rast(soil_layer_path) %>%
  check_project_crs(layer = ., data_path =  data_path, proj_layer = project_layer,
```

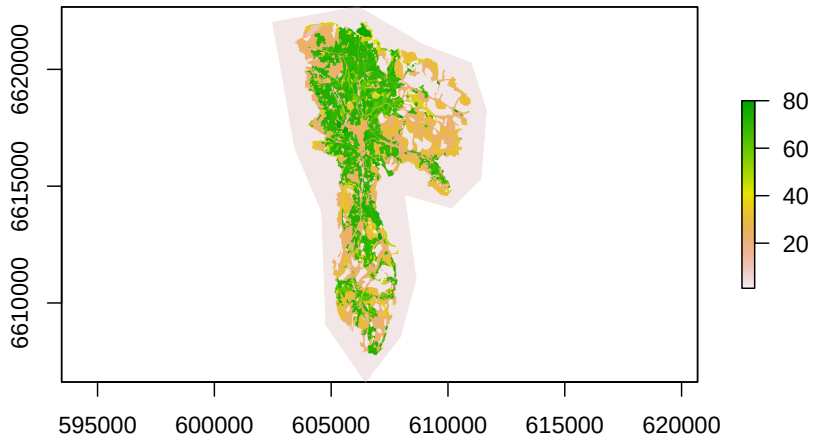


Figure 2.5: CS10 Soil map

```
                    label = 'soil', type = 'raster')
check_raster_coverage(rst = soil, vct_layer = 'hru', data_path = data_path,
                      label = 'soil', cov_frc = 0.75)
save_soil_raster(soil, data_path)
```

BuildR then generates a table with aggregated elevation, slope, soil for HRU units.

```
aggregate_hru_dem_soil(data_path)
```

Read and prepare the soil input tables and a soil/hru id table and write them into `data_path/tables.sqlite`

```
build_soil_data(soil_lookup_path, soil_data_path, data_path)
```

2.3 Calculating Contiguous Object Connectivity

SWATBuildR follows COCOA. This section contains the calculations. You can read more about it in the protocol.

2.3.1 Calculating land unit connectivity

Preparing raster layers based on the DEM and the surface channel objects that will be used in the calculation of the land object connectivity.

```
prepare_terrain_land(data_path)
```

The connection of each land object to neighboring land and water objects is calculated based on the flow accumulation and the D8 flow pointer along the object edge

```
calculate_land_connectivity(data_path)
```

2.3.1.1 Eliminate land object connections with small flow fractions:

For each land object the flow fractions are compared to connection with the largest flow fraction of that land object. Connections are removed if their fraction is smaller than `frc_thres` relative to the largest one.

This is necessary to:

1. Simplify the connectivity network
2. To reduce the risk of circuit routing between land objects. Circuit routing will be checked.

If an error due to circuit routing is triggered, then 'frc_thres' must be increased to remove connectivities that may cause this issue.

```
frc_thres <- 0.3
```

The remaining land object connections are analyzed for infinite loop routing. For each land unit the connections are propagated and checked if the end up again in the same unit.

```
# reduce_land_connections(data_path, frc_thres) %>%
#  check_infinite_loops(., data_path, 'Land')
```

```
# Analyzing land objects for infinite loop routing: Completed 6185 Land objects
# in 10M 44S
#
#    150 Land objects identified where water is routed in loops.
#
```

```
# You can resolve this issue in the following ways:
#
#  - Use the layer 'land_infinite_loops.gpkg' that was written to
# model_data/cs10_setup/optain-cs10/data/vector to identify land polygons that
# cause the issue and split them to break the loops This would require to
# restart the entire model setup procedure!
#
#   - Increase the value of 'frc_thres'.
# This reduces the number of connections of each land unit (maybe undesired!)
# and can remove the connections that route the water in loops.
#
#   - Continue with the model setup (only recommended for small number of identified units!).
# The function 'resolve_loop_issues()' will then eliminate a certain number of
# connections.
```

What do we do here ? – I think in the past we just removed all the loops, but it says here it is not recommended when there are a lot of them. Should we increase the threshold? Should we try to break up the loops? You can see the problem polygons on the map below:

```
land_inifnite_loops_shp <- sf::read_sf("model_data/cs10_setup/optain-cs10/data/vector/land_infini
mapview::mapview(land_inifnite_loops_shp)
```

To me it seems to complicated to try to break up the loops, maybe increase the threshold a bit and see what happens? and delete the rest?

Increasing threshold and re-running leads to this:

```
frc_thres <- 0.1 #     loops (   mins)
frc_thres <- 0.2 # 238 loops (30 mins)
frc_thres <- 0.3 # 150 loops (12 mins)
frc_thres <- 0.4 # 115 loops (   mins)
frc_thres <- 0.5 # 94  loops (   mins)
frc_thres <- 0.6 # 57  loops (   mins)
```

We have decided that at a threshold of 0.3 and 150, we are within the “not that many” loops territory, considering out of 6000+ HRUs, it is only around 2%. Therefore we will let BuildR continue and remove them. This has been discussed in Issue #55.

```
frc_thres <- 0.3 # 150 loops
reduce_land_connections(data_path, frc_thres) %>%
  check_infinite_loops(., data_path, 'Land')
```

2.3.1.2 Resolve infinite loops

If infinite loops were identified this routine tries to resolve the issues by selectively removing connections between land units in order to get rid of all infinite loops.

```
resolve_loop_issues(data_path)
```

2.3.2 Calculating channel/reservoir connectivity

2.3.2.1 Calculating the water object connectivity

the function returns the `cha` and `res con_out` tables in SWAT+ database format and writes them into `data_path/tables.sqlite`

```
build_water_object_connectivity(data_path)
```

2.3.3 Checking the water objects for infinite loops

From the `cha_res_con_out` tables `id_from/id_to` links are generated and checked for infinite loop routing.

```
prepare_water_links(data_path) %>%
  check_infinite_loops(., data_path, 'Water', Inf)
```

2.3.4 Terrain properties

Calculate terrain properties such as elevation, slope, catchment area, channel width/depth for channel and reservoir objects and write them into `data_path/tables.sqlite`

```
prepare_terrain_water(data_path)
```

2.4 Generate SWAT+ input

2.4.1 Generate land object SWAT+ input tables

Build the `landuse.lum` and a landuse/hru id table and write them into `data_path/tables.sqlite`

```
build_landuse(data_path)
```

Build the HRU SWAT+ input files and write them into `data_path/tables.sqlite`

```
build_hru_input(data_path)
```

Add wetlands to the HRUs and build the wetland input files and write them into `data_path/tables.sqlite`

We only use the `wetf` land use .

```
wetland_landuse <- c('wehb', 'wetf', 'wetl', 'wetn')
```

```
add_wetlands(data_path, wetland_landuse)
```

2.4.2 Generate water object SWAT+ input tables

Build the SWAT+ `cha` input files and write them into `data_path/tables.sqlite`

```
build_cha_input(data_path)
```

Build the SWAT+ res input files and write them into `data_path/tables.sqlite`

```
build_res_input(data_path)
```

Build SWAT+ routing unit con_out based on 'land_connect_fraction'.

```
build_rout_con_out(data_path)
```

Build the SWAT+ rout_unit input files and write them into `data_path/tables.sqlite`

```
build_rout_input(data_path)
```

Build the SWAT+ LSU input files and write them into `data_path/tables.sqlite`

```
build_ls_unit_input(data_path)
```

2.4.3 Build aquifer input

Build the SWAT+ aquifer input files for a single aquifer for the entire catchment. The connectivity to the channels with geomorphic flow must be added after writing the `txt` input files. **This is not implemented in the script yet.**

```
build_single_aquifer_files(data_path)
```

2.4.4 Add point source inputs

The point source locations are provided with a point vector layer in the path 'point_path'.

We have yet to complete this step. It is being tracked in Issue # 21

```
point_path <- 'model_data/input/point/cs10_pointsource.shp'
```

```
point_sf <- sf::read_sf(point_path)
point_map <- mapview::mapview(point_sf, zcol = "GRAD_P", cex = "GRAD_N")

mapview::mapview(bound_sf, alpha.regions = 0.2)+point_map
```

Map of point sources, colored by (assumed) phosphorous and size by (assumed) Nitrogen

Maximum distance of a point source to a channel or a reservoir to be included as a point source object (recall) in the model setup:

```
max_point_dist <- 500 # meters
```

Point source records can automatically be added from files in the same folder as the point source location layer. To be identified as point source data the files must be named as `<name>_<interval>.csv`, where `<name>` must be the name of a point `int` the vector layer and `<interval>` must be one of `const`, `yr`, `mon`, or `day` depending on the time intervals in the input data.

```
add_point_sources(point_path, data_path, max_point_dist)
```

2.4.5 Create SWAT+ sqlite database

2.4.5.1 Write the SWAT+Editor project database

The database will be located the 'project_path'.

```
stat <- file.remove("model_data/cs10_setup/optain-cs10/optain-cs10.sqlite")
create_swatplus_database(project_path, project_name)
```

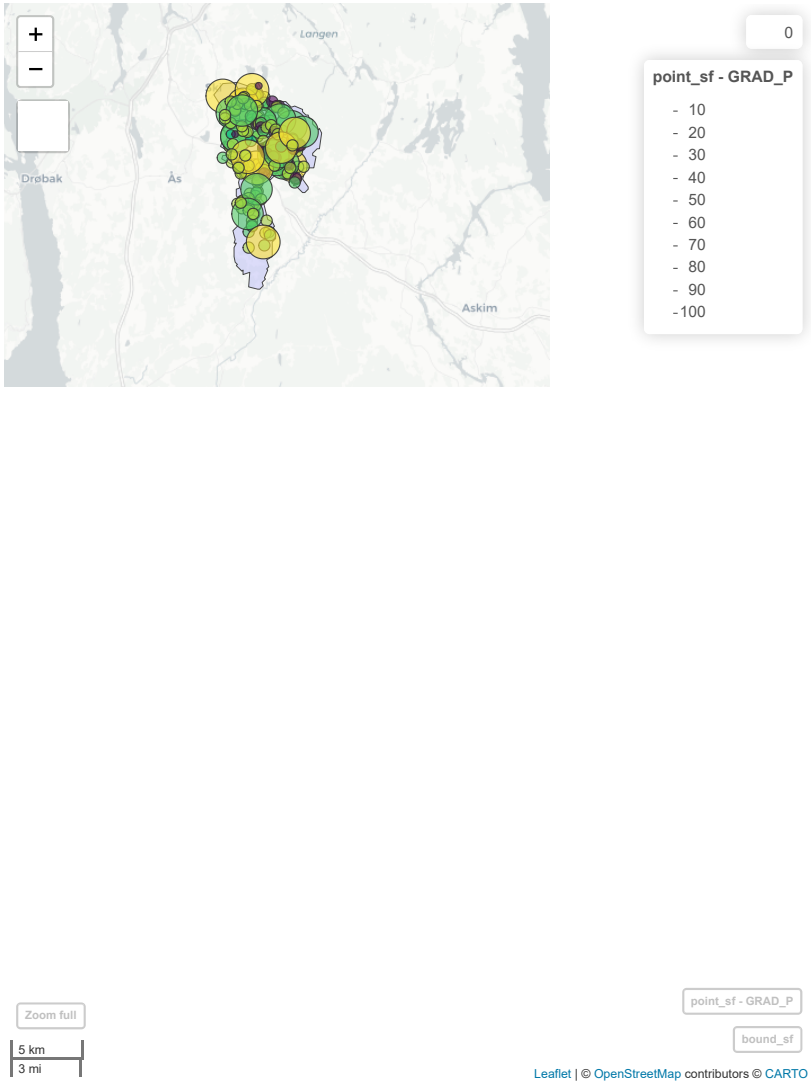


Figure 2.6: CS10 Point Sources, colored by Phosphorous emission and sized by Nitrogen emission

Chapter 3

Climate inputs and weather generator

```
run_this_chapter = FALSE
```

3.1 Preparation and loading data

To add weather and climate data to our SWAT project, we will use `svatools` (**?**)

3.1.1 Required packages

```
require(euptf2) # devtools::install_github("tkdweber/euptf2")
require(svatools) # devtools::install_github("biopsichas/svatools")
require(readr)
require(readxl)
require(sf)
require(mapview)
require(ggplot2)
require(tidyr)
```

3.1.2 Load in the file(s) and load svatools template

First we load in the template and fill it in with our values and rename it to "cs10_weather_data.xlsx". This is not done within R. No detailed documenta-

tion exists on the source of this data (yet).

There has been some discussion in issue #48

```
#temp_path <- system.file("extdata", "weather_data.xlsx", package = "svatools")
# /// fill out this template and save "cs10_weather_data.xlsx" ///
```

We are using the following projection for this project:

```
epgs_code = 25832
```

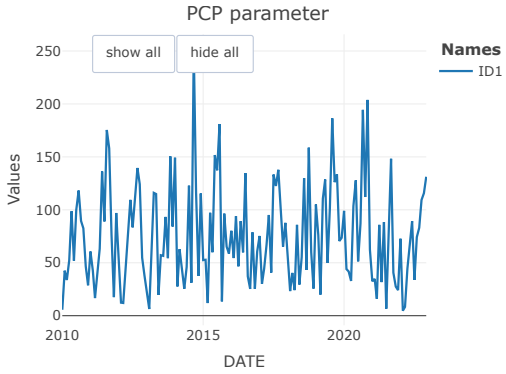
Now we can load it in with Svatools.

```
met_lst <- load_template(template_path = "model_data/input/met/cs10_weather_data.xlsx"
```

```
## [1] "Loading data from template."
## [1] "Reading station ID1 data."
## [1] "Loading of data is finished."
```

3.1.3 Proof the station

```
plot_weather(met_lst, "PCP", "month", "sum")
```



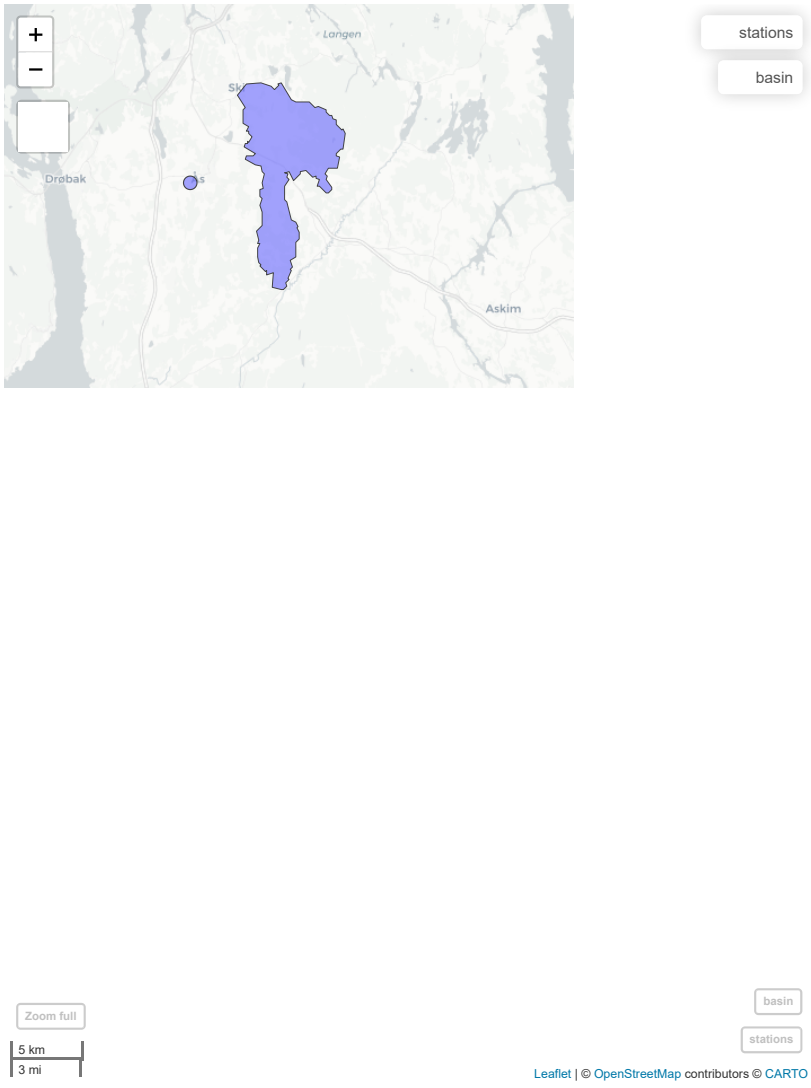
Checking the location of the station:

```
basin_path <- "model_data/input/shape/cs10_basin.shp"
basin <- st_transform(st_read(basin_path), epgs_code) %>%
  mutate(NAME = "Basin")
```

```
## Reading layer `cs10_basin' from data source
##   `C:\Users\NIBIO\Documents\GIT\swat-cs10\model_data\input\shape\cs10_basin.shp'
```

```
##   using driver `ESRI Shapefile'
## Simple feature collection with 1 feature and 1 field
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 603439.4 ymin: 6607792 xmax: 610984.4 ymax: 6622017
## Projected CRS: ETRS89 / UTM zone 32N
```

```
stations <- st_transform(met_lst$stations, epgs_code)
mapview(stations) + mapview(basin)
```



3.2 Creating the weather generator

We only have one weather station, which means only one weather generator.

```
wgn <- prepare_wgn(met_lst,
                   TMP_MAX = met_lst$data$ID1$TMP_MAX,
                   TMP_MIN = met_lst$data$ID1$TMP_MIN,
                   PCP = met_lst$data$ID1$PCP,
                   RELHUM = met_lst$data$ID1$RELHUM,
                   WNDSPD = met_lst$data$ID1$WNDSPD,
                   MAXHHR = met_lst$data$ID1$MAXHHR,
                   SLR = met_lst$data$ID1$SLR)
```

```
## [1] "Coordinate system checked and transformed to EPSG:4326."
## [1] "Working on station ID1:WS_AAS"
```

```
write.csv(wgn$wgn_st, "model_data/input/met/as_wgn_st.csv", row.names = FALSE, quote =

write.csv(wgn$wgn_data, "model_data/input/met/as_wgn_data.csv", row.names = FALSE, quo
```

3.3 Adding weather data to SWAT+ project

This adds the weather generator and weather stations to our SWAT+ project

```
db_path <- "model_data/cs10_setup/optain-cs10/optain-cs10.sqlite"
add_weather(db_path, met_lst, wgn)
```

3.4 Adding Atmospheric deposition

Instructions and documentation can be found here

Getting the data:

```
basin_path <- "model_data/input/shape/cs10_basin.shp"
df <-
  get_atmo_dep(
    basin_path,
    start_year = 2010,
    end_year = 2020,
    t_ext = "year"
  )

readr::write_csv(df, file = "model_data/input/met/atmodep.csv")
```

A plot of the results:

```
df <- readr::read_csv("model_data/input/met/atmodep.csv", show_col_types = F)
ggplot(pivot_longer(df, !DATE, names_to = "par", values_to = "values"), aes(x = DATE, y = values)
  geom_line()+
  facet_wrap(~par, scales = "free_y")+
  theme_bw()
```

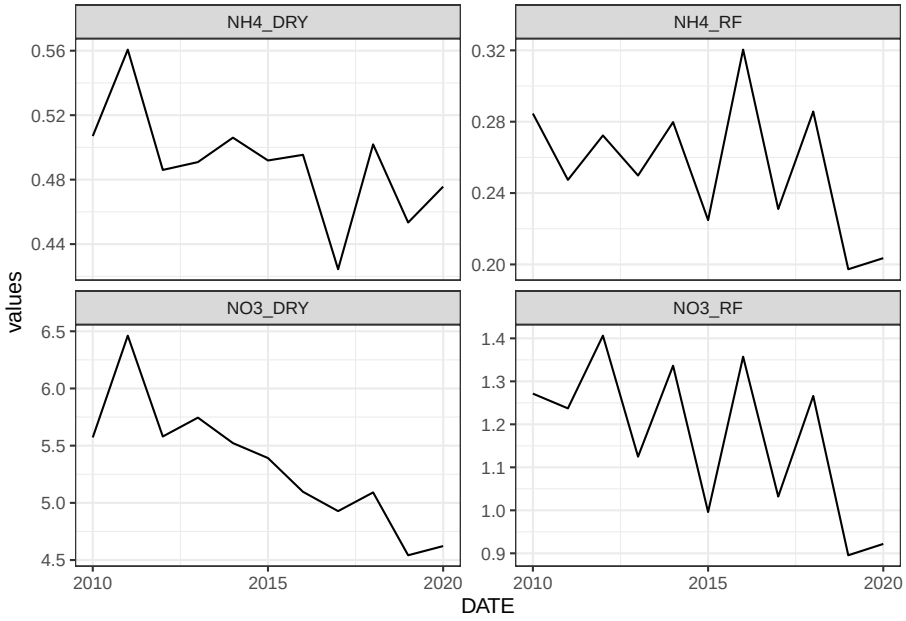


Figure 3.1: Atmospheric Deposition data grabbed by svatools

Adding the data to the SQLITE

