Rhydro

Contents

Using R in hydrology (EGU2017 short course)

Instructors: Shaun Harrigan, Katie Smith, Berry Boessenkool and Daniel Klotz
Organizer: Berry Boessenkool
Contact: Questions and feedback are welcome via berry-b@gmx.de
Last updated: 2017-04-24

Please let us know your current R knowledge level by filling out the short survey at
bit.ly/knowR

These slides and all other course materials can be found at
github.com/brry/rhydro
Download the github course repository with all the materials including the datasets and presentation source code.

This is an R Markdown Notebook.
For discussions, please visit the Hydrology in R Facebook group.
Before running the code blocks below, we suggest to get package installation instructions by running:

```
source("https://raw.githubusercontent.com/brry/rhydro/master/checkpc.R")
```

Aim and contents of this workshop

We want to:

- Show off how awesome R is for hydrology (it's R-some!^^)
- Convince you to start or continue using R
- Provide all the code for you as a starting point

We can not:

- Teach you actual R coding (90 mins is too short for a tutorial)

We have prepared:

- Good coding practice, report generation (Rstudio, `rmarkdown`, R notebook)
- Using R as GIS (reading a rainfall shapefile + Kriging, `sf` + `leaflet` + `mapview` + `OSMscale`)
- River discharge time-series visualisation and extreme value statistics (`animation` + `extremeStat`)
- Hydrological modelling with `airGR`
- Trend anaysis including flow duration curve (`trend` + `hydroTSM`)

top

Report

Good coding practice, report generation (Rstudio, `rmarkdown`, R notebook) **Daniel Klotz** BOKU Vienna (daniel.klotz@boku.ac.at)

Why to use R

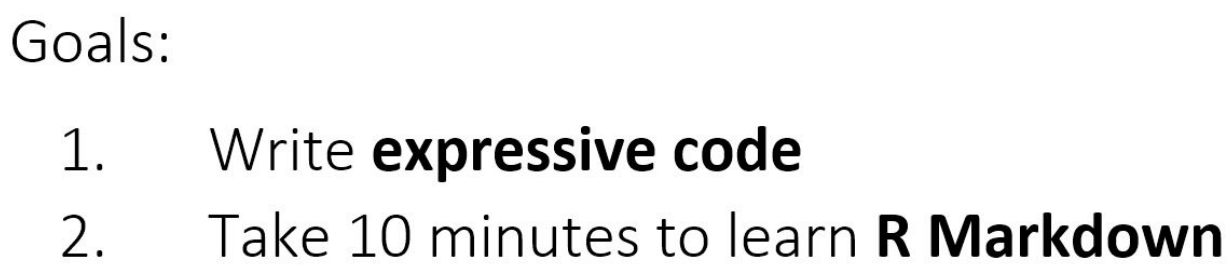


Figure 1:

Why I did not use R:

Whats great about R:

```
library(ggplot2)
test_data <- mpg
test_plot <- ggplot(test_data, aes(displ, hwy, colour = class)) +
```

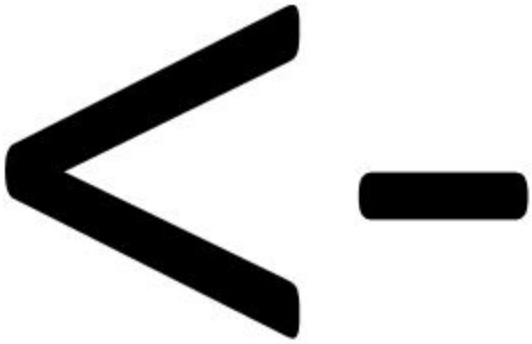
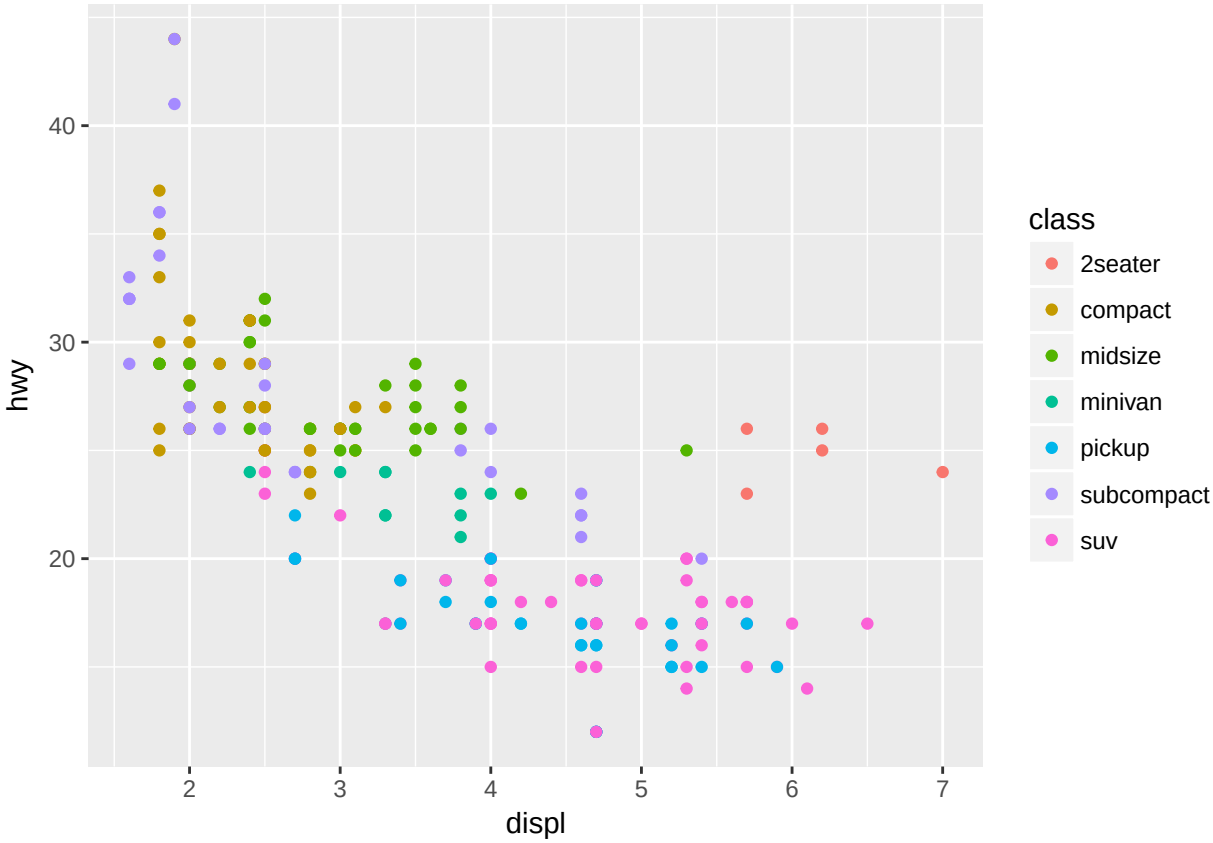


Figure 2:

```
  geom_point()
test_plot
```



Why I decided to use R:

Previously:

Figure 3: pipe

```
  aggregation_function <- function(x) {
    round(mean(x),2)
  }
  mtcars_subset <- subset(mtcars,hp > 100)
  mtcars_aggregated <- aggregate(. ~ cyl, data = mtcars_subset, FUN = aggregation_function)
  car_data1 <- transform(mtcars_aggregated, kpl = mpg*0.4251)
  print(car_data1)
```

```
##   cyl   mpg   disp     hp drat   wt  qsec   vs   am gear carb       kpl
## 1   4 25.90 108.05 111.00 3.94 2.15 17.75 1.00 1.00 4.50 2.00 11.010090
## 2   6 19.74 183.31 122.29 3.59 3.12 17.98 0.57 0.43 3.86 3.43  8.391474
## 3   8 15.10 353.10 209.21 3.23 4.00 16.77 0.00 0.14 3.29 3.50  6.419010
```

Now:

```
library(magrittr)
car_data2 <-
  mtcars %>%
  subset(hp > 100) %>%
  aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%
  transform(kpl = mpg %>% multiply_by(0.4251)) %>%
  print
```

```
##   cyl   mpg   disp     hp drat   wt  qsec   vs   am gear carb       kpl
## 1   4 25.90 108.05 111.00 3.94 2.15 17.75 1.00 1.00 4.50 2.00 11.010090
## 2   6 19.74 183.31 122.29 3.59 3.12 17.98 0.57 0.43 3.86 3.43  8.391474
## 3   8 15.10 353.10 209.21 3.23 4.00 16.77 0.00 0.14 3.29 3.50  6.419010
```

btw: You can integrate other programming languages with ease. Here an example from Yihui Xie for the use of `Fortran` in Rmarkdown:

1. Compile Code:

```
C Fortran test
      subroutine fexp(n, x)
      double precision x
```

```
C  output
      integer n, i
C  input value
      do 10 i=1,n
         x=dexp(dcos(dsin(dble(float(i)))))
  10  continue
      return
      end
```

2. Run Code:

```
res = .Fortran("fexp", n=100000L, x=0)
str(res)
```

Be happy with the result: > ## List of 2 > ## $ n: int 100000 > ## $ x: num 2.72

Markdown

HTML: HyperText **Markdown** Language

John Gruber:

```latex
\section{Introduction}

We did a \emph{cool} study,
and our main findings:


\begin{enumerate}
\item You can never remember
  how to escape backslashes.
\item A dollar sign is \$,
  an ampersand \&, and
  a \textbackslash{}.
\item How about ~? Use $\sim$.
\end{enumerate}
```

```markdown
# Introduction

We did a _cool_ study
our main findings:

1. You do not need to
   a lot of rules.
2. A dollar sign is $
   an ampersand is &,
   a backslash \.
3. A tilde is ~.

Write content instead
markup languages.
```

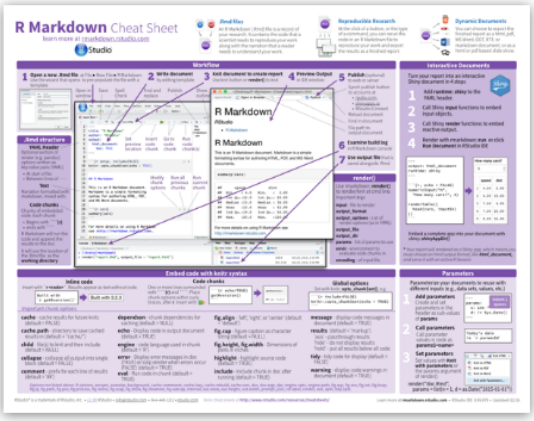
Comparison: Markdown vs. Latex

Rstudio provides cheat-sheets with the most important informations about many of their "favorite" packages & software:

R Markdown Cheat Sheet

R Markdown is an authoring format that makes it easy to write reusable reports with R. You combine your R code with narration written in markdown (an easy-to-write plain text format) and then export the results as an html, pdf, or Word file. You can even use R Markdown to build interactive documents and slideshows. Updated 02/16. (Old Version).

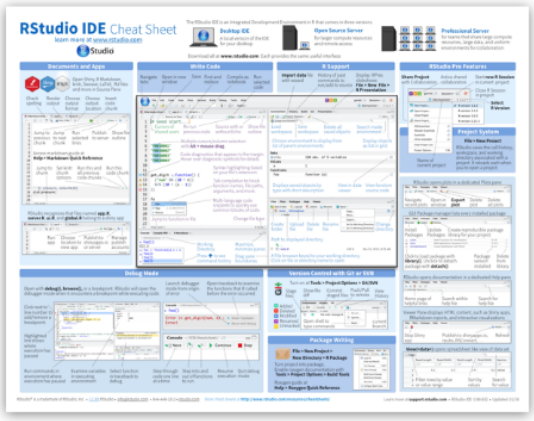
DOWNLOAD



RStudio IDE Cheat Sheet

The RStudio IDE is the most popular integrated development environment for R. Do you want to write, run, and debug your own R code? Work collaboratively on R projects with version control? Build packages or create documents and apps? No matter what you do with R, the RStudio IDE can help you do it faster. This cheat sheet will guide you through the most useful features of the IDE, as well as the long list of keyboard shortcuts built into the RStudio IDE. Updated 01/16.

DOWNLOAD

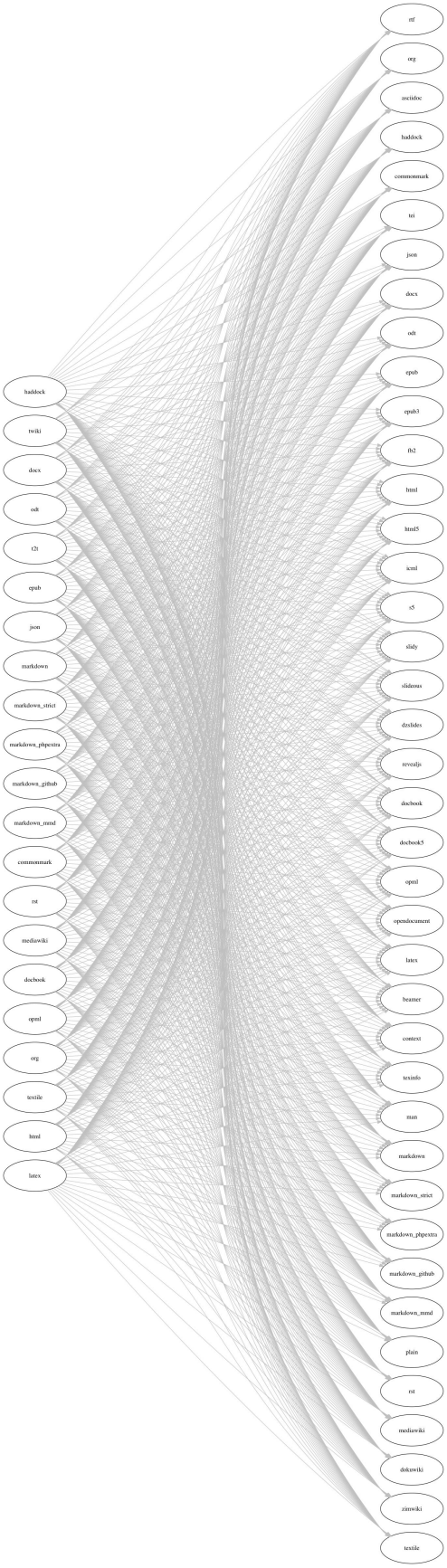


** Rmarkdown **

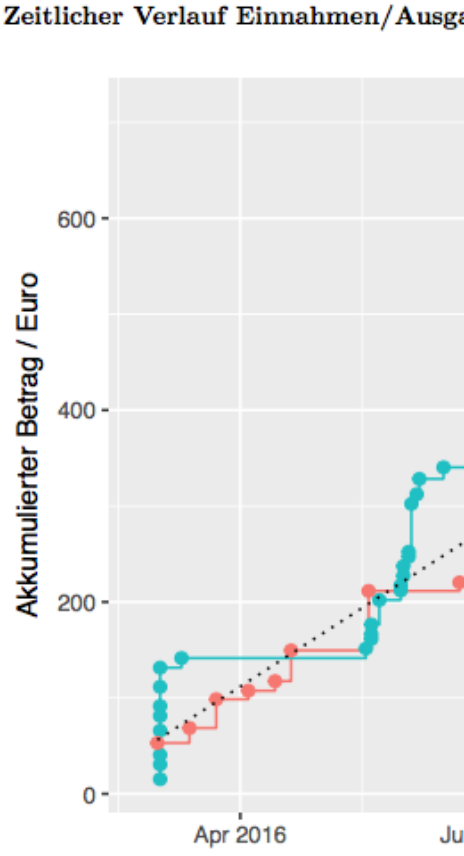
In Rstudio:

“Native” Formats:

- html
- pdf
- word



Much more possible if you adress pandoc directly:



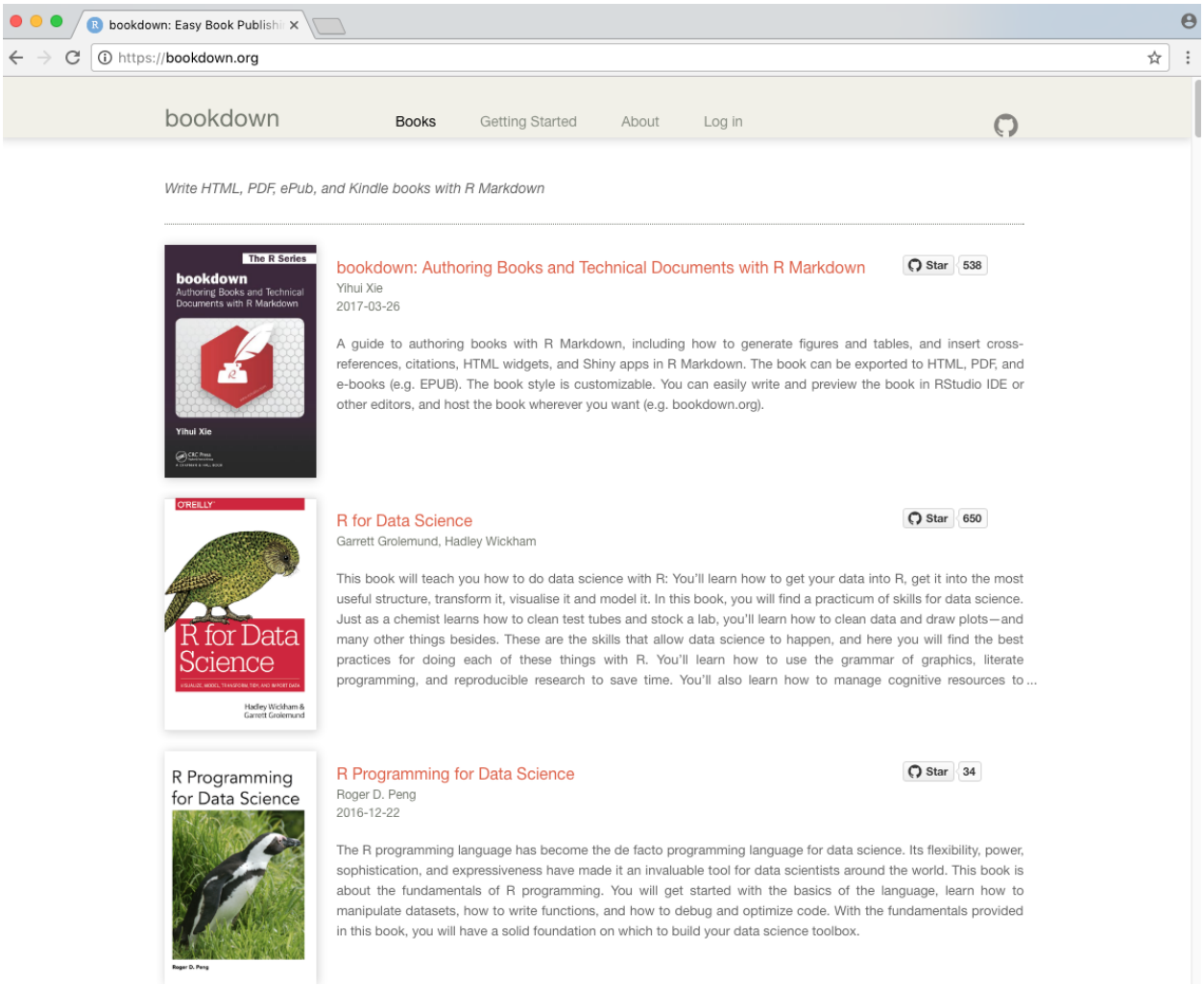
Zur Zeit haben wir ein Guthaben von -9.7

