visCOS: documentation

Daniel Klotz, Johannes Wesemann, Mathew Herrnegger

Contents

Hello, traveler

visCOS (*vis*ual *c*omparison of *o*bservations and *s*imulations) is still under (heavy) development. It is thus not recommended to use the package, yet.

There exist many hydrological R-packages.

Among them there are several that have the goal of providing aid to the calibration of hydrological models. In particular: The `hydroGOF` packages provides a set of commonly used (and disputed) objective functions. The `hydroTSM` package allows for the analysis, interpolation and plotting of hydrological time series. Topically, **visCOS** can be positioned between those two. In concrete, visCOS is an R-package that provides summaries and visualisation to aid the parameter estimation for conceptual rainfall-runoff models in general and COSERO in special. COSERO is a HBV-like model that was developed at the institute for water management, hydrology and hydraulic engineering (IWHW) at BOKU, Vienna. The name is an abbreviation for "**Co**nceptual **Se**mi-Distributed **R**ainfall Run**o**ff Model". This page provides a hub to access the different examples and the entire program code.

1 Cooking Data

Within **visCOS** "cooking data" is is used as a synonym for the process of transforming *raw data* into *cooked data*. Here, *cooked data* is data in the COSERO data.frame format. *Cooked data* can be eaten or served. That is, is can be used for the furhter analysis and to generate figures. The package provides a set of basic functions cooking data.

1.1 Raw Data

Raw-data are time-series of observations o and model simulations s. Usually raw data is saved in some simple file format, e.g. *.txt* or *.csv*. `R` natively includes many options to read those files, e.g. with the `read.table` functions (simply enter `?read.table` in your `R` terminalto get an overview). For larger (unstructured) files we recommend the read options of the `data.table` package. In our tests it was fastest and most flexible alternative. If the data is better structured (and large) you might also want to try the `readr` package.

1.2 Cooked Data

In order to learn how to cook data it is usefull to know what cooked data is. This section will describe the basic data structure used to enable visualizations in **visCOS**

Currently **visCOS** only allows to compare between *numbered catchments*! The data **must** include an integer number at the end of its name, e.g. `QObs_001` and `QSim_001`).

1.3 Examples

The function `get_viscos_example` can be used to get some exemplary data from within **visCOS**:

```
library(visCOS)
require(magrittr)
options(width=80)
runoff_example_raw <- get_viscos_example()
head(runoff_example_raw)
```

```
##   yyyy mm dd hh min QOBS_0001 QOSI_0001 QSIM_0001 QOBS_0002 QOSI_0002 QSIM_0002
## 1 2007  1  1  0   0      2.98      3.48      3.48      2.56      3.11      3.11
## 2 2007  1  1  1   0      2.89      3.48      3.48      2.56      3.11      3.11
## 3 2007  1  1  2   0      2.64      3.48      3.48      2.57      3.11      3.11
## 4 2007  1  1  3   0      2.51      3.48      3.48      2.57      3.11      3.11
## 5 2007  1  1  4   0      2.42      3.48      3.48      2.57      3.11      3.11
## 6 2007  1  1  5   0      2.34      3.49      3.49      2.57      3.11      3.11
```

The next step would be to adapt the options (`viscos_options`, but in this case the options are already set as in the data:

```
 viscos_options() %>% unlist()
```

```
##         name_o         name_s      data_unit   name_COSyear  name_COSmonth
##         "qobs"         "qsim"       "(m^3/s)"        "yyyy"           "mm"
##    name_COSday   name_COShour    name_COSmin name_COSposix name_COSperiod
##           "dd"           "hh"          "min"   "posixdate"       "period"
##   missing_data        color_o        color_s    of_limits1     of_limits2
##         "-999"    "steelblue"       "orange"           "0"            "1"
```

However, a glimpse of the data above shows that some of the columns (`QOSI_0001` and `QOSI_0002`) are not needed for the analysis. All the junk data, i.e. data columns that are not defined within the options and unobserved columns can be removed using the `remove_junk` function. Additionally **visCOS** needs to have two different ways to define the date of a given row (see: Introduction). In the runoff_example each part of the date definition is defined with a column: “yyyy” - year, “mm” - month, “hh” - hour, “min” - minute. The other needed format is the POSIXct`format (see: [link](https://stat.ethz.ch/R-manual/R-devel/library/base/html/as.POSIXlt.html)) format often used within R. Here all the information is saved within one column Given one date format is available in the data.frame the other can be generated by using`complete_dates‘.

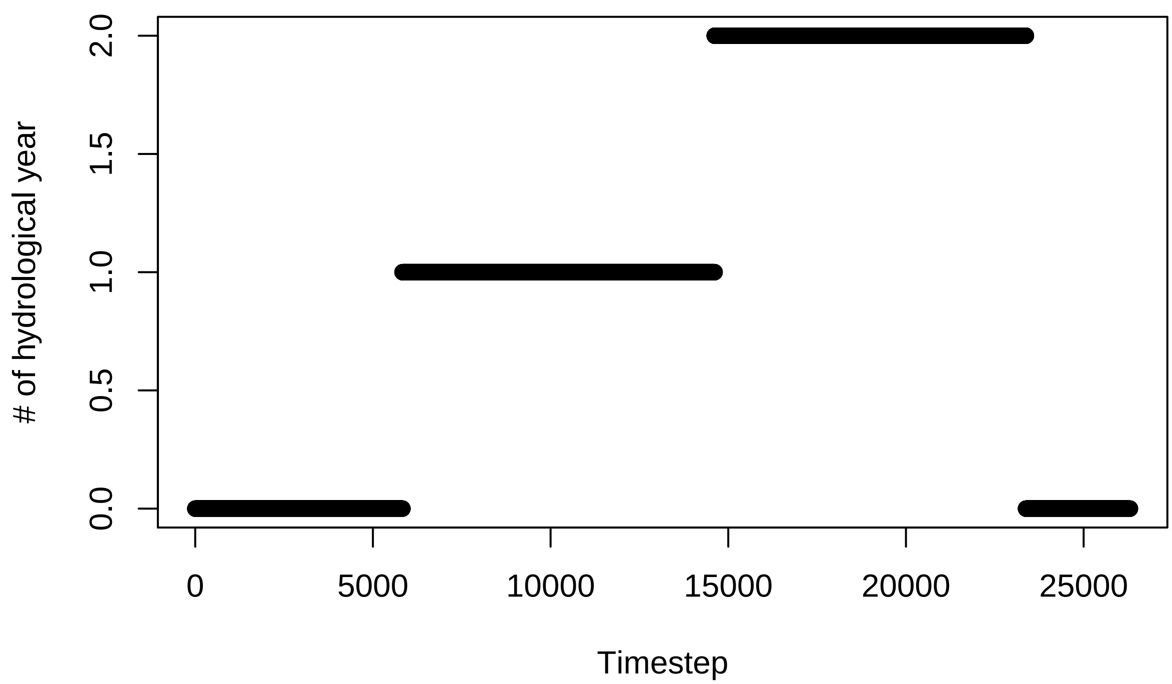
```
runoff_example_raw %>%
  remove_junk() %>%
  complete_dates() %>%
  head()
```

```
##   yyyy mm dd hh min QOBS_0001 QSIM_0001 QOBS_0002 QSIM_0002           posixdate
## 1 2007  1  1  0   0      2.98      3.48      2.56      3.11 2007-01-01 00:00:00
## 2 2007  1  1  1   0      2.89      3.48      2.56      3.11 2007-01-01 01:00:00
## 3 2007  1  1  2   0      2.64      3.48      2.57      3.11 2007-01-01 02:00:00
## 4 2007  1  1  3   0      2.51      3.48      2.57      3.11 2007-01-01 03:00:00
```

```
## 5 2007  1  1  4   0       2.42       3.48       2.57       3.11 2007-01-01 04:00:00
## 6 2007  1  1  5   0       2.34       3.49       2.57       3.11 2007-01-01 05:00:00
```

Lastly, if a seasonal or periodical analysis of the data wants to be done, these periods can be defined with the `mark_periods` function. In the example case, the hydrological years, going from September till August, will be defined:

```
cooked_runoff_example <- runoff_example_raw %>%
  remove_junk() %>%
  complete_dates() %>%
  mark_periods(., start_month = 9, end_month = 8)
# here is an example plot to visualise the periods
plot(cooked_runoff_example$period,
     xlab = "Timestep",
     ylab = "# of hydrological year")
```



The example data starts with beginning of September, so right with the first hydrological year. The end of the year 2010 is not completely inside a hydrological year, so the period counter jumps to one.

1.4 Code

```
# ---------------------------------------------------------------------------
# Code for data cooking
# authors: Daniel Klotz, Johannes Wesemann, Mathew Herrnegger
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This section defines these functions and explains their usage. Currently the following steps can be performed from within **visCOS**:

- Provide example data
- Remove junk
- Prepare date formats
- Define periods

1.4.1 Get Example Data

visCOS provides some exemplary data. All the available functions can be tested with it. `get_viscos_example` is a small wrapper function to get the data from within the package via `read.csv`

```
# --------------------------------------------------------------------------
#' Get runoff example
#'
#' Get exemplary runoff data to test the different functions of visCOS
#' @export
get_viscos_example <- function() {
  file_path <- system.file("extdata",
                           "runoff_example.csv",
                           package = "visCOS")
  runoff_example <- read.csv(file_path)
  return(runoff_example)
}
```

1.4.2 Remove not needed columns

This function removes all columns not specified in the `viscos_options`, as well basins where no `data_o` is available. **Note** that the routine is **not** case sensitive. It does not distinguish between small and capital letters!

The first part of the code loads the dependencies and makes sure that the `cos_data` variable is a data.frame (see: chapter about *defensive coding*).

The body of `remove_junk` works as following: First, the names of the columns of the `cos_data` are determined and helper function `get_regex_for_cos_data` is invoked to get regular expressions for the columns defined within `viscos_otopns`. Then the `idx` of the different wanted data is extracted and used to select the appropriate columns.

Afterwards, the "sub" function `only_observed_basins` is being executed (see next chapter)

```
# --------------------------------------------------------------------------
#' removes junk in cos_data
#'
#' Removes all columns which are not foreseen (see: viscos_options) from
#' runoff data
#'
#' @import magrittr
#' @param cos_data The cos_data data.frame (see vignette for info)
#' @return data.frame object without the chunk
#' @export
remove_junk <- function(cos_data) {
  assert_dataframe(cos_data)
  #
  names_in_data <- cos_data %>% names
  regex_columns <- get_regex_for_cos_data() # see: helpers
```

```
  #
  idx <- regex_columns %>%
    grep(.,names_in_data, ignore.case = TRUE)
  no_junk_cos_data <- cos_data[ , idx]
  return( only_observed_basins(no_junk_cos_data) )
}
```

1.4.2.1 Get only observed basins

This function removes basins for which no data in the `o_data` column. No observation are columns where all the entries are either as defined in `viscos_options("missing_data")` or `NA's`. At the end of the function the former are transformed into `NA's`.

```
# ---------------------------------------------------------------------------
# remove basins without observations
#
# Removes basins without observation (-999/NA values) from the provided data.frame
#
# @param cos_data A raw cos_data data.frame, which may contains basins
# without observations.
# \strong{Note:} It is assumed that all available basins are simulated!
# @return data.frame without the observation-free basins
#
# @import magrittr
# @import pasta
only_observed_basins <- function(cos_data) {
  require("magrittr")
  require("pasta")
  assert_dataframe(cos_data)
  # set NA values to viscos_options("missing_data") and check if there are
  # cloumns wihtouth observervation
  chosen_cols <- which( names(cos_data) != viscos_options("name_COSposix") )
  rows_with_na <- is.na(cos_data[,chosen_cols])
  data_wihtouth_posix <- cos_data[ ,chosen_cols]
  data_wihtouth_posix[rows_with_na] <- viscos_options("missing_data")
  colmax <- sapply(X = data_wihtouth_posix, FUN = max)
  #
  # if there are columns withouth observations remove them from the data:
  if ( any(colmax < 0.0) ){
    name_o <- viscos_options("name_o")
    neg_o_names <- which(colmax < 0.0) %>%
      names
    neg_s_names <- gsub(name_o,viscos_options("name_s"),neg_o_names,ignore.case = TRUE )
    data_selection <- neg_o_names %|% neg_s_names %>%
      paste(.,collapse = "|") %>%
      grepl(., names(cos_data), ignore.case = TRUE) %>%
      not(.)
    data_only_observed <- cos_data[ ,data_selection]

  } else {
    data_only_observed <- cos_data
  }
  # set all missing data values to NA for use in hydroGOF
  idx_NA <- data_only_observed %>%
```

```
    equals(viscos_options("missing_data"))
  data_only_observed[idx_NA] <- NA
  return(data_only_observed)
}
```

1.4.3 Complete the date formats

This function is not finished yet! Within cos_data data.frame (see: introduction) the date has to be defined in two formats: With 5 columns, where the columns are year-month-day-hour-minute and names as defined in `viscos_options()` and with one column in `POSIXct` format (see: link), that is named as defined in `viscos_options("name_COSposix")`. The idea of the `complete_dates` fucntion is to provide a internal posibility to create the one format out of the ohter and thus *complete the dates*.

However, currently it is only possible to convert the 5-columns representation into `POSIXct` dates via the internally defined `implode_cosdate` function.

Also by using *UTC* a **fixed time-zone** is assumed within visCOS to avoid problems with leaps in time (summer/winter time).

```
# -----------------------------------------------------------------------------
#' Complete the date-formats with POSIXct or COSdate
#'
#' Complete the data-formats of your data.frame `POSIXct` and/or `COSdate`
#'
#' @param cos_data The data.frame, which contains the runoff information
#' @param name_cosyear string with the name of the `COSdate` year column
#' @param name_posix string with the name of the POSIXct column
#' @return The new runoff data.frame with the added data-format.
#'
#' @import magrittr
#'
#' @export
complete_dates <- function(cos_data) {
  # make sure that magrittr is loaded:
  assert_dataframe(cos_data)
  # check for COSdates and stop if non-logical expression are obtained
  OK_COSdate <- any(
    unlist(viscos_options("name_COSyear",
                          "name_COSmonth",
                          "name_COSmonth",
                          "name_COShour",
                          "name_COSmin")
                          )
    %in%
    names(cos_data)
  )
  OK_POSIXdates <- any(names(cos_data) == viscos_options("name_COSposix"))
  if ( !is.logical(OK_COSdate) | !is.logical(OK_POSIXdates) ) {
    stop("Something is wrong :( \n
         some of the date-columns could not be processed!")
  }
  # choose function depending on which formats are available!
  if (!OK_COSdate & !OK_POSIXdates) {
    stop("Something is wrong :( \n
```

```
          The 5 cosero date columns and the POSIXct colum could not be found")
  } else if (OK_COSdate & !OK_POSIXdates) {
    cos_data <- implode_cosdate(cos_data) # see following chapter
  } else if (!OK_COSdate & OK_POSIXdates) {
    stop("POSIXct to COSdates not yet supported :(")
  }
  return(cos_data)
}
```

1.4.3.1 Implode cosdate

This function is used to transform the “old-school” 5 column format into the widely spread POSIXct format. The function is not exported and should only be called from within `complete_date()`!.