```
db_path <- "model_data/cs10_setup/optain-cs10/optain-cs10.sqlite"
add_atmo_dep(df, db_path)
```

Big thank you to Svajunas for his svatools package, making our lives much easier!

Chapter 4

Writing SWAT+ Input Files

The SWATBuildR cannot write the actual text files for our model setup, we have to use the SWAT+ Editor for this. Once the input files have been written, we will try to **ONLY modify these parameters with R** and **never edit parameters in the Editor**. The reason for this is because changing the values of the text files will **not** change the values of the `sqlite` database. If the input files are re-written by the editor, our changes to the text files will be **overwritten**.

4.1 Loading the project in SWAT+ Editor

1. After clicking on the Run Model / Save Scenario button, you arrive at the "Confirm Simulation Settings" page. Here you need to choose where to write your input files. We have chosen `cs10_setup/swat_input`.
2. We have also changed our simulation time period, but this is not required.
3. Make sure to un-check "Run SWAT" and "Analyze output for visualization"
4. "Save settings & Run Selected"

Your SWAT+ input files will be generated, and you will be prompted to Save Scenario. **This is not recommended**, because in our testing, the scenario saving would recursively generate new scenarios within the same folder, until all file space had been exhausted on the drive. This extremely deep folder **brings windows to its knees**, even when trying to delete it using explorer (Use Powershell instead).

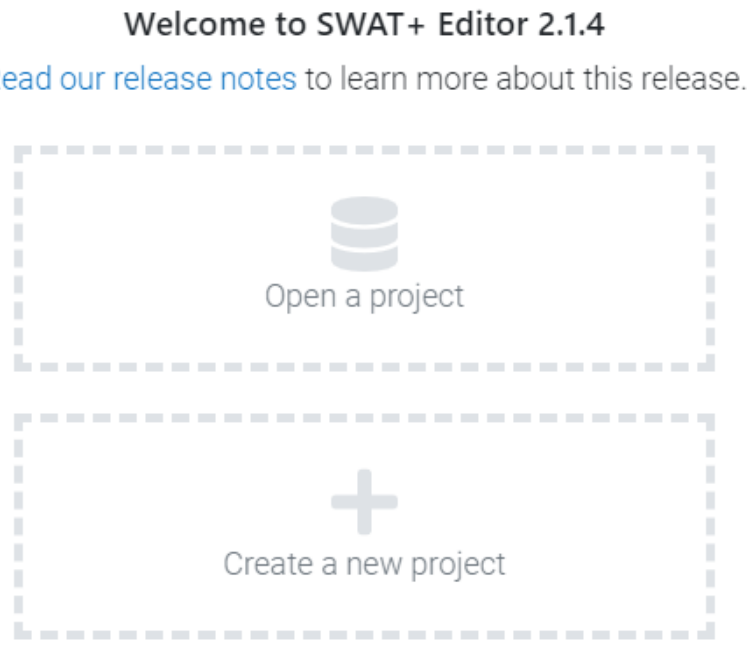


Figure 4.1: Open a project

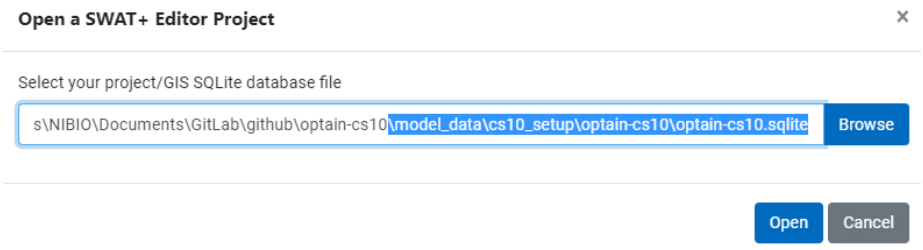


Figure 4.2: Path to the project sqlite

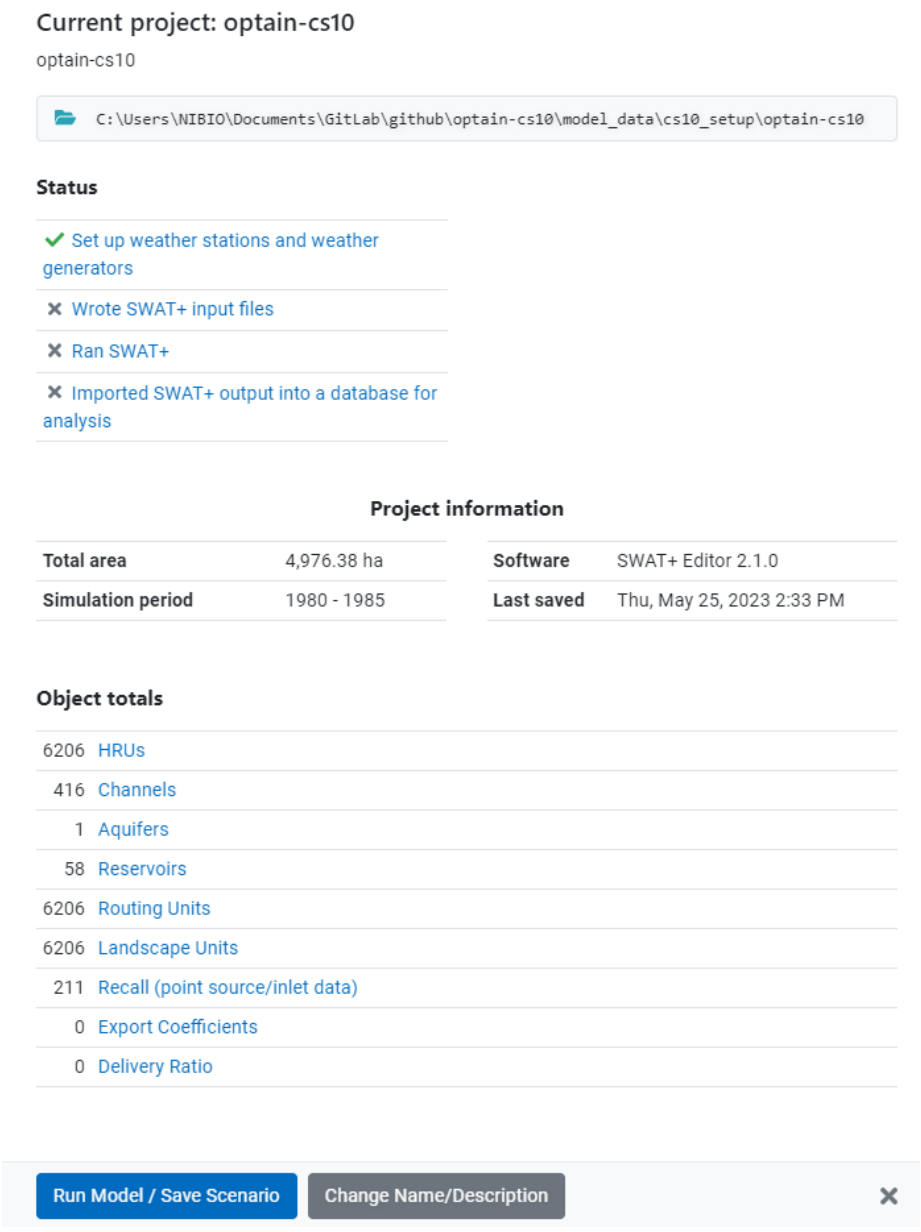


Figure 4.3: Project Information. The weather generator has been complete by svatools in Section ??.

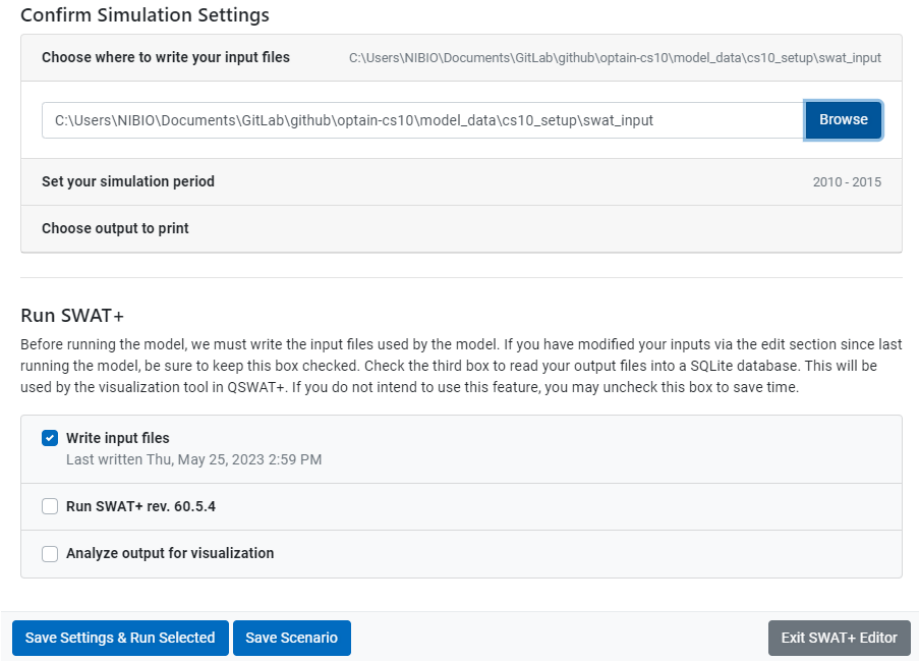


Figure 4.4: Confirm Simulation Settings. Make sure to follow the instructions carefully here.

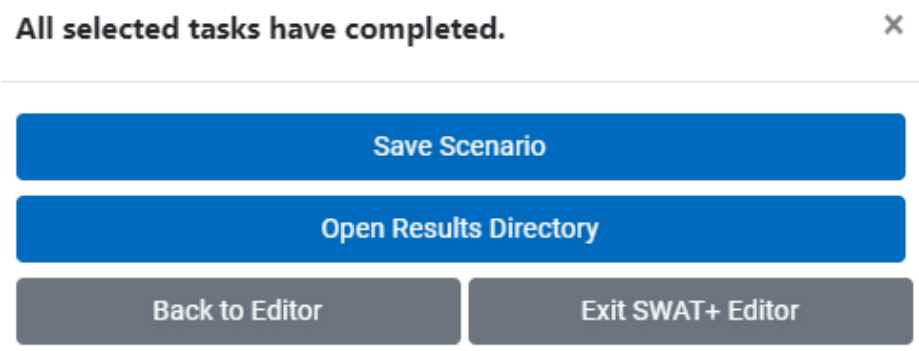


Figure 4.5: Option to save scenario. Not Recommended.

4.2 SWAT+ Test Run

We will copy in our SWAT+ executable, our input files, and our weather files into a folder and run SWAT+ as a test.

> For some reason, when writing the SWAT+ input files, it does not write the weather station data. I am not sure if this is intended or not. the SWAT+ model still runs fine, however maybe it is using the weather generator? The meteo input files are located in the same directory as the `sqlite`, probably because `svatools` put them there? #TODO

Our current SWAT+ version is `rev60.5.4_64rel.exe` but this is **subject to change**

Make a run directory and enable code running.

```
# if set to true, following code will be evalutated.
run_this_chapter = FALSE

dir.create("model_data/cs10_setup/run_swat", showWarnings = F)
```

Copy all required files into this directory:

```
sta_files <- list.files("model_data/cs10_setup/optain-cs10/",
             pattern = "sta_", full.names = T)

cli_files <- list.files("model_data/cs10_setup/optain-cs10/",
                        pattern = ".cli", full.names = T)

input_files <- list.files("model_data/cs10_setup/swat_input/",
                       full.names = T)

path_to_swat <- "model_data/cs10_setup/rev60.5.4_64rel.exe"


source_files <- c(sta_files, cli_files, input_files, path_to_swat)


status <- file.copy(from = source_files,
                    to = "model_data/cs10_setup/run_swat/",
                    overwrite = T)
```

Lets run some checks on these files

```
if(any(status == FALSE)){
  warning("Some files were not copied:")
  print(source_files[which(status==FALSE)])
}

print(
  paste(
    length(which(status)),
    "of",
    length(source_files),
    "files were copied into the run folder, amounting to a size of ",
    round(sum(file.info(source_files)$size * 1e-6), 2),
    "megabytes"
  )
)
```

Now lets run the model to make sure it works

```
# update time sim
time_sim <- readLines("model_data/cs10_setup/run_swat/time.sim")
time_sim[3] <- "       0      2010         0      2011         0   "
writeLines(text = time_sim, con = "model_data/cs10_setup/run_swat/time.sim")

msg <- processx::run(command = "rev60.5.4_64rel.exe",
              wd = "model_data/cs10_setup/run_swat/")


simout <- readLines("model_data/cs10_setup/run_swat/simulation.out")

cat(tail(simout), sep = "\n")
```

It will be useful to test SWAT+ along our journey, so we will reuse this code in the function `test_swat()`

```
source("model_data/code/test_swat.R")
```

We can now continue with our final buildR step

4.3 Link aquifers and channels with geomorphic flow

A SWATBuildR model setup only has one single aquifer (in its current version). This aquifer is linked with all channels through a channel-aquifer-link file

(`aqu_cha.lin`) in order to maintain recharge from the aquifer into the channels using the geomorphic flow option of SWAT+.

The required input file cannot be written with the SWAT+Editor. **Therefore it has to be generated in a step after writing the model text input files with the SWAT+Editor.**

Path of the `TxtInOut` folder (project folder where the SWAT+ text files are written with the SWAT+Editor)

```
txt_path <- 'model_data/cs10_setup/swat_input/'
```

Linking the aquifer to the channels

```
project_path <- 'model_data/cs10_setup'
project_name <- 'optain-cs10'
source('model_data/swat_buildR/init.R')

link_aquifer_channels(txt_path)
```

This created the file `aqu_cha.lin` and changed `file.cio` to point to it (row 16, column 3)

Let us test to see if SWAT+ still runs

```
simout <- test_swat()

cat(tail(simout), sep = "\n")
```

Success.

```
# remove the swat+ run
unlink("model_data/cs10_setup/run_swat/", recursive = T)
```

Chapter 5

Crop Map Generation

As OPTAIN takes into account the individual field, we need to know what is growing on which field and when. Unfortunately our data foundation for this task is quite lackluster, but we will try out best to do so. in the following chapter.

```
require(sf)
require(dplyr)
require(readr)
require(stringr)
require(readxl)
require(reshape2)
require(tibble)
require(DT)
require(ggplot2)
require(gifski)

sf_theme <- theme(axis.text.x=element_blank(), #remove x axis labels
        axis.ticks.x=element_blank(), #remove x axis ticks
        axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
        )
```

5.1 Calculating Field Area

We have data based on what area certain crops have per farm. In order to relate this to our spatial land use map, we require the area of both the farms, and the fields within these farms.

The basis for these calculations comes from the BuildR output from Section

```
lu_sf <- read_sf("model_data/cs10_setup/optain-cs10/data/vector/land.shp")
lu_map <- ggplot() + geom_sf(lu_sf, mapping = aes(fill = type)) +
  theme(legend.position = "left") + sf_theme + theme(legend.position = "none")
print(lu_map)
```

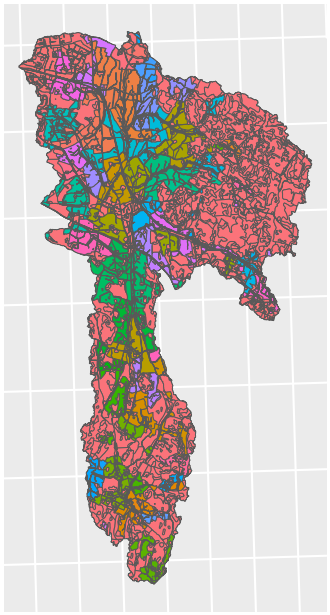


Figure 5.1: Land use map for CS10, colored by type.

5.1.1 Calculate Field Area and Assign IDs

This stage has been completed in QGIS. An R-implementation is being considered.

```
farm_area_sf <-
  read_sf("model_data/input/crop/buildr_output_dissolved_into_farms.shp")
field_lu_plot <-
  ggplot() + geom_sf(farm_area_sf, mapping = aes(fill = farm_id)) +
  theme(legend.position = "none") + sf_theme
print(field_lu_plot)
```

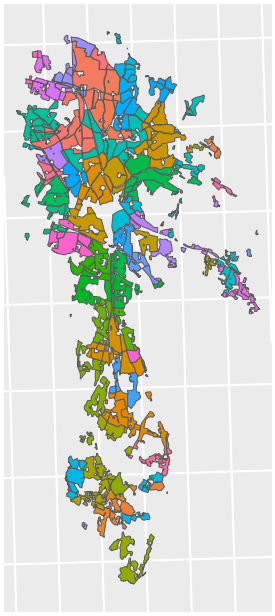


Figure 5.2: Farms in the CS10 catchment, colored by ID

5.1.2 Calculate Field area and assign IDs

This stage has been completed in QGIS. An R-implementation is being considered.

5.2 Generating the Crop Rotation

The following algorithm takes information from reporting farm, and their respective area in the SWAT+ setup, and combines these data sets to generate a plausible crop rotation.

Note, pasture refers to meadow (TODO: fix!)

```
crop_data <- read_excel("model_data/input/crop/crop_rotation_source_data.xlsx")
datatable(crop_data)
```

Show 10 entries Search:

	farm_id	area_daa	pastrure_area_daa	crop1	crop2	crop3	percent_crop1	percent_crop2	percent_crop3
1	a_011	197.225		swht	barl		69.54314720812182	30.45685279187818	
2	a_052	162.422	108	barl	wwht	swht	54	37	
3	a_212	0.29		oats			100		
4	a_211	94.772		oats	swht		84.07258064516128	15.92741935483871	
5	a_096	134.911		swht	barl		50	50	
6	a_193	37.891		oats	barl	wwht	68.44181459566076	19.72386587771203	11.83431952662
7	a_088	246.452		swht	oats		52.00764818355641	47.99235181644359	
8	a_031	415.489		barl	oats	swht	36.97590361445783	32.87951807228916	29.95180722891
9	a_178	115	80	swht	oats		79	21	
10	a_176	177.228		swht			100		

Showing 1 to 10 of 174 entries Previous 1 2 3 4 5 … 18 Next

5.2.1 Tidy up the source data:

We are converting the source data to tidy format in the following code snippet

```
crop_names <- melt(
   crop_data,
```

```
  id.vars = c("farm_id", "year", "area_daa", "pastrure_area_daa"),
  measure.vars = c("crop1", "crop2", "crop3"),
  variable.name = "crop_status", value.name = "crop"
) %>% tibble()
crop_area <- melt(
  crop_data,
  id.vars = c("farm_id", "year", "area_daa", "pastrure_area_daa"),
  measure.vars = c("percent_crop1", "percent_crop2", "percent_crop3"),
  variable.name = "crop_status", value.name = "area"
) %>% tibble()
crop_area$crop_status <- crop_area$crop_status %>% str_remove("percent_")
# unstable, TODO fix!
colnames(crop_area)[6] <- "crop_area_percent"
crop_tidy <- left_join(crop_names, crop_area, by = c("farm_id", "year", "area_daa","pastrure_ar
 # unstable, TODO fix!
colnames(crop_tidy)[3] <- "farm_area_daa"
crop_tidy$crop_area_percent <- (crop_tidy$crop_area_percent/100) %>% round(2)
crop_tidy <- crop_tidy %>% filter(crop %>% is.na() == FALSE)
datatable(crop_tidy)
```

Show 10 entries Search:

	farm_id	year	farm_area_daa	pastrure_area_daa	crop	crop_area_percent
1	a_011	2016	197.225		swht	0.7
2	a_052	2016	162.422	108	barl	0.54
3	a_212	2016	0.29		oats	1
4	a_211	2016	94.772		oats	0.84
5	a_096	2016	134.911		swht	0.5
6	a_193	2016	37.891		oats	0.68
7	a_088	2016	246.452		swht	0.52
8	a_031	2016	415.489		barl	0.37
9	a_178	2016	115	80	swht	0.79
10	a_176	2016	177.228		swht	1

Showing 1 to 10 of 334 entries Previous 1 2 3 4 5 … 34 Next

5.2.2 Tidy up the BuildR output

The maps shown in Section **??** were exported to CSV in QGIS. An R-implementation might be written in here sometime (#TODO).

This CSV data needs to be re-formatted to be tidy as well.

```
fields <-
  read_csv(
    "model_data/input/crop/buildr_landuse_dissolved_into_fields.csv",
    show_col_types = F
  ) %>% dplyr::select(type, farm_id, field_area_daa)

farms <-
  read_csv(
    "model_data/input/crop/buildr_landuse_dissolved_into_farms.csv",
    show_col_types = F
  ) %>% dplyr::select(type, farm_id , farm_area_)
# fix farm_area_ TODO
# unstable, fix! TODO
colnames(farms)[3] <- "farm_area_daa"
farms$farm_area_daa <- farms$farm_area_daa %>% round(2)
fields$field_area_daa <- fields$field_area_daa %>% round(2)
reported_data <- left_join(fields, farms, by = "farm_id")

# unstable, fix! TODO
colnames(reported_data)[1] <- "field_id"

reported_data <- reported_data %>% dplyr::select(field_id, farm_id, field_area_daa, farm_area_daa

datatable(reported_data)
```

Show 10 entries Search:

	field_id	farm_id	field_area_daa	farm_area_daa
1	a_025f_2	a_025	13.22	393.72
2	a_105f_3	a_105	1.17	85.2
3	a_105f_1	a_105	59.2	85.2
4	a_105f_4	a_105	20.27	85.2
5	a_187f_1	a_187	8.01	8.01
6	a_206f_1	a_206	7.1	7.1
7	a_121f_2	a_121	6.98	10.66
8	a_132f_1	a_132	42.78	92.58
9	a_182f_3	a_182	7.17	88.77
10	a_182f_4	a_182	6.29	88.77

Showing 1 to 10 of 501 entries

Previous 1 2 3 4 5 … 51 Next

5.2.3 Field classification

We are now ready to perform our classification. We will store our results in this predefined dataframe:

```
classed_fields <-
  tibble(
    farm_id = NA,
    field_id   = NA,
    farm_area_daa = NA,
    field_area_daa = NA,
    crop  = NA,
    year = NA
  )
```

The following reporting farms will be used for the classification:

```
farms <- reported_data$farm_id %>% unique()
```

For the following years (AKA we have data for these years):

```
years <- c(2016, 2017, 2018, 2019)
```

This for loop runs through all the fields and classifies them. The output will not be printed in this file, but will be saved in a log file, that you can dig out of the repository, located here:

```
# TODO add the log file generation
```

The following code is not executed:

```
# For every year
for(c_year in years){
  # For every farm,
  for (farm in farms) {
    # Do the following:

    # Filter the source and buildR data to the given year and farm
    crop_filter <- crop_tidy %>% filter(year == c_year) %>% filter(farm_id == farm)
    hru_filter <- reported_data %>% filter(farm_id == farm)

    # Behavior if the farm has no reporting data:
    if(crop_filter$farm_id %>% length() == 0){

      # Set all fields to winter wheat.

      farm_fields <- hru_filter %>%
        dplyr::select(farm_id, field_id, farm_area_daa, field_area_daa) %>%
```

```
      arrange(desc(field_area_daa))

    farm_fields$crop = "wwht"
    farm_fields$year = c_year

    # Add them to the results dataframe
    classed_fields <- rbind(classed_fields, farm_fields)

    # and skipping to next farm
    next()
  }

  # Behavior, if there is reporting data on the farm"

  # Find the area of the farm (all elements in vector are the same, so ok
  # to just grab the first)
  farm_area_hru <- hru_filter$farm_area_daa[1]

  # Find the discrepency between "SHOULD BE" crop area and "CURRENTLY IS"
  # crop area. We want this as an absolute value.
  area_dis <- (hru_filter$farm_area_daa[1]-crop_filter$farm_area_daa[1]) %>%
    round(2) %>% abs()

  # Some farms have no reported area. in this case we set it to some random
  # negative number.
  if(area_dis %>% is.na()){area_dis = -999}

  # Extracting the area of the pasture and farm (all vectors same value, ok
  # to just take the first)
  past_area <- crop_filter$pastrure_area_daa[1]
  farm_area <- crop_filter$farm_area_daa[1]

  # Extract relevant fields
  farm_fields <- hru_filter %>%
    dplyr::select(farm_id, field_id, farm_area_daa, field_area_daa) %>%
    # and sort by area
    arrange(desc(field_area_daa))

  # pre-define crops column to be NA
  farm_fields$crop = NA

  # Bollean check to see if the field has been meadow in a previous year. If
  # it has, then we want to continue planting meadow on it (This was a
  # decision we made.)
  has_meadow <- (classed_fields %>% filter(field_id %in% farm_fields$field_id)
```

```
                  %>% filter(crop == "meadow") %>%
                    dplyr::select(field_id) %>% pull() %>% length() > 0)


  ### Determining how much land the crops should cover.

  # This if statement checks if is currently a pasture portion for the reporting
  # farm, or if there has been in the past.
  if((past_area %>% is.na() == FALSE) | has_meadow) {

    # If it does, or did, then this special routine is enacted:

    # Determines the pasture percentage of the total farm
    past_percent <- (past_area/farm_area)

    # If it cannot be calculated, then it is set to 0
    if(past_percent %>% is.na()){past_percent = 0}

    # The crop fractions provided to us in the source data did not account for
    # the pasture fraction. therefore we need to reduce the other crop
    # fractions by this amount.
    adjuster <- past_percent / length(crop_filter$crop)
    crop_filter$crop_area_percent <- crop_filter$crop_area_percent-adjuster

    # creating a new entry for pasture and adding it to the now updated crop\
    # fractions list
    past_row <-
      tibble(
        farm_id = farm,
        year = c_year,
        farm_area_daa = crop_filter$farm_area_daa[1],
        pastrure_area_daa = past_area,
        crop = "meadow",
        crop_area_percent = past_percent
      )

    crop_filter <- rbind(crop_filter, past_row)

    # Creating an updated "SHOULD BE" and "CURRENTLY IS" crop coverage
    # datafram
    crop_coverage <- tibble(
      crop = crop_filter$crop,
      to_cover = (farm_area_hru*crop_filter$crop_area_percent))

    # Setting the initial actual coverage to 0 ("CURRENTLY IS")
```

```
    crop_coverage$actual <- 0

    # This extracts the fields that were previously pasture fields.
    previously_pasture <- classed_fields %>%
      filter(field_id %in% farm_fields$field_id) %>%
      filter(crop == "meadow") %>%
      dplyr::select(field_id) %>% pull()

    if (length(previously_pasture) > 0) {
      # if some of the fields were previously pasture, they are set to pasture
      # again:
      farm_fields$crop[
        which(farm_fields$field_id %in% previously_pasture)] = "meadow"

      # We save the amount of area the meadow crops cover, so that we can
      # adjust the other "SHOULD BE" fractions correctly.
      custom_actual <- farm_fields %>% filter(crop == "meadow") %>%
        dplyr::select(field_area_daa) %>% pull() %>% sum(na.rm = T)

      crop_coverage$actual[which(crop_coverage$crop=="meadow")] = custom_actual
    }

    # Now the algorithm continues as normal.

    # This is the routine that is run, when no meadow is detected:
    }else{

      # creating a tibble which tracks how much land the crops SHOULD cover
      crop_coverage <- tibble(
          crop = crop_filter$crop,
          to_cover = (farm_area_hru * crop_filter$crop_area_percent)
        )

    # pre define crop coverage to 0 (CURRENT)
    crop_coverage$actual <- 0
  }

  ### Crop classification

  # Now we know how much land the crops SHOULD cover, it is time to assign
  # the fields accordingly. We do this with a WHILE loop, which keeps going
  # until we have classified every field.

  while(any(farm_fields$crop %>% is.na())){
```

```
    # run through all the crops in the crop coverage list and determine or
    # update their current coverage
    for (c_crop in crop_coverage$crop) {

      # figure out which index we are in the list
      crop_index <- which(c_crop ==  crop_coverage$crop)

      # Determine/UPDATE the current actual coverage of the crop (sum)
      crop_coverage$actual[crop_index] <-
        farm_fields %>% filter(crop == c_crop) %>%
        dplyr::select(field_area_daa) %>% sum()
    }

    # Calculate the difference between crop SHOULD BE coverage and ACTUAL
    # coverage
    crop_coverage$diff <- crop_coverage$actual-crop_coverage$to_cover

    # determine the maximum difference.
    max_diff <- crop_coverage$diff %>% min(na.rm = T)

    # determine which crop has the maximum difference
    max_diff_crop <- crop_coverage$crop[which(crop_coverage$diff == max_diff)
                                        %>% min(na.rm = T)]

    # determine the biggest field left un-classified.
    biggest_na_field <- farm_fields %>% filter(crop %>% is.na()) %>%
      arrange(desc(field_area_daa)) %>% nth(1)

    # Set the field which is still NA, and also the biggest to the crop with
    # the maximum difference (This code seems weird, and could be
    # improved?)
    farm_fields$crop[which((farm_fields$crop %>% is.na() == TRUE) &
                           farm_fields$field_area_daa == biggest_na_field$field_area_daa
    )] <- max_diff_crop
  }

  # After having classified the field, we update the actual coverage value
  # the respective crop that has been classified.
  for (c_crop in crop_coverage$crop) {
    crop_index <- which(c_crop ==  crop_coverage$crop)
    crop_coverage$actual[crop_index] <-
      farm_fields %>% filter(crop == c_crop) %>% dplyr::select(field_area_daa) %>% sum()
  }

  # save the year of this crop assignment in the dataframe
```

```
    farm_fields$year = c_year

    # add the classification to the result dataframe
    classed_fields <- rbind(classed_fields, farm_fields)

    # and move onto the next farm
  }
  # for every year
}

# Remove the first NA line
classed_fields <- classed_fields[-1, ]
```

Now we can have a look at our results (not evaluated)

```
datatable(classed_fields)
```

We can now extract the crop rotation from this datatable. (not evaluated)

```
# the format of our final dataframe
final_df <- tibble(field = NA,
                   y_2016 = NA,
                   y_2017 = NA,
                   y_2018 = NA,
                   y_2019 = NA)

# fields we need to extract
field_ids <- classed_fields$field_id %>% unique()
```

We will do this with this for loop (note this is bad R-practice. Should be done with something like `pivot_longer` (TODO))

```
for(c_field_id in field_ids) {

  crop_rotation <- tibble(field = c_field_id)
  crops <- classed_fields %>% filter(field_id == c_field_id) %>%
    dplyr::select(crop) %>% t() %>% as_tibble(.name_repair = "minimal")

  colnames(crops) <- c("y_2016", "y_2017", "y_2018", "y_2019")

  crop_rotation <- cbind(crop_rotation, crops) %>% as_tibble()

  final_df <- rbind(final_df, crop_rotation)
}
```

```
# Remove the first NA line
crop_rotation <- final_df[-1,]

# Save the crop rotation in CSV format
write_csv(x = crop_rotation, file = "model_data/input/crop/cs10_crop_rotation.csv")
```

Here is a look at the crop rotation:

```
lu_shp <- 'model_data/input/crop/land_with_cr.shp'

lu_sf <- read_sf(lu_shp)

years <- paste0("y_", 2016:2019)

for (year in years) {
  index <- which(colnames(lu_sf) == year)
  lu_sf_filt <- lu_sf[c(index,length(lu_sf))]

  plot <- ggplot(data = lu_sf_filt) +
    geom_sf(color = "black", aes(fill = get(year)))+
    guides(fill=guide_legend(title=year))+
    ggtitle("CS10 Catchment", "crop rotation")+
    theme(legend.position="right")+
      theme(axis.text.x=element_blank(), #remove x axis labels
        axis.ticks.x=element_blank(), #remove x axis ticks
        axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
        )

  print(plot)
}
```

5.2.4 Extrapolating the Crop Rotation

For OPTAIN, the crop rotation needs to span from 1988 to 2020. We currently have 2016 to 2019.

