A Short Workflow Exercise with RavenR v1.3

Dr. James R. Craig, Robert Chlumsky

May 2019

This short document is intended to get you started with using RavenR to aid your analysis with the Raven model. This will get you up and running with the RavenR package and comfortable running a few commands. Some knowledge or R is presumed in this document. If you are not comfortable with R, take a look at any number of R training and Introductory resources, such as the tRaining repository) on Github. If you are looking for a longer version of this exercise for use on the RavenR package, see the RavenR Tutorial file maintained with the RavenR package (**currently under construction**).

This exercise will use the Nith River modelled output available from within the RavenR package, thus the functions to read in data from csv files are not required. However, it is recommended that you download the Nith river model files, and try to both run the model and read in the output files. The Nith river model can be downloaded from the Raven Tutorial #2.

As you go through this tutorial, don't just follow along blindly. Try to pay attention to what you are doing and how you are doing it.

Getting Acquainted with RavenR

Start a new RStudio session by opening RStudio. If you don't have RavenR yet installed in your R library, run the following commands to install the latest version of RavenR from Github (you will need the **devtools** library to be installed and loaded as well, so install this library first if you haven't yet).

```
# install.packages("devtools")
library(devtools)
devtools::install_github("rchlumsk/RavenR")
```

Load the RavenR library from the console and view its contents with the following commands:

```
library(RavenR)
ls("package:RavenR") # view all functions in RavenR
```

You can look at what any one of these functions does by typing out the name of the function beginning with a question mark, which will show the help information at the right of the RStudio environment.

```
?flow.scatterplot
```

Sample Data in RavenR

The RavenR package contains a number of sample data files, which are useful for training purposes and testing of functions. The package contains sample data both in R format (under RavenR/data) and as raw data files in their native formats (RavenR/inst/extdata). The sample data set from the RavenR package (in R format) can be loaded in using the data function (with either quotes or just the name of the data), e.g.,

```
data(forcing.data)
?forcing.data
data("hydrograph.data")
?hydrograph.data
```

Notice as well that the sample data set in R format also has a built in help file to describe the data.

To pull out the raw data from the RavenR package, we will use a syntax to find the data by file name in the RavenR package directory, which ends up looking more similar to a raw file call. This raw data file comes from the **extdata** folder in the RavenR package, **not the data folder**. Note that this is done so that the sample data in raw format can be used and tested with functions, and the syntax to locate the data file is more portable across various computer operating systems.

```
# read in hydrograph sample csv data from RavenR package
ff <- system.file("extdata","run1_Hydrographs.csv", package="RavenR")

# read in sample rvi file from the RavenR package
rvi <- system.file("extdata", "Nith.rvi", package="RavenR")
```

The **system.file** command will simply build a file path for where this data file is located on your machine with the RavenR package installation, which can then be passed to any function as required to provide a file location. This command will be used throughout this tutorial in place of local files for portability, however, your own data files may be swapped in place of the system.file locations.

Data and Plotting

Now you are ready to start using RavenR to directly visualize and manipulate model output. This section of the exercise will make use of raw sample data in the RavenR package to illustrate some of the diagnostics and plotting capabilityes of RavenR.