```
# ---------------------------------------------------------------------------
implode_cosdate <- function(cos_data) {
  require("magrittr", quietly = TRUE)
  assert_dataframe(cos_data)
  name_string <-  cos_data %>% names %>% tolower
  #
  POSIXdate <- paste(cos_data[[viscos_options("name_COSyear")]],
                     sprintf("%02d",cos_data[[viscos_options("name_COSmonth")]]),
                     sprintf("%02d",cos_data[[viscos_options("name_COSday")]]),
                     sprintf("%02d",cos_data[[viscos_options("name_COShour")]]),
                     sprintf("%02d",cos_data[[viscos_options("name_COSmin")]]),
                     sep= "" ) %>%
    as.POSIXct(format = "%Y%m%d%H%M", origin = .[1], scale = "hourly", tz = "UTC")
  cos_data[[viscos_options("name_COSposix")]] <- POSIXdate
  return(cos_data)
}
```

1.4.4 Remove leading zeros in column names

This internal function removes leading zeros from column names of the `cos_data` data.frame. The function has no defensive code but uses `remove_junk` (see: above). It should therefore be used with care!

```
# ---------------------------------------------------------------------------
# remove leading zeros from the names of cos_data (data.frame)
remove_leading_zeros <- function(cos_data) {
  # pre: ==================================================================
  require("magrittr", quietly = TRUE)
  require("pasta", quietly = TRUE)
  cos_data %<>% remove_junk
  name_o <- viscos_options("name_o")
  search_o_or_s <- paste0(name_o,"|", viscos_options("name_s"))
  #
  runoff_names <- cos_data %>% names
  runoff_lowercase_names <- runoff_names %>% tolower
  del_leading_zeros <- function(string) sub("^[0]+", "",string)
  # calc: =================================================================
  idx_o <- grep(name_o , runoff_lowercase_names)
  separator <- runoff_lowercase_names %>%
    extract( idx_o[1] ) %>%
    gsub(name_o, "", .) %>%
```

```
    gsub("\\d", "", .)
  runoff_nums <- runoff_lowercase_names %>%
    gsub(search_o_or_s, "",.) %>%
    gsub(separator, "", .) %>%
    gsub("\\D", "", .)
  search_runoff_nums <- "[" %&% paste(runoff_nums, collapse = "") %&% "]"
  runoff_only_names <- runoff_names %>%
    gsub(search_runoff_nums, "", .) %>%
    gsub(separator,"",.)
  runoff_new_numbers <- del_leading_zeros(runoff_nums)
  #
  new_names <- runoff_new_numbers %>%
    gsub("\\d+", separator,.) %>%
    paste0(runoff_only_names, ., runoff_new_numbers)
  names(cos_data) <- new_names
  return(cos_data)
}
```

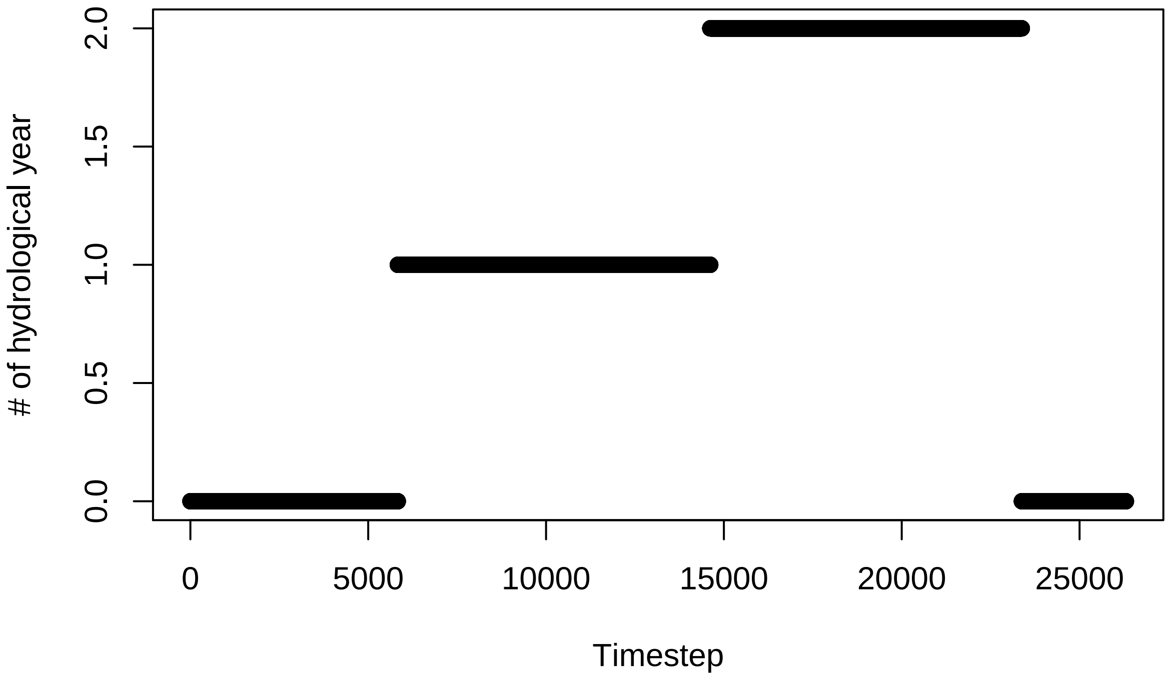
1.4.5 Mark the needed periods

This function creates a period column. The period column consists of increasing integers for each period. Rows with zeros are considered to be out-of-period. The names of the period column is defined with `viscos_options("name_COSperiod")`.

The function has three inputs: The data.frame `cos_data` and the two integers `start_month` and `end_month`. The latter define the to-be monhts that are within the period. Everything between `start_month` and `end_month` is consideren to be in-period.

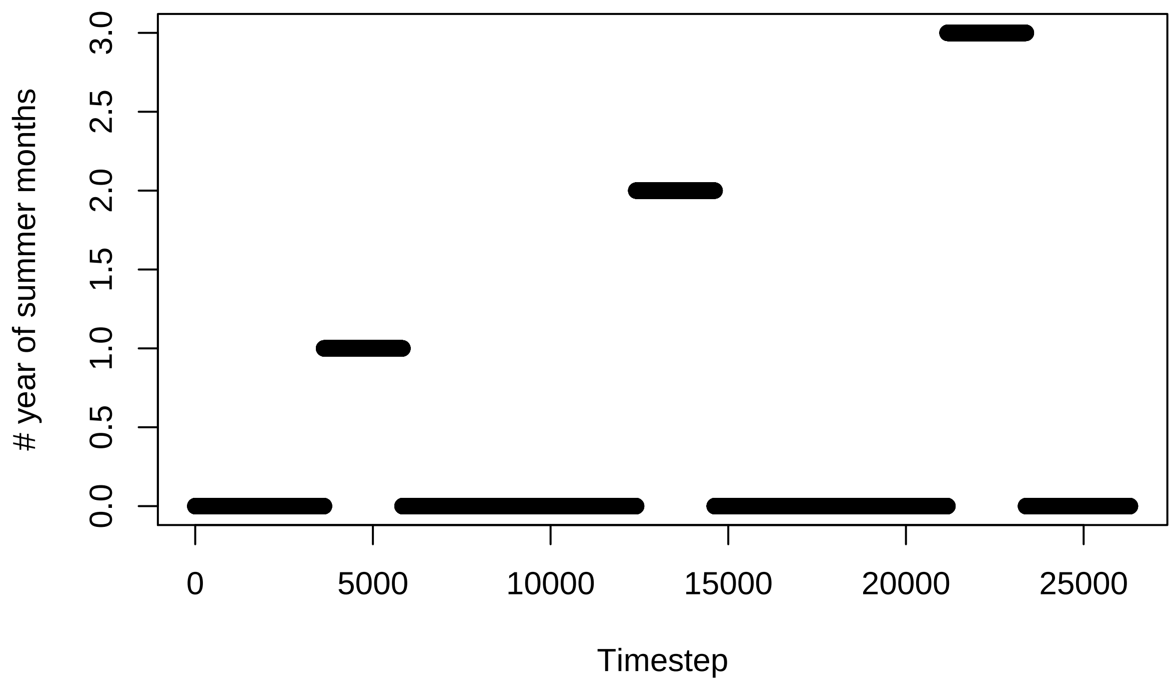
Here are two examples that display the slected periods and the numbering:

```
require(magrittr)
require(visCOS)
# example 1: hydrological years (september till august)
ex1 <- get_viscos_example() %>% mark_periods(start_month = 9, end_month = 8)
plot(ex1$period, xlab = "Timestep", ylab = "# of hydrological year")
```




```r
# note that the last year is not complete, so the counter jumps back to 0

# example 2: summer months (june till august)
ex2 <- get_viscos_example() %>% mark_periods(start_month = 6, end_month = 8)
plot(ex2$period, xlab = "Timestep", ylab = "# year of summer months")
```



This is currently computed in the following way:

Before starting the acutal computation the variables `period_range` and `out_of_period` are defined. The `period_range` are the months ordered in the given range. The `out_of_period` variable marks all months which are not within the chosen period. With these variables the periods can be "marked" in two steps:

1. All the starting months within cos_data are marked and the cumulative sum is used to count the periods within the data.frame. At the beginning of the first period, the counter is at "1" and becomes "2" with the beginning of the second period and so on.
2. The `out_of_period` of all years is set back to zero again by checking which months of the data are equal to the `out_of_period` entries.

One problem with this solution is that the last year is not extracted properly if the `start_month`is higher than the `end_month`. To compensate this problem the `dplyr` shenanigans are added as an inofficial third step. Another quirk is, that with this solution the the first and last period are included, even if they are not complete.

This solution is not realy satisfying. But, life is short and it at the time it seemed to be the best that the authors could come out with. Suggestions for improvements are welcome!

```
# ----------------------------------------------------------------------------
#' calculate periods
#'
#' Mark the periods within cos_data.
# The marking uses a monthly resolution, which are defined by the integers
#' `start_month` and `end_month`.
#'
#' @param cos_data The data.frame, which contains the runoff information
#' @return The runoff data.frame reduced and ordered according to the
```

```r
#' hydrological years within the data.
#' \strong{Note:} The periods columns are formatted as characters!
#'
#' @import dplyr
#' @import magrittr
#'
#' @export
mark_periods <- function(cos_data, start_month = 10, end_month = 9) {
  # pre: ================================================================
  assert_dataframe(cos_data)
  name_year <- viscos_options("name_COSyear")
  name_month <- viscos_options("name_COSmonth")
  cos_data %<>% remove_junk %>% complete_dates()
  eval_diff <- function(a) {c(a[1],diff(a))}
  # calc: ===============================================================
  # (I) get labels for the months:
  if (start_month <= end_month ) {
    period_range <- seq(start_month,end_month)
    out_of_period <- seq(1,12) %>% extract( !(seq(1,12) %in% period_range) )
  } else if (start_month > end_month) {
    range_1 <- seq(start_month,12)
    range_2 <- seq(1,end_month)
    period_range <- c(range_1,range_2)
    out_of_period <- seq(1,12) %>% extract( !(seq(1,12) %in% period_range) )
  }
  # (II) mark periods:
  start_months_in_data <- cos_data[[name_month]] %in% c(start_month)
  cos_data[[viscos_options("name_COSperiod")]] <- start_months_in_data %>%
    eval_diff() %>%
    pmax(.,0) %>%
    cumsum
  out_period_in_data <- cos_data[[name_month]] %in% out_of_period
  cos_data$period[out_period_in_data] <- 0
  # (III) corrections for last year
  max_year <- max(cos_data[[name_year]])
  marked_cos_data <- cos_data %>%
    dplyr::mutate(
      period = ifelse(
        ((.[[name_year]] == max_year) &  (.[[name_month]] > end_month)),
        0,
        period)
    )
  return(marked_cos_data)
}
```

1.4.6 Transform the `cos_data` into `xts`

This function is just a small wrapper around the `xts()` function for internal use in **viCOS**.

A notable quirk of the function is it puts all column-names to lower cases and removes leading zeros in their enumeration.