```
crop_rotation <- read_csv("model_data/input/crop/cs10_crop_rotation.csv",
                          show_col_types = F)

rotation <- crop_rotation %>% dplyr::select(-field)

crop_rotation_extrapolated <-
```

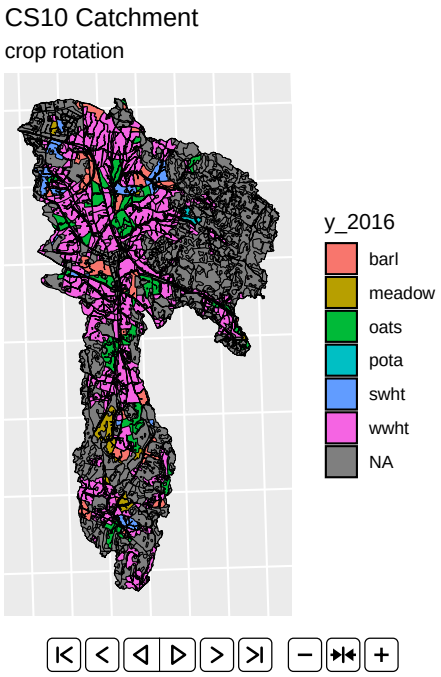


Figure 5.3: Generated crop map for years 2016-2019

```
  cbind(crop_rotation,
        rotation,
        rotation,
        rotation,
        rotation,
        rotation,
        rotation,
        rotation,
        rotation$y_2016)

# set the column names to be in the correct format
colnames(crop_rotation_extrapolated) <- c("field", paste0("y_", 1988:2020))
```

5.3 Merging crop rotation with BuildR output

The output of `SWATbuildR` "land.shp" needs to be connected to the newly generated crop map.

```r
lu <- read_sf("model_data/cs10_setup/optain-cs10/data/vector/land.shp")

# temporary rename to field, for the left join
# unstable, fix! TODO
colnames(lu)[2] = "field"

# join the crop rotation and land use layer by their field ID
lu_cr <- left_join(lu, crop_rotation_extrapolated, by = "field" )

# reset column name.
# unstable, fix! TODO
colnames(lu_cr)[2] = "lu"

# Write the new shape file
write_sf(lu_cr, "model_data/input/crop/land_with_cr.shp")
```

Related Issues

Extrapolating the crop rotation #41

Chapter 6

Landuse.lum update

This section covers the modifications made to the landuse file. For some reason it was written in a tutorial fashion, as if the reader were new to R.

```
require(tidyverse)
require(reshape2)
require(sf)
require(DT)
require(dplyr) # require dplyr last to overwrite plyr count()
require(ggplot2)
```

ggplot theme(s):

```
sf_theme <- theme(axis.text.x=element_blank(), #remove x axis labels
        axis.ticks.x=element_blank(), #remove x axis ticks
        axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
        )
```

6.1 Data preperation

We read in the land use file. We want to set `skip = 1` to ignore the SWAT+editor text, and set `header = T`. We then convert it to a `tibble` format for better printing to console

```
landuse_lum <-
  read.table("model_data/cs10_setup/swat_input/landuse.lum", skip = 1, header = T) %>% tibble::as
```

```
head(landuse_lum)
```

```
## # A tibble: 6 x 14
##   name       cal_group plnt_com mgt   cn2   cons_prac urban urb_ro ov_mann tile
##   <chr>      <chr>     <chr>    <chr> <chr> <chr>     <chr> <chr>  <chr>   <chr>
## 1 a_001f_1_~ null      nopl_co~ null  null  null      null  null   null    mw24~
## 2 a_001f_2_~ null      nopl_co~ null  null  null      null  null   null    mw24~
## 3 a_001f_3_~ null      nopl_co~ null  null  null      null  null   null    mw24~
## 4 a_001f_4_~ null      nopl_co~ null  null  null      null  null   null    mw24~
## 5 a_001f_5_~ null      nopl_co~ null  null  null      null  null   null    mw24~
## 6 a_001f_6_~ null      nopl_co~ null  null  null      null  null   null    mw24~
## # i 4 more variables: sep <chr>, vfs <chr>, grww <chr>, bmp <chr>
```

Our `field_id` for our cropland does not match `name` of `landuse_lum` so we need to parse it out, we can do this many ways, but a safe way is to split it by "_" and combine the first 3 splits with that same underscore:

```
splitted <- landuse_lum$name %>% str_split("_")

landuse_lum$field_id <-
  paste(splitted %>% map(1), splitted %>% map(2), splitted %>% map(3), sep = "_")

head(landuse_lum$field_id)
```

```
## [1] "a_001f_1" "a_001f_2" "a_001f_3" "a_001f_4" "a_001f_5" "a_001f_6"
```

But wait, this does not work for our "non-fields". So lets find out which ones they are, and set them to `NA`

```
not_fields <- which(!grepl(x=landuse_lum$field_id, "a_"))
landuse_lum$field_id[not_fields] <- NA
```

So, what non-fields do we have?

```
landuse_lum$name[not_fields]
```

```
## [1] "frst_lum" "past_lum" "rngb_lum" "urml_lum" "utrn_lum" "wetf_lum"
```

Good to know. We'll keep that in mind.

6.2 Assigning landuse values

6.2.1 Setting `cal_group`

There is no info on this column, so we are going to leave it as `null`. Dicussion in Issue #18

6.2.2 Setting `plnt_com`

This step will be done by SWATFarmR in Section **??**. Dicussed in Issue #15

6.2.3 Setting `mgt`

This step will be done by SWATFarmR in Section **??**.

Discussed in Issue #14

6.2.4 Setting `urb_ro`

We want to set the `urb_ro` column for all urban land uses (in our case this would be `urml` and `utrn`) to `usgs_reg`.

This has been discussed in Issue #6

We can do this like so:

```
landuse_lum$urb_ro[which(landuse_lum$name %in% c("urml_lum", "utrn_lum"))] <-
  "usgs_reg"
```

```
## # A tibble: 2 x 15
##   name     cal_group plnt_com mgt   cn2   cons_prac urban urb_ro   ov_mann tile
##   <chr>    <chr>     <chr>    <chr> <chr> <chr>     <chr> <chr>    <chr>   <chr>
## 1 urml_lum null      null     null  null  null      null  usgs_reg null    null
## 2 utrn_lum null      null     null  null  null      null  usgs_reg null    null
## # i 5 more variables: sep <chr>, vfs <chr>, grww <chr>, bmp <chr>,
## #   field_id <chr>
```

Very good. In our case this was only two – could be done by hand, but that will not be the case for all of our land uses.

6.2.5 Setting `urban`

This one is easy, we set our `urban` column to an urban parameter set of the same name. The rest we leave as `null`

This has been discussed in Issue #5

```
landuse_lum$urban[which(landuse_lum$name =="utrn_lum")] <- "utrn"
landuse_lum$urban[which(landuse_lum$name =="urml_lum")] <- "urml"
```

```
## # A tibble: 1 x 15
##   name     cal_group plnt_com mgt   cn2   cons_prac urban urb_ro   ov_mann tile
##   <chr>    <chr>     <chr>    <chr> <chr> <chr>     <chr> <chr>    <chr>   <chr>
## 1 urml_lum null      null     null  null  null      urml  usgs_reg null    null
## # i 5 more variables: sep <chr>, vfs <chr>, grww <chr>, bmp <chr>,
## #   field_id <chr>
```

Lets make sure other land uses still have `null`:

```
## # A tibble: 1 x 15
##   name     cal_group plnt_com  mgt   cn2   cons_prac urban urb_ro ov_mann tile
##   <chr>    <chr>     <chr>     <chr> <chr> <chr>     <chr> <chr>  <chr>   <chr>
## 1 frst_lum null      nopl_comm null  null  null      null  null   null    null
## # i 5 more variables: sep <chr>, vfs <chr>, grww <chr>, bmp <chr>,
## #   field_id <chr>
```

Looks good.

6.2.6 Setting Manning's n (`ovn`)

This has been discussed in Issue #13

Lets get the easy ones out of the way

```
landuse_lum$ov_mann[which(landuse_lum$name =="past_lum")] <- "densegrass"
landuse_lum$ov_mann[which(landuse_lum$name =="rngb_lum")] <- "rangeland_20cover"
landuse_lum$ov_mann[which(landuse_lum$name =="urml_lum")] <- "urban_rubble"
landuse_lum$ov_mann[which(landuse_lum$name =="urtn_lum")] <- "urban_asphalt"
```

Don't fall asleep yet! For `wetf` we want `forest_heavy` but with a higher value. this means we need to add a new entry. And since we are doing this fancy Rmarkdown stuff, we're going to do it with code! Lets jump in and get at this file. (REMEBER TO REMOVE THE TEMP)

```
ovn_table_path <- "model_data/cs10_setup/swat_input/ovn_table.lum"
ovn_table_path_new <- "model_data/cs10_setup/swat_input_mod/ovn_table.lum"

ovn_table <- readLines(ovn_table_path)
```

Good, now where is this forest heavy entry? and what does the format look like?

```
index <- grepl(x=ovn_table, "forest_heavy") %>% which %>% min()

ovn_table[c(2, index)] %>% print()
```

```
## [1] "name                  ovn_mean       ovn_min       ovn_max  description"
## [2] "forest_heavy           0.80000       0.70000       0.90000  Forest_heavy"
```

Now, lets make our own and add it in. But only if it doesn't exist it (Like if the script has been run before...)

```
if(grepl(x = ovn_table, "forest_heavy_cs10") %>% which %>% length() == 0) {
  entry <-
    "forest_heavy_cs10      0.90000       0.80000       0.95000  Forest_heavy_mod"
  ovn_table <- c(ovn_table, entry)
}
```

Did it work? Yes:

```
ovn_table %>% tail()
```

```
## [1] "forest_med             0.60000       0.50000       0.70000  Forest_medimum_good"
## [2] "forest_heavy           0.80000       0.70000       0.90000  Forest_heavy"
## [3] "urban_asphalt          0.01100       0.01100       0.01100  Urban_asphalt"
## [4] "urban_concrete         0.01200       0.01200       0.01200  Urban_concrete"
## [5] "urban_rubble           0.02400       0.02400       0.02400  Urban_rubble"
## [6] "forest_heavy_cs10      0.90000       0.80000       0.95000  Forest_heavy_mod"
```

Now lets write this new table

```
writeLines(ovn_table, con = ovn_table_path_new)
```

And now we can enter our wetland class:

```
landuse_lum$ov_mann[which(landuse_lum$name =="wetf_lum")] <- "forest_heavy_cs10"
```

For the fields we are going to need the crop rotation info which we created in section ??.

```
crop_rotation <- read_csv("model_data/input/crop/cs10_crop_rotation.csv",
    show_col_types = F)

head(crop_rotation)
```

```
## # A tibble: 6 x 5
##   field    y_2016 y_2017 y_2018 y_2019
##   <chr>    <chr>  <chr>  <chr>  <chr>
## 1 a_025f_1 wwht   wwht   wwht   wwht
## 2 a_025f_3 wwht   wwht   wwht   wwht
## 3 a_025f_4 wwht   wwht   wwht   wwht
## 4 a_025f_2 wwht   wwht   wwht   wwht
## 5 a_105f_1 wwht   wwht   wwht   wwht
## 6 a_105f_4 wwht   wwht   wwht   wwht
```

We have decided to classify `ovn` based on the degree to which a crop rotation contains `meadow`, conventional crops (`wwht`, `pota`), and conservation crops (all others). To do this we need to analyze the crop rotation.

Now it gets a little complicated, but trust me its not actually as bad as it looks. We need to count how many times certain crops show up in the crop rotation of **a certain field**. Lets do it for **conventional crops** first.

Now, lets get into it

```
conv_crop_count <- crop_rotation %>% melt(. ,"field") %>% group_by(field) %>%
  filter(value %in% c("wwht", "pota")) %>% dplyr::count() %>% dplyr::rename(conv = n)

head(conv_crop_count)
```

```
## # A tibble: 6 x 2
## # Groups:   field [6]
##   field     conv
##   <chr>    <int>
## 1 a_001f_1     4
## 2 a_001f_2     4
## 3 a_001f_3     4
## 4 a_001f_4     4
## 5 a_001f_5     4
## 6 a_001f_6     4
```

What happened here? well we took our `crop_rotation`, and melted it by `field` using `melt`. This function converts the data into *tidy format* (**?**). This format makes it easier to apply the following operations on **a field basis**. What does this look like?

```
crop_rotation %>% melt(. ,"field") %>% arrange(field) %>% head()
```

```
##      field variable value
## 1 a_001f_1   y_2016  wwht
## 2 a_001f_1   y_2017  wwht
## 3 a_001f_1   y_2018  wwht
## 4 a_001f_1   y_2019  wwht
## 5 a_001f_2   y_2016  wwht
## 6 a_001f_2   y_2017  wwht
```

`melt` is a function from `reshape2` which is why we `required` it. And what does that "`.`" mean in there, to the left of `"field"`?? That is a code word for the object to the left of the "pipe" (in this case `crop_rotation`). the pipe is "`%>%`" and passes the object to the left of it, into the function to the right of it. *look it up*!

When `arranged` by field, we can see that every field gets one entry per crop per year. This is a good format to count how many times we have a certain type of crop on a field.

Next we `group_by` the `field` – this means all the following operations will be done on a **field basis**. Then we filter our `value` (which is the crop name). For conventional we needed to filter in any fields with the crop `"wwht"` or `"pota"`.

What does this look like?

```
crop_rotation %>% melt(. ,"field") %>% group_by(field) %>%
  filter(value %in% c("wwht", "pota"))
```

```
## # A tibble: 1,519 x 3
## # Groups:   field [415]
##    field    variable value
##    <chr>    <fct>    <chr>
##  1 a_025f_1 y_2016   wwht
##  2 a_025f_3 y_2016   wwht
##  3 a_025f_4 y_2016   wwht
##  4 a_025f_2 y_2016   wwht
##  5 a_105f_1 y_2016   wwht
##  6 a_105f_4 y_2016   wwht
##  7 a_105f_2 y_2016   wwht
##  8 a_105f_3 y_2016   wwht
```

```
##  9 a_187f_1 y_2016   wwht
## 10 a_206f_1 y_2016   wwht
## # i 1,509 more rows
```

Looks good. We only have crops with winter wheat and potatoes, for every year. Exactly what we need, now we just need to `count()` them.

```
conv_crop_count <- crop_rotation %>% melt(. ,"field") %>% group_by(field) %>%
  filter(value %in% c("wwht", "pota")) %>% dplyr::count()

head(conv_crop_count)
```

```
## # A tibble: 6 x 2
## # Groups:   field [6]
##   field        n
##   <chr>    <int>
## 1 a_001f_1     4
## 2 a_001f_2     4
## 3 a_001f_3     4
## 4 a_001f_4     4
## 5 a_001f_5     4
## 6 a_001f_6     4
```

And we are back where we started! See, not that complicated. One last thing we need to do is rename `n` to `conv`, we do that like so:

```
conv_crop_count <- crop_rotation %>% melt(. ,"field") %>% group_by(field) %>%
  filter(value %in% c("wwht", "pota")) %>% dplyr::count() %>% dplyr::rename(conv = n)

head(conv_crop_count)
```

```
## # A tibble: 6 x 2
## # Groups:   field [6]
##   field     conv
##   <chr>    <int>
## 1 a_001f_1     4
## 2 a_001f_2     4
## 3 a_001f_3     4
## 4 a_001f_4     4
## 5 a_001f_5     4
## 6 a_001f_6     4
```

Now lets go ahead and do the same thing for the two other categories: `cons` and`meadow`.

```
meadow_crop_count <- crop_rotation %>% melt(. ,"field") %>% group_by(field) %>%
  filter(value == "meadow" ) %>% dplyr::count() %>%
  dplyr::rename(meadow = n)

cons_crop_count <- crop_rotation %>% melt(. ,"field") %>% group_by(field) %>%
  filter(!(value %in% c("wwht", "pota", "meadow"))) %>% dplyr::count() %>%
  dplyr::rename(cons = n)

cons_crop_count %>% head()
```

```
## # A tibble: 6 x 2
## # Groups:   field [6]
##   field     cons
##   <chr>    <int>
## 1 a_002f_1     2
## 2 a_002f_2     2
## 3 a_002f_3     4
## 4 a_002f_4     4
## 5 a_002f_5     2
## 6 a_002f_6     4
```

```
meadow_crop_count %>% head()
```

```
## # A tibble: 6 x 2
## # Groups:   field [6]
##   field    meadow
##   <chr>     <int>
## 1 a_012f_3      4
## 2 a_012f_6      4
## 3 a_016f_3      4
## 4 a_017f_2      4
## 5 a_017f_3      4
## 6 a_046f_1      4
```

`meadow` was a simple filter, all we have to do was grab crops with the `meadow` name. For `cons` crops, we just grabbed the remaining crops that were not `conv` or `meadow`.

Great. We have these 3 separate, we need to combine them. we can do that with `left_join` and join by the `field` column which contains our IDs

```
# create a base dataframe to join to
crop_fractions <- crop_rotation %>% dplyr::select(field) %>% distinct()
```

```
# join our 3 data frames
crop_fractions <- left_join(crop_fractions, conv_crop_count, by = "field")
crop_fractions <- left_join(crop_fractions, cons_crop_count, by = "field")
crop_fractions <- left_join(crop_fractions, meadow_crop_count, by = "field")

# lets have a look
crop_fractions %>% head()
```

```
## # A tibble: 6 x 4
##   field     conv  cons meadow
##   <chr>    <int> <int>  <int>
## 1 a_025f_1     4    NA     NA
## 2 a_025f_3     4    NA     NA
## 3 a_025f_4     4    NA     NA
## 4 a_025f_2     4    NA     NA
## 5 a_105f_1     4    NA     NA
## 6 a_105f_4     4    NA     NA
```

That does not look good! when it seems like when the `count` is 0, it is returned as `NA`. Lets fix that...

```
crop_fractions <- crop_fractions %>%
  mutate(cons = ifelse(is.na(cons), 0, cons))
crop_fractions <- crop_fractions %>%
  mutate(conv = ifelse(is.na(conv), 0, conv))
crop_fractions <- crop_fractions %>%
  mutate(meadow = ifelse(is.na(meadow), 0, meadow))
```

What are we doing here? we are `mutating` the the 3 columns using an `ifelse` statement. The statement is simple. If the value `is.na` then we set it to `0`. Otherwise, we set it to the same value it had before.

did it work?

```
crop_fractions %>% head()
```

```
## # A tibble: 6 x 4
##   field     conv  cons meadow
##   <chr>    <dbl> <dbl>  <dbl>
## 1 a_025f_1     4     0      0
## 2 a_025f_3     4     0      0
## 3 a_025f_4     4     0      0
## 4 a_025f_2     4     0      0
## 5 a_105f_1     4     0      0
## 6 a_105f_4     4     0      0
```

Very good. Now we need to decide what to classify our fields at. lets Define some `functions` to do that.

```
is_meadow <- function(conv, cons, meadow) {
  ((meadow > cons) & (meadow > conv)) %>% return()
}

is_cons <- function(conv, cons, meadow) {
  ((cons >= meadow) & (cons > conv)) %>% return()
}

is_conv <- function(conv, cons, meadow) {
  ((conv >= meadow) & (conv >= cons)) %>% return()
}
```

The `&` sign means that both conditions need to be met. and `>=` you should know already. Lets use those functions in action. lets do the meadow first. We will create a new dataframe from crop fractions, named `field_class`.

```
field_class <- crop_fractions %>% mutate(meadow = is_meadow(conv,cons,meadow))

head(field_class)
```

```
## # A tibble: 6 x 4
##   field     conv  cons meadow
##   <chr>    <dbl> <dbl> <lgl>
## 1 a_025f_1     4     0 FALSE
## 2 a_025f_3     4     0 FALSE
## 3 a_025f_4     4     0 FALSE
## 4 a_025f_2     4     0 FALSE
## 5 a_105f_1     4     0 FALSE
## 6 a_105f_4     4     0 FALSE
```

Ok, so none of those first 6 fields are `meadow` dominated. What about `cons`? (lets stick with our `field_class` dataframe and just keep adding on)

```
field_class <- field_class %>% mutate(cons = is_cons(conv,cons,meadow))
head(field_class)
```

```
## # A tibble: 6 x 4
##   field     conv cons  meadow
##   <chr>    <dbl> <lgl> <lgl>
## 1 a_025f_1     4 FALSE FALSE
## 2 a_025f_3     4 FALSE FALSE
```

```
## 3 a_025f_4     4 FALSE FALSE
## 4 a_025f_2     4 FALSE FALSE
## 5 a_105f_1     4 FALSE FALSE
## 6 a_105f_4     4 FALSE FALSE
```

Nope, not `cons` either. Then it must be `conv` dominant.

```
field_class <- field_class %>% mutate(conv = is_conv(conv,cons,meadow))

field_class %>% head()
```

```
## # A tibble: 6 x 4
##   field    conv  cons  meadow
##   <chr>    <lgl> <lgl> <lgl>
## 1 a_025f_1 TRUE  FALSE FALSE
## 2 a_025f_3 TRUE  FALSE FALSE
## 3 a_025f_4 TRUE  FALSE FALSE
## 4 a_025f_2 TRUE  FALSE FALSE
## 5 a_105f_1 TRUE  FALSE FALSE
## 6 a_105f_4 TRUE  FALSE FALSE
```

Correct! now lets just double check that we didnt have any cases where `all` are `TRUE` or where `all` are `FALSE`

```
field_class %>% dplyr::select(conv, cons, meadow) %>% isTRUE %>% all()
```

```
## [1] FALSE
```

```
field_class %>% dplyr::select(conv, cons, meadow) %>% isFALSE() %>% all()
```

```
## [1] FALSE
```

Looks good to me! carry on:

We have our field classifications now, they will be very useful to us later. Lets `pull` them out of our dataframe to store them as a nice `list`.

```
cons_fields <- field_class %>% filter(cons) %>% dplyr::select(field) %>% pull()
conv_fields <- field_class %>% filter(conv) %>% dplyr::select(field) %>% pull()
meadow_fields <- field_class %>% filter(meadow) %>% dplyr::select(field) %>% pull()

cons_fields %>% head()
```

```
## [1] "a_182f_1" "a_182f_5" "a_182f_3" "a_182f_4" "a_182f_2" "a_182f_7"
```

```
conv_fields %>% head()
```

```
## [1] "a_025f_1" "a_025f_3" "a_025f_4" "a_025f_2" "a_105f_1" "a_105f_4"
```

```
meadow_fields %>% head()
```

```
## [1] "a_182f_6" "a_182f_8" "a_017f_3" "a_017f_2" "a_012f_3" "a_012f_6"
```

Fantastic! Now that was a big task, but it makes the next bit very easy. Lets assign the correct `ovn` to the types of fields we have classified:

6.2.6.1 Agricultural Fields: Meadow

```
landuse_lum$ov_mann[which(landuse_lum$field_id %in% meadow_fields)] <- "shortgrass"
```

6.2.6.2 Agricultural Fields: Conventional

```
landuse_lum$ov_mann[which(landuse_lum$field_id %in% conv_fields)] <- "convtill_nores"
```

6.2.6.3 Agricultural Fields: Conservational

```
landuse_lum$ov_mann[which(landuse_lum$field_id %in% cons_fields)] <- "falldisk_res"
```

6.2.6.4 Forest

Initially we wanted certain classes for the forest land use based on how good the soil quality was. But to do this, we would need multiple land uses, which we do not have. We will just use `forest_medium` for all `frst`. We can only change this if we go back to `buildR` and define more generic forest classes.

```
landuse_lum$ov_mann[which(landuse_lum$name =="frst_lum")] <- "forest_med"
```

6.2.6.5 Summary

With all that data processing out of the way, lets have a quick look at what we've done:

```
landuse_lum$ov_mann[which(landuse_lum$name == "past_lum")] <-
  "densegrass"
landuse_lum$ov_mann[which(landuse_lum$name == "rngb_lum")] <-
  "rangeland_20cover"
landuse_lum$ov_mann[which(landuse_lum$name == "urml_lum")] <-
  "urban_rubble"
landuse_lum$ov_mann[which(landuse_lum$name == "urtn_lum")] <-
  "urban_asphalt"
landuse_lum$ov_mann[which(landuse_lum$name == "frst_lum")] <-
  "forest_med"
landuse_lum$ov_mann[which(landuse_lum$field_id %in% meadow_fields)] <-
  "shortgrass"
landuse_lum$ov_mann[which(landuse_lum$field_id %in% conv_fields)] <-
  "convtill_nores"
landuse_lum$ov_mann[which(landuse_lum$field_id %in% cons_fields)] <-
  "falldisk_res"
landuse_lum$ov_mann[which(landuse_lum$name == "wetf_lum")] <-
  "forest_heavy_cs10"
```

Sweet. Next column..

6.2.7 Setting cn2:

This has been discussed in Issue Issue #1

You know the drill – I am sure you know how to read this code by now.