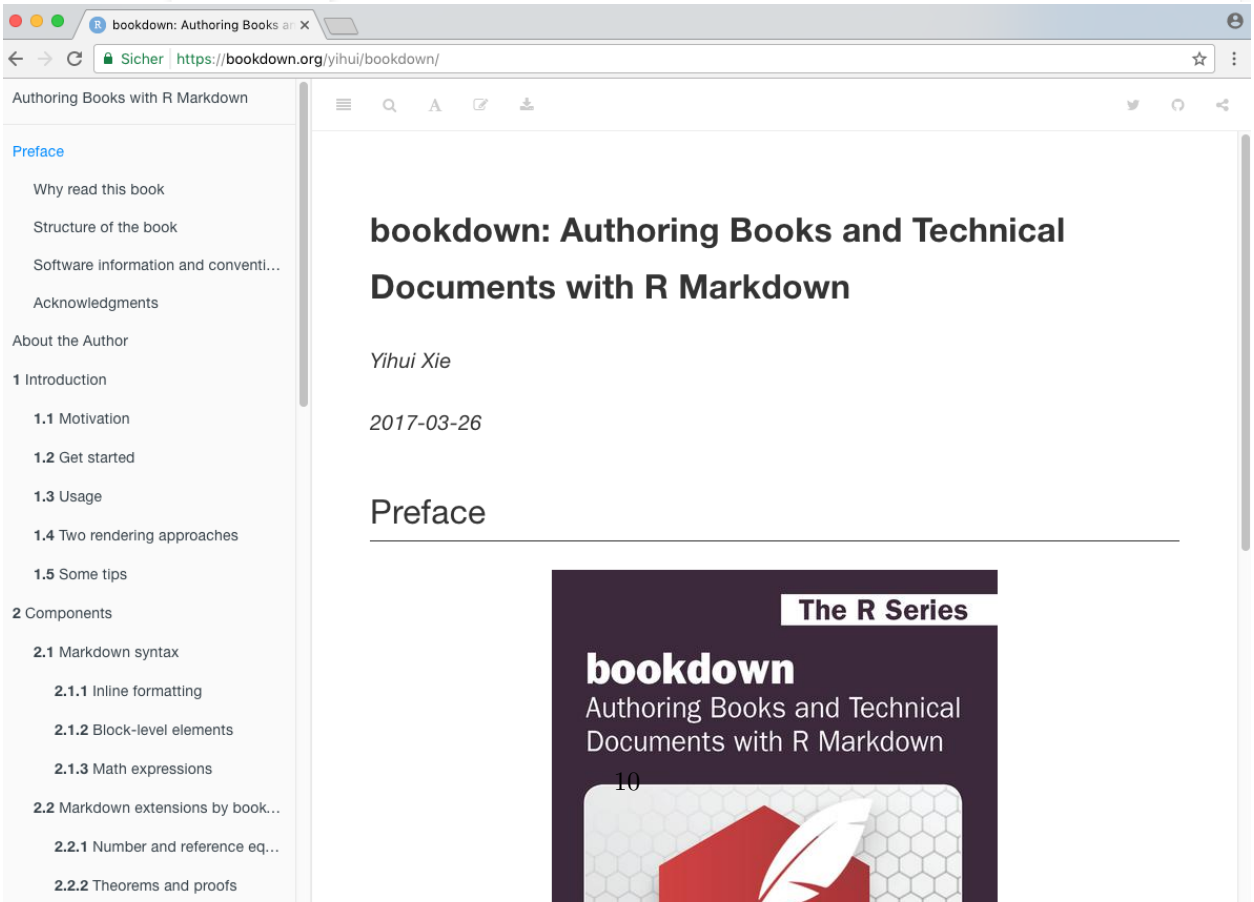
Information in the text can be automatically updated with the rest of the document: [

Examples

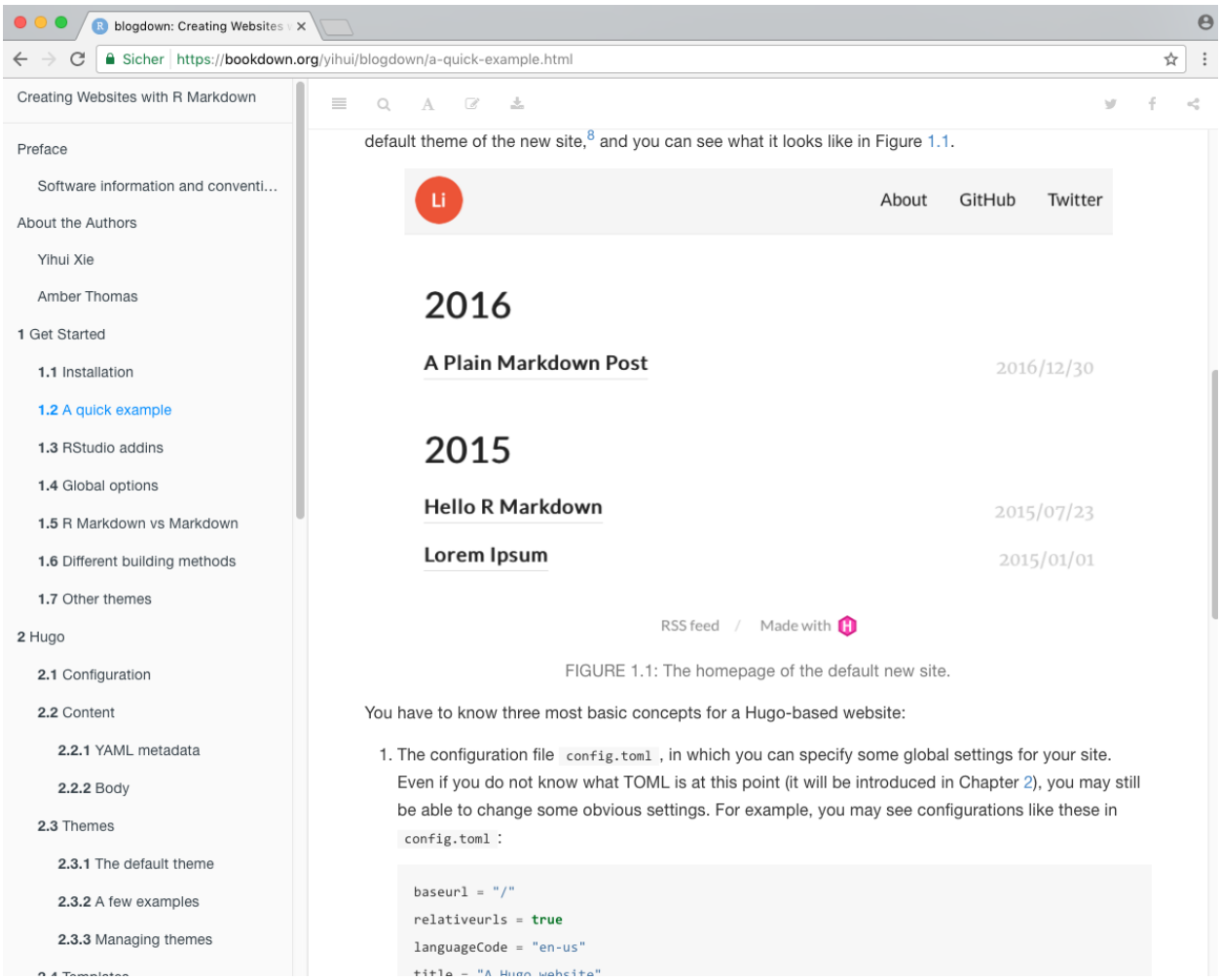
Small Websites

Books (blogdown)


bookdown: Easy Book Publishi
https://bookdown.org
bookdown
Books
Getting Started
About
Log in
Write HTML, PDF, ePub, and Kindle books with R Markdown
bookdown: Authoring Books and Technical Documents with R Markdown
Star 538
Yihui Xie
2017-03-26
A guide to authoring books with R Markdown, including how to generate figures and tables, and insert cross-references, citations, HTML widgets, and Shiny apps in R Markdown. The book can be exported to HTML, PDF, and e-books (e.g. EPUB). The book style is customizable. You can easily write and preview the book in RStudio IDE or other editors, and host the book wherever you want (e.g. bookdown.org).
R for Data Science
Star 650
Garrett Grolemund, Hadley Wickham
This book will teach you how to do data science with R: You'll learn how to get your data into R, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you'll learn how to clean data and draw plots—and many other things besides. These are the skills that allow data science to happen, and here you will find the best practices for doing each of these things with R. You'll learn how to use the grammar of graphics, literate programming, and reproducible research to save time. You'll also learn how to manage cognitive resources to ...
R Programming for Data Science
Star 34
Roger D. Peng
2016-12-22
The R programming language has become the de facto programming language for data science. Its flexibility, power, sophistication, and expressiveness have made it an invaluable tool for data scientists around the world. This book is about the fundamentals of R programming. You will get started with the basics of the language, learn how to manipulate datasets, how to write functions, and how to debug and optimize code. With the fundamentals provided in this book, you will have a solid foundation on which to build your data science toolbox.


bookdown: Authoring Books ar
Sicher
https://bookdown.org/yihui/bookdown/
Authoring Books with R Markdown
Preface
Why read this book
Structure of the book
Software information and conventi...
Acknowledgments
About the Author
1 Introduction
1.1 Motivation
1.2 Get started
1.3 Usage
1.4 Two rendering approaches
1.5 Some tips
2 Components
2.1 Markdown syntax
2.1.1 Inline formatting
2.1.2 Block-level elements
2.1.3 Math expressions
2.2 Markdown extensions by book...
2.2.1 Number and reference eq...
2.2.2 Theorems and proofs
bookdown: Authoring Books and Technical Documents with R Markdown
Yihui Xie
2017-03-26
Preface
The R Series
bookdown
Authoring Books and Technical Documents with R Markdown

Blogs (hugo)



Presentations

An Example R Markdown Document

(A Subtitle Would Go Here if This Were a Class)

Steven V. Miller

Department of Political Science

Sheena Easton and Game Theory

Sheena Easton describes the following scenario for her baby:

1. Takes the morning train
2. Works from nine 'til five
3. Takes another train home again
4. Finds Sheena Easton waiting for him

Sheena Easton and her baby are playing a zero-sum (total conflict) game.

- Akin to Holmes-Moriarty game (see: von Neumann and Morgenstern)
- Solution: mixed strategy

Rick Astley's Re-election Platform

Widgets

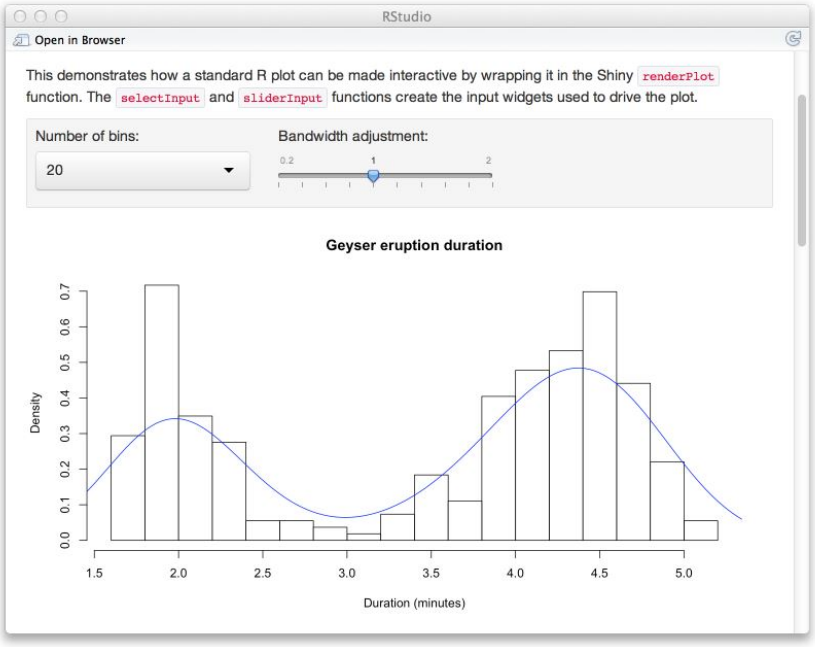
Apps (Shiny)

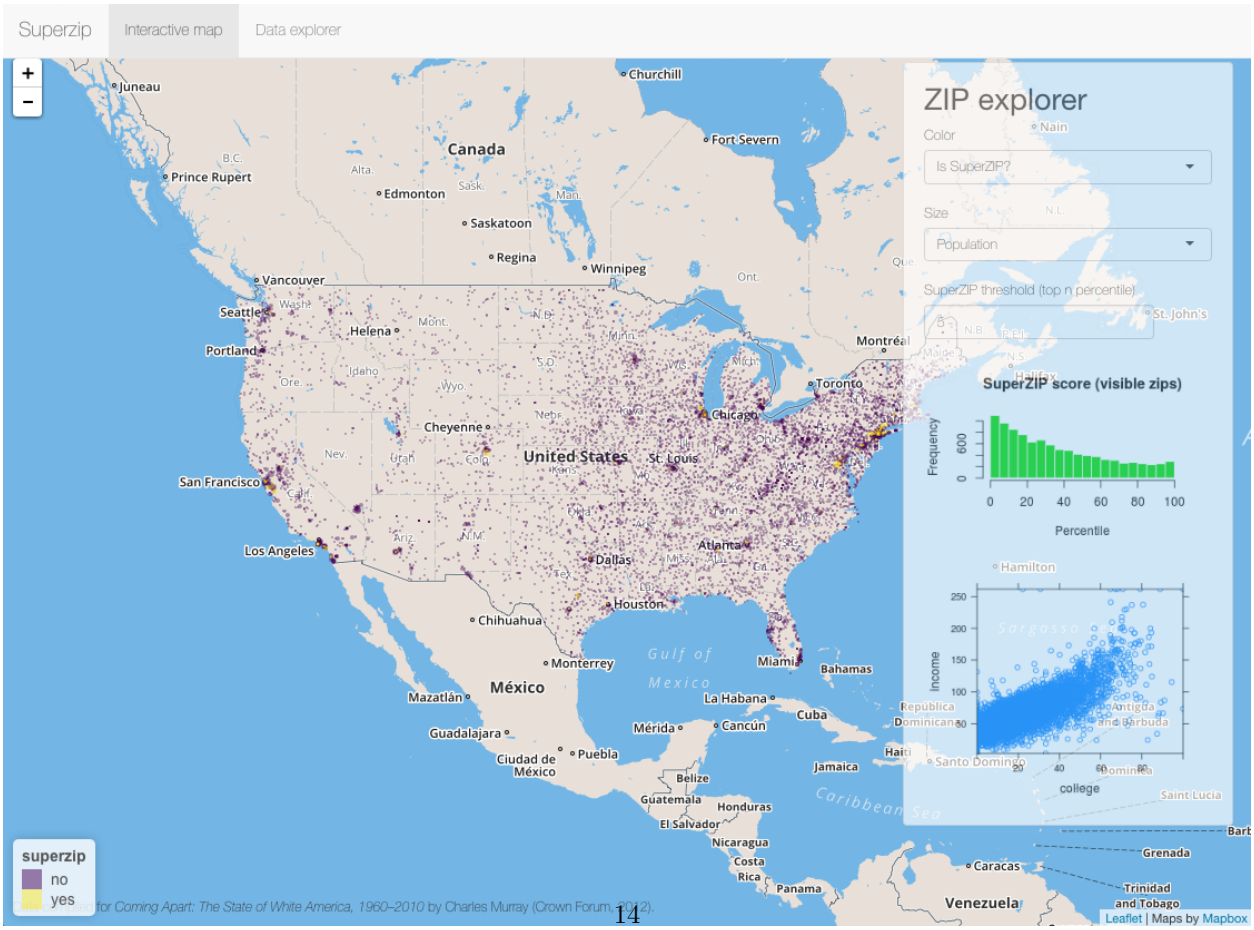
Interactive Documents

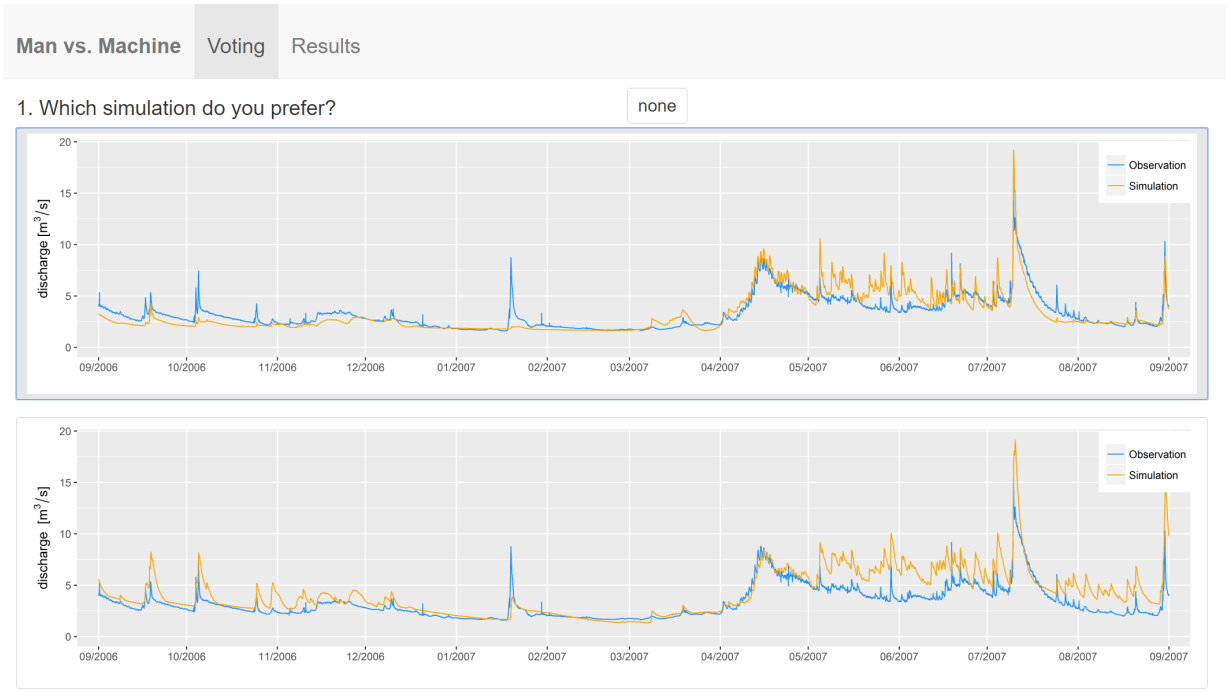
Overview

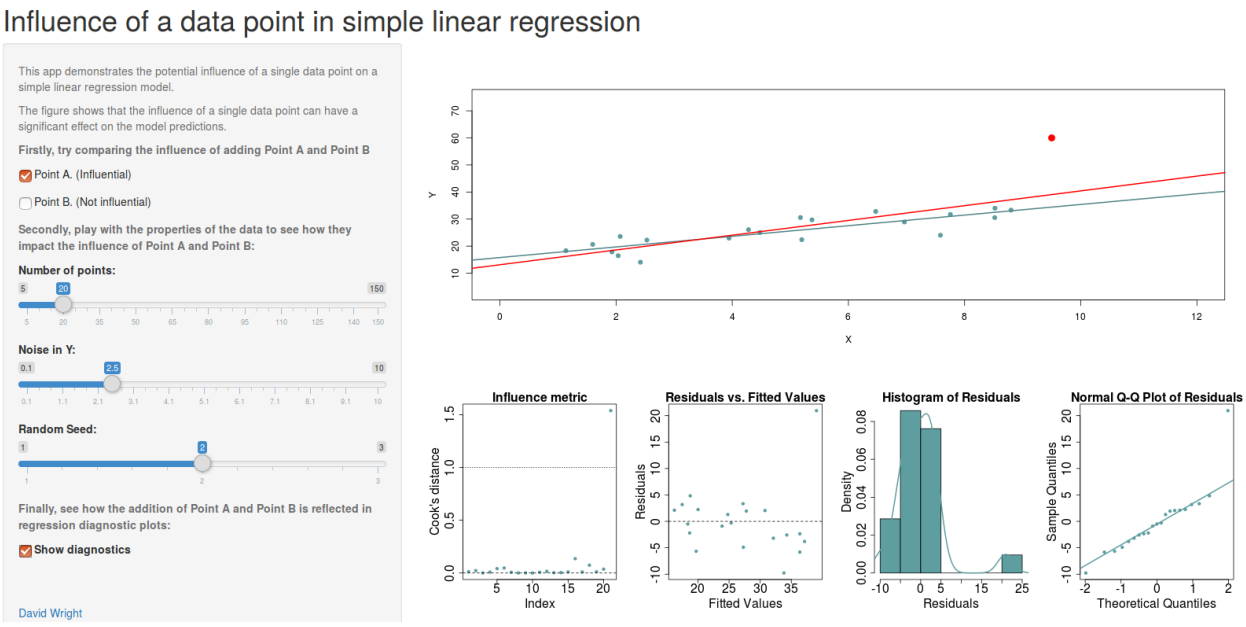
R Markdown has been extended to support fully interactive documents. Unlike the more traditional workflow of creating static reports, you can now create documents that allow your readers to change the parameters underlying your analysis and see the results immediately.