```r
# ---------------------------------------------------------------------
#' Convert cos_data to xts-format
```

```r
#'
#' Converts the cos_data (class: data_frame) into an xts object
#'
#' @param cos_data data_frame of the cos_data (see: xxx)
#' @return xts object of the cos_data data.frame
#'
#' @import zoo
#' @importFrom xts xts
#' @import magrittr
cos_data_as_xts <- function(cos_data) {
  # pre: ==============================================================
  assert_dataframe(cos_data)
  assert_junk(cos_data)
  assert_complete_date(cos_data)
  # calc: =============================================================
  # set every- name to lover capitals and generate xts frame
  new_names <- names(cos_data) %>% tolower
  name_posix <- viscos_options("name_COSposix") %>% tolower
  cos_data <- remove_leading_zeros(cos_data) %>%
    magrittr::set_names(new_names)
  cos_data_as_xts <- xts(x = cos_data[], order.by = cos_data[[name_posix]])
  return(cos_data_as_xts)
}
```

2 Options

visCOS provides a set of global options for controlling the package. They are implemented with the help of the `GlobalOptions` package. This section defines the code for the global options and explains the individual options:

```r
#' visCOS global options
#'
#' Get and set the global options of visCOS
#'
#' These are the options you can adapt by executing the function
#' (default values)
#' \preformatted{
#'   viscos_options(
#'    # data.frame column names
#'       name_o = "qobs", # name of the first time-series data (observations)
#'       name_s = "qsim", # name of the second time-series data (simulations)
#'       name_COSyear = "yyyy", # name of year-column
#'       name_COSmonth = "mm",  # name of month-column
#'       name_COSday = "dd",  # name of day-column
#'       name_COShour = "hh", # name of hour-column
#'       name_COSmin = "min", # name of minute-column
#'       name_COSposix = "posixdate", # name of the complete-date-column
#'       name_COSperiod = "period", # name of the marked-period column
#'       missing_data = -999, # marker for missing data in the o_columns
#'    # plot options
#'       color_o = "steelblue", # color associated with the first o time-series data
#'       color_s= "orange",  # color associated with the second s time-series data
```

```r
#'        of_limits = c(0,1) # limits of the plotted objective functions
#'    )
#' }
#'
#' @examples
#' viscos_options("name_o")
#' viscos_options(name_o = "OtherData")
#' viscos_options("name_o")
#' @export
viscos_options <- GlobalOptions::setGlobalOptions(
  # data.frame column names
  name_o = "qobs",
  name_s = "qsim",
  data_unit = "(m^3/s)",
  name_COSyear = "yyyy",
  name_COSmonth = "mm",
  name_COSday = "dd",
  name_COShour = "hh",
  name_COSmin = "min",
  name_COSposix = "posixdate",
  name_COSperiod = "period",
  missing_data = -999,
 # plot options
  color_o = "steelblue",
  color_s = "orange",
  of_limits = c(0,1)
)
```

3 Ojective Functions

This chapter defines the objective functions that are used in **visCOS**.

Objective functions, in short of, are an important part of the hydrological model calibration. Their importance arises from the approximate nature of the models and the large uncertainties of the process. Hydrological models are not only imperfect, in the sense that they simplify nature, but in most cases structurally different than the reality so that different models or their respective parametrisations approximate the hydrograph equally well. Thus, over the time a many objective functions have been developed to either make the model-results better interpretable/comparable or to address specific problems of given objective functions.

3.1 Code

This section defines the code for different objective functions. If possible the calculation is done with the help of the `hydroGOF` package, if not an R-code solution is tried. Currently 4 main objective functions are provided in **visCOS**. They can be directly extracted from the `cos_data` data.frame via the `main_of_` functions. Other objective functions are provided to, but no special extraction and visualisation functions are provided for them.

For the explanation and definition of the objective function it is assumed that o are the observations (defined by `name_o` in visCOS) and s are the simulation(defined by `name_s` in visCOS).

```r
#' Objective Functions
#'
#' Different objective Functions, provided by visCOS. A detailed description
```

```r
#' of each of the provided objective function is provided in the respective
#' vignette
#'
#' @param o The reference data or observations (o_data)
#' @param s The created data or the simulations (s_data)
#' @name of_overview
NULL
```

3.1.1 The "Main" Objective Functions

Currently the main objective functions are the Nash-Sutcliffe Efficiency, the Kling-Gupta Efficiency, the percentage bias and the correlation.

3.1.1.1 Nash-Sutcliffe Efficiency

The Nash-Sutcliffe Criterion NSE is by far the most used efficiency criterion in hydrology. In the hydrological context o usually represents a set of runoff-observation and s a set of simulations. The NSE is defined in the same way as the general definition of the coefficient of determination R^2:

$$NSE = \frac{\sum_{t=1}^{T} \big(o(t) - s(t)\big)^2}{\sum_{t=1}^{T} \big(o(t) - \bar{o}\big)^2}.$$

The variable $\bar{o}$ represents the average of o. The NSE can be seen as the relational the estimator s and the estimator resulting form the average of the data. It can can have values between minus infinity and 1 with 1 being the perfect fit, 0 when the mean of s is as good as the mean of o and negative values are even worse.

The code for the NSE computation is:

```r
#' Nash-Sutcliffe Efficiency
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_nse <- function(o,s) {
  as.numeric( NSE(s,o) )
}
```

3.1.1.2 Kling-Gupta Efficiency

The Kling-Gupta Efficiency KGE was introduced by Gupta et al. (2009) to alleviate some of the shortcomings of the NSE. In their paper they argue why the NSE tends to overate simulations with small variance (note: in the context of the paper simulations = s) and propose their efficiency criterion instead.

The KGE is defined as:

$$KGE = 1 - ED,$$

with

$$ED = \sqrt{\big(corr(o,s) - 1\big)^2 + \big(\alpha(o,s) - 1\big)^2 + \big(\beta(o,s) - 1\big)^2}.$$

In which $\alpha(o,s) = \frac{\sigma_s}{\sigma_o}$ is the standard deviation σ of s divided by the σ of o, $\beta(o,s) = \mu_s/\mu_o$ withμ being the arithmetic mean and $corr(o,s)$ as the Pearson's correlation coefficient (see below). The value range and the quality is similar to the NSE.

The code for the KGE computation is:

```
#' Kling-Gupta Efficiency
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_kge <- function(o,s) {
  as.numeric( KGE(s,o) )
}
```

3.1.1.3 Percentage Bias

The percentage of bias p_{bias} is defined as the sum of the differences between o and s divided by the sum of o:

$$p_{bias} = 100 * \frac{\sum_{t=1}^{T}[o(t) - s(t)]}{\sum_{t=1}^{T} o(t)}.$$

The $100*$ is just a scaling factor applied to express p_{bias} as a percentage.

The code for the p_{bias} computation is:

```
#' Percentage Bias
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_p_bias <- function(o,s) {
  as.numeric( pbias(s,o) )
}
```

3.1.1.4 Pearson's correlation coefficient

Pearson's correlation coefficient, r or $corr(o,s)$, is a measure of the linear relationship between o and s. It is defined as:

$$r \equiv corr(o,s) = \frac{cov(o,s)}{\sigma_s * \sigma_o},$$

where $cov(...)$ denotes the covariance. The correlation coefficient can take on values between -1 and 1. The former corresponds to an inverse and the latter to a direct relationship and the closer the values is to zero the weaker is the implied correlation.

The code for the correlation is:

```
#' Correlation
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_cor <- function(o,s) {
```

```
  diag( cor(o,s) )
}
```

3.1.2 Ohter Objective Functions

Descriptions shall follow

3.1.2.1 Root Mean Sqaured Error

$$\frac{\sum_{t=1}^{T}\big(o(t)-s(t)\big)^2}{T}$$

```
#' Root Mean Sqaured Error
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_rmse <- function(o,s) {
  as.numeric( rmse(s,o) )
}
```

3.1.2.2 Inverted Nash-Sutcliffe Efficiency

$$nse^{-1} = \frac{\sum_{t=1}^{T}\big(s(t)-o(t)\big)^2}{\sum_{t=1}^{T}\big(s(t)-\bar{s}\big)^2}$$

```
#' Inverted Nash-Sutcliffe Efficiency
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_invert_nse <- function(o,s) {
  as.numeric( NSE(o,s) )
}
```

3.1.2.3 Ratio of the Standard Deviations

$$rsd = \frac{\sigma_s}{\sigma_o}$$

```
#' Ratio of Standard Deviations
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_rsd <- function(o,s) {
  as.numeric( rSD(s,o) )
}
```

3.1.2.4 Ratio of the Means

$$rmeans = \mu_s/\mu_o$$

```
#' Ratio of Means
#'
#' @rdname of_overview
#' @export
of_rmeans <- function(o,s) {
  as.numeric( mean(s)/mean(o) )
}
```

3.1.2.5 Volumetric Efficiency

The volumetric efficiency, VE, uses the absolute distance between observation and simulation instead of the quadratic and is bound between 0 to 1.

$$VE = 1 - \frac{\sum_{t=1}^{T} abs\big(s(t) - o(t)\big)}{\sum_{t=1}^{T} o(t)}$$

```
#' Volumetric Efficiency
#'
#' @rdname of_overview
#' @import hydroGOF
#' @export
of_ve <- function(o,s) {
  as.numeric( VE(s,o) )
}
```

3.2 References

- **Percentage Bias:** Yapo P. O., Gupta H. V., Sorooshian S., 1996. Automatic calibration of conceptual rainfall-runoff models: sensitivity to calibration data. Journal of Hydrology. v181 i1-4. 23-48
- **Nash-Sutcliffe Efficiency:** Nash, J. E. and J. V. Sutcliffe (1970), River flow forecasting through conceptual models part I -A discussion of principles, Journal of Hydrology, 10 (3), 282-290
- **Kling-Gupta Efficiency:** Gupta, Hoshin V., Harald Kling, Koray K. Yilmaz, Guillermo F. Martinez. Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling. Journal of Hydrology, Volume 377, Issues 1-2, 20 October 2009, Pages 80-91. DOI: 10.1016/j.jhydrol.2009.08.003. ISSN 0022-1694
- **Volumetric Efficiency:** Criss, R. E. and Winston, W. E. (2008), Do Nash values have value? Discussion and alternate proposals. Hydrological Processes, 22: 2723-2725. doi: 10.1002/hyp.7072

4 Time Aggregates

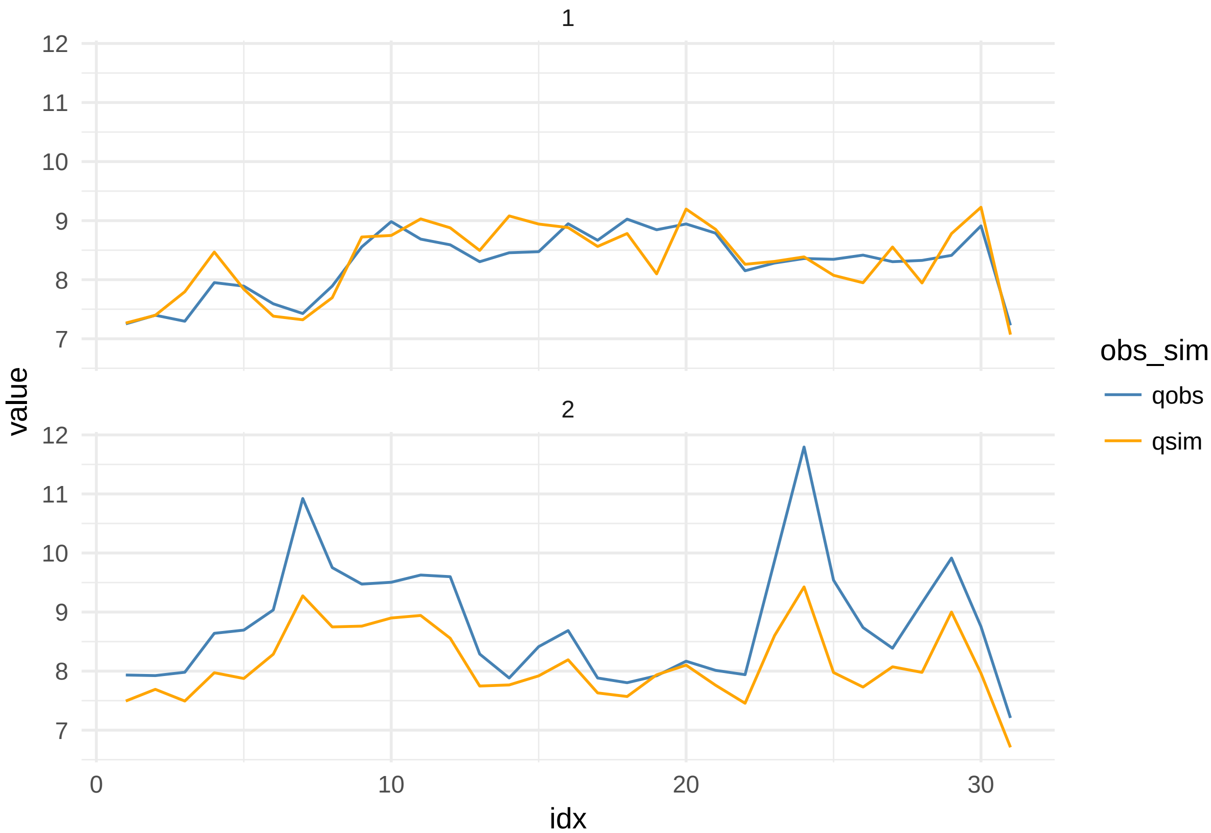
In hydrology it is often useful to summarise the data respect to a given time dimension. In **visCOS** this can be done by using the `aggregate_time` function. The function takes COSERO data.frame and aggregates them according to a chosen time dimension. Note, that the name of the dimension can be specified via the options.

4.1 Examples