```
# Brush-brush-weed-grass_mixture_with_brush_the_major_element (poor)
landuse_lum$cn2[which(landuse_lum$name == "rngb_lum")] <- "brush_p"

# Woods (poor)
landuse_lum$cn2[which(landuse_lum$name == "wetf_lum")] <- "wood_p"

# Paved_streets_and_roads;_open_ditches_(incl._right-of-way)
landuse_lum$cn2[which(landuse_lum$name == "utrn_lum")] <- "paveroad"

# Paved_parking_lots_roofs_driveways_etc_(excl_right-of-way)
landuse_lum$cn2[which(landuse_lum$name == "urml_lum")] <- "urban"

# Woods (fair)
landuse_lum$cn2[which(landuse_lum$name == "frst_lum")] <- "wood_f"

# Pasture_grassland_or_range-continuous_forage_for_grazing
landuse_lum$cn2[which(landuse_lum$name == "past_lum")] <- "pastg_g"
```

```
# Meadow-continuous_grass_protected_from_grazing_mowed_for_hay
landuse_lum$cn2[which(landuse_lum$field_id %in% meadow_fields)] <-
  "pasth"

# Row_crops
landuse_lum$cn2[which(landuse_lum$field_id %in% cons_fields)] <-
  "rc_conterres_g"

# Row_crops
landuse_lum$cn2[which(landuse_lum$field_id %in% conv_fields)] <-
  "rc_strow_p"
```

6.2.8 Setting cons_prac

This has been discussed in Issue #4

For this, we need to add some custom entries to the database:

```
cons_prac_path <- "model_data/cs10_setup/swat_input/cons_practice.lum"
cons_prac_path_new <- "model_data/cs10_setup/swat_input_mod/cons_practice.lum"

cons_prac <- readLines(cons_prac_path)
cons_prac[1:3] %>% print()
```

```
## [1] "cons_practice.lum: written by SWAT+ editor v2.1.0 on 2023-05-30 10:39 for SWAT+ rev.60.5.
## [2] "name                    usle_p   slp_len_max  description"
## [3] "up_down_slope          1.00000     121.00000  Up_and_down_slope"
```

Let us do it in a way so that it is only added if it doesn't exist yet:

```
if(grepl(x = cons_prac, "agri_conv") %>% which %>% length() == 0) {
  entry <-
    "agri_conv          1.00000      60.00000  no_convervation"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "agri_part_conv") %>% which %>% length() == 0) {
  entry <-
    "agri_part_conv     0.85000      60.00000  75_percent_convential"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "agri_half") %>% which %>% length() == 0) {
```

```
  entry <-
    "agri_half          0.70000      50.00000  50_percent_convential"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "agri_part_cons") %>% which %>% length() == 0) {
  entry <-
    "agri_part_cons     0.50000      30.00000  75_percent_consveration"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "agri_cons") %>% which %>% length() == 0) {
  entry <-
    "agri_cons     0.30000      30.00000  100_percent_consveration"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "past_cons") %>% which %>% length() == 0) {
  entry <-
    "past_cons   0.1   60   pasture"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "rngb_cons") %>% which %>% length() == 0) {
  entry <-
    "rngb_cons   0.2   60   rangeland"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "frst_cons") %>% which %>% length() == 0) {
  entry <-
    "frst_cons   0.1   60   forest"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "urml_cons") %>% which %>% length() == 0) {
  entry <-
    "urml_cons   1   60   cs10urban"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "utrn_cons") %>% which %>% length() == 0) {
  entry <-
    "utrn_cons   1   60   road"
  cons_prac <- c(cons_prac, entry)
```

```
}

if (grepl(x = cons_prac, "cs10_meadow") %>% which %>% length() == 0) {
  entry <-
    "cs10_meadow   0.2   60   cs10_meadow"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "wetf_cons") %>% which %>% length() == 0) {
  entry <-
    "wetf_cons   0.05   30   wetlands"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "cs10_sed_pond") %>% which %>% length() == 0) {
  entry <-
    "cs10_sed_pond   0.1   30   cs10_sed_pond"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "cs10_cons_wetl") %>% which %>% length() == 0) {
  entry <-
    "cs10_cons_wetl   0.1   30   cs10_cons_wetl"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "cs10_buff_grass") %>% which %>% length() == 0) {
  entry <-
    "cs10_buff_grass   0.25   10   cs10_buff_grass"
  cons_prac <- c(cons_prac, entry)
}

if (grepl(x = cons_prac, "cs_10_buff_wood") %>% which %>% length() == 0) {
  entry <-
    "cs_10_buff_wood   0.2   10   cs_10_buff_wood"
  cons_prac <- c(cons_prac, entry)
}
```

And write the updated file:

```
writeLines(cons_prac, con = cons_prac_path_new)
```

Now we need to define the 0, 25, 50, 75, 100 percent conventional crops. We can use our old “crop_fractions” dataframe:

```
head(crop_fractions)
```

```
## # A tibble: 6 x 4
##   field     conv  cons meadow
##   <chr>    <dbl> <dbl>  <dbl>
## 1 a_025f_1     4     0      0
## 2 a_025f_3     4     0      0
## 3 a_025f_4     4     0      0
## 4 a_025f_2     4     0      0
## 5 a_105f_1     4     0      0
## 6 a_105f_4     4     0      0
```

Lets derive which fields get classified as what

```
p100_conv <- crop_fractions %>% filter(conv==4) %>% dplyr::select(field) %>% pull()
p75_conv <- crop_fractions %>% filter(conv==3) %>% dplyr::select(field) %>% pull()
p50_conv <- crop_fractions %>% filter(conv==2) %>% dplyr::select(field) %>% pull()
p25_conv <- crop_fractions %>% filter(conv==1) %>% dplyr::select(field) %>% pull()
p0_conv <- crop_fractions %>% filter(conv==0) %>% dplyr::select(field) %>% pull()
```

Now we can assign the all the land uses

```
# Agricultural
landuse_lum$cons_prac[which(landuse_lum$field_id %in% p100_conv)] <-
  "agri_conv"
landuse_lum$cons_prac[which(landuse_lum$field_id %in% p75_conv)] <-
  "agri_part_conv"
landuse_lum$cons_prac[which(landuse_lum$field_id %in% p50_conv)] <-
  "agri_half"
landuse_lum$cons_prac[which(landuse_lum$field_id %in% p25_conv)] <-
  "agri_part_cons"
landuse_lum$cons_prac[which(landuse_lum$field_id %in% p0_conv)] <-
  "agri_cons"

# Meadow
landuse_lum$cons_prac[which(field_class$meadow)] <-
  "cs10_meadow"

# Generic
landuse_lum$cons_prac[which(landuse_lum$name == "past_lum")] <-
  "past_cons"
landuse_lum$cons_prac[which(landuse_lum$name == "rngb_lum")] <-
  "rngb_cons"
landuse_lum$cons_prac[which(landuse_lum$name == "frst_lum")] <-
```

```
  "frst_cons"
landuse_lum$cons_prac[which(landuse_lum$name == "urml_lum")] <-
  "urml_cons"
landuse_lum$cons_prac[which(landuse_lum$name == "utrn_lum")] <-
  "utrn_cons"
landuse_lum$cons_prac[which(landuse_lum$name == "wetf_lum")] <-
  "wetf_cons"

# Measures
landuse_lum$cons_prac[which(landuse_lum$name == "cs10_sed_pond")] <-
  "cs10_sed_pond"
landuse_lum$cons_prac[which(landuse_lum$name == "cs10_cons_wetl")] <-
  "cs10_cons_wetl"
landuse_lum$cons_prac[which(landuse_lum$name == "cs10_buff_grass")] <-
  "cs10_buff_grass"
landuse_lum$cons_prac[which(landuse_lum$name == "cs_10_buff_wood")] <-
  "cs_10_buff_wood"
```

6.2.9 Setting `tile`

This column has already been completed by `SWATbuildR` in **??**

6.2.10 Setting `sep`

This column has been left as `null` as discussed in Issue #9

6.2.11 Setting `vfs`

This column has been left as `null` as discussed in Issue #10

6.2.12 Setting `grww`

This column has been left as `null` as discussed in Issue #11

6.2.13 Setting `bmp`

This column has been left as `null` as discussed in Issue #12

6.3 Writing Changes

We are done with our modifications however, we need to remove the `field_id` column

```
landuse_lum <- landuse_lum %>% dplyr::select(-field_id)
```

lets take a look at the final product:

```
datatable(landuse_lum)
```

Show 10 entries Search:

	name	cal_group	plnt_com	mgt	cn2	cons_prac	urban	urb_ro	ov_mann	tile	se
1	a_001f_1_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
2	a_001f_2_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
3	a_001f_3_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
4	a_001f_4_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
5	a_001f_5_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
6	a_001f_6_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
7	a_001f_7_drn_lum	null	nopl_comm	null	rc_strow_p	agri_conv	null	null	convtill_nores	mw24_1000	nu
8	a_002f_1_drn_lum	null	nopl_comm	null	rc_strow_p	agri_half	null	null	convtill_nores	mw24_1000	nu
9	a_002f_2_lum	null	nopl_comm	null	rc_strow_p	agri_half	null	null	convtill_nores	null	nu
10	a_002f_2_drn_lum	null	nopl_comm	null	rc_strow_p	agri_half	null	null	convtill_nores	mw24_1000	nu

Showing 1 to 10 of 577 entries Previous 1 2 3 4 5 … 58 Next

Lets write out changes in a new directory where we will store any SWAT+ input files which we have modified from their source BuildR form.

```
dir.create("model_data/cs10_setup/swat_input_mod", showWarnings = F)
new_lum_path <- "model_data/cs10_setup/swat_input_mod/landuse.lum"

write.table(landuse_lum, file = new_lum_path, sep = "\t", quote = F, row.names = F)
```

But wait – we need to keep that pesky header, otherwise the FarmR will be very angry with us.

```
lum_lines <- readLines(new_lum_path)

header <- "header header HEADER, delete me and you will regret it FOREVER!"

lum_lines2<- c(header,lum_lines)

writeLines(text = lum_lines2, con = new_lum_path)
```

Done!

Now that We've changed another input file, we should test if our model still runs. We will do that with our custom `test_swat_mod()` function. This function is basically the same as the previous `test_swat()` function, only that it incorporates the modified SWAT input files

```
source("model_data/code/test_swat_mod.R")

simout <- test_swat_mod()

cat(tail(simout), sep = "\n")
```

Related Milestone

landuse.lum update

Related minor issues

Don't write rownames! - #50

Update the land use file - #40

Chapter 7

Field Site Loggers

7.1 Introduction

This chapter is concerned with preparing the measured data for the OPTAIN field scale models. This will involve loading the data, transforming it into a usable format, and performing a quality check. The end of this chapter will result in data fit for use in SWAP as well as a tangentially related antecedent precipitation index (API) model for use in the SWAT+ modelling work, related to scheduling management operations using SWATFarmR .

7.1.1 Prerequisites

The following packages are required for this chapter:

```
require(readr) # for reading in data
require(dplyr) # for manipulating data
require(stringr) # for manipulating text
require(purrr) # for extracting from nested lists
require(plotly) # for diagnostic plotting
require(lubridate) # for date conversions
require(reshape2) # for data conversions
require(wesanderson) # plot colors
require(DT) # for tables
require(mapview) # for plotting
require(sf) # for shapefiles

show_rows <- 500 # number of rows to show in the tables
```

As well as the following custom functions:

```
source("model_data/code/plot_logger.R")
```

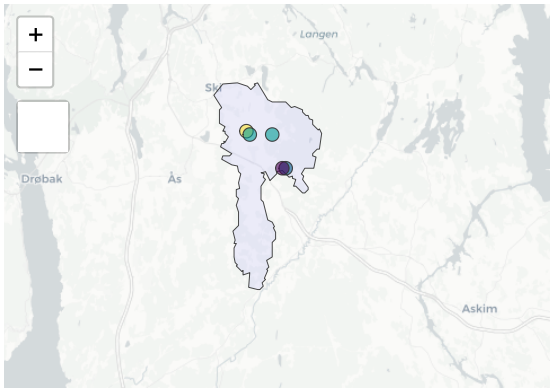
7.1.2 Locations

The loggers are described as such:

1. Logger3(600409): deep rocky sand (forest?)
2. Logger4(600410): Tormods farm (agri)
3. Logger2(600411): John Ivar lower area (agri)
4. Logger1(600413): John Ivar behind the barn (agri)

And are located here:

```
field_sites_shp <- read_sf("model_data/field_sites/geospatial/cs10_field_sites.shp")
basin_shp <- read_sf("model_data/input/shape/cs10_basin.shp")

basin_map <- mapview(basin_shp, alpha.regions = .1, legend = FALSE)
field_map <- mapview(field_sites_shp, legend = FALSE)

basin_map+field_map
```



7.2 Data Processing

7.2.1 Loading Data

To start we need to load in our data from the *GroPoint Profile* data loggers. The logger is approx. 1m long and has 7 temperature sensors, and 6 soil moisture

sensors located along its segments. **?**

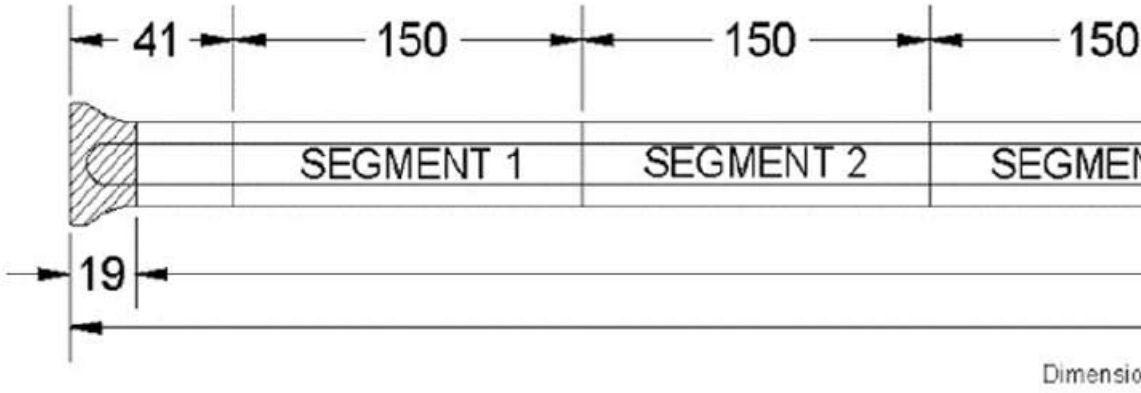


Figure 7.1: GroPoint Segment Schematic (GroPoint Profile User Manual, Page 6)

The logger data has been modified before import into R. The file names have been changed and column headers have been added. Missing values stored as "Error" have been removed.

Columns headers have assigned with variable name, depth, and unit of depth separated by an underscore.

```
path <- "model_data/field_sites/loggers/data/"
```

Loading the data and modifying it is performed with `readr` and `dplyr`.

```
# list of files
datasheets <- list.files(path, full.names = TRUE)

# loading in the data, with an ID column 'source'
data <- read_csv(datasheets, show_col_types = F, id = "source")

# translating the source text to logger ID
data$site <- data$source %>% str_remove(path) %>% str_remove(".csv") %>%
  str_split("_") %>% map(1) %>% unlist()

# removing the source column
data <- data %>% select(-source)

# removing duplicates
data <- data %>% distinct()
```

Sensor	# Temp	Locations (from top)
GPLP-2	4	35, 100, 200, and 300
GPLP-3	6	35, 100, 200, 300, 400, and 450
GPLP-4	7	35, 100, 200, 300, 400, 500, and 600
GPLP-5	9	35, 100, 200, 300, 400, 500, 600, 700, and 750
GPLP-6	11	35, 100, 200, 300, 400, 500, 550, 650, 750, 850,
GPLP-8	14	35, 100, 200, 300, 400, 500, 550, 650, 750, 850,

Figure 7.2: GroPoint Temperature sensor depths. We use GPLP-4 with seven temperature sensors. (GroPoint Profile User Manual, Page 6)

Data preview

Search:

	datetime	mystery_col	smc_150_mm	smc_300_mm	smc_450_mı
1	10/25/2021 16:51	0	0	0	
2	10/25/2021 17:21	0	0	0	
3	10/25/2021 17:51	0	0	0	
4	10/25/2021 18:21	0	0	0	
5	10/25/2021 18:51	0	0	0	

Showing 1 to 6 of 500 entries

We need to force the date time column into the correct format. We can do this with the following command. (*Please note, this has been done for region:* `United States` *this code may need to be adjusted to fit your regional settings*)

```
data$datetime <- data$datetime %>%
  as_datetime(format = "%m/%d/%Y %H:%M", tz = "CET")
```

7.2.2 Data Re-structuring

For ease of use in R, we will re-structure the data to be in "tidy" format (Read more: (**?**)).

```
# grab the measured variable column headers
mea_var <- colnames(data)[3:15]

# melt (tidy) the data by datetime and site.
data_melt <- data %>% melt(id.vars = c("datetime", "site"),
                           measure.vars = mea_var)

# parse out the variable
data_melt$var <- data_melt$variable %>% str_split("_") %>% map(1) %>%
  unlist()

# parse out the depth of measurement
data_melt$depth <- data_melt$variable %>% str_split("_") %>% map(2) %>%
  unlist()

# remove duplicates
data_melt <- data_melt %>% distinct()
```

Data preview:

Search:

	datetime	site	variable	value	var	depth
1	2021-10-25T14:51:00Z	600409	smc_150_mm	0	smc	150
2	2021-10-25T15:21:00Z	600409	smc_150_mm	0	smc	150
3	2021-10-25T15:51:00Z	600409	smc_150_mm	0	smc	150
4	2021-10-25T16:21:00Z	600409	smc_150_mm	0	smc	150
5	2021-10-25T16:51:00Z	600409	smc_150_mm	0	smc	150
6	2021-10-25T17:21:00Z	600409	smc_150_mm	0	smc	150

Showing 1 to 6 of 500 entries

7.2.3 Summarize to Daily Means

As SWAP is a daily time step model, we do not need the hourly resolution, and can therefore simplify our analysis. We first need to parse out a (daily) date column

```
data_melt$date <- data_melt$datetime %>% as.Date()
```

And then group the data by this date, followed by an averaging function.

```
daily_data <- data_melt %>% group_by(date, var, depth, site) %>%
    summarise(value = round(mean(value, na.rm = TRUE), 1), .groups = "drop_last") %>% ungroup()
```

Additional notes:

- The values are rounded to one decimal place, as this is is the precision limit of the Logger.
- The averaging is performed per date, variable, depth, and site
- The functionality of `.groups = "drop_last"` is unknown to me, however it does not change anything other than quieting a warning message.
- The data is "ungrouped" at the end, to avoid problems with plotting the data (with plotly) later on.

Data preview:

Search:

	date	var	depth	site	value	
1	2021-10-25	smc	150	600409	0	
2	2021-10-25	smc	150	600410	0	
3	2021-10-25	smc	150	600411	0	
4	2021-10-25	smc	150	600413	0	
5	2021-10-25	smc	300	600409	0	
6	2021-10-25	smc	300	600410	0	

Showing 1 to 6 of 500 entries

7.2.4 Including Air Temperature and Precipitation Data

We are going to add the temp / precipitation data from out SWAT+ setup to give us insight on temperature and moisture dynamics for the data quality check.

Also important to note, is that all this code is only necessary, because of the

strange format SWAT+ uses for its weather input. If it would just use a standard format (like everything else) then all this would not be needed.

Reading in the SWAT source precipitation data:

```
swat_pcp <- read.table("model_data/cs10_setup/optain-cs10/sta_id1.pcp", skip = 3, header = F, sep

swat_tmp <- read.table("model_data/cs10_setup/optain-cs10/sta_id1.tmp", skip = 3, header = F, sep
```

Data preview:

Search:

	V1	V2	V3	
1	2010	1	0	
2	2010	2	0	
3	2010	3	0	
4	2010	4	0	
5	2010	5	0	
6	2010	6	0	

Showing 1 to 6 of 500 entries

Parsing out the first and last day and year.

```
FIRST_DOY <- swat_pcp[1,][[2]]-1
FIRST_YEAR <- swat_pcp[1,][[1]]

LAST_DOY <-  swat_pcp[length(swat_pcp$V1),][[2]]-1
LAST_YEAR <- swat_pcp[length(swat_pcp$V1),][[1]]
```

Converting these to an R format. Note that R uses a 0-based index *only for* dates. (very confusing!).

```
first_day <- as.Date(FIRST_DOY, origin = paste0(FIRST_YEAR, "-01-01"))
last_day <- as.Date(LAST_DOY, origin = paste0(LAST_YEAR, "-01-01"))

date_range <- seq(from = first_day, to = last_day, by = "day")
```

Creating a dataframe from the source data

```
pcp_data <- tibble(date = date_range, pcp = swat_pcp$V3)
```

And filtering it to only cover the same range as our logger data.

```
pcp_data <- pcp_data %>% filter(date %in% daily_data$date)
```

And we do the same for temperature

```
FIRST_DOY <- swat_tmp[1,][[2]]-1
FIRST_YEAR <- swat_tmp[1,][[1]]
LAST_DOY <-  swat_tmp[length(swat_tmp$V1),][[2]]-1
LAST_YEAR <- swat_tmp[length(swat_tmp$V1),][[1]]
first_day <- as.Date(FIRST_DOY, origin = paste0(FIRST_YEAR, "-01-01"))
last_day <- as.Date(LAST_DOY, origin = paste0(LAST_YEAR, "-01-01"))
date_range <- seq(from = first_day, to = last_day, by = "day")
tmp_data <- tibble(date = date_range,
                   tmp_max = swat_tmp$V3,
                   tmp_min = swat_tmp$V4)
tmp_data <- tmp_data %>% filter(date %in% daily_data$date)
```

Note: this does (yet) include data from 2023, even though the loggers do.

7.3 Data quality check

In this section we are proofing the quality of the data.

7.3.1 Logger 600409 “Deep Rocky Sand”

```
plot_logger(daily_data, "600409", pcp_data, tmp_data)
```

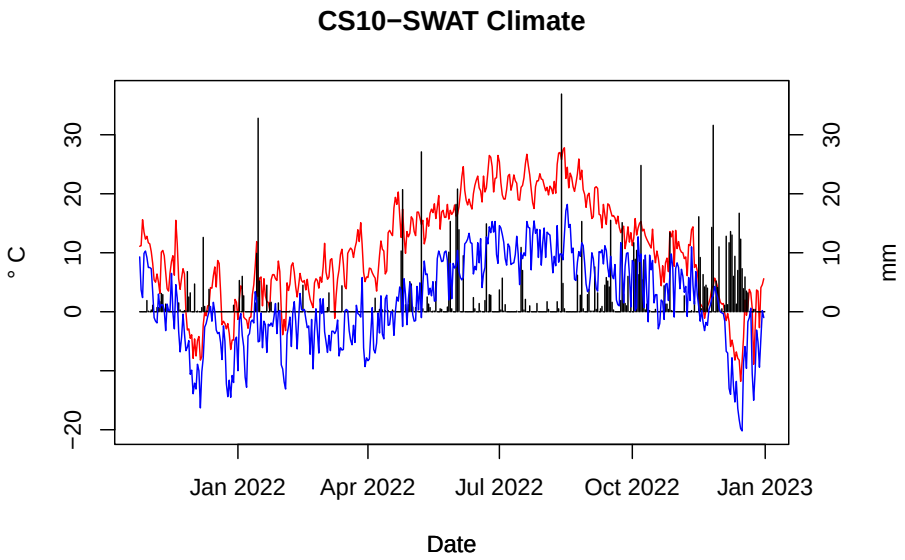


Figure 7.3: Temperature range and precipitation for CS10

7.3.1.1 Soil Temperature

- Remove first few days with unrealistic data

7.3.1.2 Soil Moisture Content

- First few days need to be removed
- 150mm depth seems usable before Sept2022 – However Attila (instrument installer) mentioned this sensor might be in the air (which would explain low moisture levels), so potentially useless
- 350mm depth seems usable, breaks around/after Sept2022
- 450mm depth seems useless, constantly getting wetter for an entire year, spiking post Sept2022
- 600mm depth seems useless, constantly getting wetter for an entire year, spiking post Sept2022
- 750mm depth seems useless, constantly getting wetter for an entire year.
- 900mm depth seems useless, constantly getting wetter for an entire year.

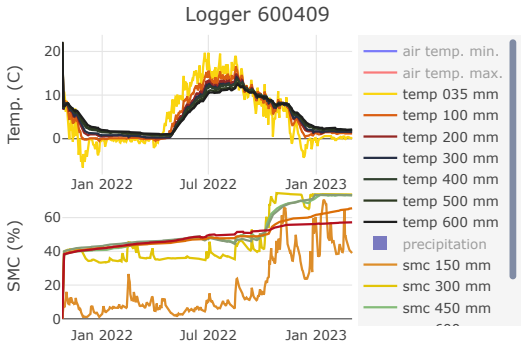


Figure 7.4: Logger 600409

7.3.2 Logger 600410 “Tormods farm”

```
plot_logger(daily_data, "600410", pcp_data, tmp_data)
```

7.3.2.1 Soil Temperature

- Remove first few days with unrealistic data

7.3.2.2 Soil Moisture Content

- First few days need to be removed
- Depth 150 probably useless, as sensor is in free air. (According to Attila)
- Logger seems to have broken Sometime after Sept1 2022, for all depth levels.
- Data pre-Sept22 seems usable (Csilla ?)

7.3.3 Logger 600411 “John Ivar Lower Area”

```
plot_logger(daily_data, "600411", pcp_data, tmp_data)
```

7.3.3.1 Soil Temperature

- Need to remove the first few days
- Need to remove the malfunction in April-May 2022

7.3.3.2 Soil Moisture Content

- First few days need to be removed
- Need to remove the malfunction in April-May 2022
- Depths 150 is out of soil, must be removed.
- Logger seems to have broken August 16th, 2022, data seems useful before that date (Csilla, agree?)

7.3.4 Logger 600413 “John Ivar behind the barn”

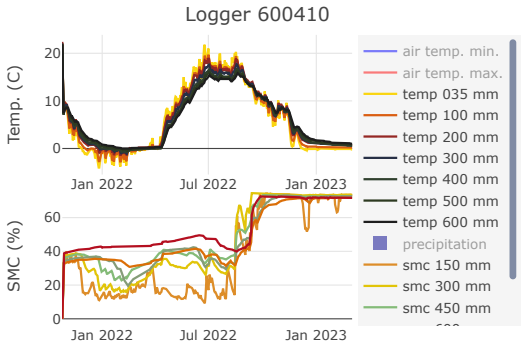


Figure 7.5: Logger 600410

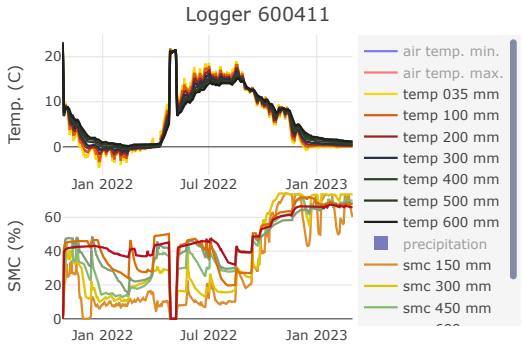


Figure 7.6: Logger 600411

```
plot_logger(daily_data, "600413", pcp_data, tmp_data)
```

7.3.4.1 Soil Temperature

- Remove the first few days with unrealistic data

7.3.4.2 Soil Moisture Content

- Remove first few days
- Remove 150mm, seems out of earth
- Hard to tell when sensor broke, earliest June 24th, latest Sept.12 2022, what do you think Csilla ?

7.4 Data Cleanup

7.4.1 General cleanup

Removing the first few days of the simulation

```
start_date <- "2021-11-01"
data_clean <- daily_data %>% filter(date > start_date)
```

To be on the safe side, I am setting the logger cut-off date to XXXXXXXXX This could be tuned "per logger" later on if one feels confident.