R Markdown leverages Shiny at its core to make this possible. Here is a simple example of an R Markdown document with an interactive plot:









R based hydrology tools from the Intelligent Water Decisions Research Group of the University of Adelaide:

- RShiny `influence regression app` by David Wright: https://davidpwright.shinyapps.io/LinearRegressionInfluenceExample/
- `Evapotranspiration` R package that enables the use of 17 well-known ET models in a consistent manner.

top

GIS

Using R as GIS (reading a rainfall shapefile + Kriging, `sf` + `leaflet` + `mapview` + `OSMscale`) **Berry Boessenkool** Potsdam University, Germany (berry-b@gmx.de)

Shapefiles

Reading shapefiles with `maptools::readShapeSpatial` and `rgdal::readOGR` is obsolete. Instead, use `sf::st_read`. `sf` is on CRAN since oct 2016. Main reaction when using sf: "Wow, that is fast!" Download the shapefile or better: download the whole github course repository

```
rain <- sf::st_read("data/PrecBrandenburg/niederschlag.shp")
```

```
## Reading layer `niederschlag' from data source `/home/berry/Dropbox/R/rhydro/presentations/data/PrecB
## converted into: POLYGON
## Simple feature collection with 277 features and 1 field
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 3250175 ymin: 5690642 xmax: 3483631 ymax: 5932731
## epsg (SRID):    NA
## proj4string:    +proj=tmerc +lat_0=0 +lon_0=15 +k=0.9996 +x_0=3500000 +y_0=0 +ellps=GRS80 +units=m +
```

Central points of rainfall Thiessen polygons

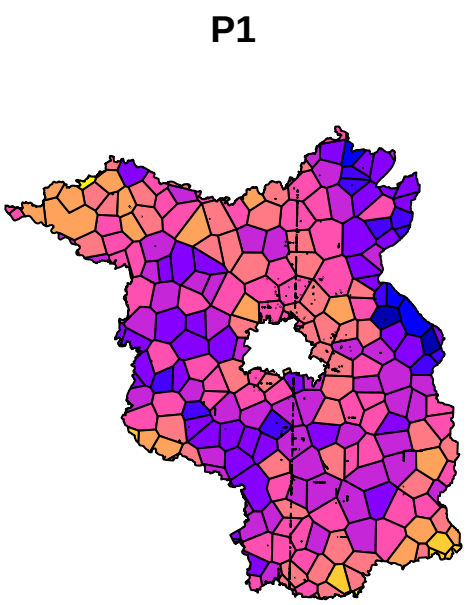
```
centroids <- sf::st_centroid(rain)
centroids <- sf::st_coordinates(centroids)
centroids <- as.data.frame(centroids)
```

top

Plotting, maps

Static plot:

```
plot(rain[,1])
```



Static map:

```
prj <- sf::st_crs(rain)$proj4string
#cent_ll <- OSMscale::projectPoints(Y,X, data=centroids, to=OSMscale::pll(), from=prj)
#map_static <- OSMscale::pointsMap(y,x, cent_ll, fx=0.08, type="maptoolkit-topo", zoom=6)
#save(map_static, file="data/map_static.Rdata")
#load("data/map_static.Rdata")
#OSMscale::pointsMap(y,x, cent_ll, map=map_static)
```

Interactive map:

```
library(leaflet)
cent_ll$info <- paste0(sample(letters,nrow(cent_ll),TRUE), ", ", round(cent_ll$x,2),
                       ", ", round(cent_ll$y,2))
leaflet(cent_ll) %>% addTiles() %>% addCircleMarkers(lng=~x, lat=~y, popup=~info)
```

Interactive map of shapefile:

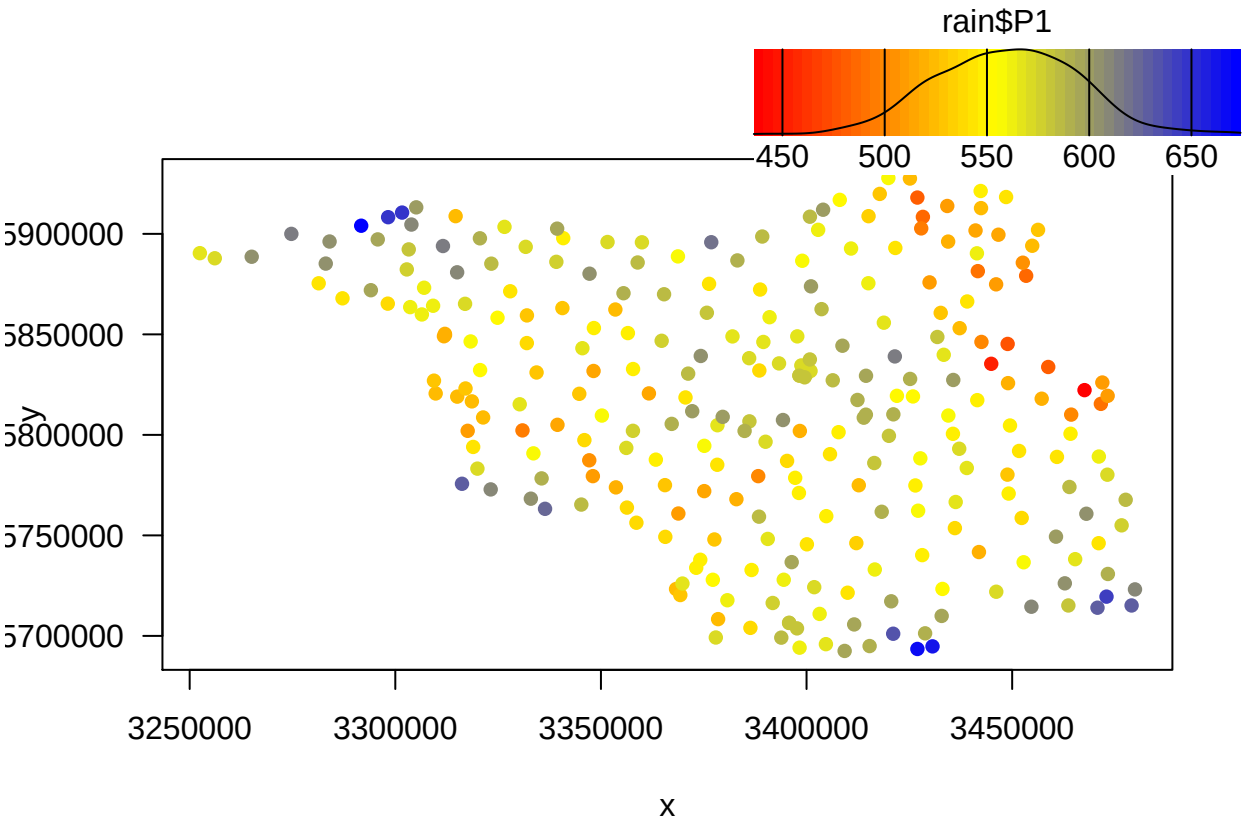
```
# make sure to have installed the development version of mapview:
# devtools::install_github("environmentalinformatics-marburg/mapview", ref = "develop")
library(berryFunctions) # classify, seqPal
col <- seqPal(n=100, colors=c("red","yellow","blue"))[classify(rain$P1)$index]
mapview::mapview(rain, col.regions=col)
```

top

Kriging

Plot original points colored by third dimension:

```
pcol <- colorRampPalette(c("red","yellow","blue"))(50)
x <- centroids$X # use cent_ll$x for projected data
y <- centroids$Y
berryFunctions::colPoints(x, y, rain$P1, add=FALSE, col=pcol, y1=0.8)
```



Calculate the variogram and fit a semivariance curve

```
library(geoR)
```

```
## ----------------------------------------------------------
##  Analysis of Geostatistical Data
##  For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
##  geoR version 1.7-5.2 (built on 2016-05-02) is now loaded
## ----------------------------------------------------------
```

```
geoprec <- as.geodata(cbind(x,y,rain$P1))
vario <- variog(geoprec, max.dist=130000) # other maxdist for lat-lon data
```

```
## variog: computing omnidirectional variogram
```

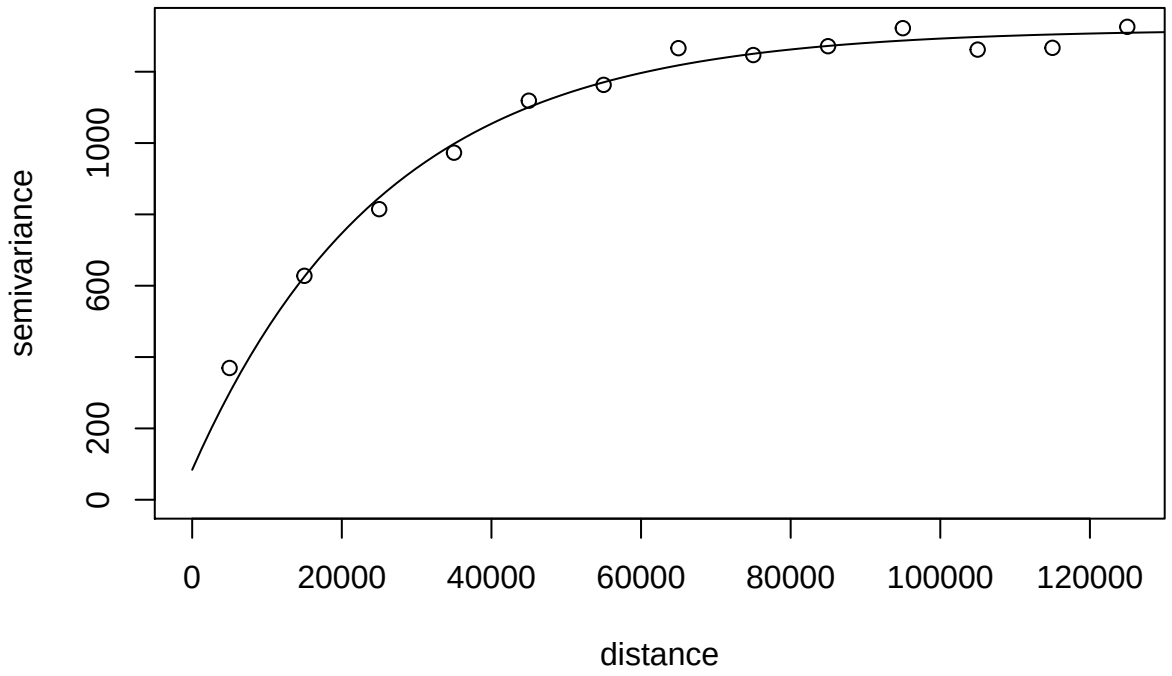
```
fit <- variofit(vario)
```

```
## variofit: covariance model used is matern
## variofit: weights used: npairs
## variofit: minimisation function used: optim
```

```
## Warning in variofit(vario): initial values not provided - running the
## default search
```

```
## variofit: searching for best initial value ... selected values:
##                sigmasq   phi        tausq kappa
## initial.value "1325.81" "19999.05" "0"   "0.5"
## status        "est"     "est"      "est" "fix"
## loss value: 104819060.429841
```

```
plot(vario)
lines(fit)
```



Determine a useful resolution (keep in mind that computing time rises exponentially with grid size)

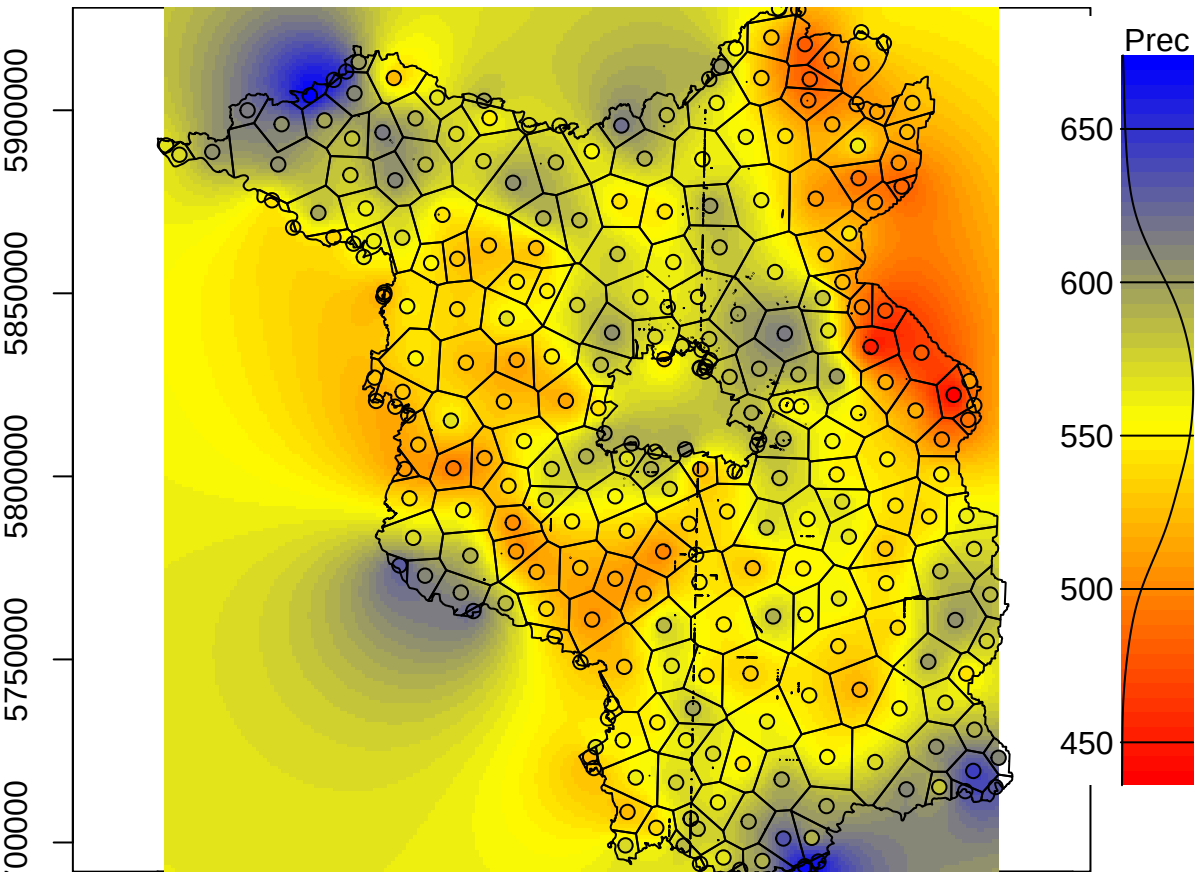
```
# distance to closest other point:
d <- sapply(1:length(x), function(i)
            min(berryFunctions::distance(x[i], y[i], x[-i], y[-i])) )
# for lat-long data use (2017-Apr only available in github version of OSMscale)
# d <- OSMscale::maxEarthDist(y,x, data=cent_ll, fun=min)
hist(d/1000, breaks=20, main="distance to closest gauge [km]")
mean(d/1000) # 8 km
```

Perform kriging on a grid with that resolution

```
res <- 1000 # 1 km, since stations are 8 km apart on average
grid <- expand.grid(seq(min(x),max(x),res),
                    seq(min(y),max(y),res))
krico <- krige.control(type.krige="OK", obj.model=fit)
#krobj <- krige.conv(geoprec, loc=grid, krige=krico)
#save(krobj, file="data/krobj.Rdata")
load("data/krobj.Rdata") # line above is too slow for recreation each time
```

Plot the interpolated values with `image` or an equivalent function (see Rclick 4.15) and add contour lines.

```
par(mar=c(0,3,0,3))
geoR:::image.kriging(krobj, col=pcol)
colPoints(x, y, rain$P1, col=pcol, legargs=list(horiz=F, title="Prec",y1=0.1,x1=0.9))
points(x,y)
plot(rain, col=NA, add=TRUE)
```



top

Discharge

River discharge time-series visualisation and extreme value statistics (`animation` + `extremeStat`)
Berry Boessenkool

Read, plot and aggregate data

Datasets from the UK National River Flow Archive http://nrfa.ceh.ac.uk/data/station/meanflow/39072
Download discharge csv or better: download the whole github course repository

Read and transform data

```
Q <- read.table("data/discharge39072.csv", skip=19, header=TRUE, sep=",", fill=TRUE)[,1:2]
colnames(Q) <- c("date","discharge")
Q$date <- as.Date(Q$date, format="%Y-%m-%d")
```

Examine data

```
head(Q)
```

```
##         date discharge
## 1 1979-07-20     33.44
## 2 1979-07-21     32.45
## 3 1979-07-22     33.15
```

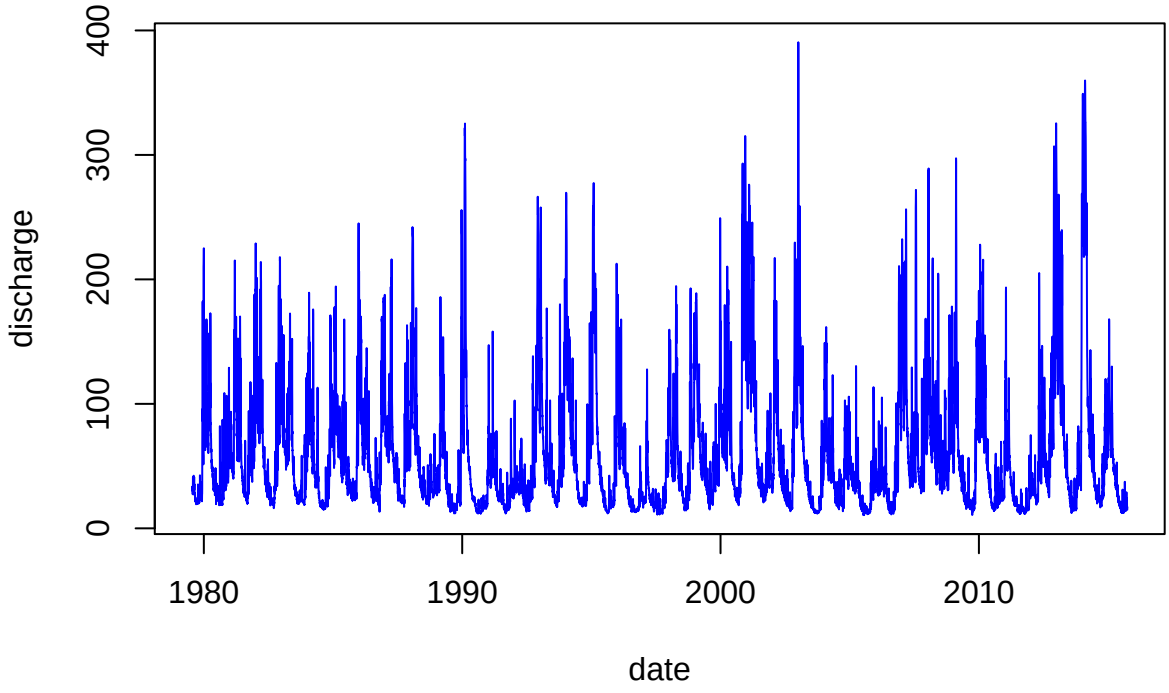
```
## 4 1979-07-23     30.62
## 5 1979-07-24     30.06
## 6 1979-07-25     31.26
```

```
str(Q)
```

```
## 'data.frame':    13222 obs. of  2 variables:
##  $ date     : Date, format: "1979-07-20" "1979-07-21" ...
##  $ discharge: num  33.4 32.5 33.1 30.6 30.1 ...
```

Simple time series plot

```
plot(Q, type="l", col="blue")
```



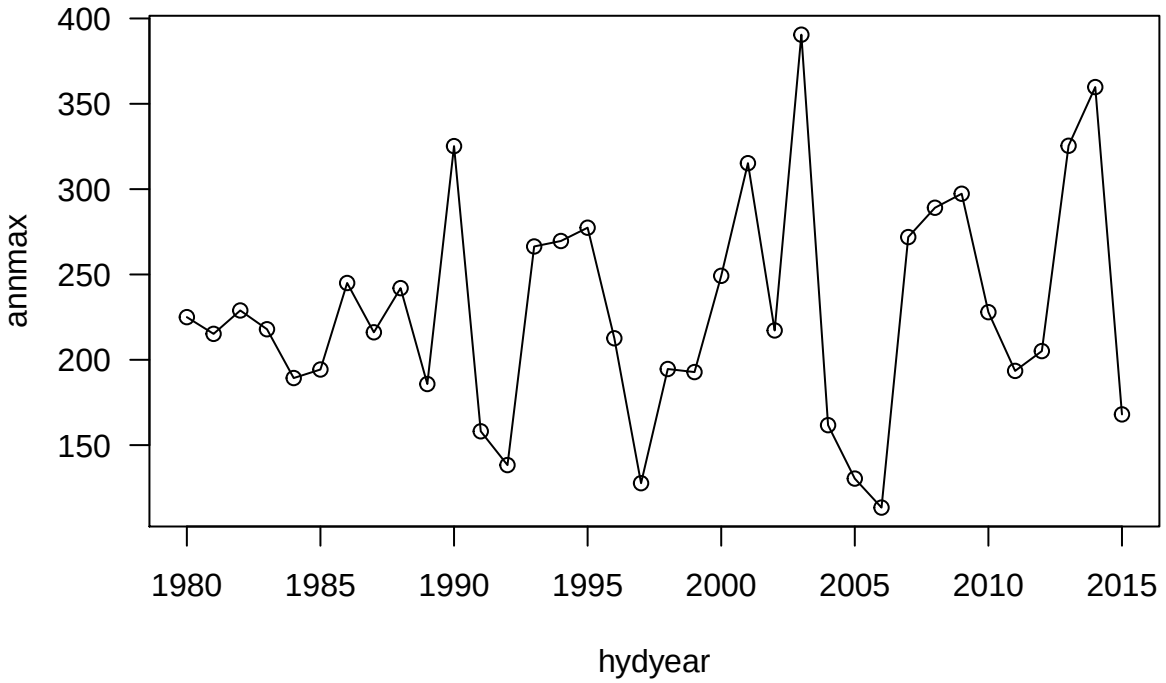
Publication-ready graphics

```
png("DischargeVis.png", width=20, height=10, units="cm", res=500)
#pdf("DischargeVis.pdf", width=20/2.5, height=10/2.5) # vector graph
par(mar=c(3.5,3.5,2.5,0.2), mgp=c(2.3,0.7,0), las=1)
plot(Q, type="l", col="blue", main="NRFA: Thames\nRoyal Windsor Park",
     xlab="Date", ylab="Discharge  [m\U{00B3}/s]")
dev.off()
```

```
## pdf
##   2
```

Annual maxima, German hydrological year split at Oct 31

```
Q$hydyear <- as.numeric(format(Q$date+61, "%Y"))
annmax <- tapply(Q$discharge, Q$hydyear, max, na.rm=TRUE)
annmax <- annmax[-1]
hydyear <- as.numeric(names(annmax))
plot(hydyear, annmax, type="o", las=1)
```