```
require(ggplot2, quietly = TRUE)
require(visCOS, quietly = TRUE)
```

Daily runoff aggregation:

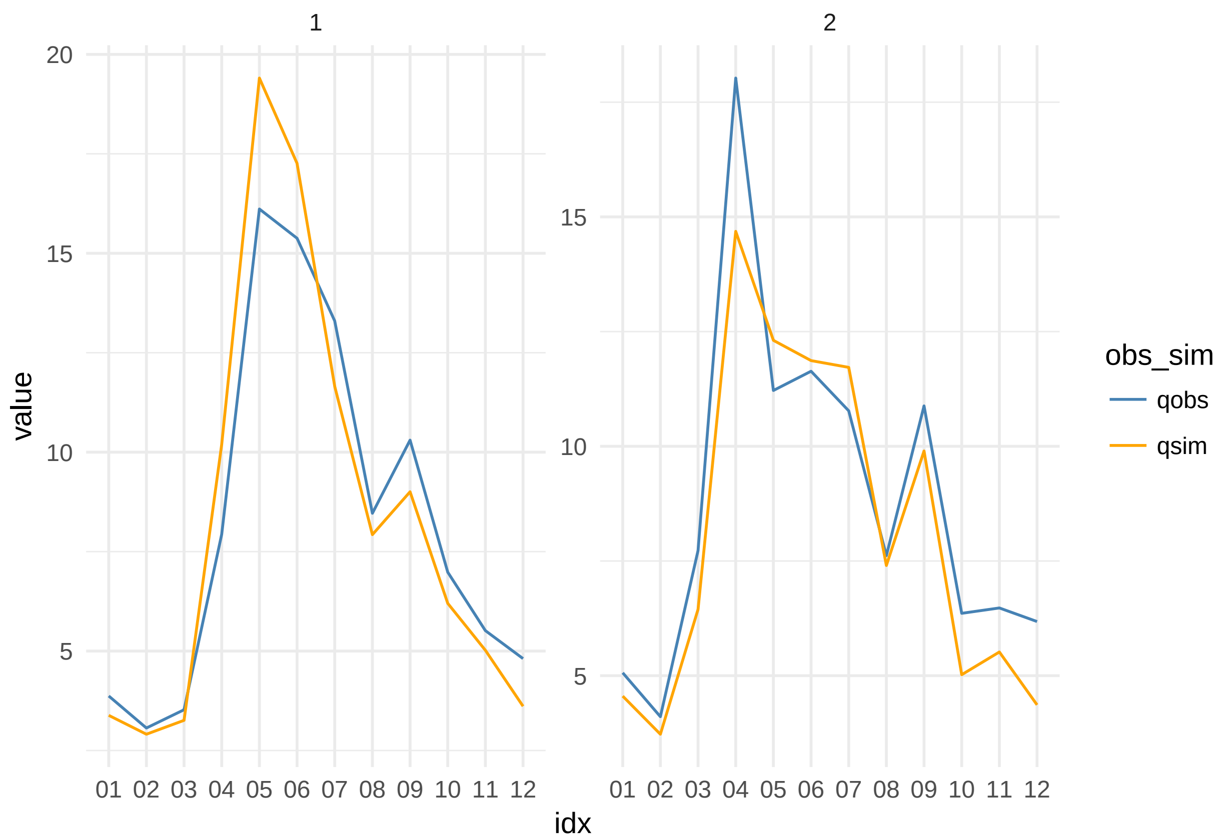
```
cos_data <- visCOS::get_viscos_example()
runoff_aggregate_dd <- aggregate_time(cos_data, "dd")
# plot data:
ggplot(runoff_aggregate_dd) +
    geom_line(aes(x = idx, y = value, col = obs_sim)) +
    scale_colour_manual(values = c(viscos_options("color_o"),
                                   viscos_options("color_s"))) +
    facet_wrap( ~ basin,ncol = 1) +
    theme_minimal()
```



Monthly runoff aggregation:

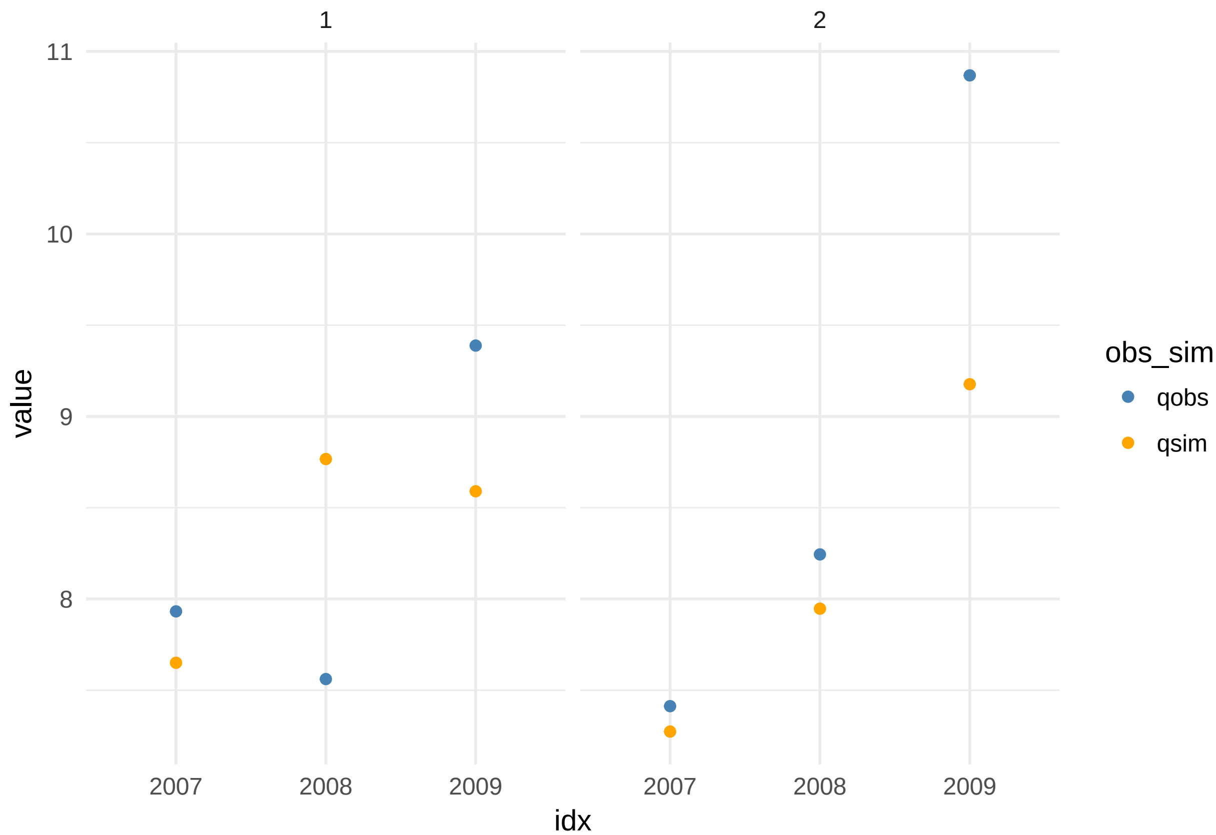
```
runoff_aggregate_mm <- aggregate_time(cos_data, "mm")
# plot data:
ggplot(runoff_aggregate_mm) +
    geom_line(aes(x = idx, y = value, col = obs_sim)) +
    scale_colour_manual(values = c(viscos_options("color_o"),
                                   viscos_options("color_s"))) +
    scale_x_discrete(limits = runoff_aggregate_mm$time_aggregate) +
    facet_wrap( ~ basin, scales = "free") +
```

```
    theme_minimal()
```



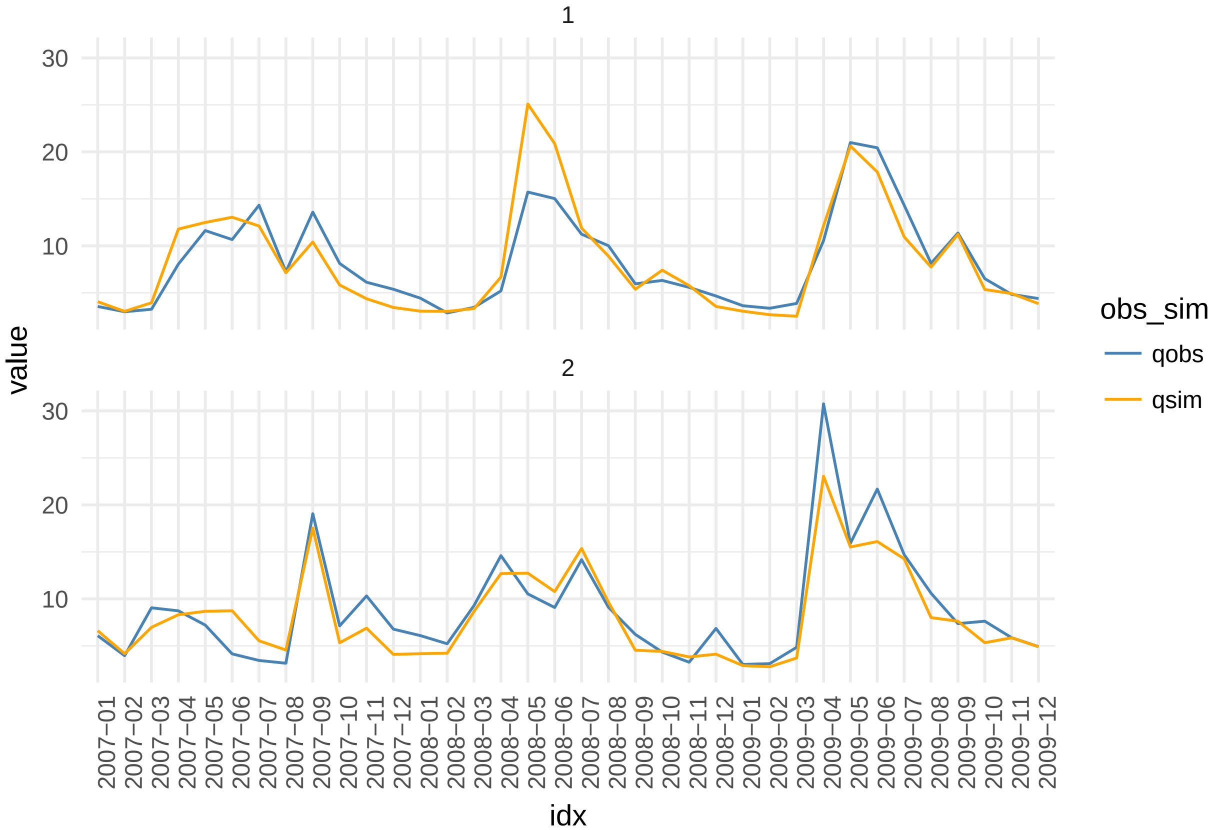
Yearly runoff aggregation:

```
runoff_aggregate_yyyy <- aggregate_time(cos_data, "yyyy")
# plot data:
ggplot(runoff_aggregate_yyyy) +
  geom_point(aes(x = idx, y = value, col = obs_sim)) +
  scale_colour_manual(values = c(viscos_options("color_o"),
                                 viscos_options("color_s"))) +
  facet_wrap( ~ basin) +
  scale_x_discrete(limits = runoff_aggregate_yyyy$time_aggregate,
                   labels = abbreviate) +
  theme_minimal()
```



Yearly and monthly runoff aggregation:

```
runoff_aggregate_yyyymm <- aggregate_time(cos_data, "yyyy-mm")
# plot data:
ggplot(runoff_aggregate_yyyymm) +
  geom_line(aes(x = idx, y = value, col = obs_sim)) +
  scale_colour_manual(values = c(viscos_options("color_o"),
                               viscos_options("color_s"))) +
  scale_x_discrete(limits = runoff_aggregate_yyyymm$time_aggregate,
                 labels = abbreviate) +
  facet_wrap( ~ basin,ncol = 1) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



4.2 Code

This section defines the code for the `aggregate_time` function. The time aggregation is done by cutting the needed information out of the date-string. This is a rough, but works nicely and seems to be more commonly used than you would expect.

```
#' Time Aggregation
#'
#' Aggregates the COSERO data.frame (\code{cos_data}) according to the
#' timely resolution defined via \code{aggregation}. Possible
#' resolution-choices are \code{'yyyy'} - year, \code{'mm'} - month and
#' \code{'dd}' - day and combinations thereof.
#'
#' @param cos_data the COSERO data.frame as used within visCOS
#' @param aggregation string that defines the resolution of the aggregation.
#' @import magrittr
#' @import ggplot2
#' @import pasta
#' @export
aggregate_time <- function(cos_data, aggregation = "mm") {
  #
  cutting_bounds <- c(Inf,-Inf)
  if (grepl("dd",aggregation)) {
    cutting_bounds[1] <- min(9,cutting_bounds[1])
    cutting_bounds[2] <- max(11,cutting_bounds[2])
```

```
  }
  if (grepl("mm",aggregation)) {
    cutting_bounds[1] <- min(6,cutting_bounds[1])
    cutting_bounds[2] <- max(7,cutting_bounds[2])
  }
  if (grepl("yyyy",aggregation)) {
    cutting_bounds[1] <- min(1,cutting_bounds[1])
    cutting_bounds[2] <- max(4,cutting_bounds[2])
  }
```

```
  ###### function and string definitions
  regex_for_cos_selection <- viscos_options("name_o") %|%  viscos_options("name_s")
  # aggregation function:
  aggregator_fun <- function(k,data_frame){
    the_aggregation <- aggregate(data_frame[[k]] ~ data_frame$date_selection, FUN = mean)
    return(the_aggregation[ ,2])
  }
  #####
  # If cos_data is not provided fully, the date is completed automatically
  # + junk is removed from the data frame
  full_cos_data <- cos_data %>%
    visCOS::complete_dates() %>%
    visCOS::remove_junk()
  # aggregate:
  cos_with_aggreggation <- cbind.data.frame(
    full_cos_data,
    date_selection = substr(full_cos_data$posixdate,
                            cutting_bounds[1],
                            cutting_bounds[2]) %>% as.factor()
    )
  names_cos_selection <- grep(
    regex_for_cos_selection,
    names(cos_with_aggreggation) %>% tolower,
    value = TRUE
    )
  selected_cos_rows <- grep(regex_for_cos_selection,
                           names(cos_with_aggreggation),
                           ignore.case = TRUE)
  time_aggregate <- selected_cos_rows %>%
    sapply(.,function(x) aggregator_fun(x,cos_with_aggreggation)) %>%
    data.frame(idx = 1:nrow(.),
               time_aggregate = unique(cos_with_aggreggation$date_selection),
               .) %>%
    set_names(., c("idx","time_aggregate",names_cos_selection))
  # melt the data in a tidy format:
  melted_time_aggregate <- time_aggregate %>%
    reshape2::melt(., id.vars = c("idx","time_aggregate")) %>%
    cbind.data.frame(.,
                     basin =  .$variable %>%
                       gsub(regex_for_cos_selection,"",.) %>%
                       gsub("\\D","",.) %>%
                       as.integer,
                     obs_sim = .$variable %>%
                       gsub(viscos_options("name_o") %&% ".*",viscos_options("name_o"),.) %>%
```

```r
                         gsub(viscos_options("name_s") %&% ".*",viscos_options("name_s"),.))
    return(melted_time_aggregate)
  }
```

5 Summaries of important objective functions

This chapter explains the code to calculate the main objective functions of used in visCOS. As explained in respective section objective functions are a pivotal part of model calibration. As of now, **visCOS** focuses on 4 main objective function: NSE, KGE, Pearson's Correlation and the Percentage bias (the respective definitions are given here).

The main objective functions for the overall data and the marked periods can be computed through the function `main_of_compute`. In order to run the function the period have to be marked first, e.g. through the `mark_periods` function. Additionally, **visCOS** already provides two different options to create plots for the main objective functions: `main_of_rasterplot` and `main_of_barplot`. Both functions create a list with 4 `ggplot` figures. Each entry in the list corresponds to one of the main objective functions and both lists can be saved to html embedded .jpgs with the `serve` function.