```
cut_off_date <- "2022-08-14"
data_clean <- data_clean %>% filter(date < cut_off_date)
```

And we will do the same for the climate data

```
# TODO dont actually do this.
pcp_data_clean <- pcp_data %>% filter(date > start_date)
pcp_data_clean <- pcp_data_clean %>% filter(date < cut_off_date)

tmp_data_clean <- tmp_data %>% filter(date > start_date)
tmp_data_clean <- tmp_data_clean %>% filter(date < cut_off_date)
```

We will remove the 150 mm depth SMC measurement, as it was not submerged properly in the soil, and is therefore of little use. The temperature reading might be useful, so we will keep it.

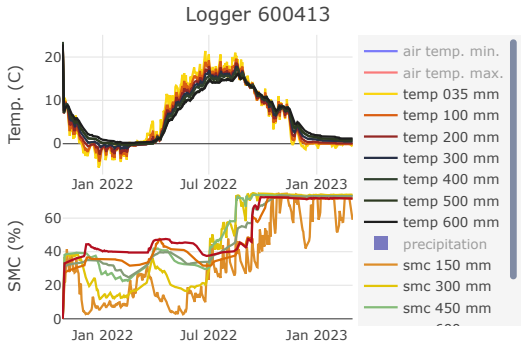


Figure 7.7: Logger 600413

```
data_clean <- data_clean %>% filter(depth != "150")
```

7.4.2 Logger specific cleanup

7.4.2.1 Logger 600409

No logger specific cleanup required

7.4.2.2 Logger 600410

No logger specific cleanup required

7.4.2.3 Logger 600411

Logger 600411 needs temperature and SMC data removed from 2022-04-21 to 2022-05-10. It seems like the logger was removed from the soil.

```
# define time range to remove
remove_dates <- seq(from = as.Date("2022-04-21"),
                    to = as.Date("2022-05-10"),
                    by = "day")

# set values in that site and date range to NA
data_clean <- data_clean %>%
  mutate(value = case_when((site == "600411" &
                              date %in% remove_dates) ~ NA, .default = value))
```

```
plot_logger(data_clean, "600411", pcp_data_clean, tmp_data_clean)
```

7.4.2.4 Logger 600413

No logger specific cleanup required

7.4.3 Saving the cleaned data

We can now save our cleaned up data. We will do so in CSV form

```
write_csv(data_clean, file = "model_data/field_sites/loggers/clean/logger_data_daily_clean.csv")
write_csv(pcp_data_clean, file = "model_data/field_sites/loggers/clean/pcp_data_clean.csv")
```

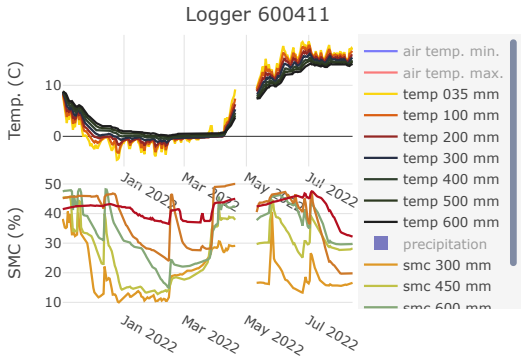


Figure 7.8: Logger 600411, cleaned data

Chapter 8

Field Site Properties

For various use-cases across the modelling project, we will require certain information about the locations of the field sites. This section gathers that information and makes it available to other sections.

8.1 Determining the HSG group of the logger sites

```
require(sf)
require(readr)
require(DT)
require(dplyr)
require(mapview)
require(ggplot2)

sf_theme <- theme(axis.text.x=element_blank(), #remove x axis labels
        axis.ticks.x=element_blank(), #remove x axis ticks
        axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
        )
```

We load the soil map of the area.

```
soil_map <- "model_data/input/soil/soil_map_UTM32N.shp"
soil_map_shp <- read_sf(soil_map)
```

We will crop it by the watershed. Therefore we load the water shed

```
cs10_basin_path <- "model_data/input/shape/cs10_basin.shp"
cs10_basin <- read_sf(cs10_basin_path)
```

And crop by by basin. (current not sure of the effect of constant geometry)

```
st_agr(soil_map_shp) = "constant"
st_agr(cs10_basin) = "constant"

soil_map_shp <- st_intersection(soil_map_shp, cs10_basin)
```

```
ggplot()+geom_sf(soil_map_shp, mapping = aes(fill = SNAM))+sf_theme
```

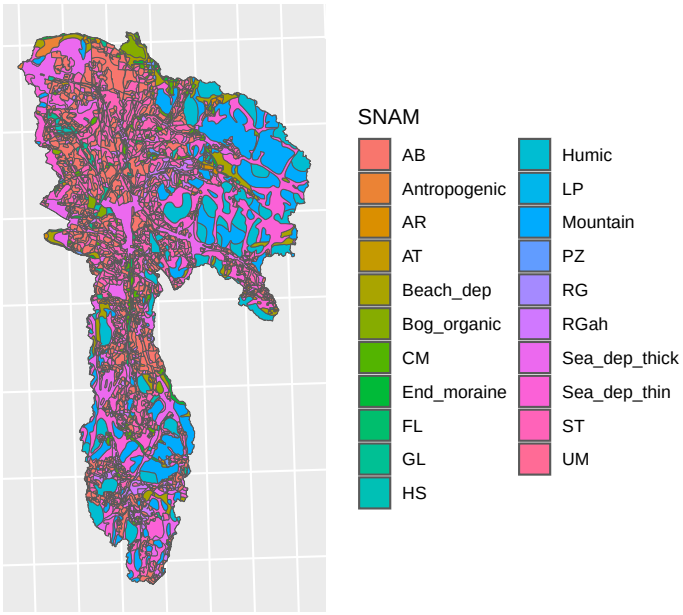


Figure 8.1: Soil map of CS10, in shapefile format

We need information on the soil itself. This is found in our user table. We only need the hydrologic soil group (HSG) `HYDGRP`, but we will keep the soil depth `SOL_ZMX` and soil texture `TEXTURE` just in case

```
usersoil_path <- "model_data/input/soil/UserSoil_Krakstad.csv"
usersoil <- read_csv(usersoil_path, show_col_types = F)
usersoil <- usersoil %>% dplyr::select(OBJECTID, SNAM, SOL_ZMX, HYDGRP, TEXTURE)
```

Search:

	OBJECTID	SNAM	SOL_ZMX	HYDGRP	TEXTURE
1	1	ARdy	1000	C	loamy_sand
2	2	ARdy-tn	1000	C	silt_loam
3	3	Mountain	500	D	sandy loam
4	4	CMdy	1000	C	sandy_loam
5	5	CMdy-ap	1000	C	sandy_loam
6	6	CMdy-sl	1000	C	silt_loam

Showing 1 to 6 of 80 entries

We can combine the two data sets with a left join by soil ID / Object ID

```
usersoil$SOIL_ID <- usersoil$OBJECTID

usersoil <- usersoil %>% dplyr::select(-OBJECTID)

soil_propery_map <- dplyr::left_join(soil_map_shp, usersoil, by = "SOIL_ID")
```

```
ggplot()+geom_sf(soil_propery_map, mapping = aes(fill = HYDGRP))+sf_theme
```

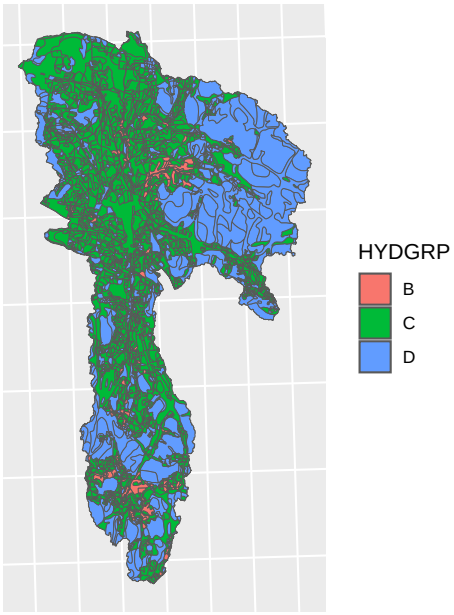


Figure 8.2: HSG of CS10

Now we need the locations of the field sites. These are stored as points in the following file:

```
cs10_field_sites_path <- "model_data/field_sites/geospatial/cs10_field_sites.shp"
cs10_field_sites <- read_sf(cs10_field_sites_path)
```

We are going to join the attributes using a spatial join

```
field_sites_attr <- st_join(cs10_field_sites, soil_propery_map)
```

Here is the result:

We have two sites covering C and D, and one site covering B. This covers all existing HSGs in the catchment, which is good. We can save this information in a dataframe.

```
field_site_attr_df <- st_drop_geometry(field_sites_attr)

datatable(field_site_attr_df)
```

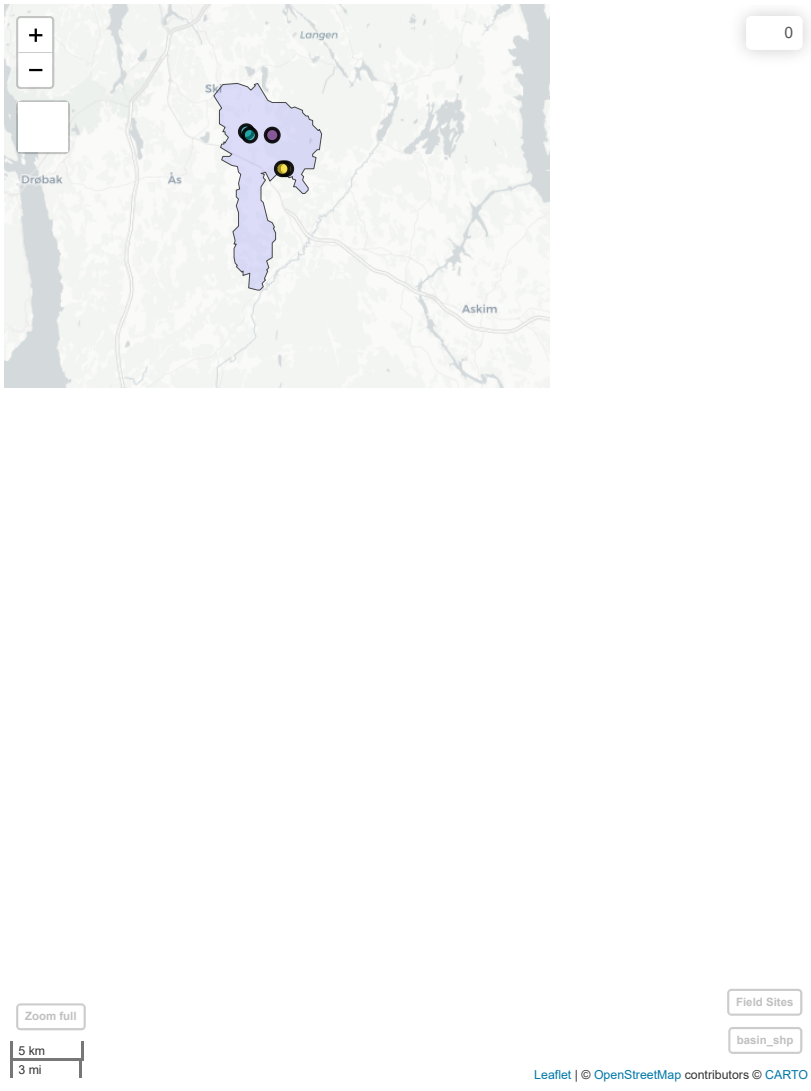


Figure 8.3: Field site locations of CS10

Show 10 entries Search:

	id	SOIL_ID	SNAM.x	Id	SNAM.y	SOL_ZMX	HYDGRP	TEXTURE
1	600413	23	ST	0	Sea_dep_thick	800	C	sandy loam
2	600411	1	AB	0	ARdy	1000	C	loamy_sand
3	600410	20	RGah	0	GLum-ar	1000	B	sandy_loam
4	600409	13	Humic	0	GLch-hu-sl	1000	D	silty_clay_loam
5	600408	13	Humic	0	GLch-hu-sl	1000	D	silty_clay_loam

Showing 1 to 5 of 5 entries Previous 1 Next

We will write this into our output folder

```
write_csv(x = field_site_attr_df, file = "model_data/field_sites/output/field_site_attr
```

And save our soil property map, and field site points (with attributes)

```
# TODO fix the weird ID column mess you made
write_sf(field_sites_attr, "model_data/field_sites/output/field_site_attr_map.shp")
```

```
## Warning in CPL_write_ogr(obj, dsn, layer, driver,
## as.character(dataset_options), : GDAL Message 6: Normalized/laundered field
## name: 'Id' to 'Id_1'
```

```
write_sf(soil_propery_map, "model_data/field_sites/output/soil_propery_map.shp")
```

Chapter 9

Antecedent Precipitation Index

9.1 Introduction

SWATFarmR cannot account for operations based on values modeled by SWAT+, such as soil moisture. This is because FarmR does not run the model itself, and creates and pre-defines all the management schedules based on measured climate data. This means we need to relate our measured climate data to our soil moisture (Because soil moisture is an important factor in planning agricultural operations). One method to do this is called the Antecedent Precipitation Index. Which is what we will creating for CS10 in this part.

As a heads up, this section requires Python!

9.1.1 Pre-requirements

Required R packages

```
require(readr)
require(dplyr)
require(hydroGOF)
library(reticulate) # Required Python packages: "pandas", "numpy", "scipy"

source("model_data/code/plot_api.R")
source("model_data/code/calc_api.R")
```

9.1.2 Setting up Python for R/Rmarkdown

A quick (non-evaluated) code block to show how a python environment can be created for use in Rmarkdown/Reticulate on your local machine.

```
# install miniconda
install_miniconda()

# create conda environment for your project
conda_create("cs10_env", # environment name
             packages = c("pandas", "numpy", "scipy"), # required packages
             conda = "auto")


conda_python("cs10_env") # returns the .exe python path for your environment

# you need to change your R environment path in this file
usethis::edit_r_environ()

# RETICULATE_PYTHON="YOUR ENVIRONMENT PATH/python.exe"

# restart your R session.

py_config()

# If everything is right, you should get something like:

# python: YOUR ENVIRONMENT PATH/python.exe libpython: YOUR ENVIRONMENT
# PATH/python39.dll pythonhome: YOUR ENVIRONMENT PATH version: 3.9.7 (default,
# Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] Architecture: 64bit numpy:
# YOUR ENVIRONMENT PATH\lib\site-packages\numpy numpy_version: 1.21.2

# After this it should work :)
```

9.2 Antecedent Precipitation Index

From (**?**)

The API is a well-known, parsimonious, recursive model for predicting soil moisture solely based on precipitation records. The API is commonly implemented using daily precipitation records, but it is possible to work at finer temporal scales (e.g. hourly) if both precipitation (model input) and soil moisture (for validation purposes) are available. The equation describing the simple version of the model is:

$$API_t = \alpha API_{t-1} + P_t$$

API_t: Soil water content at time t (today)

API_{t-1}: Soil water content at time $t-1$ (yesterday)

α: Loss coefficient. Range between 0 and 1

P_t: Precipitation at time t (today)

9.2.1 Loading data

We will load out data from the previous Section **??**

```
logger_data <- read_csv("model_data/field_sites/loggers/clean/logger_data_daily_clean.csv",
                        show_col_types=F)

# we only need SMC for this part
logger_data = logger_data %>% filter(var == "smc")

site_attributes <-
  read_csv("model_data/field_sites/output/field_site_attr_df.csv",
           show_col_types = F)

pcp_data <- read_csv("model_data/field_sites/loggers/clean/pcp_data_clean.csv",
                     show_col_types = F)

head(site_attributes)
```

```
## # A tibble: 5 x 8
##       id SOIL_ID SNAM.x    Id SNAM.y        SOL_ZMX HYDGRP TEXTURE
##    <dbl>   <dbl> <chr>  <dbl> <chr>           <dbl> <chr>  <chr>
## 1 600413      23 ST         0 Sea_dep_thick     800 C      sandy loam
## 2 600411       1 AB         0 ARdy             1000 C      loamy_sand
## 3 600410      20 RGah       0 GLum-ar          1000 B      sandy_loam
## 4 600409      13 Humic      0 GLch-hu-sl       1000 D      silty_clay_loam
## 5 600408      13 Humic      0 GLch-hu-sl       1000 D      silty_clay_loam
```

We will calculate the model per Hydrologic soil group (HSG), and start with B, which was measured by logger 600410.

```
b_data <- logger_data %>% filter(site == "600410")
```

9.2.2 Calculating the total water content

We need to calculate the total water content of the profile from the depth measurements from the following depths:

```
unique(b_data$depth)
```

```
## [1] "300" "450" "600" "750" "900"
```

Following code is modified from **?** (read more)

(This calculation could be improved by considering the soil horizons)

```
a <- b_data %>% filter(depth == "300") %>% select(value)
b <- b_data %>% filter(depth == "450") %>% select(value)
c <- b_data %>% filter(depth == "600") %>% select(value)
d <- b_data %>% filter(depth == "750") %>% select(value)
e <- b_data %>% filter(depth == "900") %>% select(value)

total_water_content <-
  a * .35 + # from 0 to 35
  b * .15 + # from 35 to 50
  c * .15 + # from 55 to 65
  d * .15 + # from 65 to 80
  e * .20   # from 80 to 100

b_col <- tibble(b_data %>% filter(depth == "300") %>% select(date), total_water_content
```

And we can add the precipitation data from **??**

```
b_col <- left_join(b_col, pcp_data, by = "date")

plot_api(obs = b_col, title = "HSG B")
```

Saving the data

```
write_csv(b_col, "model_data/field_sites/output/twc.csv")
```

9.2.3 Defining API model

The value is capped at the maximum value of the source data (50). This could be improved by capping site specific.

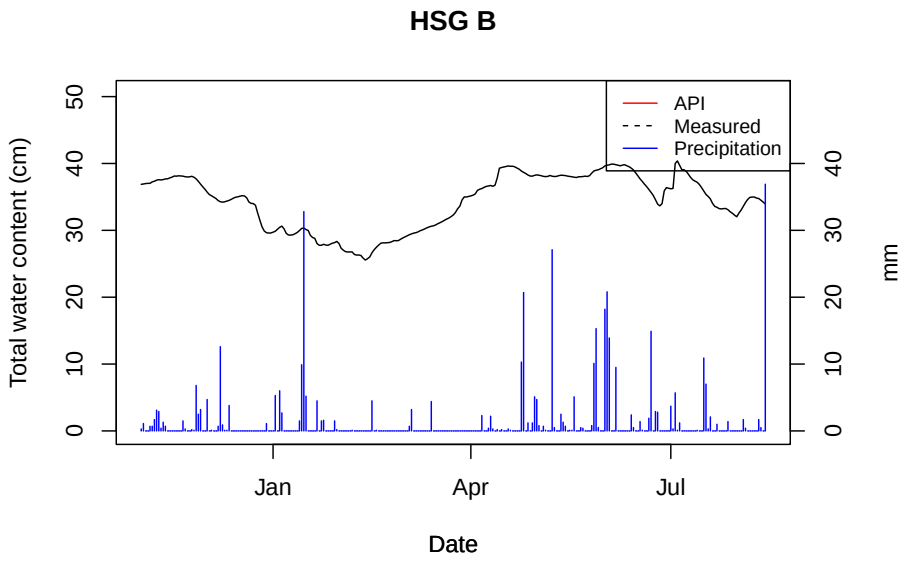


Figure 9.1: Total water content for logger 600410 (HSG B)

```
def api_model(P,alpha=0.97,ini=36.8):
    api = [ini]
    for t in range(1,len(P)):
      append_val = api[t-1]*alpha + P[t]
      if append_val > 50:
        append_val = 50
      api.append(append_val)

    return api
```

Load in our data:

```
import pandas as pd
df = pd.read_csv('model_data/field_sites/output/twc.csv')
df = df[['date','value','pcp']]
df.head()
```

```
##          date   value  pcp
## 0  2021-11-02  36.880  0.3
## 1  2021-11-03  36.930  1.1
## 2  2021-11-04  36.995  0.0
```

```
## 3  2021-11-05  37.025  0.0
## 4  2021-11-06  37.030  0.7
```

As a test, we will calculate the predictions with an alpha of 0.95:

```
import pandas as pd
from scipy.optimize import curve_fit

# guess
storage_guessed = api_model(df['pcp'], alpha=0.95, ini=36.8)

# write to text
dates = df[['date']]
guess = pd.DataFrame(dates.values.tolist(),storage_guessed)
guess.to_csv("model_data/field_sites/output/first_guess.csv")
```

Now we can have a look at it in R

```
guess <- read_csv("model_data/field_sites/output/first_guess.csv",
                  show_col_types = F,
                  skip = 1, col_names = c("val", "date"))

plot_api(api_vals = guess, obs = b_col, title = "HSG-B API guessed")
```

Now we optimize the alpha parameter using `scipy curve_fit`.

```
par_opt, par_cov = curve_fit(api_model, df['pcp'], df['value'], p0=[0.95,100])

print('Annual mean alpha value is',round(par_opt[0],2))
```

```
## Annual mean alpha value is 0.98
```

Now we can calculate the API using the optimized parameter.