Extreme value statistics

```
library(extremeStat)
```

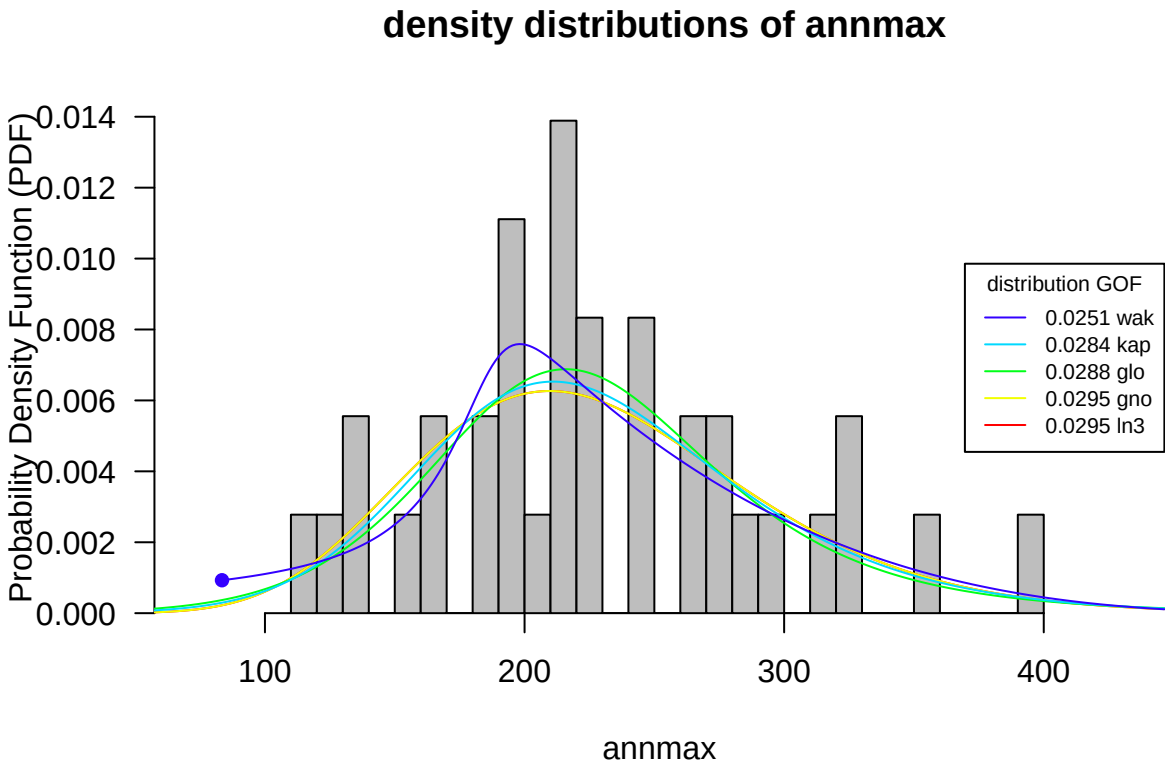
```
## # Loaded extremeStat 1.3.0 (2017-01-26). Package restructured since 0.6.0 (2016-12-13).
## # Computing functions don't plot anymore and some are renamed. See help('extremeStat-deprecated')
```

```
dlf <- distLfit(annmax)
```

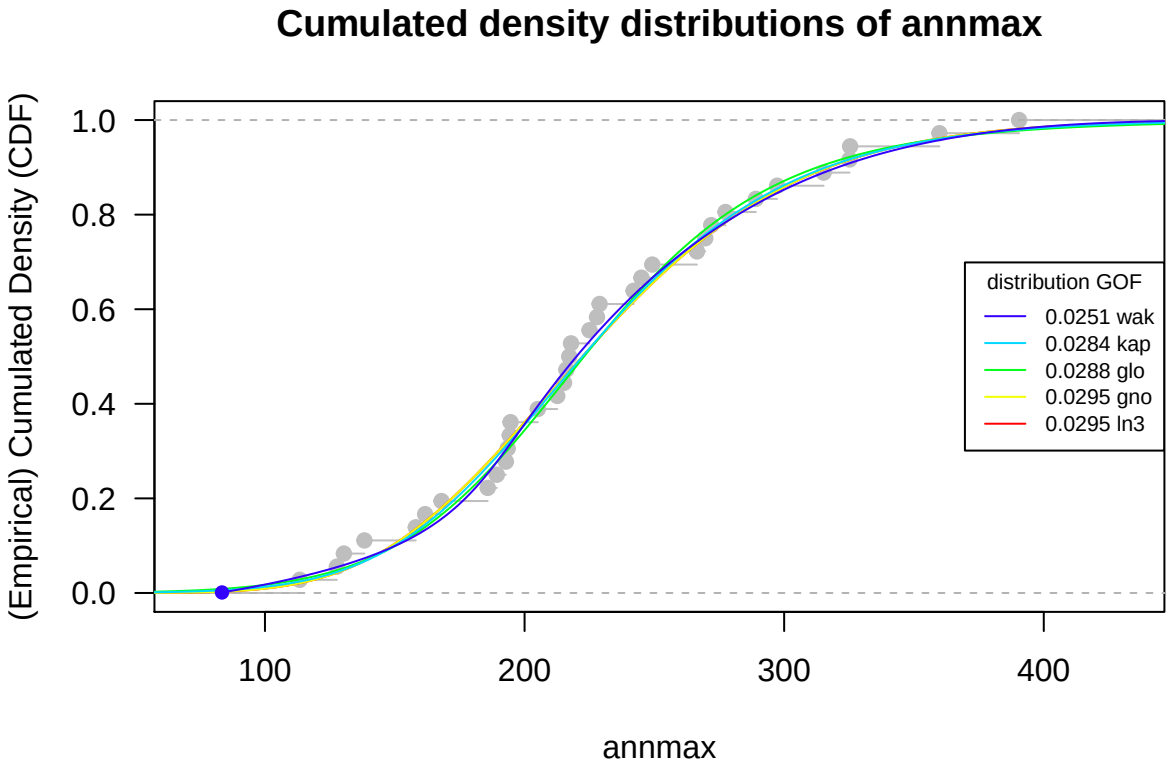
```
## Note in distLfit: dat was not a vector.
```

```
## distLfit execution took 3.4 seconds.
```

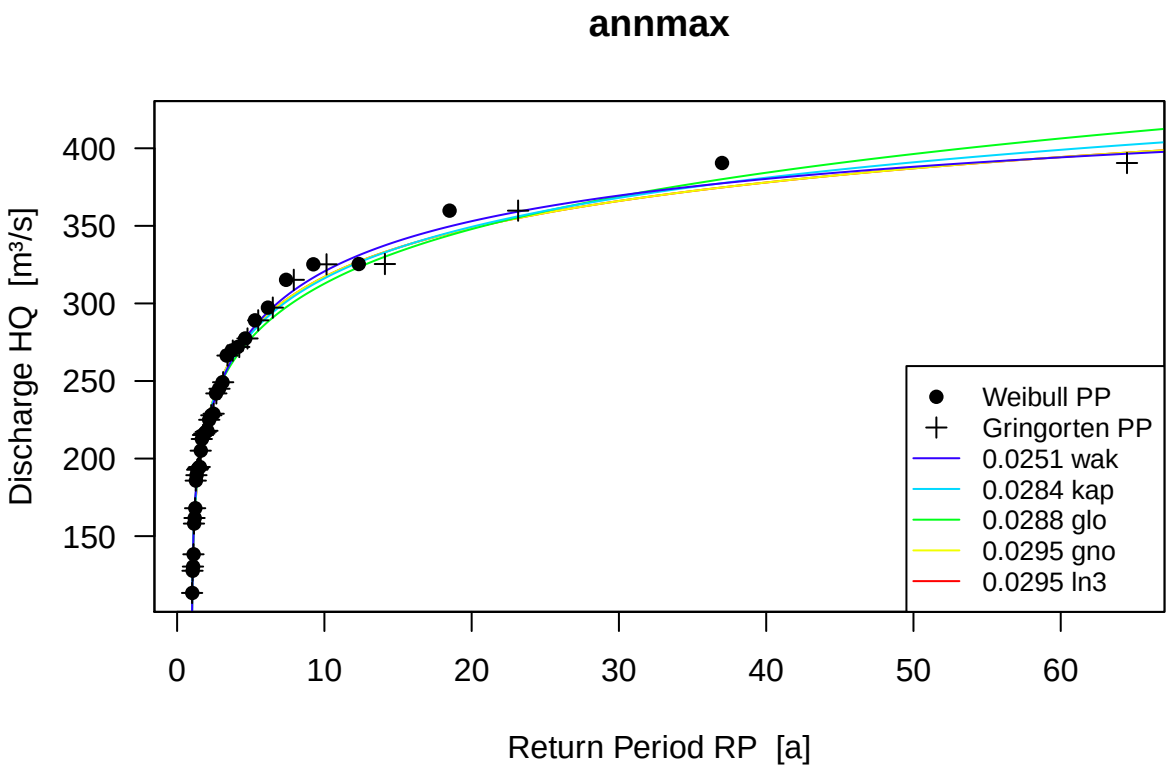
```
plotLfit(dlf)
```




```
plotLfit(dlf, cdf=TRUE)
```

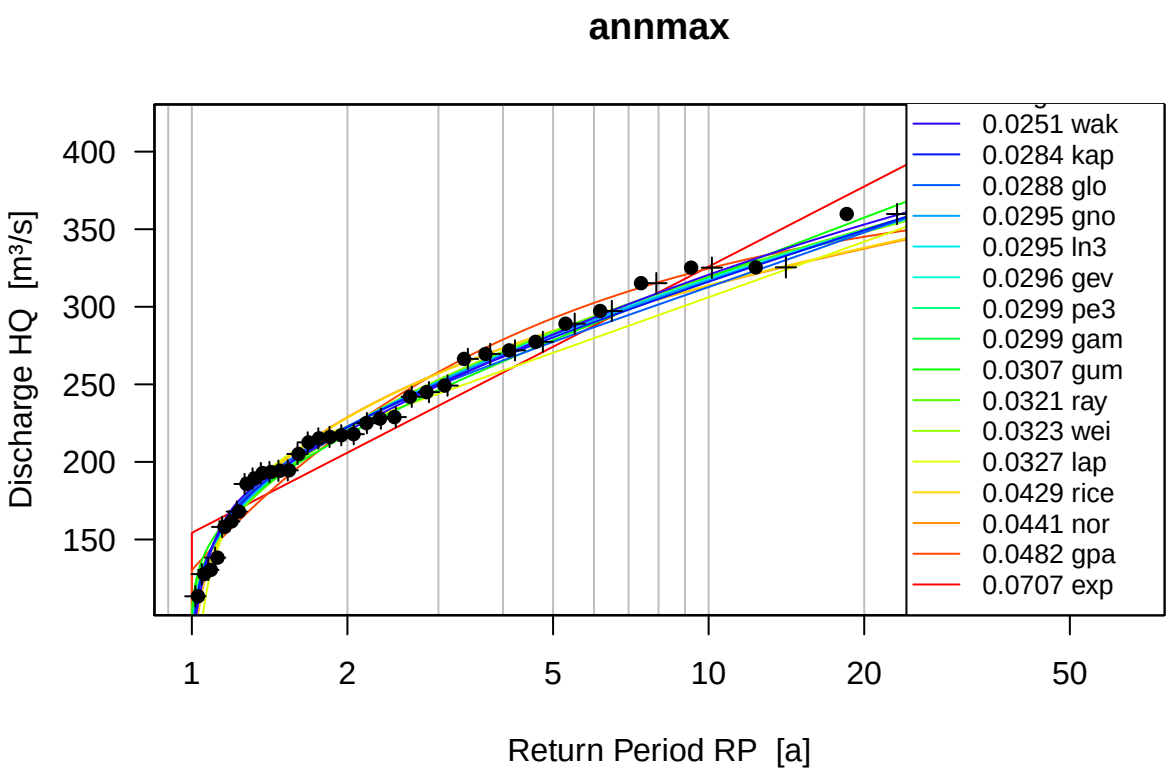



```
dle <- distLextreme(dlf=dlf, RPs=c(5,10,50,100), gpd=FALSE)
plotLextreme(dle)
```



Logarithmic plot with many fitted distribution functions

```
plotLextreme(dle, nbest=16, log=TRUE)
```



Return values (discharge estimates for given return periods)

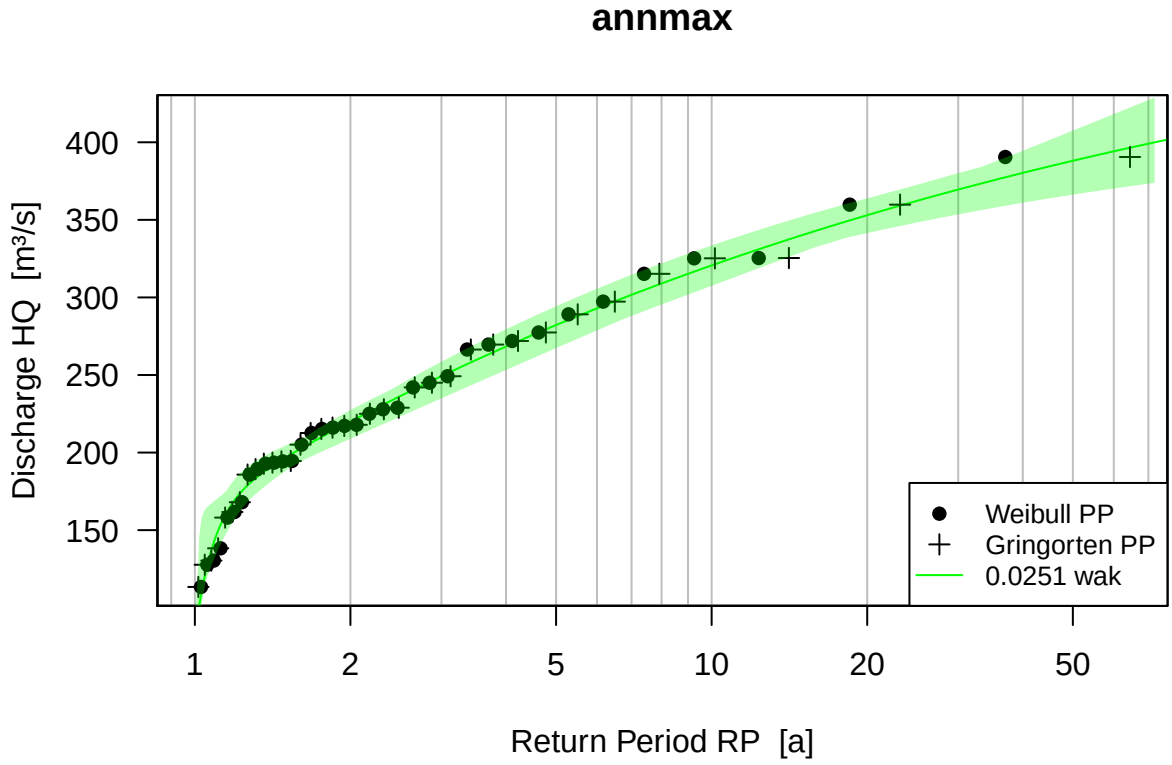
```
dle$returnlev
```

```
##                   RP.5    RP.10    RP.50   RP.100
## wak           282.1703 320.6330 388.0810 409.7826
## kap           280.2552 316.1410 390.9610 421.2308
## glo           277.3396 312.7316 396.3063 435.2280
## gno           281.8098 317.1049 387.0331 414.3959
## ln3           281.8098 317.1049 387.0331 414.3959
## gev           282.3207 318.0202 386.3711 411.5226
## pe3           282.2448 317.4862 386.0770 412.3930
## gam           282.2817 317.4259 385.7332 411.9133
## gum           278.4177 318.7478 407.5081 445.0319
## ray           283.7086 319.4134 385.4951 409.6172
## wei           284.0925 318.6432 381.6178 404.3445
## lap           270.3279 306.1250 389.2434 425.0405
## rice          284.4025 313.7770 365.4742 383.7571
## nor           284.3975 313.4448 364.4307 382.4296
## gpa           292.6228 324.6531 361.2851 368.5105
## exp           274.2330 325.8749 445.7837 497.4255
## revgum        285.4245 304.6724 333.1573 341.9241
## quantileMean 279.4553 322.7630 380.9489 388.0781
## weighted1     281.5072 316.7796 385.8692 412.5682
## weighted2     281.4350 316.8845 386.4560 413.3655
## weighted3     280.9691 317.2833 390.5207 419.4335
## weightedc          NaN      NaN      NaN      NaN
## n_full         36.0000       NA       NA       NA
## n              36.0000       NA       NA       NA
## threshold     113.4000       NA       NA       NA
```

In reality, please use non-stationary EVS!

Uncertainty band for Wakeby distribution fit estimate

```
dle_boot <- distLexBoot(dle, n=10, nbest=1)
plotLexBoot(dle_boot, distcol="green")
```



More details in the package vignette

```
vignette("extremeStat")
```

Animated movie

Download data from several discharge stations

http://nrfa.ceh.ac.uk/data/search Filter: River = Thames
http://environment.data.gov.uk/flood-monitoring/doc/reference#stations for lat-lon coordinates:

```
url <- "http://environment.data.gov.uk/flood-monitoring/id/stations?riverName=Thames"
json <- jsonlite::fromJSON(url)
str(json$items, max.level=1)
metajson <- json$items[,c("label","lat","long","dateOpened")]
```

```
meta <- read.table(header=T, as.is=T, sep=",", text="
name                , lat, lon
Kingston            , 51.415005,-0.308869
Days_Weir           , 51.638206,-1.179444
Eynsham             , 51.774692,-1.356854
West_Mill_Cricklade , 51.646694,-1.865536
Ewen                , 51.674733,-1.9904
Royal_Windsor_Park  , 51.485795,-0.589124
Reading             , 51.461325,-0.967884
")
#library(leaflet)
#leaflet(meta) %>% addTiles() %>% addCircleMarkers(~lon, ~lat, popup=~name)
#map_thames <- OSMscale::pointsMap(lat,lon,meta, fx=1, fy=0.2, plot=FALSE, zoom=6,
#                          type="maptoolkit-topo", quiet=TRUE)
```

Read datasets

```
files <- dir("data", pattern="^Thames_", full=TRUE)
thames <- lapply(files, function(f) {
 dis <- read.table(f, skip=19, header=TRUE, sep=",", fill=TRUE)[,1:2]
 name <- readLines(f, n=5)[5]
 name <- sub("station,name,Thames at ", "", name)
 name <- gsub(" ", "_", name)
 colnames(dis) <- c("date",name)
 dis$date <- as.Date(dis$date, format="%Y-%m-%d")
 dis
})
rm(files)
```

Merge datasets

```
dis <- Reduce(function(...) merge(..., all=T), thames)
```

Code to generate one movie slice

```
library(berryFunctions) # for lim0, monthAxis, textField, etc

scene <- function(i, vlcnote=TRUE, map=TRUE, cex=1.2)
{
 sel <- 0:120
 dis2 <- dis[i + sel, ]
 stat <- colnames(dis)[-1]
 col <- RColorBrewer::brewer.pal(ncol(dis)-1, name="Set1")
 names(col) <- stat
 plot(dis2$date,dis2[,2], type="n", ylim=lim0(300),  las=1,
      xaxt="n", yaxt="n", cex.lab=cex, xlab="",
      ylab="Discharge  [m\U{00B3}/s]", xaxs="i")
 axis(2, cex.axis=cex, las=1)
 Sys.setlocale("LC_TIME", "C")
 monthAxis(midmonth=TRUE, format="%b\n%Y", cex=cex, mgp=c(3,3.5,0))
 abline(h=1:6*100, v=dis2$date[format(dis2$date,"%d")=="01"], col=8)
 for(s in stat) lines(dis2$date, dis2[,s], lwd=4, col=col[s])
 xi <- seqR(sel,len=length(stat)+2)[-1]
 xi <- head(xi, -1)
 textField(dis2$date[xi], diag(as.matrix(dis2[xi,stat])), stat, cex=cex, col=col)
 box()
 if(map) berryFunctions::smallPlot(
    {
    OpenStreetMap::plot.OpenStreetMap(map_thames, removeMargin=FALSE)
    pts <- OSMscale::projectPoints(lat,lon,meta,
           to=map_thames$tiles[[1]]$projection)
   points(pts, pch=3, cex=4, lwd=5, col=col)
   },
   mar=0, bg=NA, border=NA, x1=0, x2=0.5, y1=0.8, y2=1)
 if(vlcnote) mtext("VLC: 'e': single frame forward\n'SHIFT+LEFT': few seconds back",
                   side=3, line=-9, outer=TRUE, adj=0.95, cex=cex)
}

par(mar=c(5,5,0.5,0.5), mgp=c(3,0.7,0))
scene(47200)
```

Actual movie

```
library(animation) ; library(pbapply)
saveVideo({par(mar=c(6,8,1,1), mgp=c(5.5,0.7,0))
 dummy <- pbsapply(seq(47000, by=3, len=100), scene, cex=2); rm(dummy)
}, video.name="Qmovie.mp4", interval=0.1, ffmpeg="/usr/bin/ffmpeg",
ani.width=7*200, ani.height=5*200)
```

Open video within browser

top

Hydmod

Hydrological modelling with `airGR`
Katie Smith Centre for Ecology & Hydrology (k.a.smith@ceh.ac.uk)

This is an demonstration of how to use the airGR package of hydrological models in R, as well as how to plot interactive timeseries graphs with the dygraphs package.