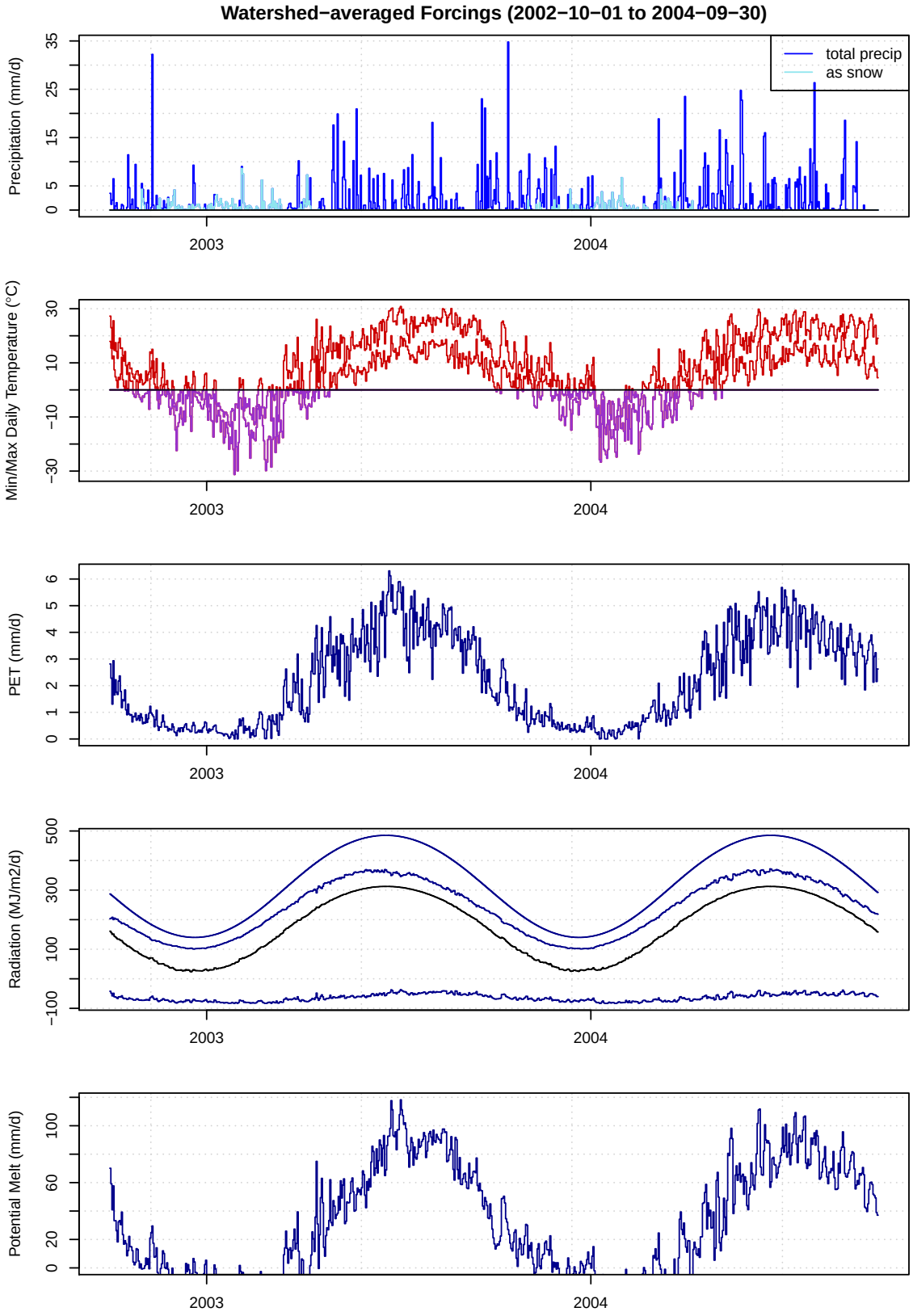
Start by finding the raw **run1_ForcingFunctions.csv** file with the **system.file** command. Note that this can be replaced with your own forcing functions file location if preferred. We will store the forcing functions data into an object called ff (and obtain just the subobject using the '$' operator), and then view the first few rows using the **head** function. We will show only the first six columns of the data for brevity.

```
ff <- system.file("extdata","run1_ForcingFunctions.csv",package="RavenR")
# ff <- "mydirectory/ForcingFunctions.csv" # replace with your own file
ff_data <- RavenR::forcings.read(ff)
head(ff_data$forcings[,1:6])
```

```
##            day_angle     rain snow    temp temp_daily_min temp_daily_max
## 2002-10-01   4.70809 3.468690    0 22.5956       17.92510        27.2662
## 2002-10-02   4.70809 3.468690    0 22.5956       17.92510        27.2662
## 2002-10-03   4.72530 1.189180    0 19.2076       15.40780        23.0075
## 2002-10-04   4.74251 2.083260    0 13.3714       11.49870        15.2440
## 2002-10-05   4.75973 6.474310    0 19.0304       12.50970        25.5510
## 2002-10-06   4.77694 0.125591    0 11.0186        7.43466        14.6024
```

Now we can plot the forcing data using the forcings.plot function. This creates an output of the five main forcings from the data set.

```
forcings.plot(ff_data$forcings)
```


Watershed−averaged Forcings (2002−10−01 to 2004−09−30)
Precipitation (mm/d)
0
5
15
25
35
total precip
as snow
2003
2004
Min/Max Daily Temperature (°C)
−30
−10
10
30
2003
2004
PET (mm/d)
0
1
2
3
4
5
6
2003
2004
Radiation (MJ/m2/d)
−100
100
300
500
2003
2004
Potential Melt (mm/d)
0
20
60
100
2003
2004

This is typically a reasonable reality check on the model forcings.