```
import numpy as np

storage_optimized = api_model(df['pcp'],par_opt[0],par_opt[1])

# Mean Absolute Error
MAE = np.mean(np.abs(df['value'] - storage_optimized))
print('Mean annual error =', round(MAE,1),'cm')
```

```
## Mean annual error = 9.1 cm
```

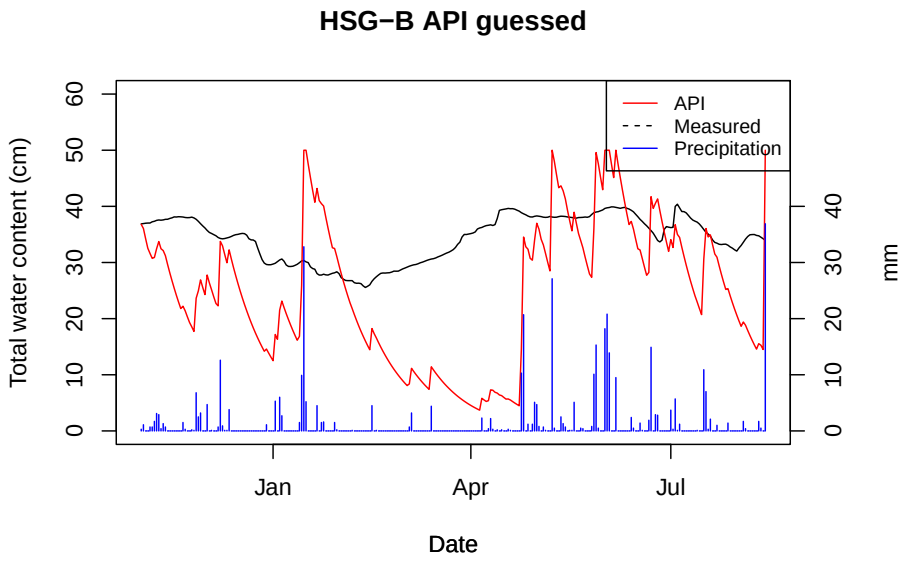


Figure 9.2: API for guessed alpha of HSG B

And save the data for R

```
dates = df[['date']]
api_optimized = pd.DataFrame(dates.values.tolist(),storage_optimized)
api_optimized.to_csv("model_data/field_sites/output/api_optimized.csv")
```

And have a look in R:

```
optimized <- read_csv("model_data/field_sites/output/api_optimized.csv",
                      show_col_types = F,
                      skip = 1, col_names = c("val", "date"))

plot_api(api_vals = optimized, obs = b_col, title = "HSG-B API optimized")
```

Not great, but also not surprising considering no account for snow melt has been done, nor do we have good logger data.

We will define this workflow in a function `calc_api.R` and run it for the next loggers.

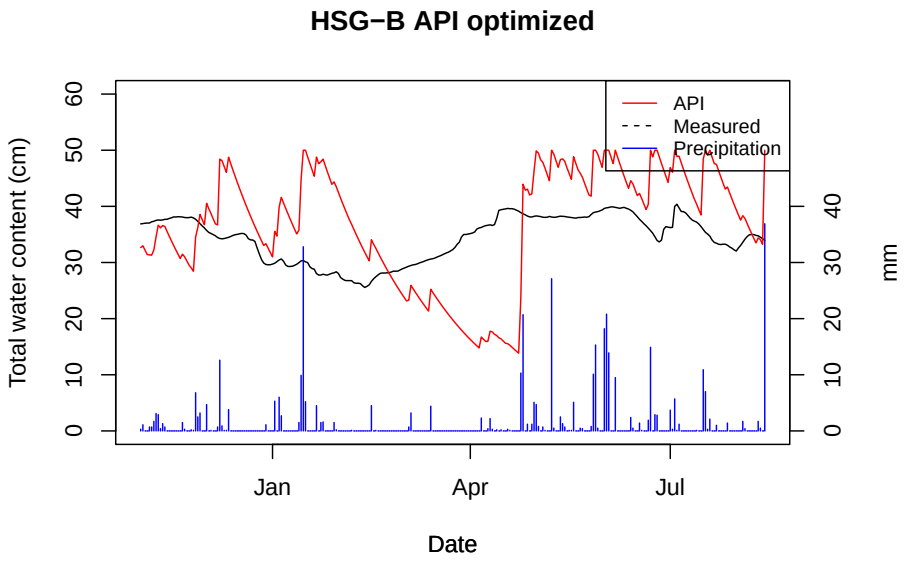


Figure 9.3: API with an optimized alpha for HSG B

```
# TODO fix warning
calc_api(logger_id = "600409")
```

```
## Warning in py_to_r.pandas.core.frame.DataFrame(x): index contains duplicated
## values: row names not set
```

```
##   HSG Alpha
##   "D"   "1"
```

This is the logger with broken data, so no surprise that the optimizer failed.

```
calc_api(logger_id = "600411")
```

```
## Warning in py_to_r.pandas.core.frame.DataFrame(x): index contains duplicated
## values: row names not set
```

```
##    HSG  Alpha
##    "C" "0.99"
```

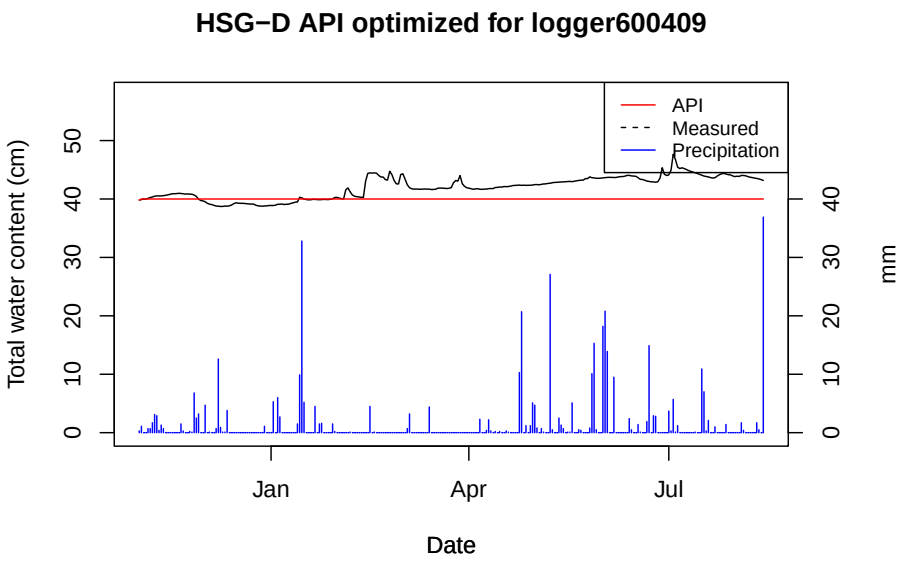


Figure 9.4: Optimized API for logger 600409

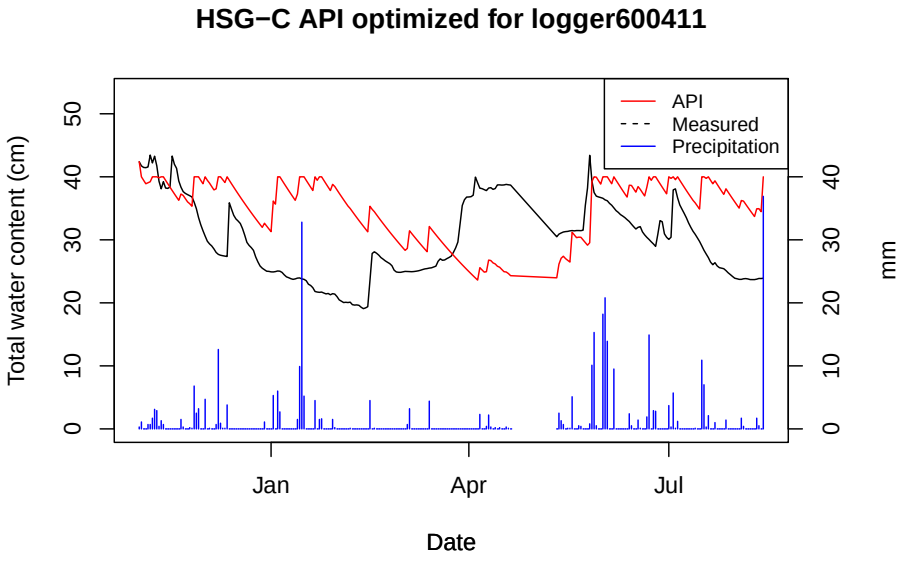


Figure 9.5: Optimized API for logger 600411

```
calc_api(logger_id = "600413")
```

```
## Warning in py_to_r.pandas.core.frame.DataFrame(x): index contains duplicated
## values: row names not set
```

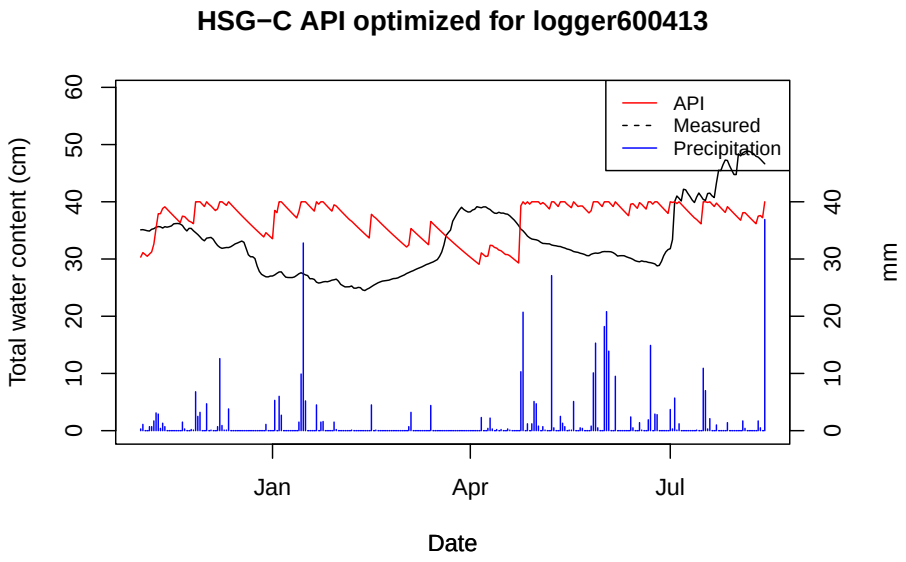


Figure 9.6: Optimized API for logger 600413

```
##    HSG  Alpha
##    "C" "0.99"
```

9.3 Conclusions

For HSG B we got an alpha of **0.98**, for C we have two samples, both of which got **0.99**, and D has two samples as well, but we currently only have data for one. For the one D site we had, the logger data was very poor and the **covariance could not be estimated**.

Overall, the API does not seem to be a great estimator of soil moisture **for our specific case**. Our logger data is poor and has been cut short due to sensor malfunctions. With a **longer time series** covering more seasons, it is likely that the curve fitting could be improved upon, and the overall dynamics could be appraised.;

Secondly, and more critically: the API does not account for **snow melt**, which plays a significant role in a Boreal catchments such as CS10. You can see this in action (I think) in March where the logger data steadily goes up, while the API steadily goes down. Snow melt could surely be accounted for by adding *another* model to estimate it – the question is more if it is worth it, considering the poor underlying data of the loggers.

Another complication is, if SWATFarmR does not account for snow melt in its management scheduling, then there is no point in estimating it. And if snow melt is not incorporated into the scheduling, then the operation timing will not be very accurate anyway (since snow melt is such a dominating factor in operation timing).

UPDATE post Q+A Session

We can modify the output of Micha's script in Section **??** by replacing API with our own custom variable. This variable could be snow melt or something else entirely (like a trafficability thing?) This is pretty straight forward, we just need to add the index to the FarmR using `add_variable()`. One thing we need to keep in mind is that this needs to be compatible for the climate scenarios (i.e. we can only base it off climate variables which we also have future data on.)

Some things to consider:

- Run SWAT+ for the weather data + scenarios data and use the calculated snow melt
- Re-create the snow melt model of SWAT+ and use that
- Use HBV or Persist models
- Use some tractability index for Norway (Attila?)

Chapter 10

Model Parameterization

These sections are not detailed enough to warrant their own dedicated chapter.

10.1 Parameter Changes

These sections mostly involve changing a few parameter values and are therefore in small sub sections without R-code. Some of this might change later.

10.1.1 Channel parameters revised

This an ongoing unresolved issue #49.

So far nothing has been changed.

10.1.2 Crop parameters verified

The crop database has been updated and stored in the following file:

```
"model_data/cs10_setup/swat_input_mod/plants.plt"
```

Discussions to this topic can be found in issue #27.

From the whole database, we only use crops defined in our management files:

Crop management: oats, barl, wwht, swht, pota, fesc

Generic: frst, fesc and others which are minor

The parameters for these crops have been updated to reflect the colder growing conditions.

10.1.3 Soil physical parameters in final form

This has been mentioned in issue #28, and has been closed without discussion. Seems like the parameters are in final form, but no further documentation exists at this point.

10.1.4 Soil chemical parameters in final form

This step will be verified in issue #46. It has been discussed in #19.

The following changes have been implemented, but are subject to change:

Table 10.1: Changes made to `soilnut1` of `nutrients.sol`

Parameter	Default	CS10
`lab_p`	5	20
`nitrate`	7	8.5
`inorgp`	3.5	35.1
`watersol_p`	0.15	0.4

```
nutrients_sol <- readLines("model_data/cs10_setup/swat_input/nutrients.sol")

header <- nutrients_sol[1]

nutrients_sol_df <- read.table("model_data/cs10_setup/swat_input/nutrients.sol", header

nutrients_sol_df$lab_p <- 20
nutrients_sol_df$nitrate <- 8.5
nutrients_sol_df$inorgp <- 35.1
nutrients_sol_df$watersol_p <- 0.4

write.table(
  nutrients_sol_df,
  "model_data/cs10_setup/swat_input_mod/nutrients.sol",
  quote = F,
  sep = "   ",
  row.names = F,
)

header_new <- paste("modified by the CS10 workflow in section 9.1")

nutrients_sol <- readLines("model_data/cs10_setup/swat_input_mod/nutrients.sol")

writeLines(c(header_new, nutrients_sol), "model_data/cs10_setup/swat_input_mod/nutrient
```

Testing if the setup still works:

```
simout <- test_swat_mod()
cat(tail(simout), sep = "\n")
```

10.1.5 Impoundment parameters defined

Has been postponed by #20, will be dealt with in #54

10.1.6 Water diversions defined

Deemed as not relevant for this catchment

10.1.7 Point sources parameters added

Is an ongoing issue in #21

10.1.8 Tile drainage parameters defined

This has been discussed in Issue #7 and is an ongoing issue in #53.

```
old_path <- "model_data/cs10_setup/swat_input/tiledrain.str"
new_path <-  "model_data/cs10_setup/swat_input_mod/tiledrain.str"

tiledrain_str <- readLines(old_path)

header <- paste("modified by CS10 workflow in section 9.1")

tiledrain_df <-
  read.table(
    old_path,
    sep = "",
    header = T,
    skip = 1,
    fill = T,
    colClasses = "character"
  )

tiledrain_df$dp <- 800 # from 1000
tiledrain_df$t_fc <- 12 # from 24
tiledrain_df$lag <- 30 # from 96
tiledrain_df$rad <- 200 # from 100
```

```
tiledrain_df$dist <- 8000 # from 30
tiledrain_df$drain <- 40 # from 10
tiledrain_df$pump <- 0 # from 1
tiledrain_df$lat_ksat <- 2 # from 2 (no change)

write.table(header, file = new_path, quote = F, col.names = F, row.names = F)
write.table(
  x = tiledrain_df,
  file = new_path,
  sep = "    ",
  quote = F,
  append = T,
  row.names = F
)
```

```
## Warning in write.table(x = tiledrain_df, file = new_path, sep = " ", quote = F,
## : appending column names to file
```

Testing if the setup still works:

```
simout <- test_swat_mod()
cat(tail(simout), sep = "\n")
```

10.1.9 Atmospheric deposition defined

Has been done in Section **??**

10.1.10 Additional settings verified

Refers to chapter 3.11 in The Protocol and files `parameters.bsn` `codes.bsn`
Has been discussed in #22

The following changes have been made:

Table 10.2: Changes made to `codes.bsn`

Parameter	Default	CS10
`pet`	1	2
`rte_cha`	0	1
`cn`	0	1
`tiledrain`	0	1
`soil_p`	0	1

Parameter	Default	CS10
atmo_dep	a	m

```
old_path <- "model_data/cs10_setup/swat_input/codes.bsn"
new_path <- "model_data/cs10_setup/swat_input_mod/codes.bsn"

codes_bsn <- readLines(old_path)

header <- codes_bsn[1]

header <- paste("modified by CS10 workflow in section 9.1")

codes_df <- read.table(old_path, skip = 1, header = T, sep = "",
                       colClasses = "character")
```

PET method (pet) The OPTAIN project recommends the PET calculation of Pennman-Monteith, which is code 2.

```
codes_df$pet <- 2
```

Channel routing network (rte_cha) Recommended is to start with Muskingum (code 1) and apply variable storage method if it causes problems.

```
codes_df$rte_cha <- 1
```

Stream water quality (wq_cha) Recommended to test both for OPTAIN (1/0)

Daily curve number calculation (cn) Code 1 is recommended. (Plant ET)

```
codes_df$cn <- 1
```

OPTAIN recommends new version

```
codes_df$soil_p <- 1
```

Lapse rate is being tested in **??**

Plant growth stress is being testing in **??**

Tile drainage

This should be set to 1, but that causes a crash. We will fix this bug in a dedicated section

```
codes_df$tiledrain <- 0
```

Atmospheric Deposition

`atmodep` was done in Section **??** and is annual

```
codes_df$atmo_dep <- "y"
```

Writing the changes

```
write.table(header, file = new_path, col.names = F, row.names = F, quote = F)


write.table(
  codes_df,
  file = new_path,
  col.names = T,
  append = T,
  quote = F,
  sep = "   ",
  row.names = F
)
```

```
## Warning in write.table(codes_df, file = new_path, col.names = T, append = T, :
## appending column names to file
```

Testing if the setup still works:

```
simout <- test_swat_mod()
cat(tail(simout), sep = "\n")
```

10.2 Fixing tiledrain

```
run_this_chapter = FALSE
```

Problem: `tiledrain=1` in `codes.bsn` causes crash.

Reason: Some HRUs have drains, but are on a shallow soil type. This causes the drains to be below the soil, leading to a (nondescript) error.

Likely source of error: Misalignment of the soil and land use map.

Discussion of this issue can be found in #53

```
require(mapview)
require(sf)
require(stringr)
require(dplyr)
require(purrr)
require(DT)
```

10.2.1 Finding the problem HRUs

Our drains are set to 80cm depth. Which of our soils are less than that?

```
usersoil <- readr::read_csv("model_data/input/soil/UserSoil_Krakstad.csv", show_col_types = F)

shallow_soils <- usersoil$SNAM[which(usersoil$SOL_ZMX<800)]

print(shallow_soils)
```

```
##  [1] "Mountain"     "CMlen-dy"     "Antropogenic" "Sea_dep_thin" "Humic"
##  [6] "LVlen-sl"     "End_moraine"  "RGlen-dy-ar"  "RGlep-dy"     "STlen-dy"
## [11] "STlen-lv-sl"  "STlen-rt-sl"  "STlen-um-sl"
```

Which of our HRUs use these soils, and are drained?

```
hru_data <- read.table( "model_data/cs10_setup/swat_input/hru-data.hru",
                        skip = 1, header = T, sep = "")

shallow_hrus <- which(hru_data$soil %in% shallow_soils)

drained_hrus <- grepl(x = hru_data$lu_mgt, pattern = "_drn") %>% which()

# Intersection of HRUs which are both shallow and drained
problem_hrus <- base::intersect(shallow_hrus, drained_hrus)

length(problem_hrus)
```

```
## [1] 7
```

Just 16 HRUs have an issue. Lets get some info on these 16:

```
problem_hru_data <- hru_data[problem_hrus,]

datatable(problem_hru_data)
```

Show 10 entries Search:

	id	name	topo	hydro	soil	lu_mgt	soil_plant_init	surf_stor	snow	field
704	704	hru0704	topohru0704	hyd0704	Sea_dep_thin	a_037f_2_drn_lum	null	null	snow001	null
2784	2784	hru2784	topohru2784	hyd2784	RGlen-dy-ar	a_072f_1_drn_lum	null	null	snow001	null
3314	3314	hru3314	topohru3314	hyd3314	STlen-rt-sl	a_060f_2_drn_lum	null	null	snow001	null
3367	3367	hru3367	topohru3367	hyd3367	STlen-lv-sl	a_072f_1_drn_lum	null	null	snow001	null
3376	3376	hru3376	topohru3376	hyd3376	STlen-rt-sl	a_060f_2_drn_lum	null	null	snow001	null
3686	3686	hru3686	topohru3686	hyd3686	STlen-rt-sl	a_115f_1_drn_lum	null	null	snow001	null
5861	5861	hru5861	topohru5861	hyd5861	End_moraine	a_210f_2_drn_lum	null	null	snow001	null

Showing 1 to 7 of 7 entries Previous 1 Next

We need to link the soil type to the hru vector file

```
hru_shp <- read_sf("model_data/cs10_setup/optain-cs10/data/vector/hru.shp")
hru_soil <- hru_data %>% select(soil, name)
hru_shp <- left_join(hru_shp, hru_soil, by = "name")
```

And separate out our problematic HRUs and affected LUs

```
problem_shps <- hru_shp %>% filter(name %in% problem_hru_data$name)
problem_hru_ids <- problem_hru_data$lu_mgt %>% str_remove("_drn_lum")
problem_lus <- hru_shp %>% filter(type %in% problem_hru_ids)
```

Next we remove the problem HRUs and affect LUs from our hru_shp (this is for the upocoming map, to remove overlaps).

```
hru_shp <- hru_shp %>% filter((hru_shp$name %in% problem_lus$name) == FALSE)
hru_shp <- hru_shp %>% filter((hru_shp$name %in% problem_shps$name) == FALSE)
```

We also remove the problem HRUs from the affected LUs

```
problem_lus <- problem_lus %>% filter((problem_lus$name %in% problem_shps$name) == FALSE)
```

Data processing done, lets have a look:

```
hru_map <-mapview(
  hru_shp,
  map.types = "Esri.WorldImagery",
  color = "green",
  lwd = 1,
  legend = FALSE,
  zcol = "soil",
  alpha.region = 0
)

problem_map <-
  mapview(
    problem_shps,
    color = "red",
    lwd = 3,
    legend = FALSE,
    zcol = "name",
    layer.name = "Problem HRUs",
    alpha.region = 0
  )

problem_lus_map <-
  mapview(
    problem_lus,
    color = "orange",
    lwd = 1,
    legend = FALSE,
    zcol = "name",
```

```
    layer.name = "Problem LUs",
    alpha.region = 0
  )

full_map <- hru_map + problem_lus_map+ problem_map

full_map
```

Resolution to this problem has been discussed in issue #53

We will extract the hru and landuse file to modify further.

```
lum_fixed <- read.table("model_data/cs10_setup/swat_input_mod/landuse.lum", skip = 1, s

hru_fixed <- read.table("model_data/cs10_setup/swat_input/hru-data.hru", skip = 1, sep
```

10.2.2 Fixing the problem HRUs

`hru0218` with landuse `a_008f_2_drn_lum` – Remove the drains since it is an isolated LU

```
lum_fixed$tile[which(lum_fixed$name=="a_008f_2_drn_lum")] = "null"
```

`hru3637` with landuse `a_148f_1_drn_lum` – Remove the drains since it is an isolated LU.

```
lum_fixed$tile[which(lum_fixed$name=="a_148f_1_drn_lum")] = "null"
```

`hru3686` of landuse `a_115f_1_drn_lum` has the wrong soil type, changing it to `"STlv-sl"` of its connected field.

```
hru_fixed$soil[which(hru_fixed$name=="hru3686")] = "STlv-sl"
```

`hru4880` of landuse `a_182f_5_drn_lum` has the wrong soil type for the land use. Changing to the "`ARdy`" of neighboring fields

```
hru_fixed$soil[which(hru_fixed$name=="hru4880")] = "ARdy"
```

`hru5033` with isolated landuse `a_112f_1_drn_lum` can have its drains removed.

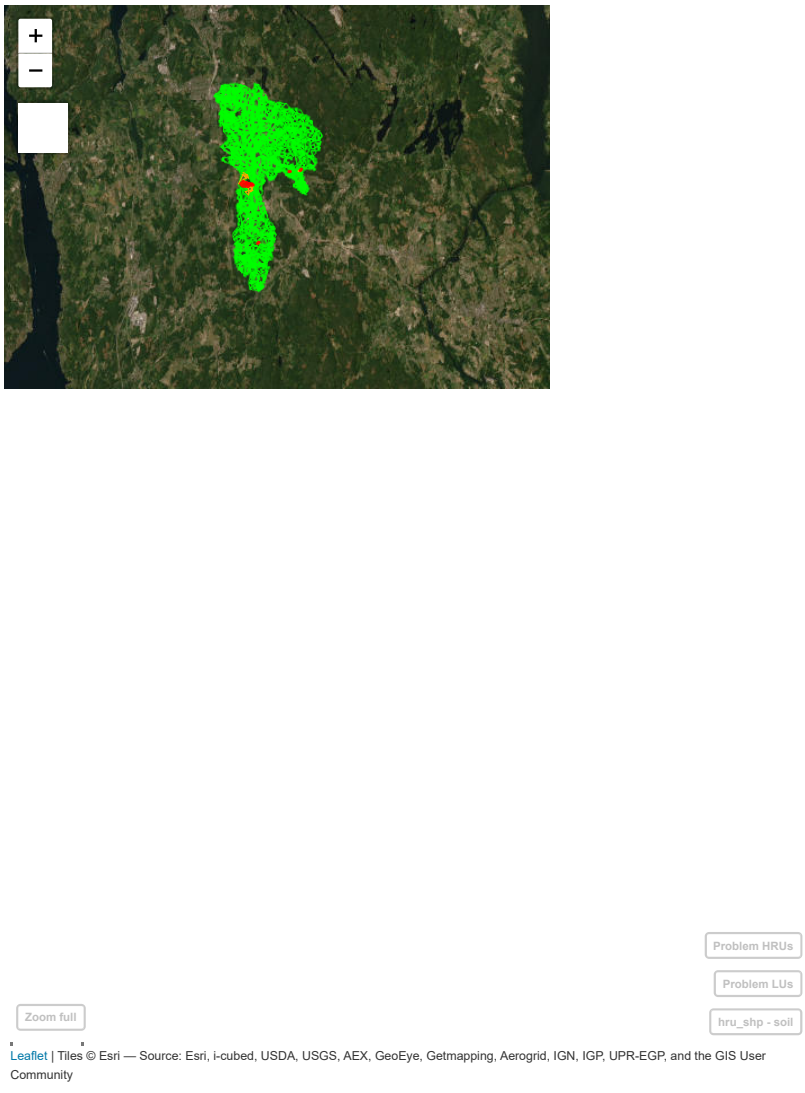


Figure 10.1: Map displaying problematic HRUs (red), and HRUs sharing landuses with problematic HRUs (orange). Green HRUs has no issues.

```
lum_fixed$tile[which(lum_fixed$name=="a_112f_1_drn_lum")] = "null"
```

hru0692 of isolated land use a_037f_1_drn_lum can safely has its drains removed

```
lum_fixed$tile[which(lum_fixed$name=="a_037f_1_drn_lum")] = "null"
```

hru0704 has the wrong soil type for its landuse a_037f_2_drn_lum, which should be STlv-sl instead of Sea_dep_thin

```
hru_fixed$soil[which(hru_fixed$name=="hru0704")] = "STlv-sl"
```

hru3314 of landuse a_060f_2_drn_lum has the wrong soil type. It should be STlv-sl, not STlen-rt-sl.

The same is true for hru3376.