First we need to load some packages

```
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(dygraphs)
library(RColorBrewer)
```

Data

Now we'll load in some observational flow data from the River Thames (naturalised) in England - with thanks to the National River Flow Archive: http://nrfa.ceh.ac.uk/data/search

```
observed_data <- read.csv("data/Qobs_390010.csv")
head(observed_data)
```

```
##         DATE Qobs
## 1 01/01/1883  238
## 2 02/01/1883  280
## 3 03/01/1883  292
## 4 04/01/1883  292
## 5 05/01/1883  276
## 6 06/01/1883  272
```

```
observed_data$DATE <- strptime(observed_data$DATE, format = "%d/%m/%Y")
```

In order to plot this as an interactive dygraph we need to change it to xts format

```
observed_data_xts <- as.xts(observed_data$Qobs, order.by = observed_data$DATE)
dygraph(observed_data_xts, main="Naturalised Streamflow Observations for the Thames at Kingston")%>%
  dyAxis("y", label="Streamflow (m3/s")%>%
  dyOptions(colors = RColorBrewer::brewer.pal(3, "Set1")[2])%>%
  dyRangeSelector()
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

Now lets read in some precipitation data - this is from CEH-GEAR: https://data.gov.uk/dataset/gridded-estimates-of-daily-and-monthly-areal-rainfall-for-the-united-kingdom-1890-2012-ceh-gear

```
precip_data <- read.csv("data/rain_1961_2014_390010.csv")
head(precip_data)
```

```
##         DATE Mean_rainfall
## 1 1961-01-01    8.63763603
## 2 1961-01-02    4.57095929
## 3 1961-01-03    2.10149133
## 4 1961-01-04    0.02007255
## 5 1961-01-05    7.18590286
## 6 1961-01-06    0.06409714
```

and some potential evapotranspiration data - this is from CHESS-PE:https://data.gov.uk/dataset/climate-hydrology-and-ecology-research-support-system-potential-evapotranspiration-dataset-for-1

```
PET_data <- read.csv("data/CHESS_PET_1961_2015_390010.csv")
head(PET_data)
```

```
##         DATE       PET
## 1 1961-01-01 0.3668075
## 2 1961-01-02 0.3534552
## 3 1961-01-03 0.4122767
## 4 1961-01-04 0.3197117
## 5 1961-01-05 0.6122423
## 6 1961-01-06 0.3556202
```

Note that our starting dates are not the same as our observational data, so we need to make a dataframe that matches the dates up. There are a lot of ways to do this. First we'll convert them to the same date format.

```
precip_data$DATE <- strptime(precip_data$DATE, "%Y-%m-%d")
PET_data$DATE <- strptime(PET_data$DATE, "%Y-%m-%d")
```

now we'll find the common period

```
first_date <- max(observed_data$DATE[1], precip_data$DATE[1], PET_data$DATE[1])
last_date <- min(observed_data$DATE[length(observed_data$DATE)], precip_data$DATE[length(precip_data$DA
```

and make a data frame of that length

```
# make an empty data frame
thames_data <- as.data.frame(matrix(NA,nrow=as.numeric((last_date-first_date)+1), ncol=4))
colnames(thames_data) <-c ("date","PET","precip","obs")
# make the date timeseries
thames_data$date <- seq(first_date, last_date, by="days")
# populate the data frame with the data
thames_data$obs <- observed_data$Qobs[which(observed_data$DATE==thames_data$date[1]):which(observed_data
thames_data$precip <- precip_data$Mean_rainfall[which(precip_data$DATE==thames_data$date[1]):which(preci
thames_data$PET <- PET_data$PET[which(PET_data$DATE==thames_data$date[1]):which(PET_data$DATE==thames_da
```

Interactive time series plot

plot the observed streamflow with the precipitation data

```
#  convert the observed discharge to runoff (so its in the same units as the precip)
# divide by catchment area (m2) and mulitply by 86.4
thames_data$obs <- (thames_data$obs/9948.0)*86.4
thames_data_xts <- as.xts(thames_data[,3:4], order.by=thames_data$date)
# initiate the dygraph
dygraph(thames_data_xts)%>%
# define the first axis
dyAxis("obs", name = "y", label = "runoff (mm/day)",
       valueRange = range(thames_data_xts[, "obs"],
                          na.rm = TRUE)* c(0.01, 1.59))%>%
# define the second axis
dyAxis("precip", name = "y2", label = "precip (mm/day)",
                   valueRange = rev(range(thames_data_xts[, "precip"],
                   na.rm = TRUE)* c(0.01, 2.99)))%>%
# plot the data
dySeries("obs",axis = 'y')%>%
dySeries("precip", axis = 'y2', stepPlot = TRUE,
         fillGraph = TRUE)%>%
dyOptions(colors = RColorBrewer::brewer.pal(3, "Set1")[2:3])%>%
dyRangeSelector()
```

Hydrological modeling

OK, enough messing with data, lets do some modelling. We are using the GR4J model, from the airGR package. This was developed by IRSTEA, France. See Perrin et al. (2003)

See this website for a good guide through the model: https://odelaigue.github.io/airGR/tutorial_1_getting_started.html

load the GR package

```
require(airGR, quietly=TRUE)
```

prepare the input data in the correct format

```
BasinObs <- thames_data
colnames(BasinObs) <- c('DatesR','E','P', 'Qobs')
```

create the InputsModel object - this defines which model we want to run, and defines the variables for the models input data

```
InputsModel <- CreateInputsModel(FUN_MOD = RunModel_GR4J,DatesR = BasinObs$DatesR,
                                 Precip = BasinObs$P,PotEvap = BasinObs$E)
str(InputsModel)
```

```
## List of 3
##  $ DatesR : POSIXlt[1:19723], format: "1961-01-01" "1961-01-02" ...
##  $ Precip : num [1:19723] 8.6376 4.571 2.1015 0.0201 7.1859 ...
##  $ PotEvap: num [1:19723] 0.367 0.353 0.412 0.32 0.612 ...
##  - attr(*, "class")= chr [1:3] "InputsModel" "daily" "GR"
```

```
# note NA values of precip and PET are NOT ALLOWED
```

create the RunOptions object - this defines options for the RunModel_GR4J function

```
# first define the period to run the model over
Ind_Run <- seq(which(BasinObs$DatesR=="1981-01-01"),
               which(BasinObs$DatesR=="2014-12-31"))
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run,
                               IndPeriod_WarmUp = NULL)
```

```
## Warning in CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel, :     Model warm-up p
##      The year preceding the run period is used
```

```
## Warning in CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel, :     Model states in
```

```
str(RunOptions)
```

```
## List of 6
##  $ IndPeriod_WarmUp: int [1:365] 6941 6942 6943 6944 6945 6946 6947 6948 6949 6950 ...
##  $ IndPeriod_Run   : int [1:12418] 7306 7307 7308 7309 7310 7311 7312 7313 7314 7315 ...
##  $ IniStates       : num [1:67] 0 0 0 0 0 0 0 0 0 0 ...
##  $ IniResLevels    : num [1:2] 0.3 0.5
##  $ Outputs_Cal     : chr "Qsim"
##  $ Outputs_Sim     : chr [1:16] "DatesR" "PotEvap" "Precip" "Prod" ...
##  - attr(*, "class")= chr [1:3] "RunOptions" "GR" "daily"
```

create the InputsCrit object - define the error metric (choose from RMSE, NSE, KGE or modified KGE (KGE2))

```
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_NSE,
                               InputsModel = InputsModel,
                               RunOptions = RunOptions,
                               Qobs = BasinObs$Qobs[Ind_Run])
str(InputsCrit)
```

```
## List of 5
##  $ BoolCrit  : logi [1:12418] TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ Qobs      : num [1:12418] 0.635 0.623 0.587 0.571 0.566 ...
##  $ transfo   : chr ""
##  $ Ind_zeroes: NULL
##  $ epsilon   : NULL
##  - attr(*, "class")= chr "InputsCrit"
```

create the CalibOptions object - choose the calibration algorithm

```
CalibOptions <- CreateCalibOptions(FUN_MOD = RunModel_GR4J,
                                   FUN_CALIB = Calibration_Michel)
str(CalibOptions)
```

```
## List of 3
##  $ FixedParam       : logi [1:4] NA NA NA NA
##  $ SearchRanges     : num [1:2, 1:4] 4.59e-05 2.18e+04 -1.09e+04 1.09e+04 4.59e-05 ...
##  $ StartParamDistrib: num [1:3, 1:4] 169.017 247.151 432.681 -2.376 -0.649 ...
##  - attr(*, "class")= chr [1:3] "CalibOptions" "GR4J" "HBAN"
```

run the calibration

```
OutputsCalib <- Calibration_Michel(InputsModel = InputsModel,
                                   RunOptions = RunOptions,
                                   InputsCrit = InputsCrit,
```

```
                                    CalibOptions = CalibOptions,
                                    FUN_MOD = RunModel_GR4J,
                                    FUN_CRIT = ErrorCrit_NSE)
```

```
## Grid-Screening in progress (
## 0%
## 20%
## 40%
## 60%
## 80%
##  100%)
##   Screening completed (81 runs)
##       Param =  432.681 ,   -0.649 ,   83.096 ,    2.384
##       Crit NSE[Q]       = 0.8923
## Steepest-descent local search in progress
##   Calibration completed (19 iterations, 216 runs)
##       Param =  607.894 ,   -0.734 ,   87.357 ,    2.315
##       Crit NSE[Q]       = 0.9246
```

NSE of 0.9246 - not bad at all!

define the parameters found by the calibration routine

```
Param <- OutputsCalib$ParamFinalR
Param
```

```
## [1] 607.8936811  -0.7336304  87.3567230   2.3153153
```

RUN THE MODEL!

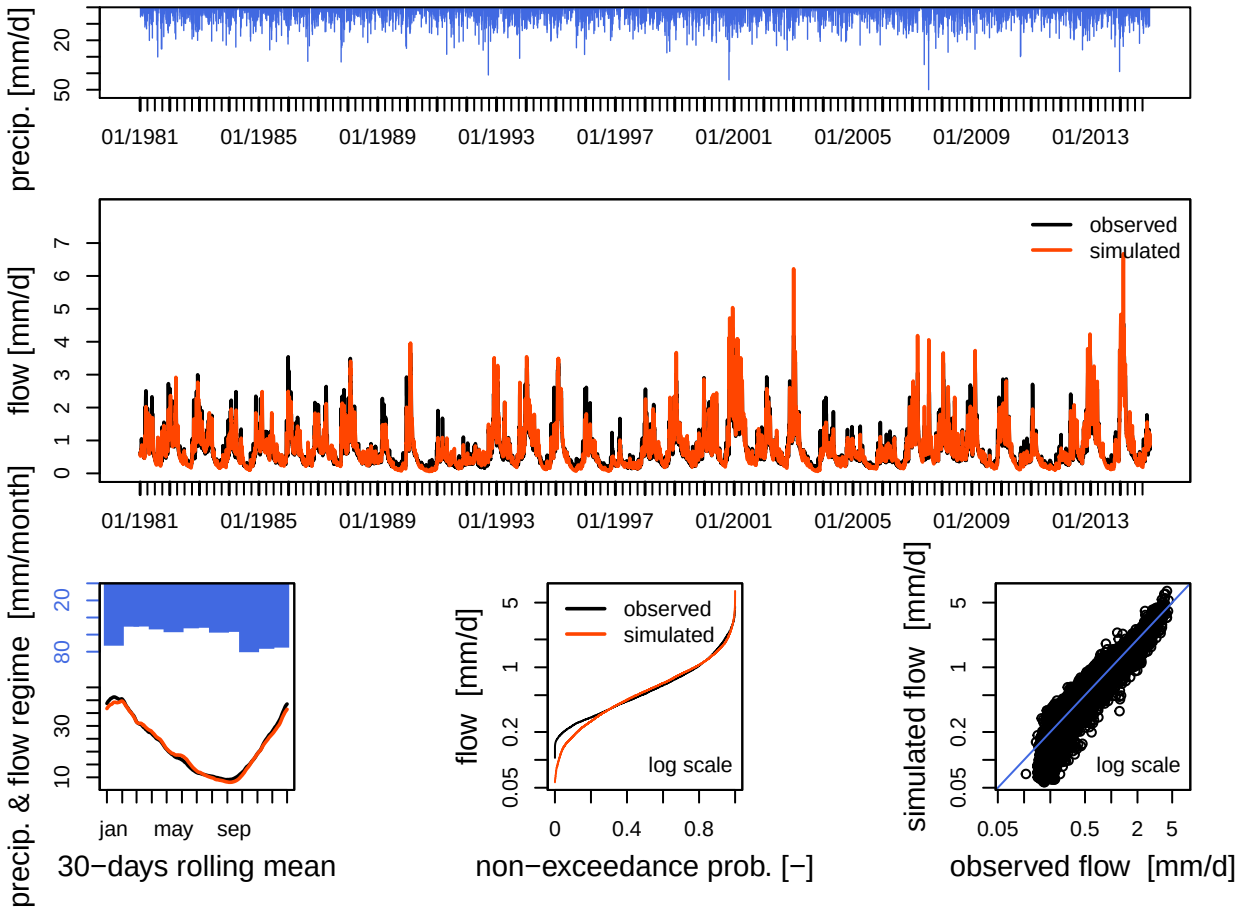
```
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions,
                              Param= Param)
str(OutputsModel)
```

```
## List of 16
##  $ DatesR  : POSIXlt[1:12418], format: "1981-01-01 00:00:00" "1981-01-02 00:00:00" ...
##  $ PotEvap : num [1:12418] 1.347 0.38 0.795 0.57 0.454 ...
##  $ Precip  : num [1:12418] 0.0972 0.1211 0.0523 0.1147 0.3248 ...
##  $ Prod    : num [1:12418] 352 351 350 350 349 ...
##  $ AE      : num [1:12418] 1.127 0.334 0.663 0.488 0.43 ...
##  $ Perc    : num [1:12418] 0.387 0.383 0.378 0.374 0.372 ...
##  $ PR      : num [1:12418] 0.387 0.383 0.378 0.374 0.372 ...
##  $ Q9      : num [1:12418] 0.355 0.35 0.345 0.341 0.338 ...
##  $ Q1      : num [1:12418] 0.0398 0.0393 0.0387 0.0382 0.0378 ...
##  $ Rout    : num [1:12418] 42.2 42 41.7 41.4 41.2 ...
##  $ Exch    : num [1:12418] -0.0592 -0.0577 -0.0564 -0.0551 -0.0539 ...
##  $ AExch   : num [1:12418] -0.099 -0.097 -0.0951 -0.0933 -0.0917 ...
##  $ QR      : num [1:12418] 0.599 0.578 0.559 0.542 0.526 ...
##  $ QD      : num [1:12418] 0 0 0 0 0 ...
```

```
##  $ Qsim    : num [1:12418] 0.599 0.578 0.559 0.542 0.526 ...
##  $ StateEnd: num [1:67] 374.4 45.5 NA NA NA ...
##  - attr(*, "class")= chr [1:3] "OutputsModel" "daily" "GR"
```

use the inbuilt plot function to look at the results

```
plot(OutputsModel, Qobs=BasinObs$Qobs[Ind_Run])
```



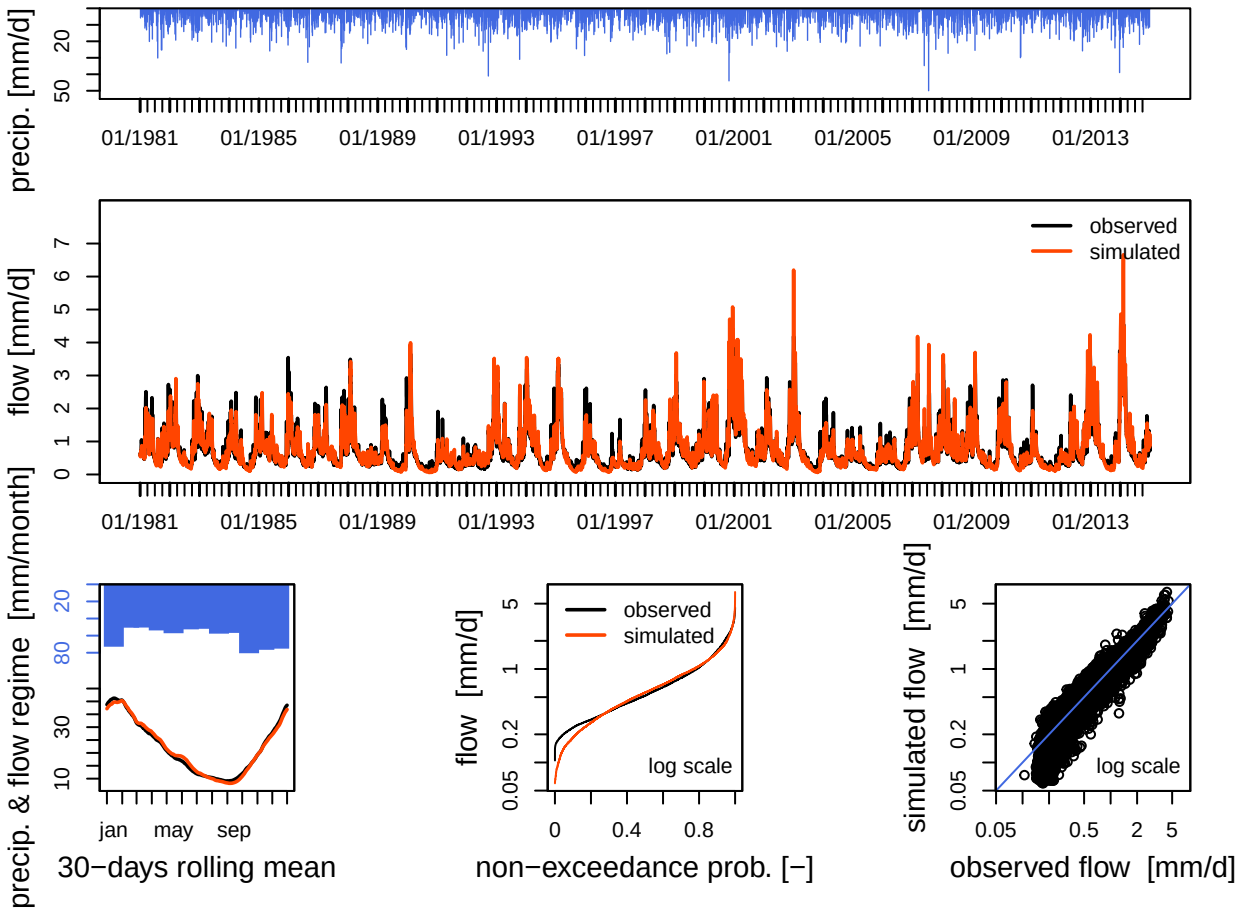
looking good - but we've got some discrepancy at the low flows end. NSE is notorious for this, it is based on the square of the flows, so over-weights the calibration to the high flows. I wonder if the modified KGE can do any better?