Examples:

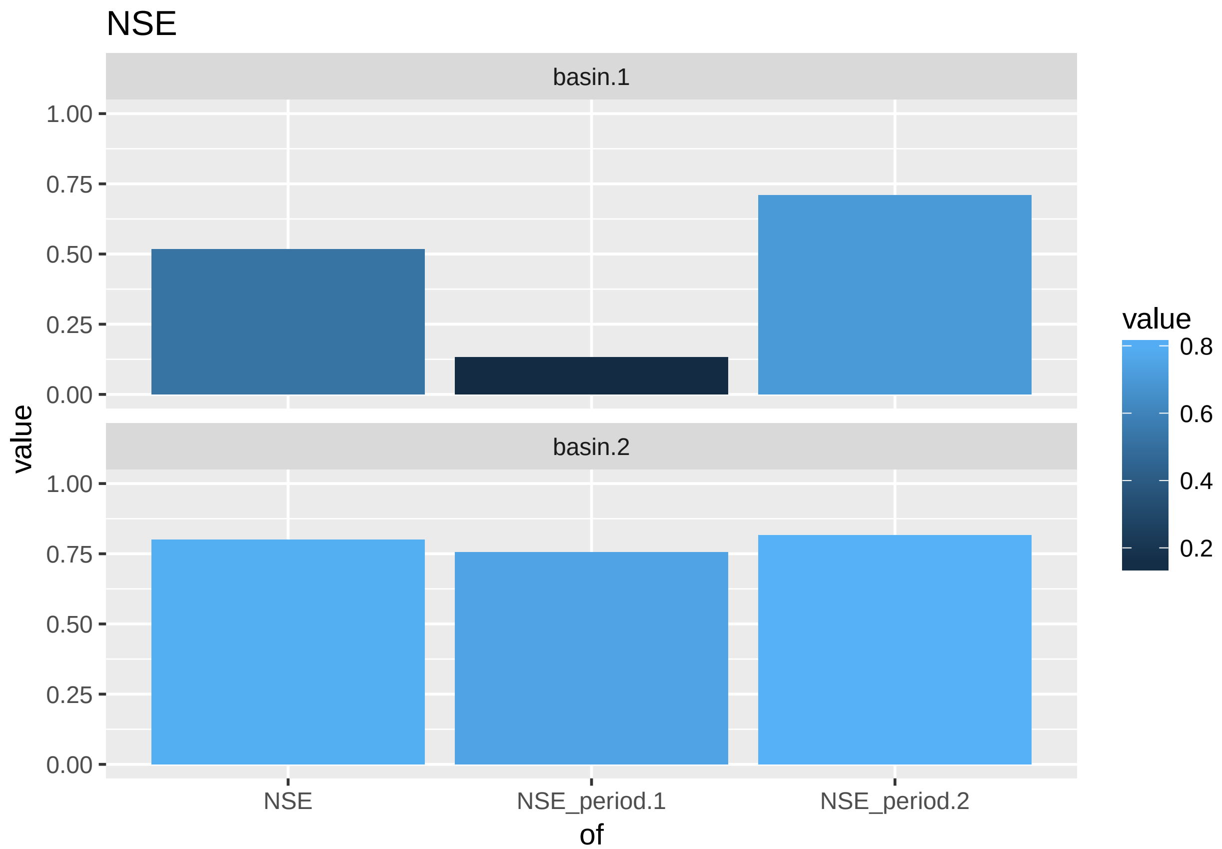
Computing the main objective functions

```r
require(visCOS)
require(magrittr)
of_values <- get_viscos_example() %>%
  mark_periods() %>%
  main_of_compute()
of_values
```

```
##                  of    basin.1     basin.2
## 1               NSE  0.5177309   0.8012919
## 2               KGE  0.7246877   0.7489862
## 3            p_bias  0.3000000 -12.5000000
## 4              CORR  0.8177663   0.9107391
## 5      NSE_period.1  0.1326604   0.7566200
## 6      NSE_period.2  0.7096110   0.8168895
## 7      KGE_period.1  0.4138710   0.8487406
## 8      KGE_period.2  0.8419666   0.6820128
## 9   p_bias_period.1  8.8000000  -8.6000000
## 10 p_bias_period.2 -6.9000000 -15.9000000
## 11    CORR_period.1  0.8243198   0.8806770
## 12    CORR_period.2  0.8577983   0.9385221
```

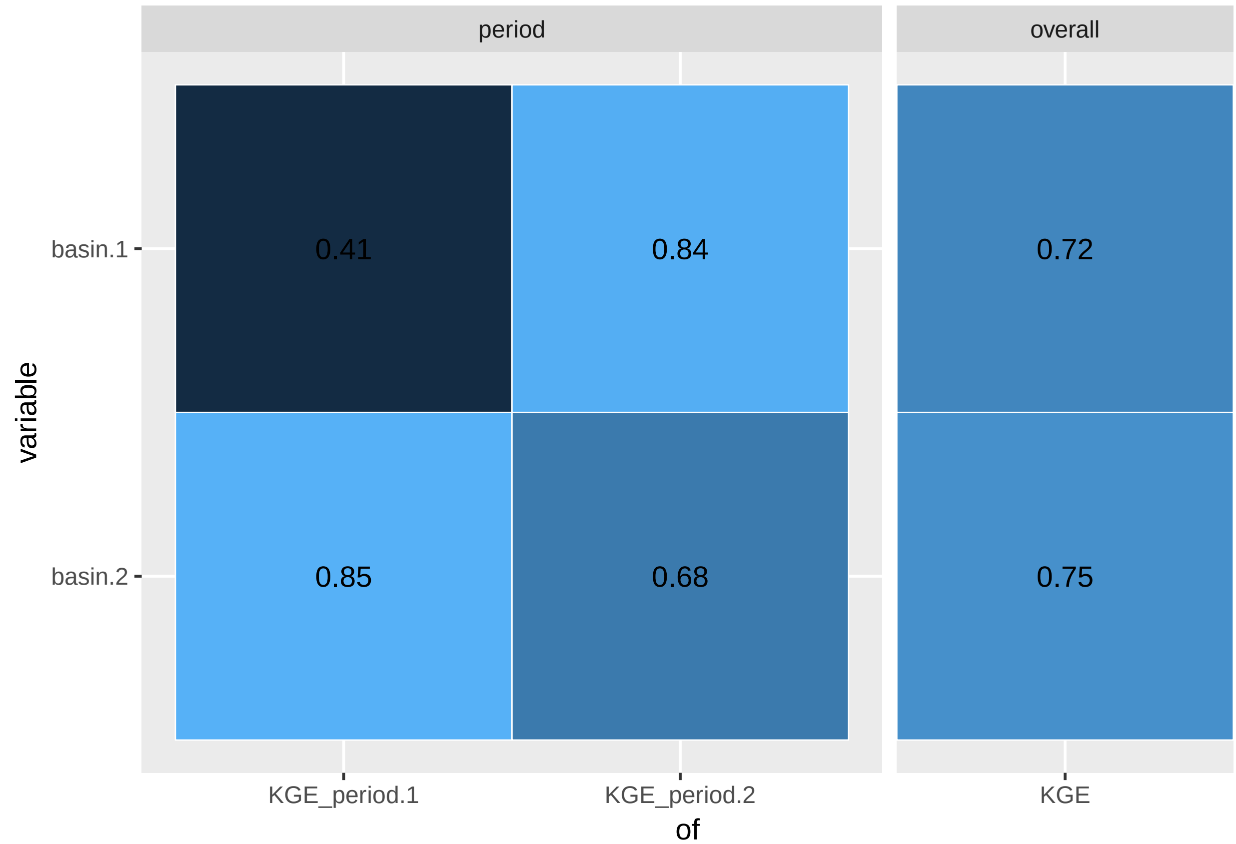
Plotting the results of main objective function with bar plots:

```r
of_values <- get_viscos_example() %>%
  mark_periods() %>%
  main_of_barplot() %>%
  extract2(1) %>%
  plot()
```



Plotting the results of main objective function with a raster:

```
of_values <- get_viscos_example() %>%
  mark_periods() %>%
  main_of_rasterplot() %>%
  extract2(2) %>%
  plot()
```



5.1 Code

This section defines the code for the `main_of`-function-family.

```
# ----------------------------------------------------------------------------
# Code for the Main Objective Functions (main_of)
# authors: Daniel Klotz, Johannes Wesemann, Mathew Herrnegger
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

5.1.1 Compute the Main Objective Functions

The purpose of this function is to extract a the main of objective functions from the `COSERO` data.frame. The objective functions are extracted for each basin separately and computed for the entire length of the data, as well as for each period separately.

The computational part of the function works as follows. In step (I) the non-marked periods of `cos_data` (columns of `viscos_options("name_COSperiod")` which are smaller then 0) are excluded from further calculations. The thereby obtained data.frame is named `evaluation_data`.

```
# ----------------------------------------------------------------------------
#' Get basic objective function for cos_data
#'
#' Calculate basic objective functions(NSE, KGE, percentage BIAS, Correlation)
#' for every basin and the chosen periods.
#'
#' @param cos_data cos_data data.frame.
```

```r
#' @return list of basic objective function evaluated for the different
#' hydrological years and over the whole timespan.
#'
#' @import hydroGOF
#' @import dplyr
#'
#' @export
main_of_compute <- function(cos_data) {
  # def: ==================================================================
  assert_dataframe(cos_data)
  name_o <- viscos_options("name_o")
  name_s <- viscos_options("name_s")
  name_period <- viscos_options("name_COSperiod")
  if (!exists(name_period, where = cos_data)) {
    stop("Error! Period-Column missing in cos_data, use `mark_periods`")
  }
  evaluation_data <- cos_data[cos_data[[name_period]] > 0, ]
  number_of_basins <- evaluation_data %>%
    names() %>%
    unique() %>%
    grepl(name_o, ., ignore.case = TRUE) %>%
    sum()
  data_periods <- evaluation_data %>%
    magrittr::extract2(name_period) %>%
    unique()
  number_of_periods <- data_periods %>% length
  # compute main-of for entire data: ========================================
  o_pick <- dplyr::select(evaluation_data,starts_with(name_o)) %>% unname()
  s_pick <- dplyr::select(evaluation_data,starts_with(name_s)) %>% unname()
  nse_all <- hydroGOF::NSE(s_pick,o_pick)
  kge_all <- hydroGOF::KGE(s_pick,o_pick)
  p_bias_all <- hydroGOF::pbias(s_pick,o_pick)
  corr_all <- cor(s_pick,o_pick) %>% diag()
  # compute periodwise main-of: =============================================
  # pre allocations: #######################################################
  NSE_period <- matrix(nrow = number_of_periods,
                       ncol = as.integer(number_of_basins),
                       data = NA)
  KGE_period <- NSE_period
  p_bias_period <- NSE_period
  CORR_period <- NSE_period
  # calculation loop, proabbly slow :( ######################################
  for (k in 1:number_of_periods) {
    o_pick <- dplyr::filter(evaluation_data,period == data_periods[k]) %>%
      dplyr::select(starts_with(name_o)) %>%
      unname()
    s_pick <- dplyr::filter(evaluation_data,period == data_periods[k]) %>%
      dplyr::select(starts_with(name_s)) %>%
      unname()
    NSE_period[k,1:number_of_basins] <- hydroGOF::NSE(s_pick,o_pick)
    KGE_period[k,1:number_of_basins] <- hydroGOF::KGE(s_pick,o_pick)
    p_bias_period[k,1:number_of_basins] <- hydroGOF::pbias(s_pick,o_pick)
    CORR_period[k,1:number_of_basins] <- cor(s_pick,o_pick) %>% diag(.)
```

```
  }
  # clean up: ==============================================================
  obj_names <- c("NSE","KGE","p_bias","CORR",
                 paste("NSE_period",1:number_of_periods,sep = "."),
                 paste("KGE_period",1:number_of_periods,sep = "."),
                 paste("p_bias_period",1:number_of_periods,sep = "."),
                 paste("CORR_period",1:number_of_periods,sep = ".")
                 )
  obj_fun <- data.frame(of = obj_names,
                        basin = rbind(nse_all,
                                      kge_all,
                                      p_bias_all,
                                      corr_all,
                                      NSE_period,
                                      KGE_period,
                                      p_bias_period,
                                      CORR_period),
                        row.names = NULL)
  return(obj_fun)
}
```

5.1.2 Plotting

```
# --------------------------------------------------------------------------
#' Plot main objective function values
#'
#' Currently two options for plotting the main objectives are provided by
#' visCOS: Plotting the different objective functions values as a set of
#' bar plots \code{barplot_of} and plotting a summary table in form of
#' a large raster of all the objective function values \code{rasterplot_of}.
#'
#' @name plot_main_of
NULL
```

5.1.2.1 Bar Plot

```
# --------------------------------------------------------------------------
#' Bar plot for the Main Objective Function Values
#'
#' @rdname of_overview
#' @export
main_of_barplot <- function(cos_data) {
  # def: ===================================================================
  assert_dataframe(cos_data)
  # functions: =============================================================
  assign_ofgroups <- function(of_melted,mof_names) {
    of_string <- as.character(of_melted$of)
    of_melted$of_group <- of_string %>%
      replace(.,startsWith(of_string,mof_names[1]),mof_names[1]) %>%
      replace(.,startsWith(of_string,mof_names[2]),mof_names[2]) %>%
      replace(.,startsWith(of_string,mof_names[3]),mof_names[3]) %>%
      replace(.,startsWith(of_string,mof_names[4]),mof_names[4])
```

```
    return(of_melted)
  }
  # plot-list function:
  barplot_fun <- function(of_name,of_melted) {
    of_to_plot <- of_melted %>% filter( of_group == of_name)
    if (of_name == "p_bias") {
      gglimits <- c(-viscos_options("of_limits")[2]*100,
                     viscos_options("of_limits")[2]*100)
    } else {
      gglimits <- viscos_options("of_limits")
    }
    plt_out <- ggplot(data = of_to_plot) +
      geom_bar(stat = "identity",
               position = "identity",
               aes(x = of, y = value, fill = value)) +
      facet_wrap(~ variable, ncol = 1) +
      ggtitle(of_name) +
      ylim(gglimits)
    return(plt_out)
  }
  # computations: ============================================================
  mof_names <- c("NSE","KGE","CORR","p_bias")
  of <- main_of_compute(cos_data)
  num_basins <- ncol(of) - 1
  of_melted <- suppressMessages( reshape2::melt(of) ) %>%
    assign_ofgroups(.,mof_names)
  # plotting =================================================================
  plot_list <- lapply(mof_names, function(x) barplot_fun(x,of_melted)) %>%
    magrittr::set_names(mof_names)
  return(plot_list)
}
```

5.1.2.2 Raster-Plot

```
#' Bar plot for the Main Objective Function Values
#'
#' @rdname of_overview
#' @import pasta
#' @export
main_of_rasterplot <- function(cos_data) {
  mof_names <- c("NSE","KGE","CORR","p_bias")
  regex_main_of <- mof_names %.% "*"
  assert_dataframe(cos_data)
  of <- main_of_compute(cos_data)
  #
  plot_list <- lapply(regex_main_of,function(x) plot_fun_raster(x,of)) %>%
    set_names(mof_names)
  return(plot_list)
}