```
hru_fixed$soil[which(hru_fixed$name=="hru3314")] = "STlv-sl"
hru_fixed$soil[which(hru_fixed$name=="hru3376")] = "STlv-sl"
```

hru2784 (and the within hru3367) is a sizable field with landuse a_072f_1_drn_lum and soil RGlen-dy-ar. Tough call here, but best bet is probably to assign soil of the other major field it shares a landuse with (STlv-sl)

```
hru_fixed$soil[which(hru_fixed$name=="hru2784")] = "STlv-sl"
hru_fixed$soil[which(hru_fixed$name=="hru3367")] = "STlv-sl"
```

hru3465 of LU a_146f_2_drn_lum has the wrong soil for the greater field. Changing it from RGlep-dy to STrt-sl

```
hru_fixed$soil[which(hru_fixed$name=="hru3465")] = "STrt-sl"
```

hru5214 of isolated land use a_020f_4_drn_lum can safely have its drains removed.

```
lum_fixed$tile[which(lum_fixed$name=="a_020f_4_drn_lum")] = "null"
```

hru5861 of landuse a_210_f2_drn_lum does not share the same soil (end moraine) as the rest of its field.

```
hru_fixed$soil[which(hru_fixed$name=="hru5861")] = "STlv-sl"
```

hru5926 of landuse a_211f_1_drn_lum has the wrong soil type (Sea-dep-thin), changing to that of the greater field (STrt-sl)

```
hru_fixed$soil[which(hru_fixed$name=="hru5926")] = "STlv-sl"
```

hru2633 of isolated LU a_131f_1_drn_lum can safely have its drains removed.

```
lum_fixed$tile[which(lum_fixed$name=="a_131f_1_drn_lum")] = "null"
```

10.2.3 Writing modified changes

Removing the appended columns, and duplicates from the left-join, adding header and writing to modified input files directory.

```
lu_header <- paste("lu table after fixing up by OPTAIN-CS10 workflow in section 9.2")

write.table(lu_header, "model_data/cs10_setup/swat_input_mod/landuse.lum", quote = F, row.names =

write.table(lum_fixed, file = "model_data/cs10_setup/swat_input_mod/landuse.lum", sep = "\t", quo
```

```
## Warning in write.table(lum_fixed, file =
## "model_data/cs10_setup/swat_input_mod/landuse.lum", : appending column names to
## file
```

Six drains have been removed.

```
hru_header <- paste("header header header, delete me and regret it forever! fixed soil types by c

write.table(hru_header, "model_data/cs10_setup/swat_input_mod/hru-data.hru", quote = F, row.names

write.table(hru_fixed, file = "model_data/cs10_setup/swat_input_mod/hru-data.hru", sep = "\t", qu
```

```
## Warning in write.table(hru_fixed, file =
## "model_data/cs10_setup/swat_input_mod/hru-data.hru", : appending column names
## to file
```

Ten soil types have been changed.

10.2.4 Re-enabling tiledrain

Load from modified input files

```
old_path <- "model_data/cs10_setup/swat_input_mod/codes.bsn"
new_path <- "model_data/cs10_setup/swat_input_mod/codes.bsn"

codes_bsn <- readLines(old_path)

header <- codes_bsn[1]

header <- paste("modified by CS10 workflow in section 9.2")

codes_df <- read.table(old_path, skip = 1, header = T, sep = "",
                       colClasses = "character")
```

Re-enable by setting to 1

```
codes_df$tiledrain <- 1
```

Writing to modified input files directory.

```
write.table(header, file = new_path, col.names = F, row.names = F, quote = F)

write.table(
  codes_df,
  file = new_path,
  col.names = T,
  append = T,
  quote = F,
  sep = "   ",
  row.names = F
)
```

```
## Warning in write.table(codes_df, file = new_path, col.names = T, append = T, :
## appending column names to file
```

10.2.5 Testing SWAT+

Now we can see if the model runs

```
source("model_data/code/test_swat_mod.R")

simout <- test_swat_mod()
cat(tail(simout), sep = "\n")
```

Success.

Chapter 11

SWATFarmR Management Schedules

WIP: Every field in the OPTAIN SWAT setup needs its own management. We will use the SWATFarmR package to define it.

```
run_this_chapter = FALSE
```

Pre-requirements

The creation of management schedules with `SWATfarmR` requires a SWAT+ model setup created by the `SWATbuildR` package.

We also require the following package and functions:

```
require(DT)
require(ggplot2)
require(readr)
require(stringr)
require(magrittr)
#remotes::install_github("chrisschuerz/SWATfarmR")
require(SWATfarmR)
require(sf)
require(tidyverse)
require(lubridate)
require(reshape2)
require(remotes)
require(dplyr)
require(data.table)
require(DT)
```

```
source('model_data/swat_farmR/functions_write_SWATfarmR_input.R')
```

11.1 Pre-processing crop rotation

This OPTAIN-provided workflow pre-processes our crop rotation data into a `SWATfarmR` compatible format. Authored by Michael Strauch, modified by Moritz Shore

11.1.1 Input files

11.1.1.1 Land use map with crop information

This map has the following requirements:

1. The map must contain the land use of each hru. In case of cropland, the names must be unique for each field (e.g., 'field_1', 'field_2', etc.)
2. The map must also contain crop infos for the period 1988 to 2020 (or 2021 if crop info available). This requires an extrapolation of the available crop sequences (the sequences derived from remote-sensing based crop classification or local data).
3. The extrapolated crop sequence for 33 years will be also used for running climate scenarios and must not contain any gaps. That means, gaps have to be closed manually!
4. The year columns must be named y_1988, y_1989, etc.
5. The crop infos for each year must match the crop_mgt names in the management operation schedules (provided in a .csv file, see below)

see also section 4.1 of the modelling protocol (**?**)

```
lu_shp <- 'model_data/input/crop/land_with_cr.shp'

lu <- st_drop_geometry(read_sf(lu_shp))

datatable(
  lu %>% head(50),
  extensions = "Scroller",
  options = list(scrollY = 200, scroller = TRUE)
)
```

Search:

	id	lu	cha_id	flow	y_1988	y_1989	y_1990	y_1991
1	1	frst						
2	2	frst						
3	3	frst						
4	4	rngb						
5	5	rngb						

Showing 1 to 6 of 50 entries

11.1.1.2 Management operation schedules for each crop

..or, if available, crop-management type.

1. All schedules must be compiled in one csv file (see example in demo data and study also section 4.2 of the modelling protocol)

2. 'crop_mgt' must start with the 4-character SWAT crop names (any further management specification is optional).

3. Each schedule must contain a 'skip' line to indicate the change of years.

4. The 'skip' line should never be the last line of a schedule.

```
mgt_csv <- 'model_data/input/management/mgt_crops_CS10.csv'

mgt_crop <- read.csv(mgt_csv)

datatable(mgt_crop,
          extensions = "Scroller",
          options = list(scrollY = 200, scroller = TRUE))
```

Search:

	crop_mgt	mon_1	day_1	mon_2	day_2	operation	op_data1
1	barl					skip	
2	barl	4	30	5	30	tillage	fldcult
3	barl	5	1	5	31	plant	barl
4	barl	5	1	5	31	fertilizer	elem_n
5	barl	5	1	5	31	fertilizer	elem_p

Showing 1 to 6 of 55 entries

11.1.1.3 Management operation schedules for generic land-use classes

Usually all non-cropland classes with vegetation cover

1. here, all schedules must be provided already in the SWATfarmR input format.

```
lu_generic_csv <- 'model_data/input/management/farmR_generic_CS10.csv' # generic land

mgt_generic <- read.csv(lu_generic_csv)

datatable(mgt_generic)
```

Show 10 entries Search:

	lulc_mgt	mon_1	day_1	mon_2	day_2	operation	op_data1	op_data2	op_data3
1	frst					initial_plant	frst	2,50000,0,0,1,10000	
2	wetl					initial_plant	wetl	2,50000,0,0,1,10000	
3	rngb					initial_plant	rngb	1,1000,0,0,1,1000	
4	rnge					initial_plant	rnge	1,1000,0,0,1,1000	
5	bsvg					initial_plant	bsvg	0.1,10,0,0,1,10	
6	past					initial_plant	fesc	1,1000,0,0,1,1000	
7	gras					initial_plant	fesc	1,1000,0,0,1,1000	
8	gras	4	10	5	5	fertilizer	elem_n	broadcast	60
9	gras	4	10	5	5	fertilizer	elem_p	broadcast	25
10	gras	6	25	7	5	harvest_only	fesc	hay_cut_low	

Showing 1 to 10 of 12 entries Previous 1 2 Next

11.1.2 Settings

11.1.2.1 Simulation Period

```
start_y <- 2010 # starting year (consider at least 3 years for warm-up!)
end_y <- 2020 # ending year
```

11.1.2.2 Prefix of cropland hrus

all names of hrus with a crop rotation must begin with this prefix in column 'lu' of your land use

```
hru_crops <- 'a_'
```

11.1.2.3 Multi-year farmland grass

Did you define any multi-year farmland grass schedules? 'y' (yes), 'n' (no)

```
m_yr_sch_existing <- 'n'
```

If yes, define also the following variables. **If not, skip next four lines**

```
crop_myr <- 'past' # name of your farmland grass
```

Maximum number of years farmland grass can grow before it is killed (should be <8)

```
max_yr <- 5
```

Do your multi-year farmland grass schedules consider the type of the following crop (summer or winter crop)?

(e.g., a '_1.5yr' schedule with a kill op in spring allows for planting a summer crop immediately afterwards)

If yes, you must define your summer crops

```
crop_s <- c('sgbt','csil','barl')
```

Do your summer crop schedules usually start with an operation in autumn (e.g. tillage)? To combine them with farmland grass, it is necessary that you provide 'half-year-schedules' ('half-year-schedules' are additional summer crop schedules

without operations in autumn) The adapted schedules should be added to the crop management table with suffix '_0.5yr' (e.g. 'csil_0.5yr')

If additional 'half-year-schedules' are not needed, because your normal summer crop schedules do not start in autumn, type 'n'

```
additional_h_yr_sch_existing <- 'n' # 'y' (yes), 'n' (no)
```

11.1.3 Checks

Check for correct positioning of 'skip' line

```
check_skip <- check_skip_position()
```

Check for date conflicts in single crop schedules

```
check_date_conflicts1()
```

Build schedules for crop sequences (Messages disabled)

```
rota_schedules <- build_rotation_schedules()
```

Check for date conflicts in the full rotation schedule. (Messages disabled)

```
check_date_conflicts2()
```

Solve minor date conflicts (where only a few days/weeks are overlapping)

(Messages disabled)

```
rota_schedules <- solve_date_conflicts()
```

Check again for date conflicts (Messages disabled)

```
check_date_conflicts2()
```

11.1.4 Writing input data

Write the SWAT farmR input table

```
write_farmR_input()
```

The output of this pre-processing stage is loaded in from this file:

```
farmR_input <- readr::read_csv("model_data/input/management/farmR_input.csv",
                               show_col_types = F)

datatable(
  head(farmR_input, 50),
  extensions = "Scroller",
  options = list(scrollY = 200, scroller = TRUE)
)
```

Search:

	land_use	management	weight	filter_attribute	condition_schedule
1	a_001f_7_lum				
2	a_001f_7_lum				(md >= 0410) * (m
3	a_001f_7_lum				
4	a_001f_7_lum				(md >= 0427) * (m
5	a_001f_7_lum				

Showing 1 to 6 of 50 entries

TODO: Figure out why this is an issue. Are you using the wrong buildR output as source material for the crop generation?

We need to fix the land use name, which entails replacing the `_drn_lum` with just `_lum` for all the agricultural fields. This is because `land.shp` from buildR does not include the `_drn` for some reason

```
# TODO: Figure out why this is an issue. Are you using the wrong buildR output
# as source material for the crop generation?

 farmR_input2 <- farmR_input %>% filter(grepl(x = land_use, "a_")) %>%
   mutate(land_use = str_replace(.$land_use, "_lum", "_drn_lum"))

 farmR_input3 <- farmR_input %>% filter(!grepl(x = land_use, "a_"))

 farmR_input4 <- rbind(farmR_input2, farmR_input3)

 write_csv(farmR_input4, file ="model_data/input/management/farmR_input2.csv")

 rm(farmR_input, farmR_input2, farmR_input3, farmR_input4)

# Yes I am aware that code is bad and ugly.
```

11.2 Initializing FarmR

We will create a dedicated directory for our SWATfarmR and copy in our latest text input files.

```
dir.create("model_data/cs10_setup/swat_farmR/", showWarnings = F)

sta_files <- list.files("model_data/cs10_setup/optain-cs10/",
                        pattern = "sta_",
                        full.names = T)

cli_files <- list.files("model_data/cs10_setup/optain-cs10/",
                        pattern = ".cli",
                        full.names = T)

input_files_base <- list.files("model_data/cs10_setup/swat_input/",
                               full.names = T)

source_files <- c(sta_files, cli_files, input_files_base)

status <- file.copy(from = source_files,
                    to = "model_data/cs10_setup/swat_farmR/",
                    overwrite = T)

# overwriting the base files with our modifications
input_files_mod <- list.files("model_data/cs10_setup/swat_input_mod/",
                              full.names = T)
```

```
status_mod <- file.copy(from = input_files_mod,
                        to = "model_data/cs10_setup/swat_farmR/",
                        overwrite = T)
```

If the FarmR has never been initialized, then use `new_farmr()` and `read_management()`, otherwise `load_farmr()`.

```
project_path <- "model_data/cs10_setup/swat_farmr/"

new_farmr(project_name = "cs10", project_path = project_path )

#load_farmr(...)

cs10$read_management(file = "model_data/input/management/farmR_input2.csv")
```

11.3 Calculating Antecedent Precipitation Index

TODO loading API from section REF

```
# using temp API

# TODO add real API

# Load dplyr. We will use functions such as 'mutate' and 'select'.
library(dplyr)
# Extract the precipitation from the farmr project
pcp <- cs10$.data$variables$pcp

# Extract the hydrologic soil group values for all HRUs
hsg <- select(cs10$.data$meta$hru_attributes, hru, hyd_grp)

# Calculate api values for the hsg classes A to D
api_A <- variable_decay(variable = pcp, n_steps = -5, decay_rate = 1)
api_B <- variable_decay(variable = pcp, n_steps = -5, decay_rate = 0.8)
api_C <- variable_decay(variable = pcp, n_steps = -5, decay_rate = 0.7)
api_D <- variable_decay(variable = pcp, n_steps = -5, decay_rate = 0.5)

# Bind the data together into one api table and name them with the hsgs
api <- bind_cols(api_A, api_B, api_C, api_D)
names(api) <- c('api_A', 'api_B', 'api_C', 'api_D')
```

```
# To add the variable to the farmR you have to tell it which variables are
# assigned to which HRUs
hru_asgn <- mutate(hsg, api = paste0('api_', hyd_grp)) %>% select(hru, api)

# Add the variable api to the farmR project
cs10$add_variable(data = api, name = 'api', assign_unit = hru_asgn, overwrite = T)
```

TODO

11.4 Scheduling Operations

```
cs10$schedule_operations(start_year = 2011, end_year = 2020, n_schedule = 2)
```

11.4.1 Write Operations

We cannot write from 1988 as it is limited to our climate data, which currently only spans back to 2010 (2011?)

```
cs10$write_operations(start_year = 2011, end_year = 2020)
```

"Writing management files: - Loading scheduled operations: 100%
- Preparing 'hru-data.hru' - Preparing 'landuse.lum' - Preparing 'schedule.mgt'
- Preparing 'plant.ini' - Writing files - Updating 'time.sim' - Updating 'file.cio'
Finished writing management files in 5M 40S"

Make sure to use the files in the SWATfarmR directory going forward.

Test to see if the setup still works:

```
# No it does not
# TODO
```

Temp fix:

```
### Change tiledrain spacing to 800
```

Related Issues

FarmR 3.1 Issues UFZ #28

FarmR 3.1 Fix UFZ #27 on the UFZ gitlab FarmR 3.0 issues – #64

Update FarmR to 3.0 - #56

No Plant ET - #48

Schedule Ops 1988-2020 - #51

Re-extrapolate the crop rotation - #47

Chapter 12

Model Verification

Preparation

Loading required packages, defining paths, and loading objects.

```
# Install/Update SWATdoctR if needed:
# remotes::install_git('https://git.ufz.de/schuerz/swatdoctr', force = FALSE)
require(SWATdoctR)
require(DT)
```

We will create a test directory to run DoctR SWAT runs and copy in our latest input files from SWATFarmR

```
doctr_path <-  "model_data/cs10_setup/doctr_run"
unlink(doctr_path, recursive = T)

dir.create(doctr_path)

file.copy(
  from = list.files("model_data/cs10_setup/swat_farmR/",full.names = T),
  to = doctr_path,
  overwrite = T,
  recursive = T
)
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
##  [76] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [106] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [121] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [136] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [151] TRUE TRUE TRUE TRUE
```

```
# Add exec:
status <- file.copy(
  from = "model_data/cs10_setup/Rev_60_5_6_64rel.exe",
  to = paste0(doctr_path, "/Rev_60_5_6_64rel.exe"),
  overwrite = T
)
```

12.1 Stage 1: Analysis of simulated climate variables

Generate a model run with water balance elements in the output.

```
run_name = "spacing800"

veri_no_stress <- run_swat_verification(project_path = doctr_path,
                                        nostress = 0,
                                        keep_folder = F,
                                        outputs = "wb")
saveRDS(veri_no_stress,
        paste0("model_data/swat_doctR/verification_runs/",
        run_name,
        ".rds"))
```

Load Model Run

```
run_name = "spacing800"


veri_no_stress <-
  readRDS(paste0("model_data/swat_doctR/verification_runs/", run_name, ".rds"))
```

Modified from the Protocol:

- The climate variables daily precipitation and daily minimum/maximum temperatures are **required** inputs of a SWAT+ model setup (more info: protocol section 2.4).

We have these present.

- Further climate inputs such as solar radiation, relative humidity and wind speed are **optional** input variables and can be essential for the calculation of the potential evapotranspiration (PET).

  We have these, see issue #42 (link)

- Climate inputs are grouped to weather stations in a model setup and are assigned to spatial objects (HRUs, channels, reservoirs, etc.) with the `nearest neighbor` method.

  We only have 1 weather station so this is not of great relevance.

- The input of weather data and the assignment of climate variables to spatial objects can be sources for several issues which must be analysed:

1. Data structure of the climate input tables, units of the climate variable, no data flag, etc. was wrong and can result in unrealistically small or large values of the climate variables in the simulation.

   Does not seem to be the case in our data

2. The nearest neighbor assignment allocates weather stations to spatial objects where the weather records do not represent the actual weather conditions in a spatial object well. This can for example be an issue in complex terrain.

   Should not be a concern for us (for now) since we only have a single met station

3. The selected method for the calculation of **PET** results in an under/overestimation of PET when compared to estimates of PET for the region. In such cases other methods for the simulation of PET which are included in SWAT+ should be tested if they better fit the regional conditions and available weather inputs (see more Additional settings).

   We are having an underestimation of ET from Ecanopy, so it might be an idea to switch the method See issue #62

4. Large implausibilities in the weather inputs can be identified in analyses of annual basin averages of the simulated climate variables. **Simulated annual and average values of climate variables must be comparable to observation data and/or region specific literature values**. Any larger deviations of precipitation can indicate errors in the input file or an inappropriate assignment of weather stations to spatial units.

We have pure observations, so no simulated data.

5. If the lapse rate option is active (Read more in Additional settings), it may be another potential reason for deviations from observations.

   We are discussing this topic in issue #39 (link). It has been disseminated in Section **??**

12.1.1 Lapse rate

From the SWAT+ changelog, we can see that setting the lapse parameter to 1 allows for adjustment of temperature and precipitation based off of elevation.

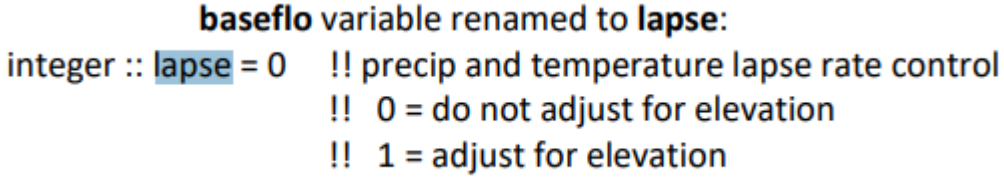


Figure 12.1: Lapse parameter from REVISION 60.5.4 documentation

The `plaps` parameter allows for adjustment of precipitation amounts per kilometer elevation change. The default is set to 0. The `tlaps` parameter allows for adjusting temperature per kilometer elevation, and it's default is 6.5C/km.

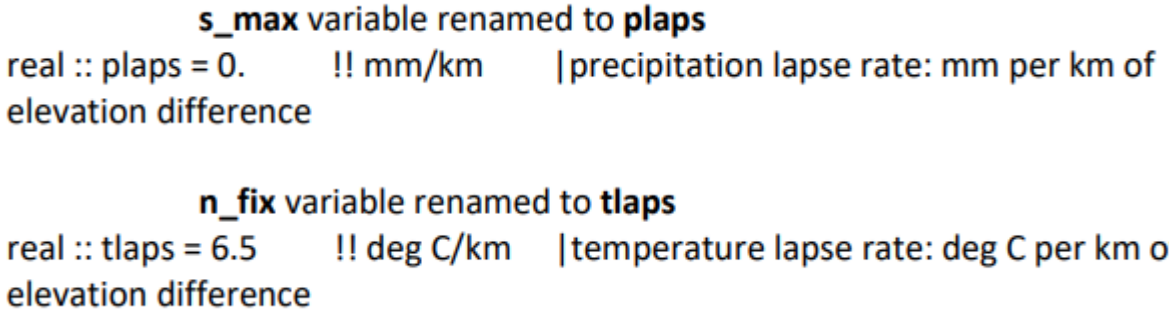


Figure 12.2: The tlaps and plaps parameters from the REVISION 60.5.4 documentation document

Let us check the elevation differences in our catchment to see if these parameters could have a meaningful effect.

```
dem_path <- "model_data/cs10_setup/optain-cs10/data/raster/dem.tif"

library(terra)
dem_rast <- rast(dem_path)

minmax(dem_rast)
```

```
##     dtm3_ns_v5
## min      50.50
## max     251.91
```

Max elevation change of 200 m is not much, but it could very well have an effect on snowfall / vs. precipitation. The max temperature adjustment would be:

```
6.5 * 0.200
```

```
## [1] 1.3
```

1.3 Degrees Celsius is significant, therefore we will enable `tlaps` and keep the default value. 200m does not seem like enough to justify using the plaps parameter, so we will leave it at 0

And load in `codes.bsn` and `parameters.bsn`

```
### Loading parameters
codes_path <- "model_data/cs10_setup/doctr_run/codes.bsn"
parameters_path <- "model_data/cs10_setup/doctr_run/parameters.bsn"
codes_bsn <-
  read.table(
    file = codes_path,
    skip = 1,
    header = T,
    sep = "",
    colClasses = "character"
  )
parameters_bsn <-
  read.table(
    file = parameters_path,
    skip = 1,
    header = T,
    sep = "",
    quote = "",
    colClasses = "character"
  )
```

```
### Setting parameters
codes_bsn$lapse = 1
parameters_bsn$tlaps = 6.5
parameters_bsn$plaps = 0

### Writing Headers
write.table(
  "codes.bsn header, modified by CS-10 workflow section 11.1",
  codes_path,
  sep = "    ",
  row.names = F,
  col.names = F,
  append = F,
  quote = F
)
write.table(
  "parameters.bsn header, modified by CS-10 workflow section 11.1",
  parameters_path,
  sep = "    ",
  row.names = F,
  col.names = F,
  append = F,
  quote = F
)
### Writing file contents
write.table(
  codes_bsn,
  codes_path,
  sep = "    ",
  row.names = F,
  col.names = T,
  append = T,
  quote = F
)
write.table(
  parameters_bsn,
  parameters_path,
  sep = "    ",
  row.names = F,
  col.names = T,
  append = T,
  quote = F
)
```

12.1.2 Climate variables

- Over or underestimated PET can indicate errors in the temperature input files (and if provided in the solar radiation, relative humidity and wind speed inputs).
- `SWATdoctR` provides the function `plot_climate_annual()` to analyse the annual simulated basin averages of climate variables.