```
# make a few changes to the calibration criteria
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_KGE2,
                               InputsModel = InputsModel,
                               RunOptions = RunOptions,
                               Qobs = BasinObs$Qobs[Ind_Run])
# rerun the calibration
OutputsCalib <- Calibration_Michel(InputsModel = InputsModel,
                                   RunOptions = RunOptions,
                                   InputsCrit = InputsCrit,
                                   CalibOptions = CalibOptions,
                                   FUN_MOD = RunModel_GR4J,
                                   FUN_CRIT = ErrorCrit_KGE2)
```

```
## Grid-Screening in progress (
## 0%
```

```
## 20%
## 40%
## 60%
## 80%
##  100%)
##   Screening completed (81 runs)
##       Param =  432.681 ,   -0.649 ,   83.096 ,    2.384
##       Crit KGE'[Q]      = 0.8589
## Steepest-descent local search in progress
##   Calibration completed (51 iterations, 501 runs)
##       Param =  598.748 ,   -0.690 ,   93.679 ,    2.297
##       Crit KGE'[Q]      = 0.9621
```

```r
# redefine the parameters
Param <- OutputsCalib$ParamFinalR
# rerun the model
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions,
                              Param= Param)
# plot again
plot(OutputsModel, Qobs=BasinObs$Qobs[Ind_Run])
```



not much different. Oh well, we can be happy with either of those metric scores. - pause for thought - which parameter set would you choose to use?!

Validation

Let's do some validation

```
# go back to the beginning, redefine the period to run on (the period we haven't used for calibration,
Ind_Run <- seq(which(BasinObs$DatesR=="1962-01-01"),
               which(BasinObs$DatesR=="1980-12-31"))
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run,
                               IndPeriod_WarmUp = NULL)
```

```
## Warning in CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel, :     Model warm-up p
##      The year preceding the run period is used
```

```
## Warning in CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel, :     Model states in
```

```
InputsCrit <- CreateInputsCrit(FUN_CRIT = ErrorCrit_KGE2,
                               InputsModel = InputsModel,
                               RunOptions = RunOptions,
                               Qobs = BasinObs$Qobs[Ind_Run])
Param <- OutputsCalib$ParamFinalR
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
```

```
                              RunOptions = RunOptions,
                              Param= Param)
OutputsCrit <- ErrorCrit_KGE2(InputsCrit = InputsCrit,
                               OutputsModel = OutputsModel)
```

```
## Crit. KGE'[Q] = 0.9039
```

```
##  SubCrit. KGE'[Q] cor(sim, obs, "pearson") = 0.9466
##  SubCrit. KGE'[Q] sd(sim)/sd(obs)          = 0.9253
##  SubCrit. KGE'[Q] mean(sim)/mean(obs)      = 1.0282
```

slightly worse than the calibration period (0.9621) but not bad at all

finally, let's run the model for the whole time period and plot a dygraph so we can look at the timeseries in more detail

```
Ind_Run <- seq(which(BasinObs$DatesR=="1962-01-01"),
               which(BasinObs$DatesR=="2014-12-31"))
RunOptions <- CreateRunOptions(FUN_MOD = RunModel_GR4J,
                               InputsModel = InputsModel,
                               IndPeriod_Run = Ind_Run,
                               IndPeriod_WarmUp = NULL)
```

```
## Warning in CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel, :     Model warm-up p
##      The year preceding the run period is used
```

```
## Warning in CreateRunOptions(FUN_MOD = RunModel_GR4J, InputsModel = InputsModel, :     Model states in
```

```
Param <- OutputsCalib$ParamFinalR
OutputsModel <- RunModel_GR4J(InputsModel = InputsModel,
                              RunOptions = RunOptions,
                              Param= Param)
plot_output_data <- as.data.frame(matrix(NA, ncol = 3,
                                         nrow = length(OutputsModel$DatesR)))
colnames(plot_output_data) <- c("Date", "Qsim", "Qobs")
plot_output_data$Date <- OutputsModel$DatesR
plot_output_data$Qsim <- OutputsModel$Qsim
plot_output_data$Qobs <- BasinObs$Qobs[Ind_Run]
plot_output_data_xts <- as.xts(plot_output_data, order.by = plot_output_data$Date)
dygraph(plot_output_data_xts, main="Observed and Simulated Runoff for the Thames at Kingston (Naturalise
  dyOptions(colors = RColorBrewer::brewer.pal(3,"Set1")[2:1])%>%
  dyAxis("y", label="Runoff (mm/day)")%>%
  dyRangeSelector()
```

Thanks for listening! Hope you get to try it out.

Any general questions? Please feel free to post them on facebook page: https://www.facebook.com/groups/1130214777123909/?ref=bookmarks

GR specific questions? Email airGR@irstea.fr

top

Trend

Exploratory Data Analysis including flow duration curve and trend analysis on time-series **Shaun Harrigan** Centre for Ecology & Hydrology (shauhar@ceh.ac.uk)

---

1. Introduction

There is no 'one best' programming language, each has individual benefits. R's strong point is importing messy 'real-world' data, cleaning it up, and exploring it with powerful statistical tools!

In this short 'Trend Analysis taster' we're going to:

- **Read** and clean up streamflow data from the Boyne catchment in Ireland
- Run an **Exploratory Data Analysis (EDA)**
  - Flow Duration Curve (FDC)
  - Check statistical assumptions
- **Plot** time-series
- Perform a **Mann-Kendall** trend test for evidence of gradual change
- Perform a **Pettitt** point test for evidence of abrupt change

The Boyne Case Study

We will use the Boyne catchment in Ireland as a case study. This catchment has a well known abrupt change in its time-series (Harrigan et al., 2014 HESS) and this case study now features in "the Dirty Dozen of freshwater science" by Wilby et al., 2017 WIREs: Water.

OK, lets go get the data...

---

2. Read in data

Data taken from HESS paper:

- Obs = Observed daily mean flow (m3/s) data from 01/01/1952 to 31/12/2009
- Sim = Simulated daily mean flow (m3/s) data from 01/01/1952 to 31/12/2009 using ensemble mean across 3 hydrological models and 328 parameter sets

Observed streamflow data provided by the Office of Public Works (OPW) in Ireland. Please refer to their Ts&Cs before re-use.

A .csv of the data used in this tutorial can be found here.

```
# Load required packages
library(trend) # Non-Parametric trend tests and change point detection (by Thorsten Pohlert)
library(hydroTSM) # Time series analysis for hydrological modelling (by Mauricio Zambrano Bigiarini )

# Read data
myData <- read.csv("data/Boyne_HESS_data_EGU2017.csv")

# Take a peak
head(myData) # Fist 6 rows
```

```
##   Day Month Year     Obs      Sim
## 1   1     1 1952 100.201 114.0215
## 2   2     1 1952 138.781 126.2769
## 3   3     1 1952 136.810 126.6293
## 4   4     1 1952 129.432 126.5246
## 5   5     1 1952 133.332 115.6216
```

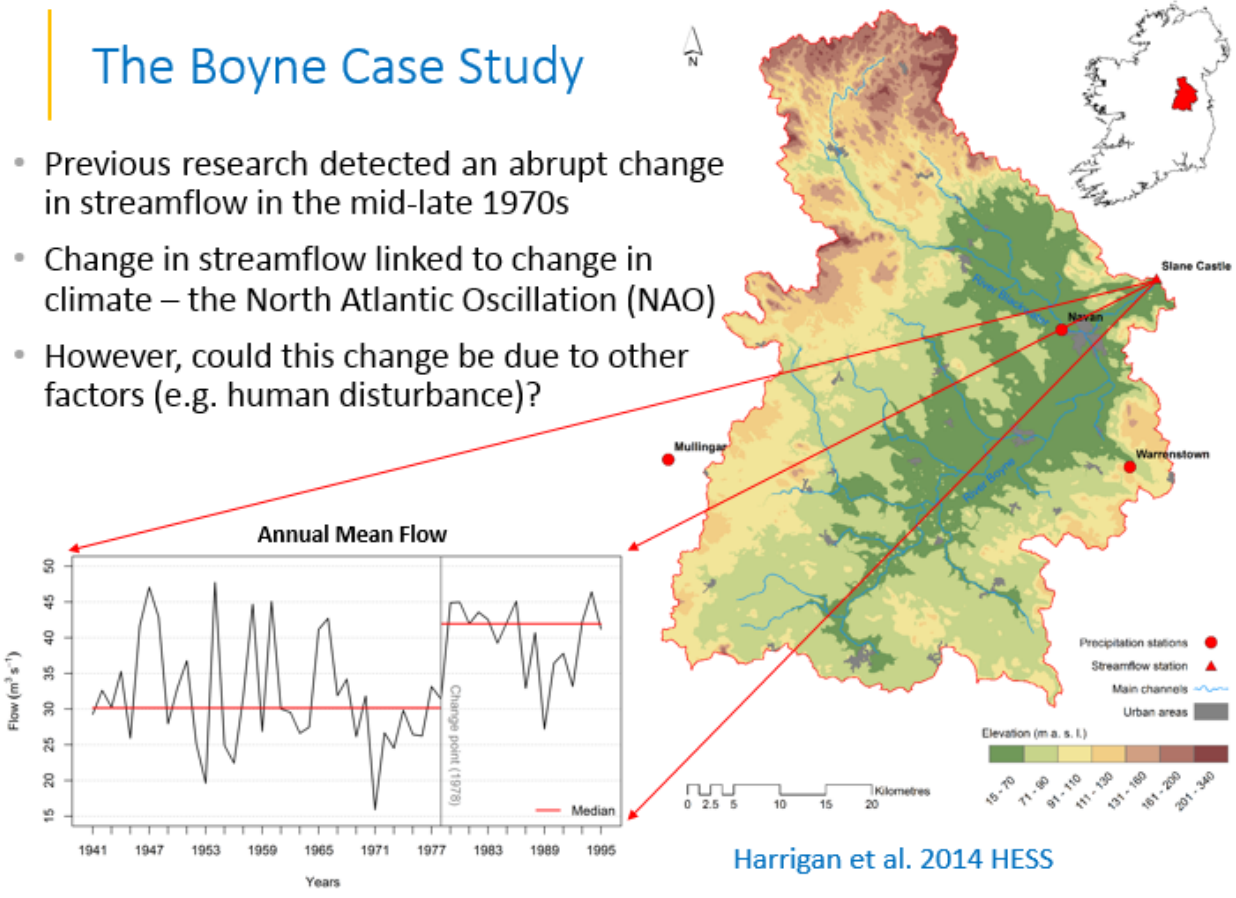

The Boyne Case Study
Previous research detected an abrupt change in streamflow in the mid-late 1970s
Change in streamflow linked to change in climate – the North Atlantic Oscillation (NAO)
However, could this change be due to other factors (e.g. human disturbance)?
Annual Mean Flow
Flow (m³ s⁻¹)
Years
Change point (1978)
Median
Slane Castle
Navan
Mullingar
Warrenstown
Precipitation stations
Streamflow station
Main channels
Urban areas
Elevation (m a. s. l.)
Kilometres
Harrigan et al. 2014 HESS

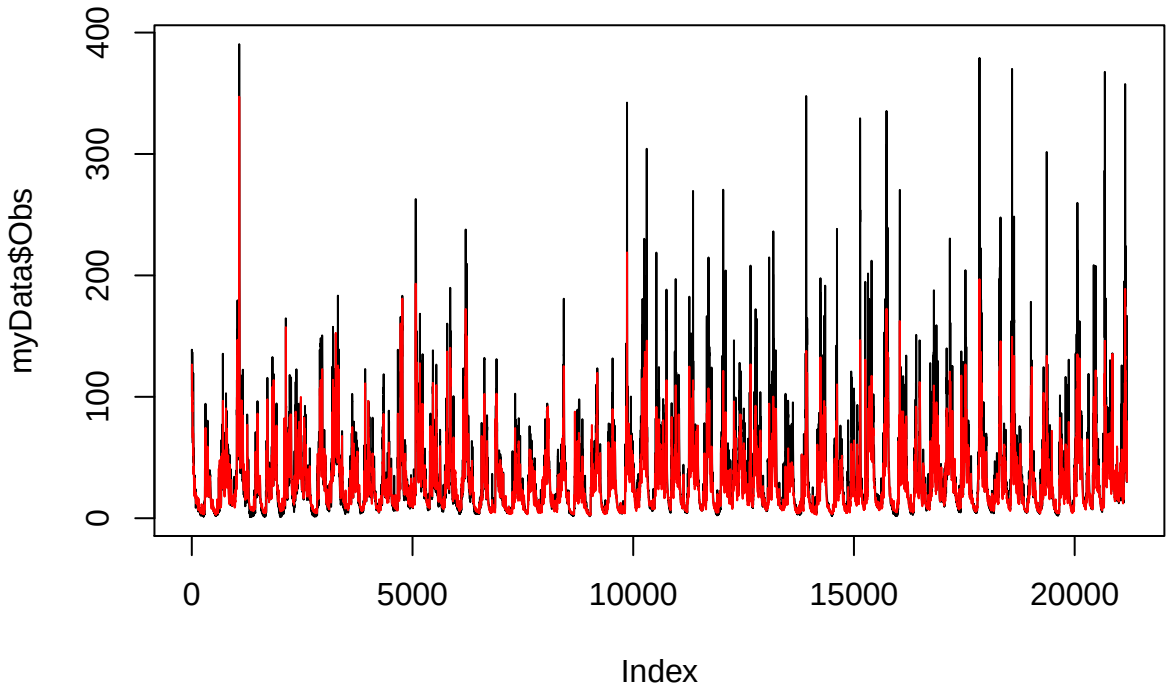
Figure 4:

```
## 6   6     1 1952 116.430 102.9900
```

```
tail(myData) # Last 6 rows
```

```
##       Day Month Year      Obs      Sim
## 21180  26    12 2009  31.1254 32.48449
## 21181  27    12 2009  30.3919 31.39742
## 21182  28    12 2009  29.8448 30.19816
## 21183  29    12 2009  29.7872 48.27515
## 21184  30    12 2009  81.2440 73.01119
## 21185  31    12 2009 166.6490 80.06223
```

```
# Quick line plot of Obs and Sim (type = 'l')
plot(myData$Obs, type = 'l')
lines(myData$Sim, col = "red") # 'lines' will add a line to active plot device
```



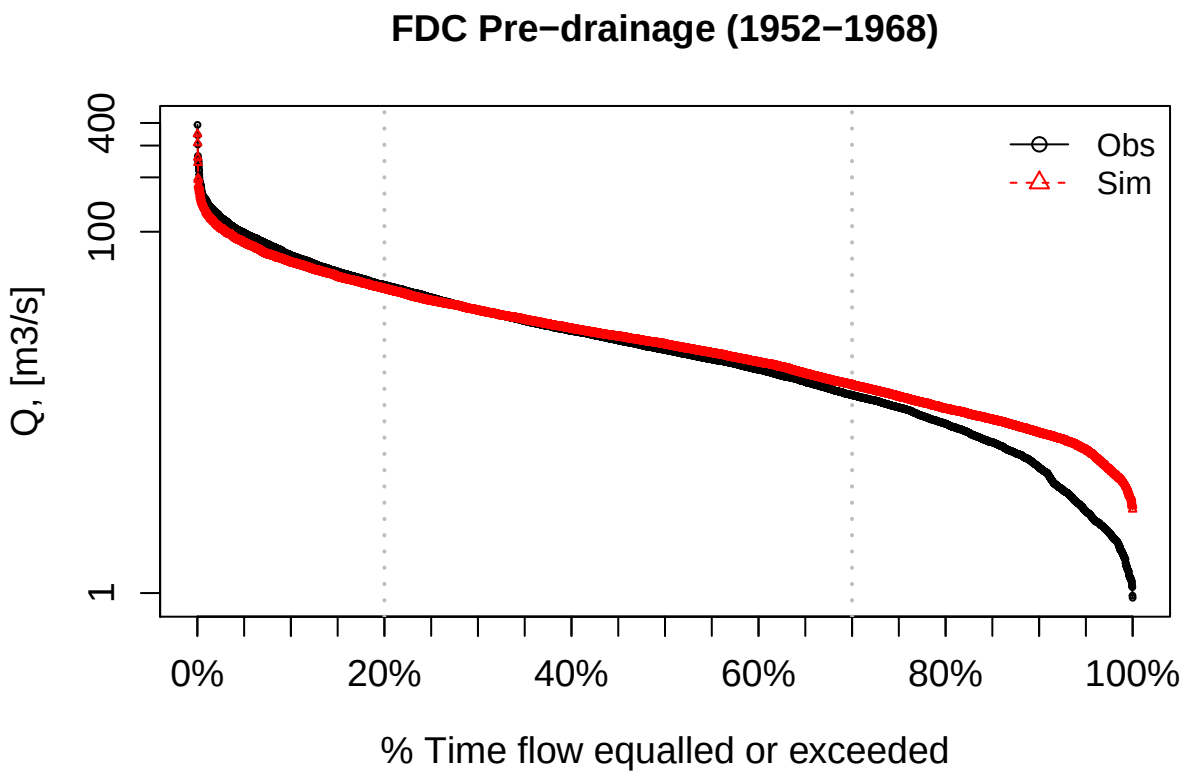
---

3. Flow Duration Curves (FDC)

We can create a FDC easily using the `fdc` function from the **hydroTSM** package. Here we will have a look at FDCs from both observed and simulated Boyne streamflow series, pre and post the known period of channel drainage (1969-1986):

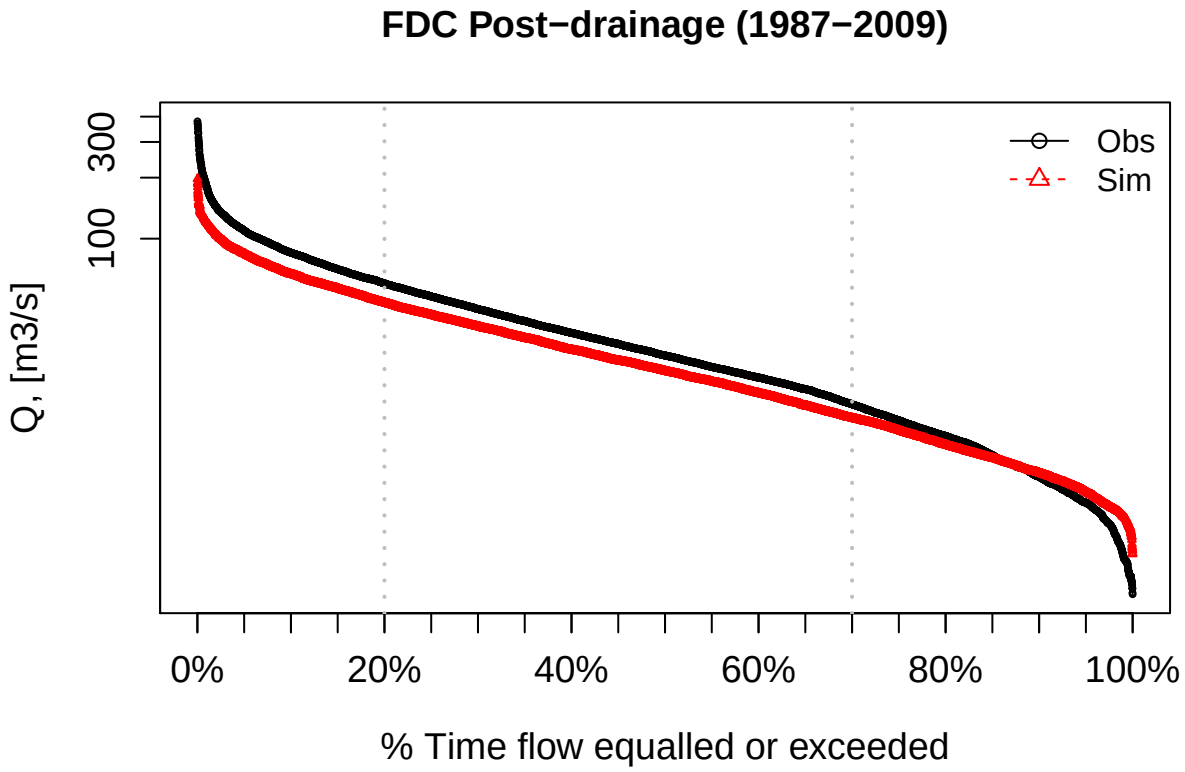
```
# Subset data pre- and post- drainage period
preDrain <- subset(myData, myData$Year >= 1952 & myData$Year <= 1968)
postDrain <- subset(myData, myData$Year >= 1987 & myData$Year <= 2009)

# Select 'Obs' and 'Sim' columns and convert to 'matrix' class (required in 'fdc' function)
preDrainFDC <- as.matrix(preDrain[ ,c(4, 5)])
postDrainFDC <- as.matrix(postDrain[ ,c(4, 5)])

# Plot pre-drainage FDC and print first 10 values
```

```
fdc(preDrainFDC, main = "FDC Pre-drainage (1952-1968)", verbose = FALSE)[c(1:10)]
```




```
##  [1] 0.04750403 0.01272142 0.01417069 0.01867955 0.01578100 0.02898551
##  [7] 0.04573269 0.06392915 0.04637681 0.03220612
```

```
# Plot post-drainage FDC and print first 10 values
fdc(postDrainFDC, main = "FDC Post-drainage (1987-2009)", verbose = FALSE)[1:10]
```




```
##  [1] 0.08608081 0.08680014 0.12264716 0.14542621 0.14926268 0.17623786
##  [7] 0.21376334 0.24505455 0.24085841 0.20441194
```

---

4. Exploratory Data Analysis (EDA)

EDA is the first step in any trend analysis, and its importance is often under appreciated. It serves 3 key purposes:

1. Visually identifying interesting patterns in the data
2. Detecting data quality problems (missing data, outliers etc.)
3. Testing key assumptions for statistical tests used later (e.g. Normality and serial correlation)

We will run a number of standard EDA tests and plots using R's built in functions:

- Extract desired hydrological indicators (`aggregate`, `mean`)
- Time-series plots (`plot` with linear regression line [`lm`] and `lowess` smoothing)
- Histograms (`hist`)
- Q-Q plot to check normality (`qqnorm`; `qqline`)
- Shapiro-Wilk test for normality (`shapiro.test`)
- Autocorrelation Function (`acf`)

4.1 Extract hydrological indicators

We will keep it simple and extract **Annual Mean Flow (AMF)** from both observed and simulated series using `aggregate` with the `mean` function.