# plot function ------------------------------------------------------------
plot_fun_raster <- function(regex_single_of,of) {
  # function definitions ===================================================
```

```
extract_single_of <- function(of){
  idx <- grep(regex_single_of,of$of)
  return(of[idx, ])
}
add_facet_info <- function(of) {
  facet_column <- nrow(of) %>%
    magrittr::subtract(1) %>%
    rep("period",.) %>%
    c("overall",.)
  return( cbind(of,facets = facet_column) )
}
reverse_basin_levels <- function(prepared_data) {
  prepared_data$variable <- factor(prepared_data$variable,
                                   levels = prepared_data$variable %>%
                                     levels() %>%
                                     rev()
                                   )
  return(prepared_data)
}
reverse_facetting_levels <- function(prepared_data) {
  prepared_data$facets <- factor(prepared_data$facets,
                                 levels = prepared_data$facets %>%
                                   levels() %>%
                                   rev()
                                 )
  return(prepared_data)
}
bind_and_round_value <- function(of,gglimits,digits) {
  dplyr::mutate(of,
                value = pmax(value,gglimits[1]) %>%
                  pmin(.,gglimits[2]) %>%
                  round(.,digits)
                )
}
# computation ============================================================
if (regex_single_of == "p_bias.*") {
  # pbias has different limits :(
  gglimits <- c(-viscos_options("of_limits")[2]*100,
                viscos_options("of_limits")[2]*100)
} else {
  gglimits <- viscos_options("of_limits")
}
#
prepared_data <- of %>%
  extract_single_of() %>%
  add_facet_info() %>%
  reshape2::melt(., id.vars = c("of","facets")) %>%
  reverse_basin_levels() %>%
  reverse_facetting_levels() %>%
  bind_and_round_value(.,gglimits,2)
# ggplot =================================================================
plt_out <- ggplot(prepared_data,
                  aes(of,variable, fill = value),
```

```
                    environmnet = environment()) +
    geom_raster(position = "identity") +
    coord_fixed(ratio = 5)  +
    facet_grid(~ facets,  scales = "free_x", space = "free") +
    theme( legend.position = "none")  +
    geom_tile(color = "white", size = 0.25 ) +
    geom_text(aes(of,variable, label = as.character(value,2)),
              color = "black")
  return(plt_out)
}
```

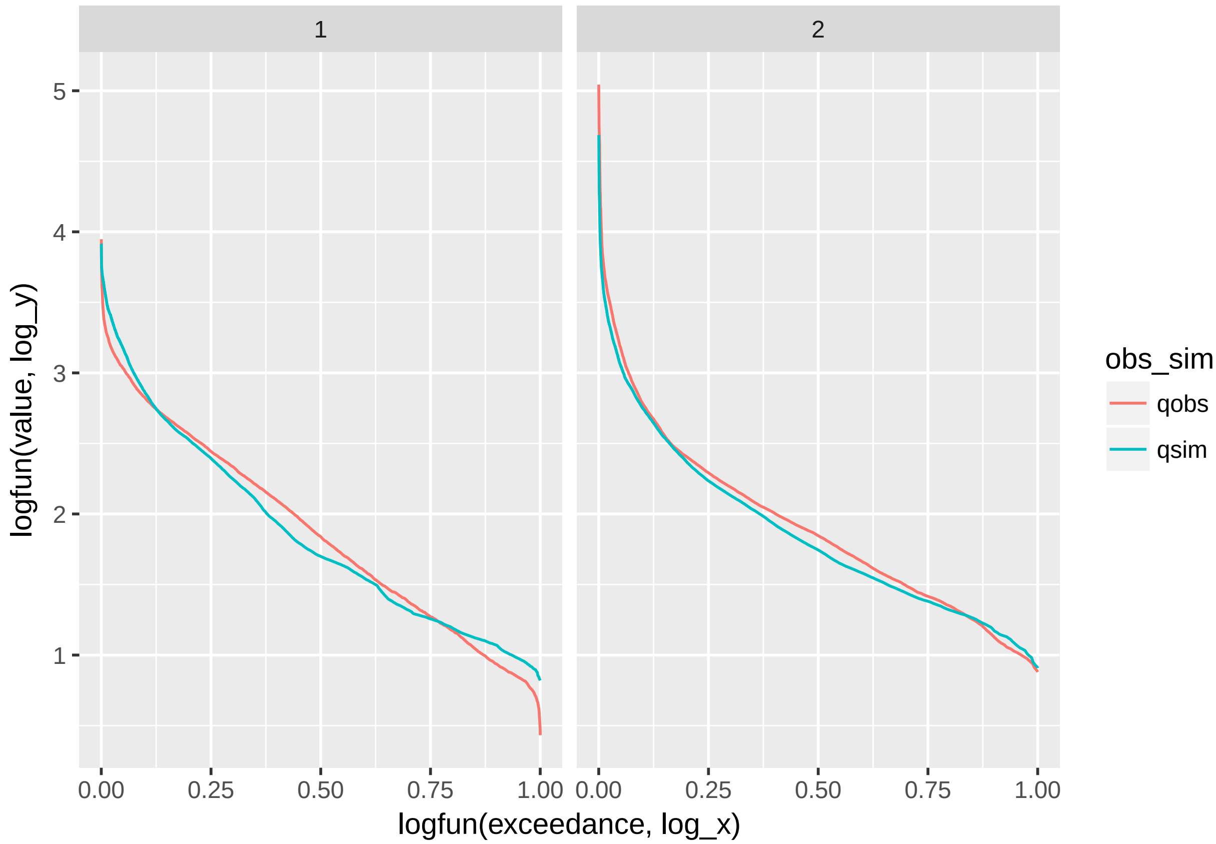
6 Flow Duration Curves

Flow-duration curves represent the relationship between the magnitue and the frequency of a streamflow. They provide an estimate of the percentage of time a given streamflow was exceeded within the evaluated time frame. Foster [1934] attributes the first use of flow duration curves to Clemens Herschel around 1880. Since they have been used for a wide array of applications. **visCOS** provides a function to compute the data for flow-duration curves and a function to plot them directly. The former function is called `fdc_compute`. It computes the flow exceedance properties and returns a data.frame. The calculations are adapted from the method used within the hydroTSM package. It is currently rather slow. The plot-function is called `fdc_plot`. Internally it uses `fdc_compute` for the data preparation and generates a facetted `ggplot` object from it. In the plot each basin is a facet and each sub-plot shows the *o*-data and *s*-data (see: Introduction).

6.1 Example

Flow duration curves can be plotted in **visCOS** in the following way:

```
  library(visCOS)
  cos_data <- get_viscos_example()
  fdc_plot(cos_data)
```



6.2 Code

6.2.1 Computing the data for the flow duration curves

```
#' Compute Flow Duration Curves
#'
#' Computes the flow duration curves (fdc) for the `cos_data` data.frame.
#' The calculations are adapted from the method used within the hydroTSM package.
#' @param cos_data A data.frame with columns as used throughout visCOS
#' @import magrittr
#' @import dplyr
#' @importFrom purrr map_df
#' @import pasta
#' @export
fdc_compute <- function(cos_data) {
  # defensive code:
  assert_dataframe(cos_data)
  # def:
  order_bound_data <- function(bound_data) {
    ordred_fdc_data <- bound_data %>%
      mutate(obs_sim = key %>%
               gsub( viscos_options("name_o") %&% ".*",
                     viscos_options("name_o"),
                     .,
```

```
                     ignore.case = TRUE ) %>%
               gsub( viscos_options("name_s") %&% ".*",
                     viscos_options("name_s"),
                     .,
                     ignore.case = TRUE ),
             basin_idx = key %>%
               gsub(viscos_options("name_o"),"",.,ignore.case = TRUE) %>%
               gsub(viscos_options("name_s"),"",.,ignore.case = TRUE) %>%
               gsub("\\D","",.) %>% as.numeric)
      return(ordred_fdc_data)
    }
    # computation:
    cos_data_only <- cos_data %>%
      select(starts_with(viscos_options("name_o")), starts_with(viscos_options("name_s")))
    exceedance_values <- map_df(cos_data_only,calc_percent_exceedance) %>%
                          tidyr::gather() %>%
                          magrittr::extract("value")
    fdc_data <- cos_data_only %>%
      tidyr::gather() %>%
      cbind.data.frame(exceedance = exceedance_values) %>%
      magrittr::set_names(c("key","value","exceedance")) %>%
      order_bound_data(.)
    return(fdc_data)
  }
```

```
# function to calculated the percent exceedance (x-axis) for the fdc
  calc_percent_exceedance <- function(q) {
    q_sorted <- sort(q)
    q_zero_index <- which(q_sorted == 0)
    nzeros <- length(q_zero_index)
    ind <- match(q, q_sorted)
    n <- length(q)
    percent_exeedence <- rep(NA, n)
    percent_exeedence[1:n] <- sapply(1:n, function(j, y) {percent_exeedence[j] <- length(which(y >= y[j
                         y = q)
    percent_exeedence <- percent_exeedence/n
    return(percent_exeedence)
  }
```

6.2.2 Plotting the Flow duration curves

```
  #' Plot Flow Duration Curves
  #'
  #' Plots the flow duration curves (fdc) for `cos_data`.
  #' The function uses `ggplot` to so and facets the different basins into
  #' separate subplots. Each subplot shows the fdc of the \eqn{o}-data and
  #' the \eqn{s}-data.
  #' @export
  #' @import ggplot2
  fdc_plot <- function(cos_data,
                       log_y = TRUE,
                       log_x = FALSE,
                       ...) {
```

```r
  # def:
  # maybe we have to account certain limits for the logs, e.g:
  # if (log_y | log_x & min(ylim) == 0) {
  #   ylim <- range(q, na.rm = TRUE)
  #   tmp <- unlist(q)
  #   tmp[which(tmp == 0)] <- NA
  #   ylim[1] <- min(tmp, na.rm = TRUE)
  # }
  logfun <- function(data,take_log){
    if(take_log){
      return(log(data))
    } else (
      return(data)
    )
  }
  # computation:
  fdc_data <- fdc_compute(cos_data)
  gplot <- ggplot(fdc_data) +
    geom_line(aes(x = logfun(exceedance,log_x), y = logfun(value,log_y), color = obs_sim)) +
    facet_wrap(~ basin_idx)
  return(gplot)
}
```

6.3 References

- Foster, H.A. (1934): Duration curves. ASCE Trans., 99, 1213-1267
- Vogel, R. M.; Fennessy, N. M. (1994): Flow-Duration Curves. I: New Interpretation and Confidence Intervals. JWRMD 120, No. 4

7 Plotting Runoff Peaks Plots

The function `peak_plot` lets users explore the highest events in among the available basins. It provides a list of ggplot2 plots, containing an overview plot (`overview`), a scatter plot (`scatter`) and detail plots of the individual events (`event_plot`). Instead of explaining the properties of each plot in detail it is best to get an intuition of the function by looking at some examples.

7.1 Examples:

For the examples 10 events are extracted from a runoff example

```r
require(visCOS)
cos_data <- get_viscos_example()
peakplots <- peak_plot(cos_data, n_events = 10L)
```

The `peakplots` list does now contain plots for each basin within the `cos_data` data.frame:

```r
names(peakplots)
```

```
## [1] "basin0001" "basin0002"
```

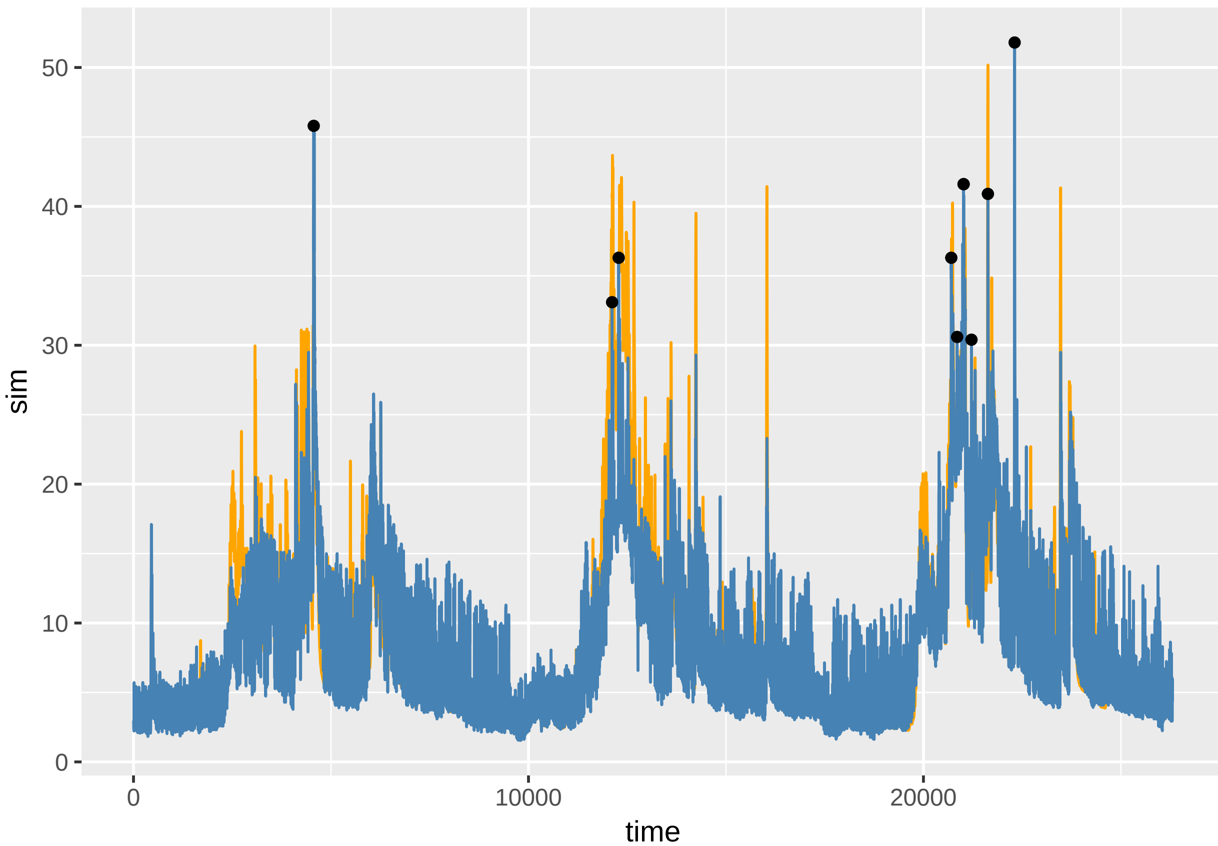
For each basin the a set of plots (`overview`,`scatter`,`event_plot`) are saved within a list for each basin. In the following the plots for basin 1 are shown:

```r
names(peakplots$basin0001)
```

```
##  [1] "overview"     "scatter"      "event_plot1"  "event_plot2"  "event_plot3"
##  [6] "event_plot4"  "event_plot5"  "event_plot6"  "event_plot7"  "event_plot8"
## [11] "event_plot9"  "event_plot10"
```

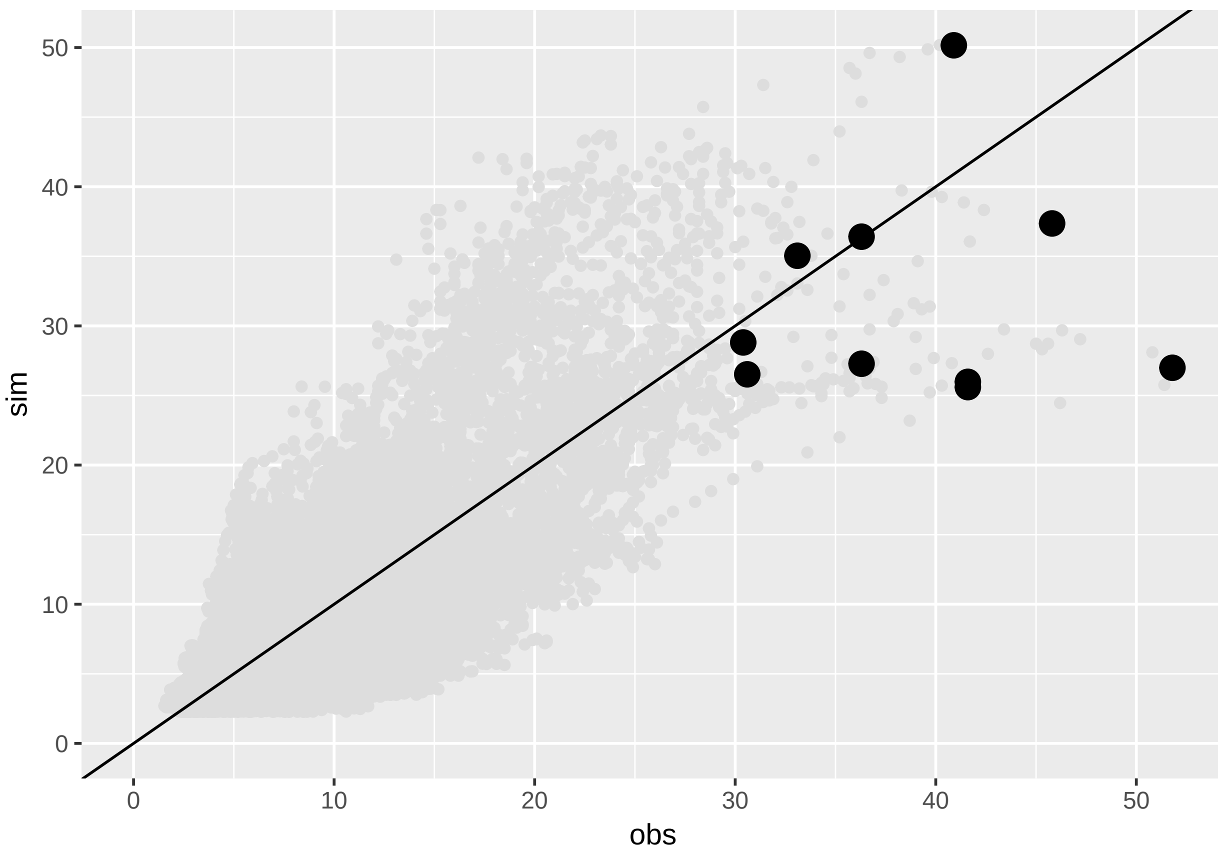
The `overview` plot shows the entire time series of `data1` and `data2` of the basin. The found events are marked with black dots. The `overview` plot for basin 1 is:

```r
peakplots$basin0001$overview
```



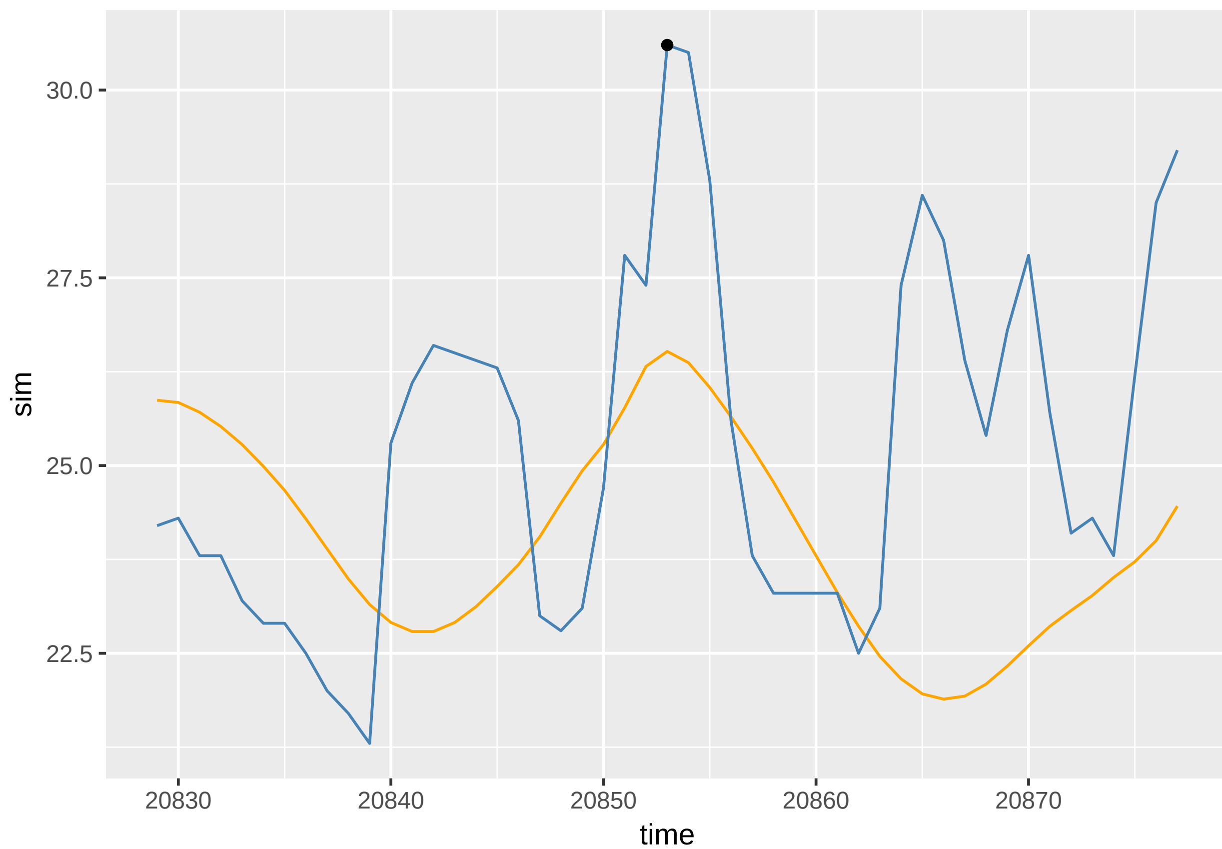
The `scatter` plot shows the found events within a scatter plot, where `data1` is the x-axis and `data2` on the y-axis. In the following an example for basin 1 is given.

```r
peakplots$basin0001$scatter
```



Detail plots for each of the found events are given in form of the `event_plot` objects. Here an example:

```
peakplots$basin0001$event_plot5
```



7.2 Code

This part of the document defines the code of `peak_plot`

```
#' Plot List for Runoff Peaks
#' @export
#'
#' @import ggplot2
#' @import dplyr
#' @import magrittr
#' @importFrom tibble tibble
peak_plot <- function(cos_data,n_events= 10L, window_size = 24L) {
  # pre:
  assert_dataframe(cos_data)
  n_events_int <- as.integer(n_events)
  window_size_int <- as.integer(window_size)
  if( is.na(n_events_int)  |
      is.nan(n_events_int) |
      is.infinite(n_events_int) |
      !is.integer(n_events_int) ) {
    stop("n_events is ill defined")
  }
  if( is.na(window_size)  |
      is.nan(window_size) |
      is.infinite(window_size) |
```

```
      !is.integer(window_size) ) {
    stop("window_size is ill defined")
  }
  data1 <- cos_data %>%
     select( starts_with(viscos_options("name_o")) )
  data2 <- cos_data %>%
     select( starts_with(viscos_options("name_s")) )
  data_numbers <- names(data1) %>%
    gsub(viscos_options("name_o"),"",.,ignore.case = TRUE) %>%
    gsub("\\D","",.,ignore.case = TRUE)
  # make plotlist:
  plotlist <- lapply(1:ncol(data1), function(x) plotlist_one_basin(data1[,x],
                                                                   data2[,x],
                                                                   n_events_int,
                                                                   window_size_int)) %>%
    set_names(.,paste("basin", data_numbers, sep =""))
  return(plotlist)
}
```

7.2.1 Generating the Plots for one basin.

This is the function for generating the different plots for one basin. At first the provided time series are grouped into a `tibble`, then the peaks of the observations are obtained via the peak_finder function and organised. Then `ggplot2` is used for plotting.

```
plotlist_one_basin <- function(qobs,qsim,n_events_int,window_size_int) {
  single_data <- tibble::tibble(time = as.integer(1:length(qobs)),
                                obs = as.double(qobs),
                                sim = as.double(qsim))
  #
  peak_idx <- find_peaks(single_data$obs,m = window_size_int)
  peak_organised <- tibble::tibble(idx = as.integer(peak_idx),
                           peak_obs = single_data$obs[peak_idx],
                           peak_sim = single_data$sim[peak_idx])
  highest_peaks_organised <- peak_organised$peak_obs %>%
    sort(decreasing = TRUE) %>%
    .[1:n_events_int] %>%
    '%in%'(peak_organised$peak_obs,.) %>%
    which( . ) %>%
    peak_organised[., ]
  #
  overview_plot <- ggplot() +
    geom_line(data = single_data,aes(x = time, y = sim), col = viscos_options("color_s")) +
    geom_line(data = single_data,aes(x = time, y = obs), col = viscos_options("color_o")) +
    geom_point(data = highest_peaks_organised, aes(idx, peak_obs))
  overview_scatter <- ggplot() +
    geom_point(data = single_data, aes(obs,sim), color = "#DDDDDD") +
    geom_abline() +
    geom_point(data = highest_peaks_organised, aes(peak_obs,peak_sim), size = 4) +
    expand_limits(x = 0, y = 0)
  sub_plots <- lapply(1:nrow(highest_peaks_organised),
                      function(x) sub_peakplot_fun(x,window_size_int,highest_peaks_organised,single_data
    set_names(.,paste("event_plot",1:length(.),sep=""))
```

```
  return(overview = append(list(overview = overview_plot,scatter = overview_scatter), sub_plots))
}
```

7.2.2 Function to find peaks

The function for finding the peaks was proposed and developed by the cross validated user "stas g" in this thread.
This is by far not the only option/possibility to approach the peak finding task. Other nice ideas for finding peaks can be found in this cross validated thread.

```
####
# peak finder function:
find_peaks <- function (x, m = 3){
  shape <- diff(sign(diff(x, na.pad = FALSE)))
  pks <- sapply(which(shape < 0), FUN = function(i){
    z <- i - m + 1
    z <- ifelse(z > 0, z, 1)
    w <- i + m + 1
    w <- ifelse(w < length(x), w, length(x))
    if(all(x[c(z : i, (i + 2) : w)] <= x[i + 1])) return(i + 1) else return(numeric(0))
  })
  pks <- unlist(pks)
  pks
}
```

7.2.3 Subplot Function

This function is a wrapper around `ggplot`, which is used to generate the individual event plots.

```
####
# sub plot function:
sub_peakplot_fun <- function(x, window_size, highest_peaks_organised, peak_data) {
  point <- highest_peaks_organised[x,]
  plot_sub <- ggplot() +
    geom_line(data = peak_data[(point$idx - window_size):(point$idx + window_size),],
              aes(x = time, y = sim),
              col = "orange") +
    geom_line(data = peak_data[(point$idx - window_size):(point$idx + window_size),],
              aes(x = time, y = obs),
              col = "steelblue") +
    geom_point(data = point, aes(idx, peak_obs))
  return(plot_sub)
}
```

7.3 References

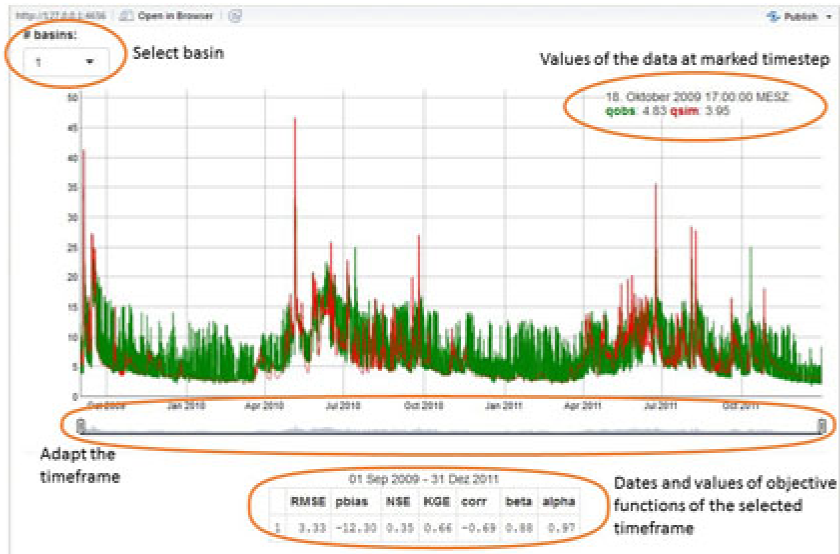
- http://stats.stackexchange.com/questions/22974/how-to-find-local-peaks-valleys-in-a-series-of-data (checked 12/2016)
- http://stats.stackexchange.com/questions/36309/how-do-i-find-peaks-in-a-dataset (checked 12/2016)

8 Exploring Objective Functions

This section defines the code of a shiny gadget. It enables the interactive exploration of (hydro-) graphs for the different basins. The gadget shows always the corresponding objective function for the selected graph. Furthermore, one can get the selected data by clicking on "done" at the end of a session. The following examples provide a good overview of what the function can do.

8.1 Example

This chapter gives examples of `explore_cos_data`. For the pre-requirements take a look at the introduction. Running the `explore_cos_data` function without any options opens a shiny gadget in the viewer:



Information on the *objective functions* can be found here.