```
fig1 <- plot_climate_annual(veri_no_stress)
plot(fig1)
```

- The **first panel** shows ET fractions. Current version of SWAT+ often has implausible ET fractions!

  Are these values plausible?

No – tracking in issue #62

- The **second panel** shows the precipitation fractions rainfall (`rainfall`) and snowfall (`snofall`).

  Are these values plausible?

Yes. Resolved in issue #43

- The **third panel** shows the annual temperature values.

  Are these values plausible?

Yes.

- The **fourth panel** shows the relative humidity values.

  Are these values plausible?

Answer: resolved in issue #42) but are they plausible? Csilla

- The **fifth panel** shows wind speed.

  Are these values plausible?

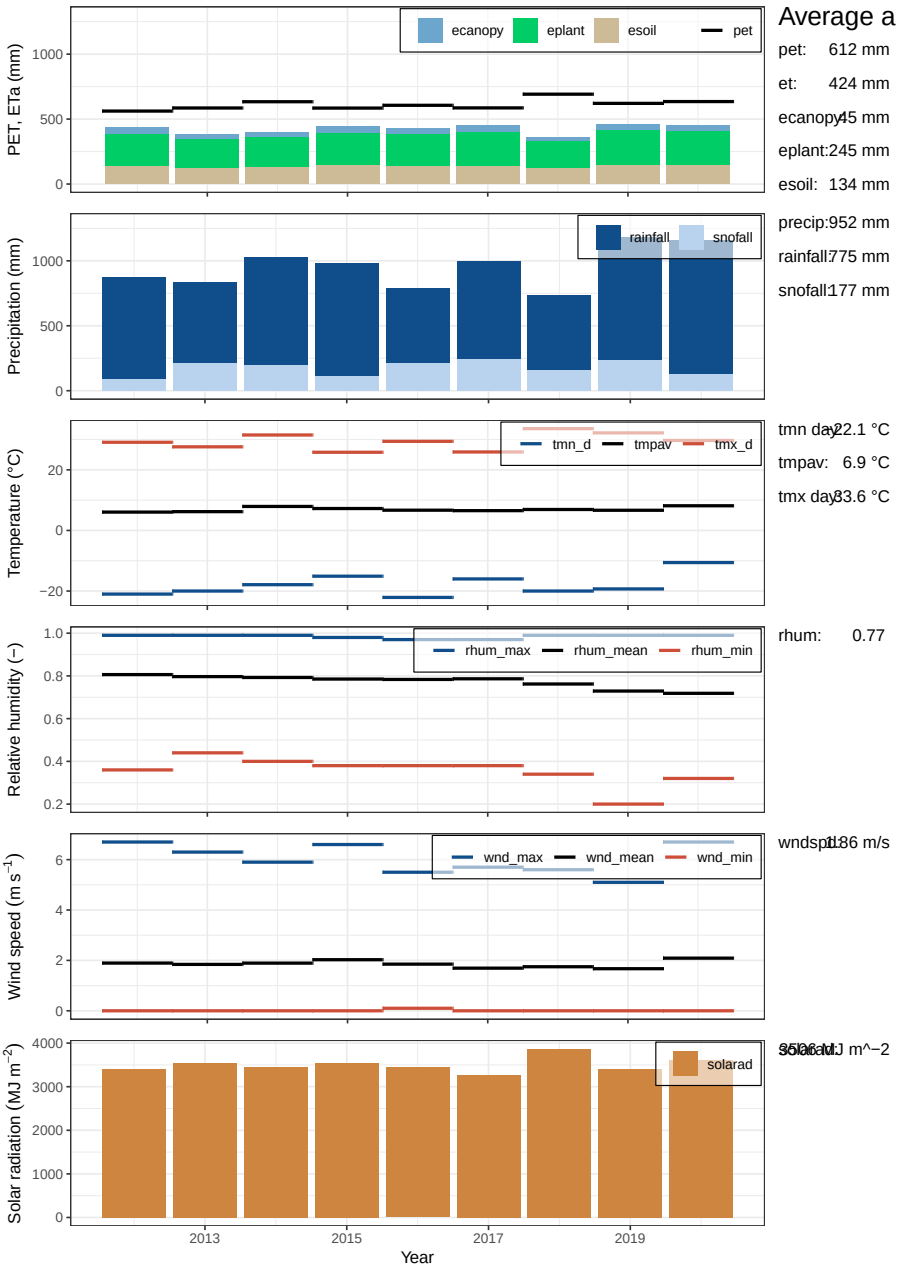


Figure 12.3: Climate plot of CS10 generated by SWATDoctR

Resolved in #42, Csilla plausible?

- The **sixth panel** shows the annual sums of solar radiation. A comparison to literature values of annual solar radiation sums for the region can indicate issues in this input.

  Are these values plausible?

Yes, values for the Oslo area seem to be around 4000 MJ – Csilla do you agree?

12.1.3 Snowfall and Snowmelt

- The analysis of mean monthly precipitation (output variable `precip`), snowfall (output variable `snofall`) and snow melt (output variable `snomlt`) sums and their comparison with region specific information (or in the best case observations) provides insight in seasonal dynamics of the precipitation input. Particularly in snow impacted catchments a first verification of snowfall is valuable to see whether precipitation in solid form is simulated, a snow storage can build up and cause increased spring runoff through snow melt. The hydrological cycle of some catchments may be dominated by spring flood events which must be reflected by the simulated processes. Any observed implausibility in such analysis can indicate issues in the weather inputs or require to pay attention in the calibration of model parameters which control the simulation of snow processes (`snofall_tmp`, `snomelt_tmp`, `snomelt_lag`).

  CS10, a Boreal catchment, is impacted by snow melt – this is relevant to the verification

```
plot_monthly_snow(sim_verify = veri_no_stress)
```

  This has been resolved in issue #43. (link). Are these values plausible csilla?

- In situations where not all required climate inputs are available which are necessary to estimate PET with PM method the estimates will be more uncertain and annual PET sums may differ to regional values. Then the use of a simpler method for the calculation of PET can be a valid solution.

  We do not need a simpler method since we have the required data

---

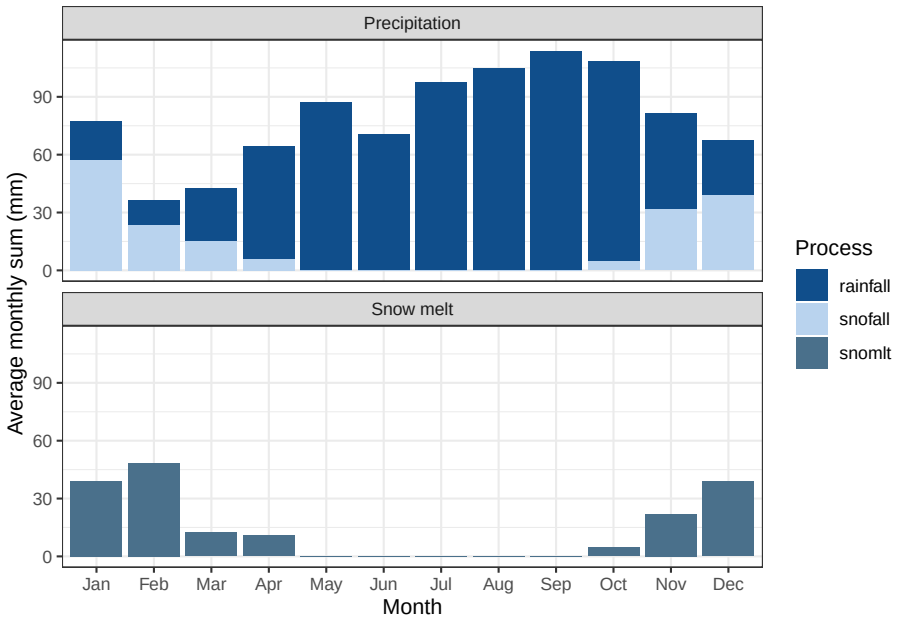


Figure 12.4: Precipitation and Snowmelt on a monthly average basis, as generated by SWATDoctR

12.2 Stage 2: Simulation of management operations

Development of management tables is complicated and complex – and thus error prone. Mistakes in this area do not produce any errors or warnings making them easy to miss.

> All operations which are triggered in a SWAT+ simulation run are written into the file 'mgt_out.txt'

To verify the operations, we are going to compare scheduled and simulated operations for specific HRUs by random sampling.

For this, we need a SWAT+ run with management outputs. If we have already run this code, we do not need to run it again, and can just load it in:

```
run_name = "spacing800all"

veri_mgt <-
  readRDS(paste0("model_data/swat_doctR/verification_runs/", run_name, ".rds"))
```

Skip this code block, unless you want to re-run SWAT+

```
veri_mgt <-
  run_swat_verification(
    project_path = doctr_path,
    keep_folder = T
  )

run_name = "spacing800all"
saveRDS(veri_mgt,
        paste0("model_data/swat_doctR/verification_runs/",
               run_name,
               ".rds"))
```

From the Protocol:

"The function `report_mgt()` generates an overview report where the scheduled and triggered operations are matched and compared for each management schedule that was implemented in the simulations. The function prepares the scheduled management operations that were written in the input file 'management.sch' in tabular form and randomly samples one HRU for each defined schedule from the triggered management operations (from the output file 'mgt_out.txt'). The comparison is only done for operations that were defined with a fixed date in the management schedule and operations which are triggered by decision tables will be excluded."

"Applying the function `report_mgt()` for the model verification simulation outputs returns a table with an overview of the operations which were scheduled but not triggered or operations where 'op_data1' differs in the scheduled and triggered operations."

```
mgt_report <- report_mgt(veri_mgt, write_report = TRUE)
```

```
## Management OK! No differences between scheduled and triggered managments identified.
```

Seems like there are no issues. I am not sure if this is a good or bad thing :D.

"The `report_mgt()` function is a good starting point to explore the triggered management. But this analysis can be error prone. Still the safest way to analyse the triggered and the scheduled managements is to compare the input and output tables. `SWATdoctR` provides the function `print_triggered_mgt()` to print the triggered managements for individual HRUs. For selecting HRUs e.g. with a specific management the helper function `get_hru_id_by_attribute()` can be useful. This table can be visually compared with the management input table ('management.sch')"

```
table <- print_triggered_mgt(sim_verify = veri_mgt, hru_id = 92)
```

```
## Triggered managament for
##   hru:        92
##   management: a_007f_1_drn_mgt_92_1
```

```
datatable(table)
```

Show 10 entries Search:

	year	mon	day	phuplant	operation	op_data1	op_data3
1	2011	5	12	0	TILLAGE	harrow	0
2	2011	5	17	0	FERT	elem_n	96
3	2011	5	17	0	FERT	elem_p	0
4	2011	5	17	0	PLANT	oats	0
5	2011	5	22	0.050951749	TILLAGE	rowcult	0
6	2011	6	15	0.383119	FERT	elem_n	38
7	2011	6	15	0.383119	FERT	elem_p	0
8	2011	8	9	1.303469	HARVEST	oats	0
9	2011	8	9	0	KILL	oats	0
10	2012	4	24	0	TILLAGE	harrow	0

Showing 1 to 10 of 92 entries

Previous 1 2 3 4 5 … 10 Next

To see the operations of a specific management schedule:

```
test <- get_hru_id_by_attribute(veri_mgt, mgt = "a_001f_1_drn_mgt_1731_1")

tables <- print_triggered_mgt(sim_verify = veri_mgt, hru_id = test$id[1])
```

```
## Triggered managament for
##   hru:        1731
##   management: a_001f_1_drn_mgt_1731_1
```

```
datatable(tables)
```

Show 10 entries Search:

	year	mon	day	phuplant	operation	op_data1	op_data3
1	2011	4	17	0.089400448	FERT	elem_n	79.1
2	2011	4	17	0.089400448	FERT	elem_p	0
3	2011	5	6	0.2004455	FERT	elem_n	77.2
4	2011	5	6	0.2004455	FERT	elem_p	0
5	2011	6	30	0.634599	FERT	elem_n	34
6	2011	6	30	0.634599	FERT	elem_p	0
7	2011	8	1	0.967464	HARVEST	wwht	0
8	2011	8	1	0	KILL	wwht	0
9	2011	10	6	0	PLANT	wwht	0
10	2011	10	15	0.036294542	TILLAGE	rowcult	0

Showing 1 to 10 of 100 entries

Previous 1 2 3 4 5 … 10 Next

"Operations which are missing in the simulated management schedules must be checked in the 'management.sch' input file. By answering the following questions for the scheduled management operations their proper implementation in the model setup can be verified:"

1. Are the date sequences in the scheduled operations correct and in a right order (mistakes in assigned month and day values)?

Answer:

2. Does the variable `op_data1` point to the correct entry in the respective input data file? Does the label exist in the input file? E.g. does defined `op_data1` exist in 'tillage.til' for tillage operations, or does defined `op_data1` exist in 'plant.plt' for plant operations

   Answer:

3. Does the variable `op_data2` point to the correct entry in the respective operations file ('.ops')? E.g. does harvest operation defined with `op_data2` exist in 'harv.ops'.

   Answer:

12.3 Stage 3: Analysis of unconstrained plant growth

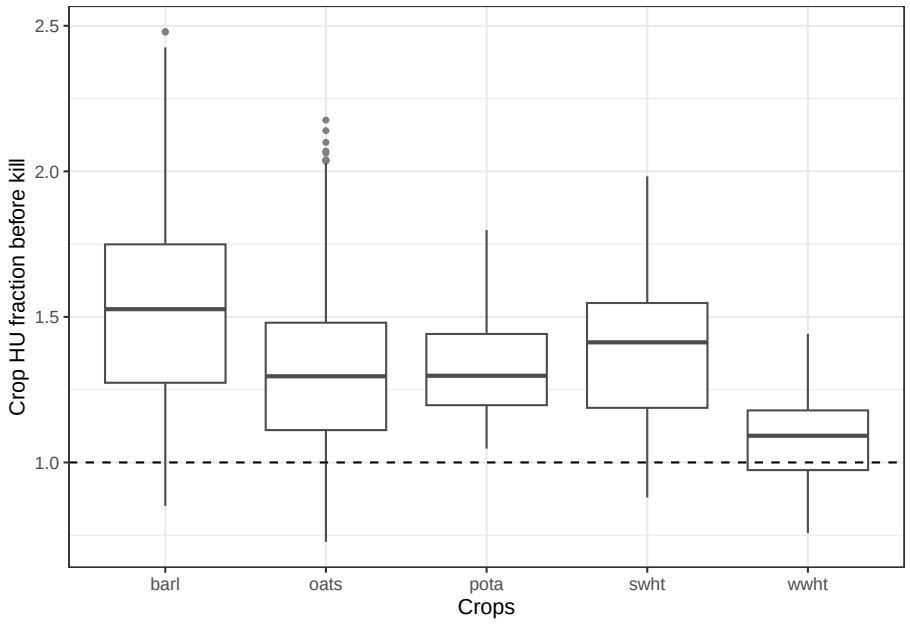
This will be a two-tiered approach (with and without stress factors)

"The function `plot_variable_at_harvkill()` function summarises the state of variables at the time of harvest/kill operations for all crops in a model setup and thus provides a general overview"

```
plot_variable_at_harvkill(veri_mgt, "yield")
```

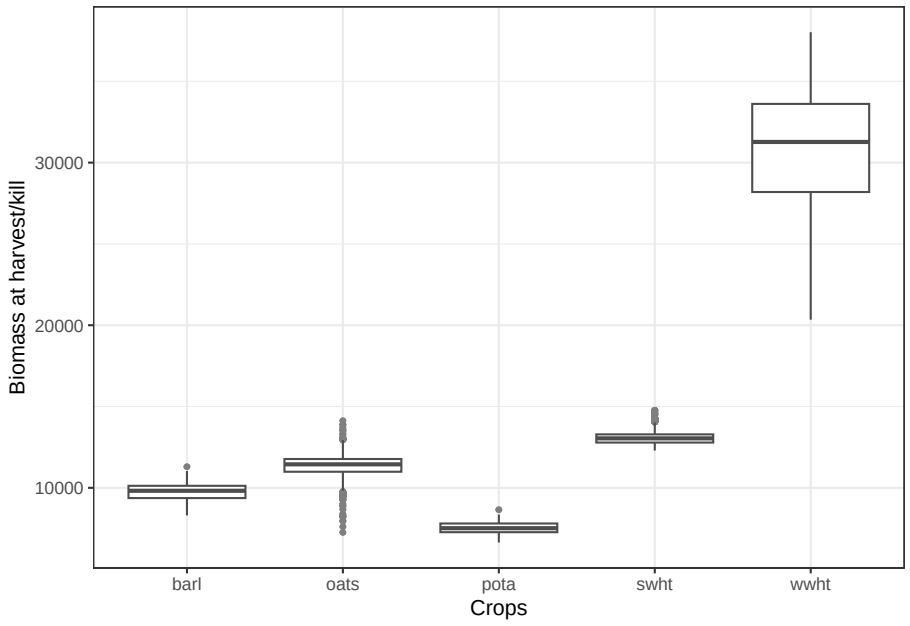
```
plot_variable_at_harvkill(veri_mgt, "phu")
```

```
## For the following crops the difference between the last harvest and
## the final kill operation was more than 1 day:
## barl
## The plotted PHU fractions may therefore not be correct.
```

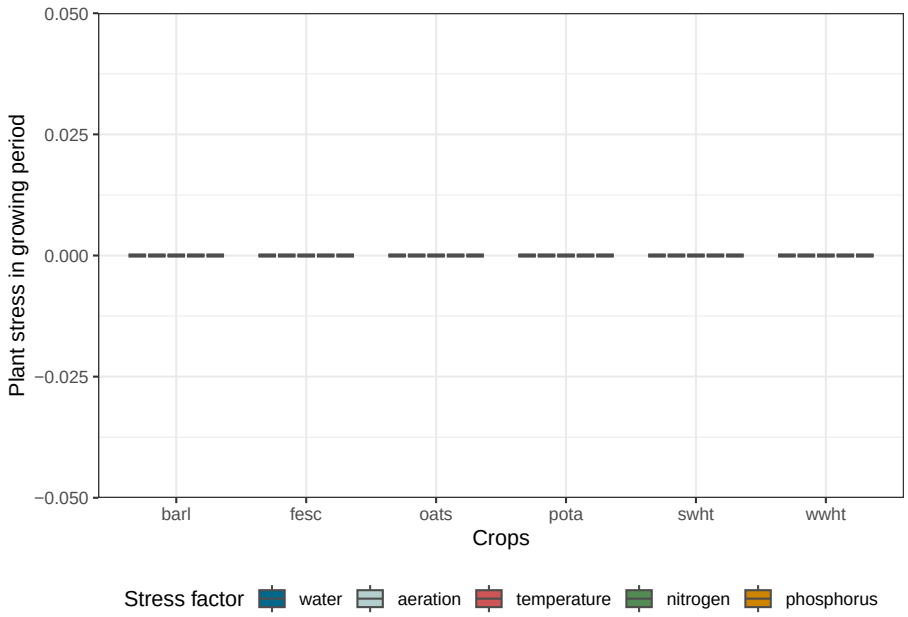



```
plot_variable_at_harvkill(veri_mgt, "bioms")
```

```
## Data for the following crops were removed before plotting:
## fesc, barl
```

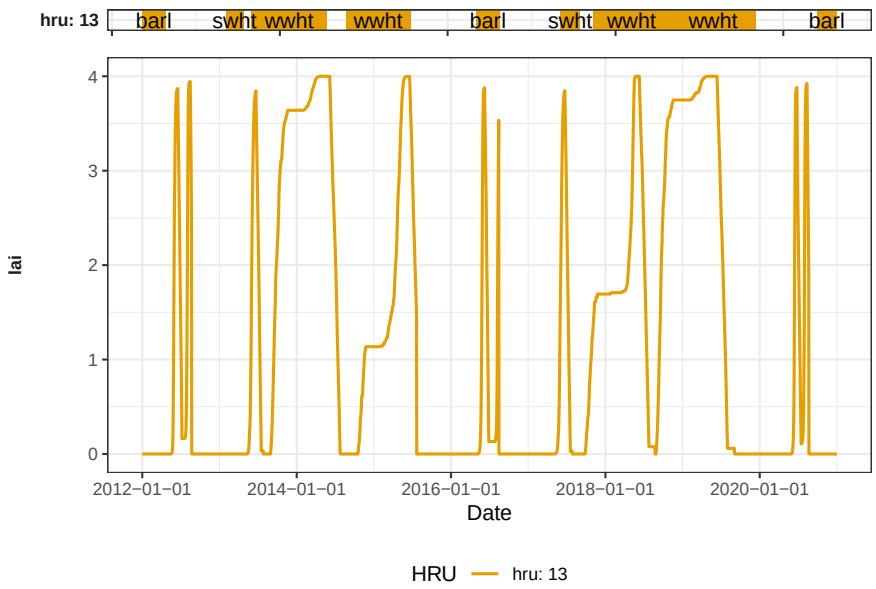


```
plot_variable_at_harvkill(veri_mgt, "stress")
```

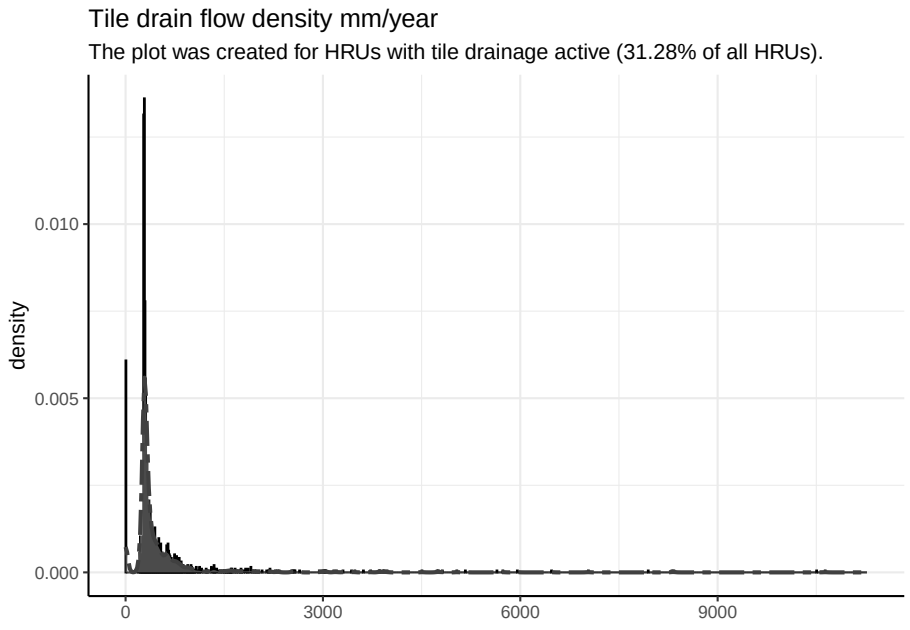


"the function `plot_hru_pw_day()` allows detailed analyses of the daily time series of HRU related variables, which then can only be performed for a few HRUs of a model setup."

```
plot_hru_pw_day(veri_mgt, 13, var =  "lai")
```




```
plot_qtile(veri_mgt)
```



update_landuse_labels currently has its issues. See here

```
update_landuse_labels(doctr_path)
```

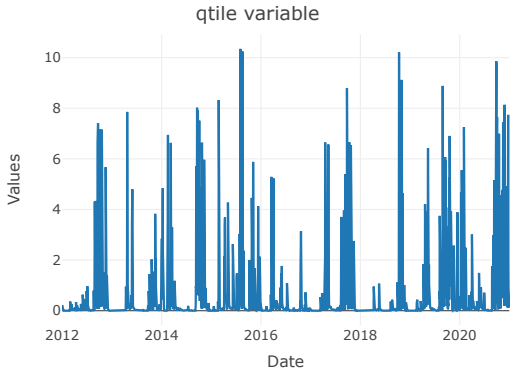
```
hru_wb_check <- check_hru_waterbalance(
  veri_mgt,
  check = c("precip", "et", "runoff", "sw", "cn"),
  ignore_lum = NULL,
  add_values = TRUE
)

datatable(hru_wb_check)
```

Show 10 entries Search:

	id	lu_mgt	precip_check	et_check	eplant_check	surq_wyld_check	perc_wyld_check	surq_check
1	1	frst_lum					perc/wyld < 22%	surq > 150% exp. surq
2	2	frst_lum					perc/wyld < 22%	surq > 150% exp. surq
3	3	frst_lum				surq/wyld > 78%	perc/wyld < 22%	surq > 150% exp. surq
4	4	rngb_lum		et < 30% precip	eplant < esoil		perc/wyld < 22%	
5	5	rngb_lum		et < 30% precip	eplant < esoil		perc/wyld < 22%	surq > 150% exp. surq
6	6	urml_lum			eplant < esoil	surq/wyld > 78%	perc/wyld < 22%	surq > 150% exp. surq
7	7	rngb_lum		et < 30% precip	eplant < esoil		perc/wyld < 22%	surq > 150% exp. surq
8	8	a_001f_7_drn_lum_8_1					perc/wyld < 22%	
9	9	a_001f_5_drn_lum_9_1					perc/wyld < 22%	
10	10	a_002f_2_drn_lum_10_1					perc/wyld < 22%	

Showing 1 to 10 of 6,206 entries Previous 1 2 3 4 5 … 621 Next

```
plot_basin_var(veri_mgt, "qtile", period = "day", fn_summarize = "mean")
```



plot_waterbalance currently broken, see here

```
plot_waterbalance(veri_mgt)
```

```
qtile_aa_dt <- print_avannual_qtile(veri_mgt)

datatable(qtile_aa_dt)
```

Show 10 entries Search:

	id	qtile	lu_mgt	mgt	soil
1	85	0	a_006f_1_drn_lum_85_1	a_006f_1_drn_mgt_85_1	Sea_dep_thick
2	91	0	a_007f_1_drn_lum_91_1	a_007f_1_drn_mgt_91_1	Sea_dep_thick
3	92	0	a_007f_1_drn_lum_92_1	a_007f_1_drn_mgt_92_1	Sea_dep_thick
4	173	0	a_007f_1_drn_lum_173_1	a_007f_1_drn_mgt_173_1	Sea_dep_thick
5	182	0	a_006f_1_drn_lum_182_1	a_006f_1_drn_mgt_182_1	Sea_dep_thick
6	191	0	a_007f_1_drn_lum_191_1	a_007f_1_drn_mgt_191_1	Sea_dep_thick
7	192	0	a_007f_1_drn_lum_192_1	a_007f_1_drn_mgt_192_1	Sea_dep_thick
8	193	0	a_006f_1_drn_lum_193_1	a_006f_1_drn_mgt_193_1	Sea_dep_thick
9	194	0	a_006f_1_drn_lum_194_1	a_006f_1_drn_mgt_194_1	Sea_dep_thick
10	195	0	a_007f_1_drn_lum_195_1	a_007f_1_drn_mgt_195_1	Sea_dep_thick

Showing 1 to 10 of 1,941 entries

Previous 1 2 3 4 5 … 195 Next

```
plot_ps(veri_mgt)
```

```
## [1] "No point sources exists in this model setup!!!"
```

Related Minor Issues

Getting drainage yet? - #16

Atmodep not read - #44

References