```
# Annual mean flow series - Obs (Note: na.rm = T is used to deal with NAs)
obsAMF <- aggregate(myData$Obs, by = list(myData$Year), FUN = function(x) { mean(x, na.rm = T)})
colnames(obsAMF) <- c("Year", "Flow")
```

```r
# Annual mean flow series - Sim
simAMF <- aggregate(myData$Sim, by = list(myData$Year), FUN = mean)
colnames(simAMF) <- c("Year", "Flow")

# Have a look!
head(obsAMF)
```

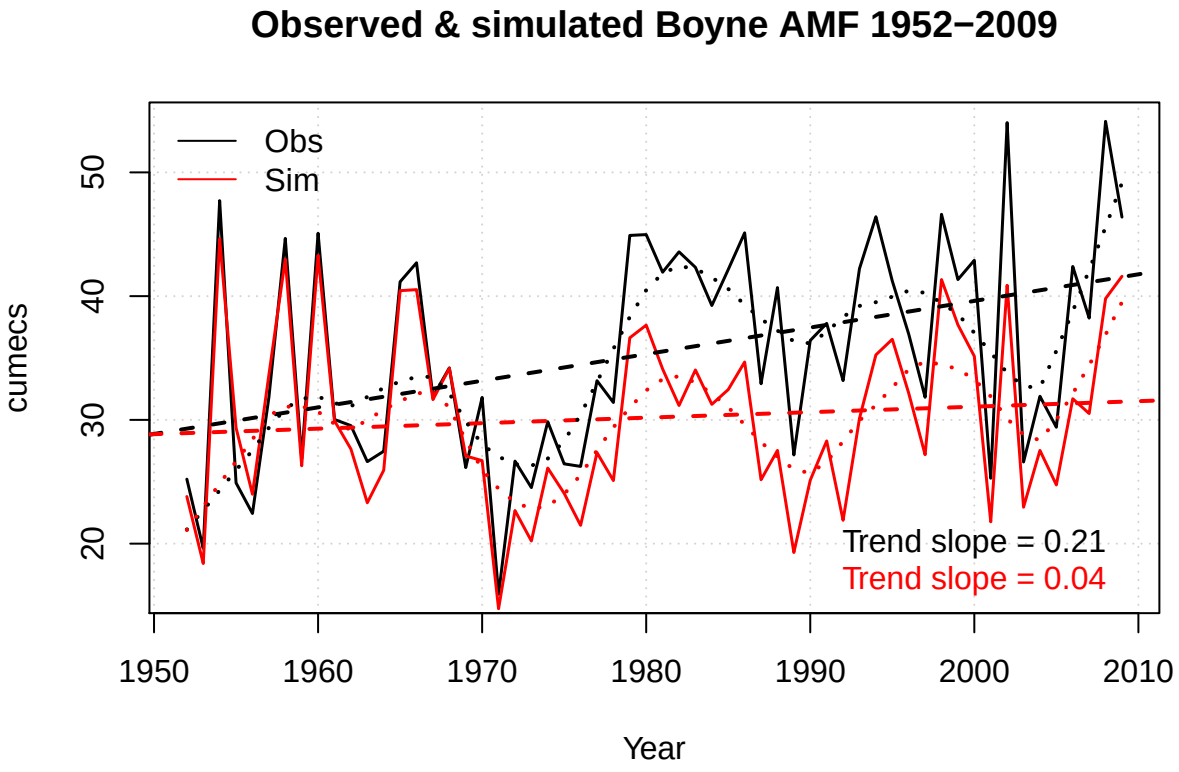
```
##   Year     Flow
## 1 1952 25.20947
## 2 1953 19.61620
## 3 1954 47.71661
## 4 1955 24.90779
## 5 1956 22.43712
## 6 1957 31.81649
```

```r
str(obsAMF)
```

```
## 'data.frame':    58 obs. of  2 variables:
##  $ Year: int  1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 ...
##  $ Flow: num  25.2 19.6 47.7 24.9 22.4 ...
```

4.2 Plot both observed and simulated series with linear and lowess lines

```r
# Timie-series plot of observed and simulated AMF
plot(obsAMF$Year, obsAMF$Flow, type = 'l', main = "Observed & simulated Boyne AMF 1952-2009", xlab = "Y

lines(simAMF$Year, simAMF$Flow, col = "red", lwd = 1.5)

# Linear regression
linRegOutObs <- lm(obsAMF$Flow~obsAMF$Year)
linRegOutSim <- lm(simAMF$Flow~simAMF$Year)

# Add linear regression line
abline(linRegOutObs, col = "black", lty = 2, lwd = 2)
abline(linRegOutSim, col = "red", lty = 2, lwd = 2)

# Add LOWESS line (f = smoother span paramter, i.e. proportion of data influencing smooth)
lines(lowess(obsAMF$Flow~obsAMF$Year, f = 0.2), col = "black", lty = 3, lwd = 2)
lines(lowess(simAMF$Flow~simAMF$Year, f = 0.2), col = "red", lty = 3, lwd = 2)


# Add legend to plot
legend("topleft",
       legend=c("Obs", "Sim"),
       col= c("black", "red"), lty=1,lwd=1, bty="n")

# Add text
text(2000, 20, paste("Trend slope = ", round(linRegOutObs$coefficients[2],2), sep = ""), col = "black")
text(2000, 17, paste("Trend slope = ", round(linRegOutSim$coefficients[2],2), sep = ""), col = "red")
```



4.3 Check for normality

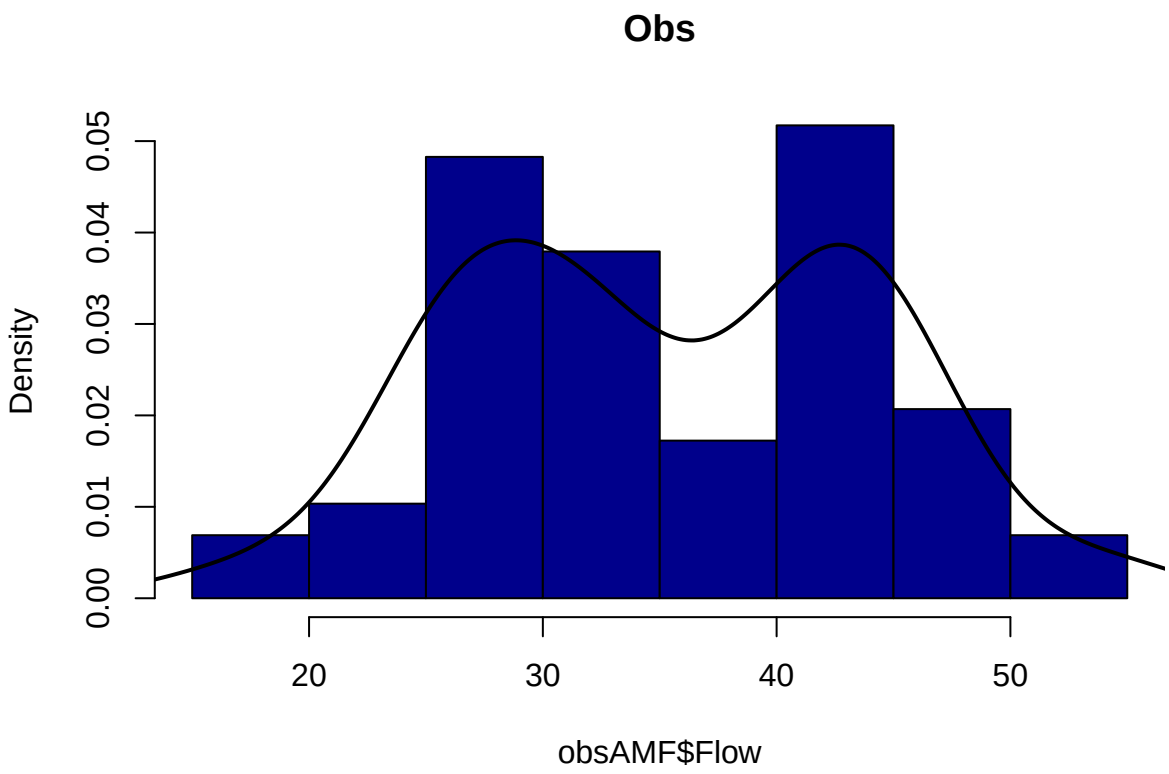
Many statistical tests require that the data follow normal (or Gaussian) distribution, for example linear regression.

However, often hydroclimatic data violate this assumption, so non-parametric tests are more widely applied (e.g. Mann-Kendall, Pettitt), as will be used below. However, it is still good practice to know the distribution of your data.

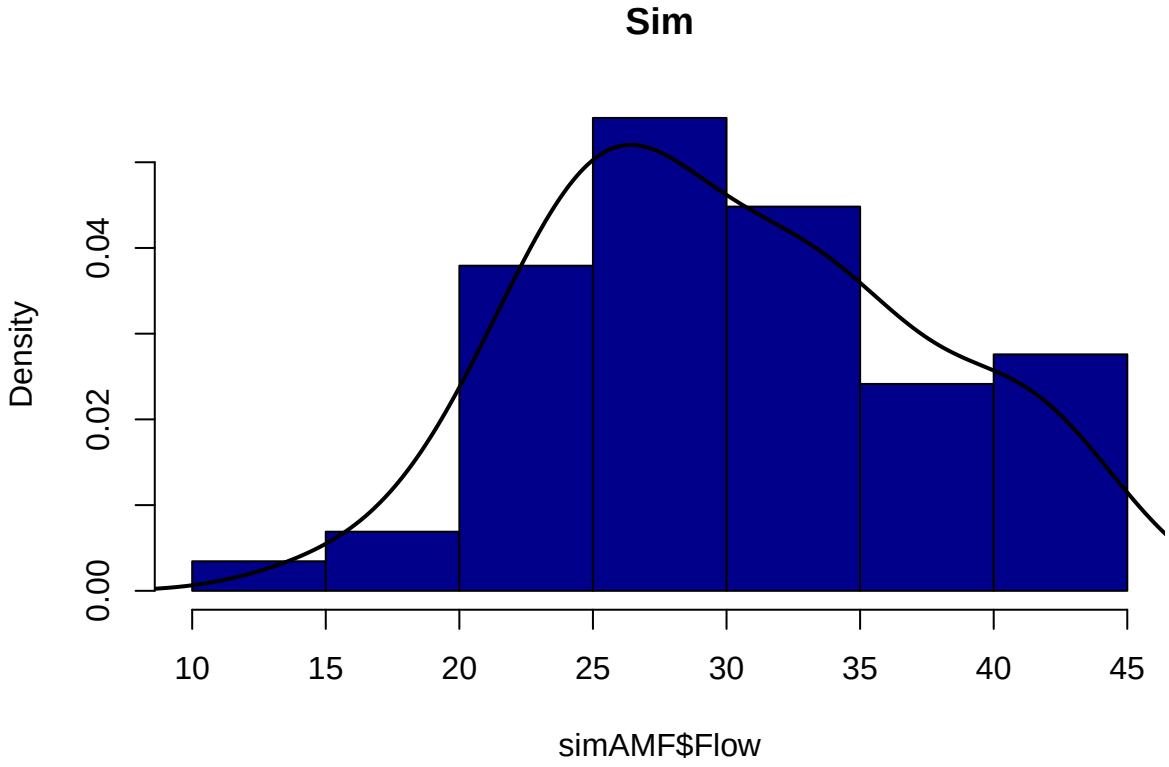
We'll use **histograms**, **Quantile-Quantile-plots (QQ plots)**, and the **shapiro-Wilk** test to examine the null hypothesis of normality.

Histograms

```
# First have a look at the historgram (prob = TRUE is for density)
# Obs histogram
hist(obsAMF$Flow, col = "darkblue", prob = TRUE, main = "Obs")
lines(density(obsAMF$Flow), lwd = 2)
```

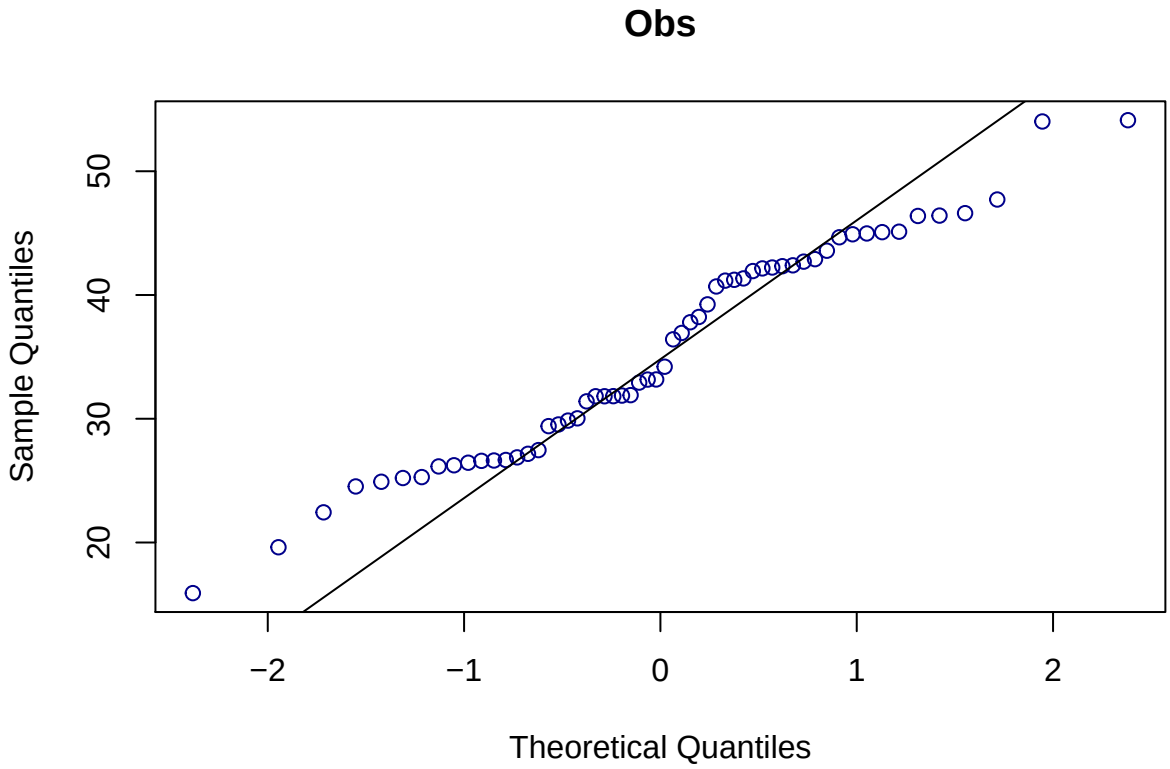



```
# Sim histogram
hist(simAMF$Flow, col = "darkblue", prob = TRUE, main = "Sim")
lines(density(simAMF$Flow), lwd = 2)
```

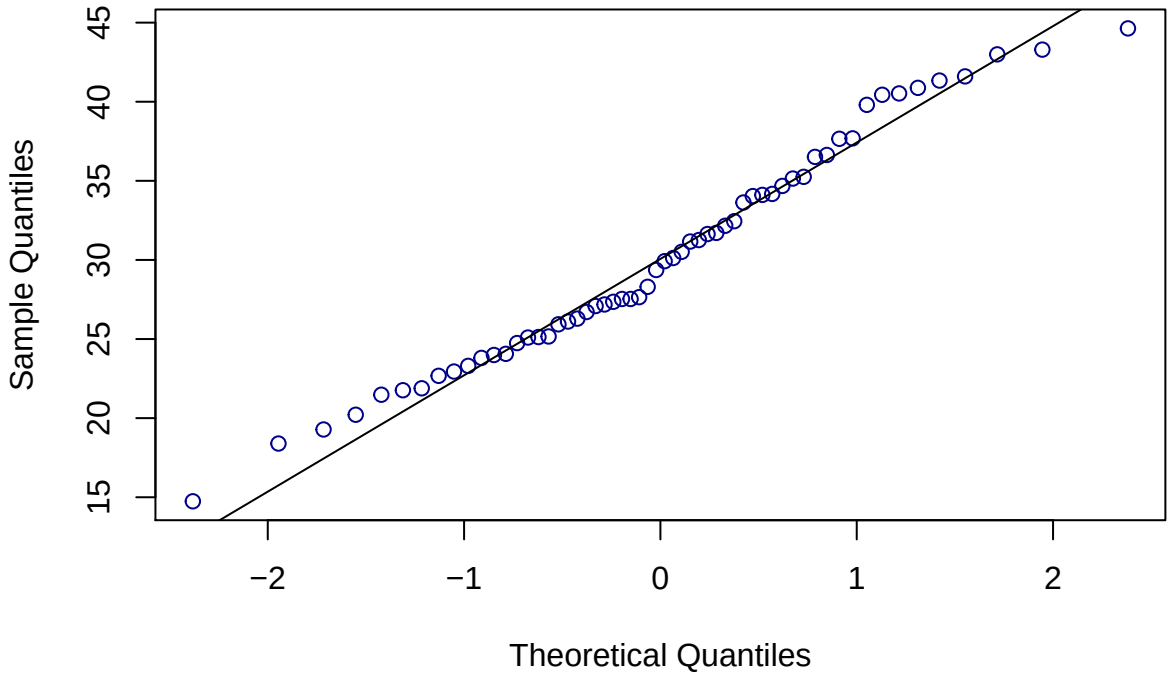


QQ-plots

```r
# QQplot for normality - Obs AMF
qqnorm(obsAMF$Flow, col = "darkblue", main = "Obs"); qqline(obsAMF$Flow, distribution = qnorm)
```




```r
# QQplot for normality - Sim AMF
qqnorm(simAMF$Flow, col = "darkblue", main = "Sim"); qqline(simAMF$Flow, distribution = qnorm)
```



Shapiro-Wilk test for normality

We will reject the null hypothesis of normality for p-value < 0.1 (according to Royston (1995) - R help page for `shapiro-test`).

```
# Shapiro-Wilk - Obs
shapiro.test(obsAMF$Flow)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  obsAMF$Flow
## W = 0.96334, p-value = 0.07707
```

```
# Shapiro-Wilk - Sim
shapiro.test(simAMF$Flow)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  simAMF$Flow
## W = 0.97729, p-value = 0.3464
```

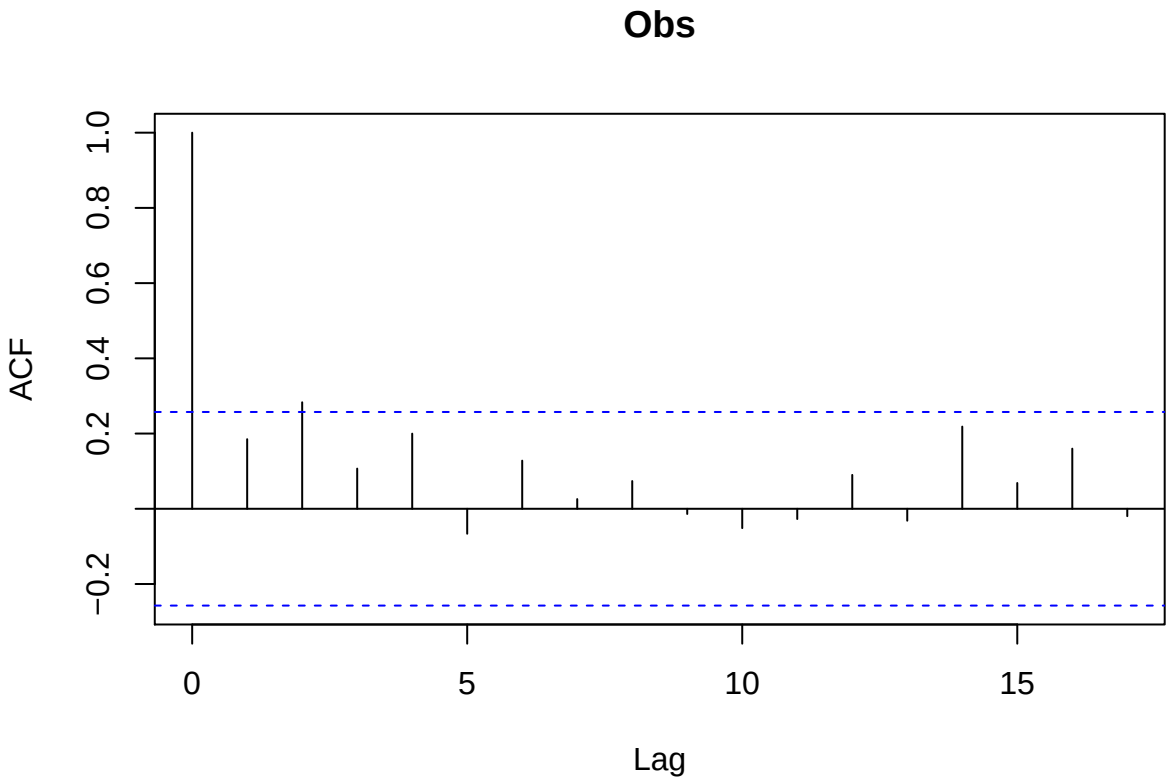
Conclusion on Normality: The simulated series can be considered broadly normally distributed, but the observed series fails the Shapiro-Wilk test and we can see from the histogram is bimodal.

4.4 Check for serial correlation

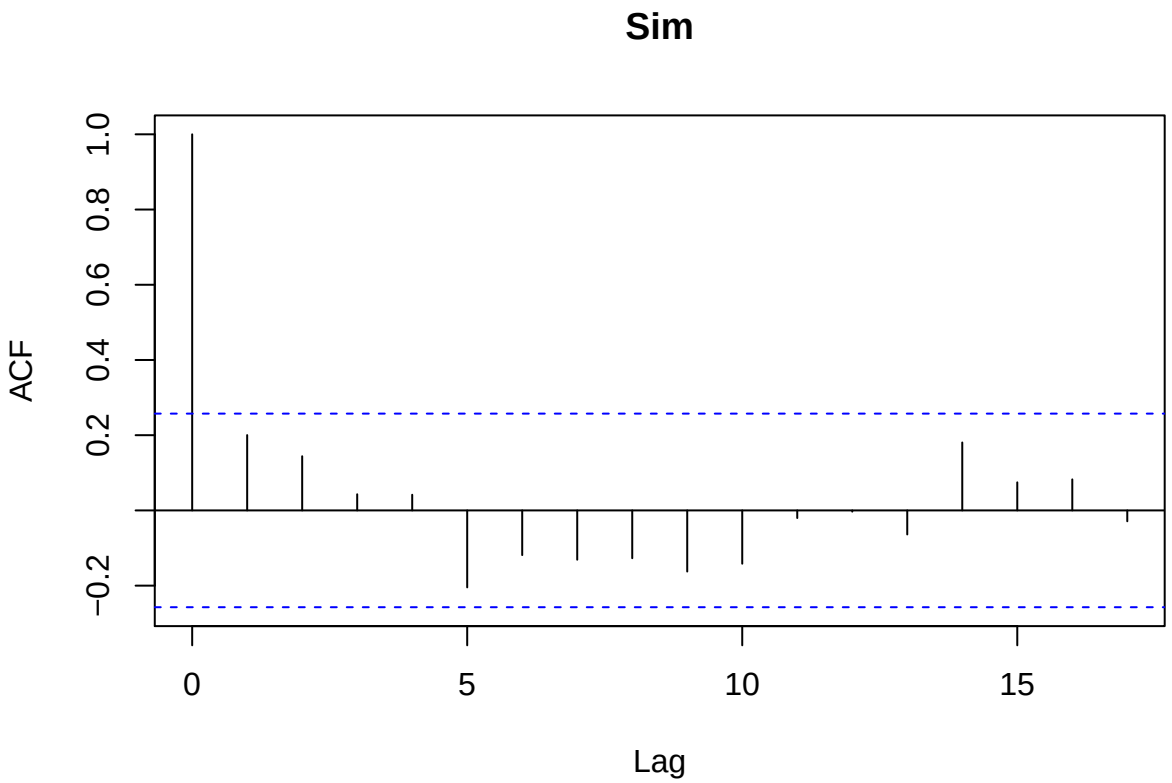
Presence of positive serial correlation (or autocorrelation) within a series can inflate Type 1 errors. That is, the flow from last year is somehow related to this year. This is common in slow responding groundwater catchments, or in regions that are strongly affected by large-scale climate process over multi-year periods (e.g. North Atlantic Oscillation (NAO)) in Europe. This tends to return too many *"statistically significant"* results, and thus misleading results.

We'll use the built in **autocorrelation function** `acf`. Typically we're only concerned if there is evidence of statistically significant positive lag-1 serial correlation at the 5% level (see dashed horizontal line in plots):