```
viscos_options(color_o = "green", color_s = "red")
explore_cos_data(runoff_example)
```

Users can select different basins via the selection box (*# basins:*) on the top-left and interactively zoom and move the graph in the center by clicking on it or moving the date switches below the graph. While doing so the objective functions (presented in the table below) are re-calculated for the chosen time window.

8.2 Code

In the following paragraphs the code of the shiny app is defined. The computations of the app are defined in the `server` part and the appearance in the `ui`.

8.2.1 Explore the data

This function represents the main part of the shiny app. The current solution **forces** users to enumerate their basins with and the shiny app needs some pre-calculation, which need to be calculated before the app is started. These calculations are made before the app as such is defined. They include:

- (I) Defensive code
- (II) Transform data into `xts` (`d_xts`).
- (III) Save numeration of basins within the variable `d_nums`.

```r
  #' explore cos_data with Objective Functions
  #'
  #' Runs a Shiny App which can be used to get an overview of a cos_data time
  #' series object.
  #'
  #' @param d_xts cos_data formatted as time series
  #'
  #' @import shiny
  #' @import miniUI
  #' @importFrom xts xts
  #' @import dplyr
  #' @import magrittr
  #' @import dygraphs
  #' @import hydroGOF
  #' @import pasta
  #' @importFrom purrr map_df
  #'
  #' @export
  #'
  #' @examples
  #' # get example data,
  #' # explore the model performance
  #' cos_data <- get_viscos_example()
  #' explore_cos_data(cos_data)
explore_cos_data <- function(cos_data,
                                   of_list = list(
                                     nse = of_nse,
                                     kge = of_kge,
                                     p_bias = of_p_bias,
                                     r = of_cor
                                   ),
                                   start_date = NULL,
                                   end_date = NULL) {
  # pre-sets
  # (I) Defense
  if (is.null(names(of_list))){
    names(of_list) <- paste("of", 1:length(of_list), sep = "_")
  }
  clean_cos_data <- cos_data %>% remove_leading_zeros
  if ( !viscos_options("name_COSposix") %in% names(clean_cos_data) ) {
    clean_cos_data %<>% complete_dates
  }
  # (II)
  d_xts <- cos_data_as_xts(clean_cos_data)
  # (III)
  idx_names <- names(d_xts) %>%
    tolower %>%
    grepl(viscos_options("name_o"),.)
  d_nums <- d_xts %>%
      names() %>%
      .[idx_names] %>%
      gsub("\\D","",.) %>%
      as.integer %>%
```

```
    unique
```

The server-side of the shiny app is rather lengthy but not too difficult. The major readability problems occur because of the unusual formatting enforced by shinyApps.

```
server <- function(input, output, session) {
  # (I) get strings used in the naming of clean_cos_data:
  unique_data_names <- names(clean_cos_data) %>%
    gsub("\\d","",.) %>%
    tolower %>%
    unique
  x_string <- unique_data_names[ unique_data_names %>%
                                   grep(viscos_options("name_o"),.) ]
  y_string <- unique_data_names[ unique_data_names  %>%
                                   grep(viscos_options("name_s"),.) ]
  # (II) select data:
  selector_x <- reactive({ x_string %&% input$basin_num %&% "$" }) # "$" terminates the searchstring;
  selector_y <- reactive({ y_string %&% input$basin_num %&% "$" })
  selected_data <- reactive({
    select(clean_cos_data,
           matches( selector_x() ),
           matches( selector_y() )
           ) %>%
      select(x = matches( selector_x() ),
             y = matches( selector_y() ))
  })
  # (III) create xts-formated table for use in dygraphs:
  xts_selected_data <- reactive ({
    xts(selected_data(),
        order.by = clean_cos_data[[viscos_options("name_COSposix")]])
  })
  # (IV) create plots:
  output$hydrographs <- renderDygraph({
    dygraph( xts_selected_data() ) %>%
      dyAxis("y",
             label = visCOS::viscos_options("data_unit")) %>%
      dySeries("x",
               label = visCOS::viscos_options("name_o"),
               color = viscos_options("color_o")) %>%
      dySeries("y",
               label = visCOS::viscos_options("name_s"),
               color = viscos_options("color_s")) %>%
      dyRangeSelector(height = 20, strokeColor = "") %>%
      dyCrosshair(direction = "vertical") %>%
       dyOptions(includeZero = TRUE, retainDateWindow = TRUE)
  })
  # (IV) get dygraph date bounds (switches):
  selcted_from <- reactive({
    if (!is.null(start_date)) {
      start_date
    } else if (!is.null(input$hydrographs_date_window)) {
      input$hydrographs_date_window[[1]]
    }
  })
```

```
  selcted_to <- reactive({
    if (!is.null(end_date)) {
      end_date
    } else if (!is.null(input$hydrographs_date_window)) {
      input$hydrographs_date_window[[2]]
    }

  })
  # (V) extract time_window for the stats header:
  output$selected_timewindow <- renderText({
    if (!is.null(input$hydrographs_date_window))
      paste(strftime(selcted_from(), format = "%d %b %Y"),
            "-",
            strftime(selcted_to(), format = "%d %b %Y"),
            sep = " ")
  })
  # (VI) calculate stats:
  sub_slctd <- reactive({
    if (!is.null(input$hydrographs_date_window))
      xts_selected_data()[paste(strftime(selcted_from(), format = "%Y-%m-%d-%H-%M"),
                                strftime(selcted_to(), format = "%Y-%m-%d-%H-%M"),
                                sep = "/")]
  })
  out_of <- reactive({
    if (!is.null(input$hydrographs_date_window)) {
        map_df(of_list, function(of_,x,y) of_(x,y),
               x = sub_slctd()$x,
               y = sub_slctd()$y ) #serve_of( sub_slctd()$x,sub_slctd()$y )
    }
  })

  output$slctd_OF <- renderTable(out_of())
  # (VII) exit when user clicks on done
   # When the Done button is clicked, return a value
  observeEvent(input$done, {
    returnValue <- list(
      selected_time = c(strftime(selcted_from(), format = "%Y-%m-%d-%H-%M"),strftime(selcted_to(), fo
      selected_data = data.frame(date = index(sub_slctd()),
                                 coredata(sub_slctd())),
      selected_of = out_of()
    )
    stopApp(returnValue)
  })
}
```

The `miniUI` is quite spartan. There is an `miniButtonBlock` that allows to select different basin, as as the `dygraph` output (i.e `hydrographs`) for the interactive exploration of the *o* and *s* data. The formatted table (`slctd_OF`) displays the different objective functions, that can be given to `explore_cos_data` .

```
ui <- miniPage(
  miniButtonBlock(selectInput("basin_num",
                              "# basin:",
                              choices = d_nums,
                              selected = 1,
```

```
                                selectize = FALSE)),
    miniContentPanel(
      fillCol(
        flex = c(4,1),
        dygraphOutput("hydrographs", width = "100%", height = "100%"),
        fillCol(
          align = "center",
          textOutput("selected_timewindow"),
          tableOutput("slctd_OF")
        )
      )
    ),
    gadgetTitleBar("test")
  )
```

```
dyCrosshair <- function(dygraph,
                        direction = c("both", "horizontal", "vertical")) {
  dyPlugin(
    dygraph = dygraph,
    name = "Crosshair",
    path = system.file("examples/plugins/crosshair.js",
                       package = "dygraphs"),
    options = list(direction = match.arg(direction))
  )
}
```

```
  runGadget(ui,server)
}
```

9 Generate Previews

Possibility to save lists of plots into .jpgs and create a linked html file.

9.1 Code

```
#' Serve is still beta
#'
#' More description shall follow
#' @export
serve <- function(plotlist, path = "", fig_width = 800L, fig_height = 500L) {
  hmtl_filename <- "summary"
  # establish html-file in chosen folder
  '%&%' <- function(a,b) paste(a,b,sep = "") # helper for easier string concatenation
  fileConn <- file(path %&% hmtl_filename %&% ".html" , "w")
  # write html header
  #writeLines(text = '<!DOCTYPE html>',fileConn)
  writeLines(text = "<HEAD>",fileConn)
  writeLines(text = "  <STYLE type='text/css'>",fileConn)
  writeLines(text = "    H1 { text-align: center}",fileConn)
  writeLines(text = "  </STYLE>",fileConn)
 writeLines(text = "</HEAD>",fileConn)
```

```
  #writeLines(text = '<html>',fileConn)
  writeLines(text = '<body>',fileConn)
  # check which kind of plotlsit we are dealing with:
  if ( all(names(plotlist) == c("NSE","KGE","p_bias","CORR")) ) {
    list_to_plot <- plotlist

  } else if ( all(grepl("basin",names(plotlist1))) ) {
    list_to_plot <- unlist(plotlist,recursive = FALSE)
  } else {
    stop("plotlist not known!")
  }
  num_plots <- length(list_to_plot)
  figure_names <- names(list_to_plot)
  ## save everything localy & link it within the html file
  jpg_filenames <- "figure"

  for (i in 1:num_plots) {
    writeLines(text = "<H1>" %&% figure_names[i] %&% "</H1>",fileConn)
    plt_name <- jpg_filenames %&% i %&% ".jpg"
    plt_pathANDname <- path %&% plt_name
    plt_hmtlInfos <- "<img src=\"" %&% plt_name %&% '" alt="plotting_failed" style="width:800px;height:5
    #
    writeLines(text = "<H1>" %&% plt_hmtlInfos  %&% "</H1>", fileConn)
    jpeg(file = plt_pathANDname, width = fig_width, height = fig_height, units = "px")
      plot(list_to_plot[[i]])
    dev.off()
  }
  close(fileConn)
}
```

10 Defensive Code

This section defines the internaly used defensive programming part of **visCOS**. This are all functions dedicated to ensure that **visCOS** is working nicely even if it is used wrongly for whatever reason. This often means that a given function has to return an error if certain criteria are not met! Useful examples for such methods can be found in Hadley Wickhams `R` package `assertthat`

10.1 Code

The subsequent functions are defined in this section

function	exported
`assert_junk`	no
`assert_complete_date`	no
`assert_dataframe`	no
`assert_of`	no

10.1.1 Check if `cos_data` is "clean"

Tests if the given data.frame (`cos_data`) contains only the columns defined within `get_regex_for_cos_data` (see: helpers and `viscos_options`). An exception throws an error. **Note** that the check is not case sensitive!

```
#' @import magrittr
assert_junk <- function(cos_data) {
  regEx <- get_regex_for_cos_data()
  assertChunk <- names(cos_data) %>% grepl(regEx, ., ignore.case = TRUE)
  if (any(assertChunk == FALSE)) {
    stop("there is still unwanted columns in the data. Try: remove_junk")
  }
}
```

10.1.2 Check date completeness

A rough check for the needed date- and/or time-columns within the provided `cos_data`. The function is rahter basic. It only checks if the names of the `viscos_options("name_COSyear")` column and `viscos_options("name_COSposix")` column exist (see: `viscos_options`). An exceptions throws an error.

```
assert_complete_date <- function(cos_data) {
  OK_COSdate <- any(names(cos_data) == viscos_options("name_COSyear"))
  OK_POSIXdates <- any(names(cos_data) == viscos_options("name_COSposix"))
  # choose error messag depending on which columns are missing!
  if (!OK_COSdate & !OK_POSIXdates) {
    stop("No COSdates and no POSIXct-dates in the data!")
  } else if (OK_COSdate & !OK_POSIXdates) {
    stop("NO POSIXct fomrated column within the cos_data!")
  } else if (!OK_COSdate & OK_POSIXdates) {
    stop("NO COSdate year within the cos_data!")
  }
}
```

10.1.3 Check if the data is a `data.frame`

Tests if `data` is a data.frame and returns an error if not.

```
# uses stop if the input: "data" is not of class "data.frame"
assert_dataframe <- function(data) {
  require("tibble", quietly = TRUE)
  if ( !is.data.frame(data)&!is.tibble(data) ) stop("data needs to be a data_frame!")
}
```

10.2 References

- Wickham, H. (2013). assertthat: Easy pre and post assertions. https://CRAN.R-project.org/package=assertthat

11 Helpers

This section collects small and helpful functions/scripts that are used throughout **visCOS**.

None of the functions is exported!

11.1 Code

The subsequent functions are defined below

function	exported
`get_regex_for_cos_data`	no
`get_basin_numbers`	no
`set_panel_size`	no

11.1.1 Get a Regular Expression String for the needed columns

This function can be called to get the names of the 8 allowed column-names within **visCOS**. `get_regex_for_cos_data` takes no input, as it gets the information directly from the global options (see: *viscos_options section*).

```
get_regex_for_cos_data <- function() {
  regex_pattern <- paste("^",viscos_options("name_COSyear"),"$|",
                         "^",viscos_options("name_COSmonth"),"$|",
                         "^",viscos_options("name_COSday"),"$|",
                         "^",viscos_options("name_COShour"),"$|",
                         "^",viscos_options("name_COSmin"),"$|",
                         viscos_options("name_o"),".*|",
                         viscos_options("name_s"),".*|",
                         viscos_options("name_COSposix"),"|",
                         viscos_options("name_COSperiod"),
                         sep = "")
  return(regex_pattern)
}
```

11.1.2 Extract the numeration of the basins

This function fetches the number of basins from the provided data.frame (`cos_data`) by removing all the non-digits characters from the column names.

```
get_basin_numbers <- function(cos_data) {
  require("magrittr", quietly = TRUE)
  assert_dataframe(cos_data)
  assert_junk(cos_data)
  #
  d_names <- names(cos_data)
  d_nums <- d_names  %>% gsub('\\D','',.) %>% unique
  d_nums <- d_nums[!(d_nums == "")] %>% as.integer
  return(d_nums)
}
```