```
# ACF plot for testing for autocorrelation - Obs AMF
acf(obsAMF$Flow, main = "Obs")
```




```
# ACF plot for testing for autocorrelation - Sim AMF
acf(simAMF$Flow, main = "Sim")
```



Conclusion on Serial Correlation: There is no statistically significant lag-1 serial correlation, although some evidence at lag-2 in observed Annual Mean Flow.

5. Statistical trend and change point analysis

Our **EDA** has revealed a large apparent difference in pattern of change in observations compared to simulated. However, we can test if these patterns of change are statistically significant.

We'll use the freely available **trend** package for 3 tasks:

1. Calculatinig the relative % change in flow over the period-of-record using the robust Thei-Sen approach (TSA) (`sens.slope`)
2. Testing for gradual change in annual mean observed and simulated flow with the Mann-Kendall test (`mk.test`)
3. Testing for abrupt change in annual mean observed and simulated flow with the Pettitt test (`pettitt.test`)

5.1 Calculate trend magnitude using the relative Theil-Sen Approach (TSArel)

The non-parametric Theil-Sen estimator is a more robust measure of trend slope than linear regression and will give a much more accurate value in the presence of outliers.

The relative Theil-Sen Approach (TSArel) for calculation of trend magnitude (%) is given:

TSArel = $(\beta * n)$ / μ * 100

where β is the slope from Theil-Sen, n is the series length, and μ is the mean of the series.

```
# First the 'trend' package requires data are in 'ts' (time-series) format
obsAMFts <- as.ts(obsAMF)
simAMFts <- as.ts(simAMF)

# Calcualte Theil-Sean slope - Obs & Sim
obsSenSlope <- sens.slope(obsAMFts[ ,2])[1] # obs
simSenSlope <- sens.slope(simAMFts[ ,2])[1] # Sim

# Calculate TSArel:
obsTSArel <- (as.numeric(obsSenSlope) * length(obsAMF$Year)) / mean(obsAMF$Flow) * 100
simTSArel <- (as.numeric(simSenSlope) * length(simAMF$Year)) / mean(simAMF$Flow) * 100

# Trend magnitude in Obs (%)
obsTSArel
```

```
## [1] 37.69257
```

```
# Trend magnitude in Sim (%)
simTSArel
```

```
## [1] 12.19653
```

Trend magnitude's in annual mean flow are TSArel = +38% and TSArel = +12% for observed and simulated, respectively!

WOW... an *almost 40% increase* in observed mean flow in the Boyne. Let's see what the statistical trend tests say...

5.2 Mann-Kendall test for (gradual) change

We'll test for gradual (monotonic) trend with the non-parametric Mann-Kendall test.

The **standardised MK statistic (MKZs)** follows the standard normal distribution with a mean of zero and variance of one. A *positive (negative)* value of MKZs indicates an *increasing (decreasing)* trend. Statistical significance was evaluated with probability of type 1 error set at the 5% significance level. We will use a two-tailed MK test (as we won't assume the direction of change beforehand).

The null hypothesis of no trend (increasing or decreasing) is rejected when |MKZs|>1.96.

```
# Obs MKZs
obsMKOut <- mk.test(obsAMFts)
obsMKZs  <- obsMKOut$Zg[2]
obsMKZs
```

```
## [1] 3.105797
```

```
# Sim MKZs
simMKOut <- mk.test(simAMFts)
simMKZs  <- simMKOut$Zg[2]
simMKZs
```

```
## [1] 1.079986
```

MK Results: MKZs for obs is 3.1 which is much higher than our significance threshold of |1.96| so **there is statistically significant increasing trend in observed mean flow.**

However, MKZs for simulated flow is 1.1, so while the trend is increasing, the change **is not** statistically significant.

5.3 Pettitt test for (abrupt) change

We'll use the Pettitt statistic to identify a single (abrupt) change point in observed and simulated series.

The Pettitt test is nonparametric and relative to other tests is less sensitive to outliers and skewed data. **The null hypothesis (no step change in time series) against the alternative (an upward or downward change point in a given year) is tested at the 5% significance level.**

```
# Pettitt test
obsPettittOut <- pettitt.test(obsAMFts[ ,2])
simPettittOut <- pettitt.test(simAMFts[ ,2])

# Find p-value of change points
obs_PettP_val <- obsPettittOut$p.value
sim_PettP_val <- simPettittOut$p.value

# Find year of change points
# Obs
obs_PettYear  <- obsPettittOut$estimate
obsChangeYear <- obsAMF$Year[obs_PettYear]
obsChangeYear # Year
```

```
## [1] 1978
```

```
obs_PettP_val # p-value
```

```
## [1] 0.001013138
## attr(,"Csingle")
## [1] TRUE
```

```
# Sim
sim_PettYear  <- simPettittOut$estimate
```

```
simChangeYear <- simAMF$Year[sim_PettYear]
simChangeYear # Year
```

```
## [1] 1978
```

```
sim_PettP_val # p-value
```

```
## [1] 0.3985291
## attr(,"Csingle")
## [1] TRUE
```

Pettitt Results: There is a **statistically significant upward change point in Boyne mean flow** and this occurred in 1978 (p = 0.001). While the Pettitt test returns 1978 as the year of change, **it's much too weak to be considered an abrupt change point** as the p-value is 0.4.

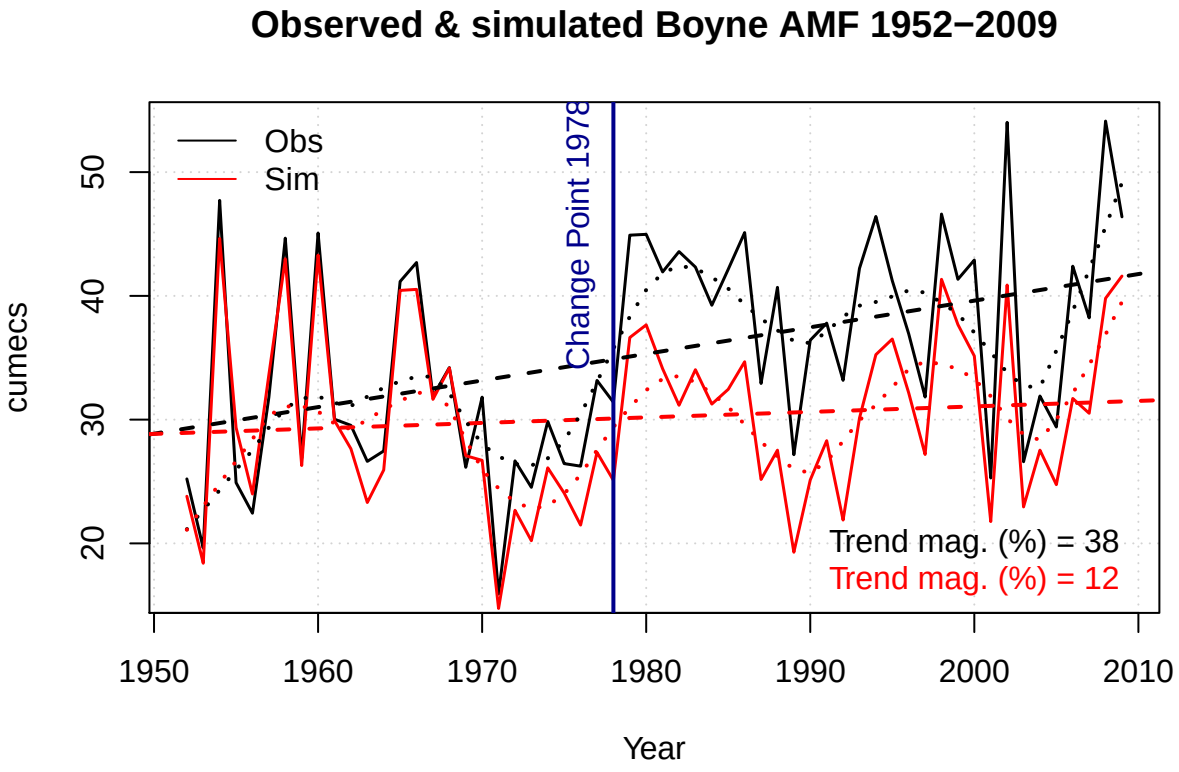
---

6. Summary of results

Test	Observed	Modelled
TSArel	38%	12%
MKZs	3.1	1.1
Pettitt	1978 (p = 0.001)	1978 (p = 0.4)

```
# Timie-series plot of observed and simulated AMF
plot(obsAMF$Year, obsAMF$Flow, type = 'l', main = "Observed & simulated Boyne AMF 1952-2009", xlab = "Y

lines(simAMF$Year, simAMF$Flow, col = "red", lwd = 1.5)

# Linear regression
linRegOutObs <- lm(obsAMF$Flow~obsAMF$Year)
linRegOutSim <- lm(simAMF$Flow~simAMF$Year)

# Add linear regression line
abline(linRegOutObs, col = "black", lty = 2, lwd = 2)
abline(linRegOutSim, col = "red", lty = 2, lwd = 2)

# Add LOWESS line (f = smoother span paramter, i.e. proportion of data influencing smooth)
lines(lowess(obsAMF$Flow~obsAMF$Year, f = 0.2), col = "black", lty = 3, lwd = 2)
lines(lowess(simAMF$Flow~simAMF$Year, f = 0.2), col = "red", lty = 3, lwd = 2)

# Add vertical change point line
abline(v = 1978, col = "darkblue", lwd = 2)

text(1976, 45, "Change Point 1978", col = "darkblue", srt = 90) # 'srt' rotates text 90 degrees


# Add legend to plot
legend("topleft",
       legend=c("Obs", "Sim"),
       col= c("black", "red"), lty=1,lwd=1, bty="n")
```

```
# Add text
text(2000, 20, paste("Trend mag. (%) = ", round(obsTSArel,0), sep = ""), col = "black")
text(2000, 17, paste("Trend mag. (%) = ", round(simTSArel,0), sep = ""), col = "red")
```



7. Further topics

7.1 Block-bootstrapping (BBS) statistics

If the serial correlation assumption is severely violated, block-bootstrapping (BBS) offers a viable method - example below:

```
# Need 'boot' package
library(boot) # bootstrapping
library(Kendall) # Another package with the 'Mann-Kendall' test - but this works better with 'tsboot'

# My MKZs Function ------------------------------------------------------------
myMKZsFun <- function(x) {

  S <- MannKendall(x)$S
  varS <- MannKendall(x)$varS

  #Calculate the Zs statistic
  if (S > 0) {
    Zs <- (S-1)/sqrt(varS)
  } else if (S < 0) {
    Zs <- (S+1)/sqrt(varS)
  } else {
    Zs <- 0
  }
```

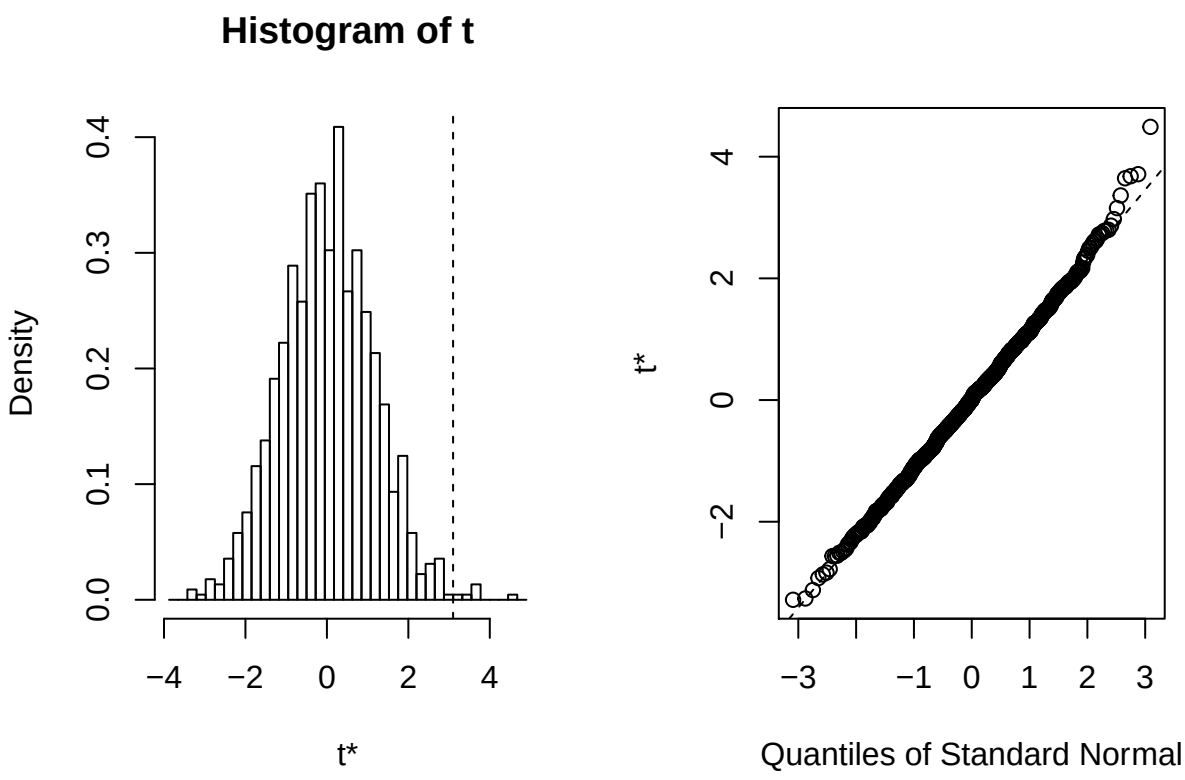
```
  return(Zs)  # MkZs

}

# end function ----------------------------------------------------------------


# Set.seed() for reproducability
set.seed(50)

# BBS with block length = 4 and 1000 resamples
b4 <- tsboot((obsAMF[ ,2]), myMKZsFun, R=1000,  sim="fixed", l=3)

# sort boot output
bs4 <- sort(b4$t)

# BBS critical values
bs4[25] # lower
```

```
## [1] -2.174693
```

```
bs4[975] # upper
```

```
## [1] 2.32295
```

```
# Plot bootstrap output
plot(b4)
```

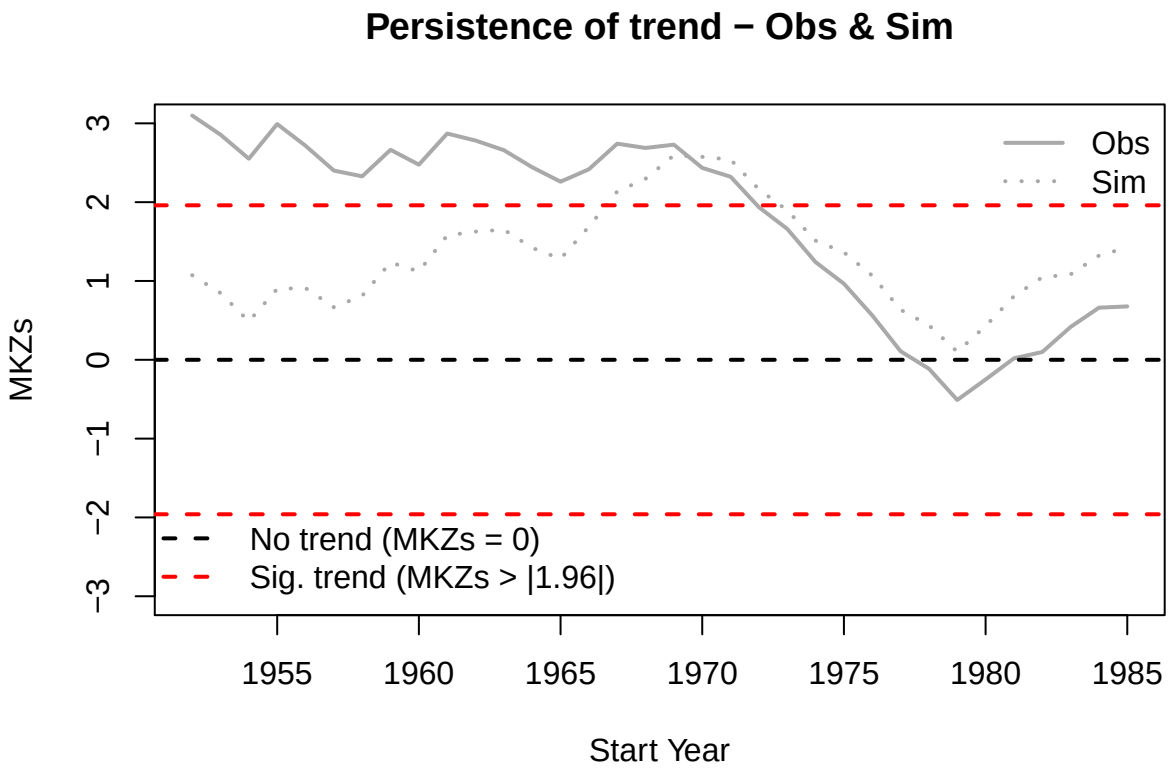


7.2 Multiple Mann-Kendall tests

R is much more powerful than just a statistical tool - it's also a capable programming environment so we're able to perform individual tasks, such as calculating a trend statistic, for many iterations using loops etc.

Here is a quick example of how the dependence of the Mann-Kendall Zs statistic on the period-of-record used. We test for changes in observed and modelled flow from all possible start years up to 1985: 1952-2009, 1953-2009,...,1985-2009.

```
# Create output matrix to store trend results
mat <- matrix(NA, ncol = 3, nrow = 34)
colnames(mat) <- c("StartYear", "obsMKZs", "simMKZs")
mat[ ,1] <- 1952:1985

# Loop Mann-Kendall test for all possible start years in both obs and sim
for(i in 1:34) {

  mat[i,2] <- myMKZsFun(obsAMF[ ,2][i:58]) # Obs
  mat[i,3] <- myMKZsFun(simAMF[ ,2][i:58]) # Sim

} # end i

# Plot
plot( mat[ ,1], mat[ ,2], type = 'l', col = "darkgray", lty = 1, lwd = 2, ylim = c(-3, 3), main = "Pers
lines(mat[ ,1], mat[ ,3], col = "darkgray", lty = 3, lwd = 2)

# Add lines to show statistically significant +/- trends and zero trend
abline(h =  1.96, col = "red", lty = 2, lwd = 2)
abline(h = -1.96, col = "red", lty = 2, lwd = 2)
abline(h = 0, col = "black", lty = 2, lwd = 2)

# First legend
legend.text <- c("No trend (MKZs = 0)", "Sig. trend (MKZs > |1.96|)")
legend("bottomleft",                    # Set location of the legend
       legend = legend.text,            # Specify text
       col = c("black","red"),          # Set colors for legend
       lty = c(2,2),                    # Set type of lines in legend
       merge = TRUE, bty = 'n', lwd = c(2, 2))

# Second legend
legend("topright",
       legend=c("Obs", "Sim"),
       col= c("darkgray", "darkgray"), lty=c(1, 3),lwd=c(2,2), bty="n")
```



END of tutorial - I hope you'll agree R is a nice (free!) alternative to e.g. Matlab :)

---

top

Discussion

Please give us feedback at bit.ly/feedbackR

For discussions, please use the Hydrology in R Facebook group.

Further resources can be found on the github page: https://github.com/brry/rhydro#resources