We can similarly access the hydrograph fit. Here the hydrograph sample data is located with the usual **system.file** command, then read into R with the **hyd.read** function intended for reading Hydrographs file. The flows from a specific subbasin can be extracted using the hyd.extract function, which is done here for subbasin 36. The precipitation can be extracted similarly.

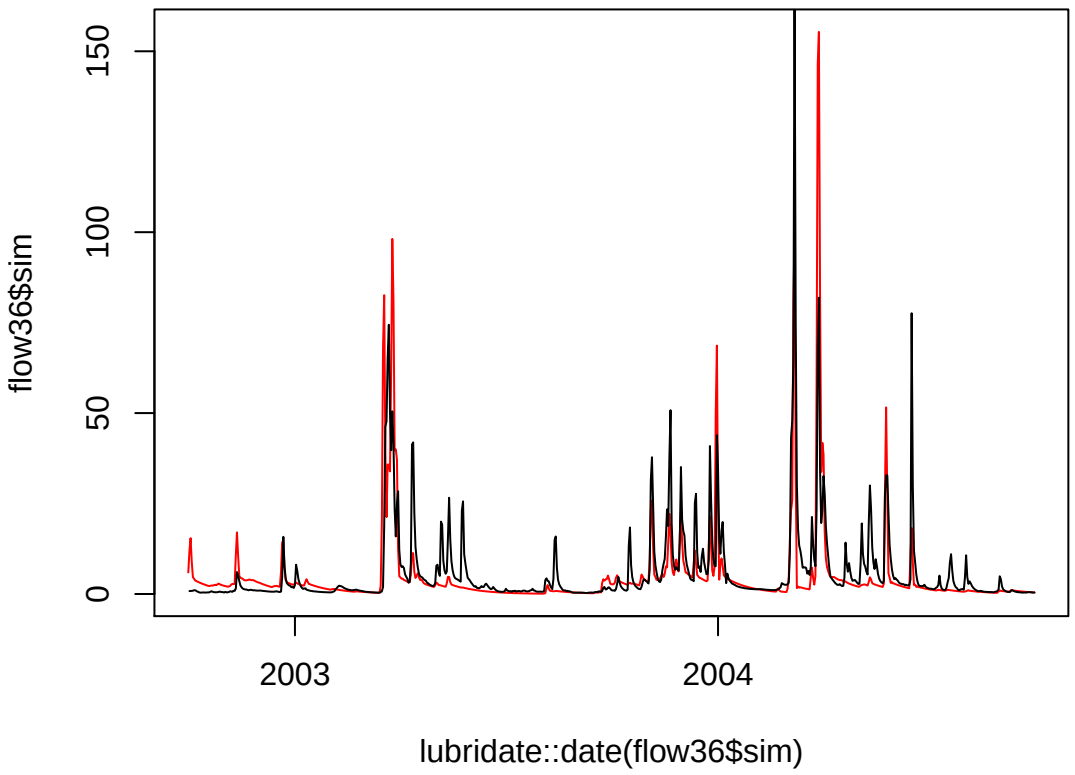
```
ff <- system.file("extdata","run1_Hydrographs.csv",package="RavenR")
# ff <- "mydirectory/Hydrographs.csv" # replace with your own file
hy <- hyd.read(ff)
head(hy$hyd)
```

```
##              precip    Sub36 Sub36_obs    Sub43 Sub43_obs
## 2002-10-01       NA  5.96354        NA 11.25050        NA
## 2002-10-02 3.468690 11.96430     0.801 18.59070      3.07
## 2002-10-03 1.189180 15.43700     0.828 25.74430      2.99
## 2002-10-04 2.083260  8.76948     0.860 18.68610      3.06
## 2002-10-05 6.474310  4.66501     0.903  9.82648      2.93
## 2002-10-06 0.125591  4.20829     1.040  7.90952      3.15
```

```
flow36 <- hyd.extract("Sub36",hy)
precip <- hyd.extract("precip",hy)$sim
```

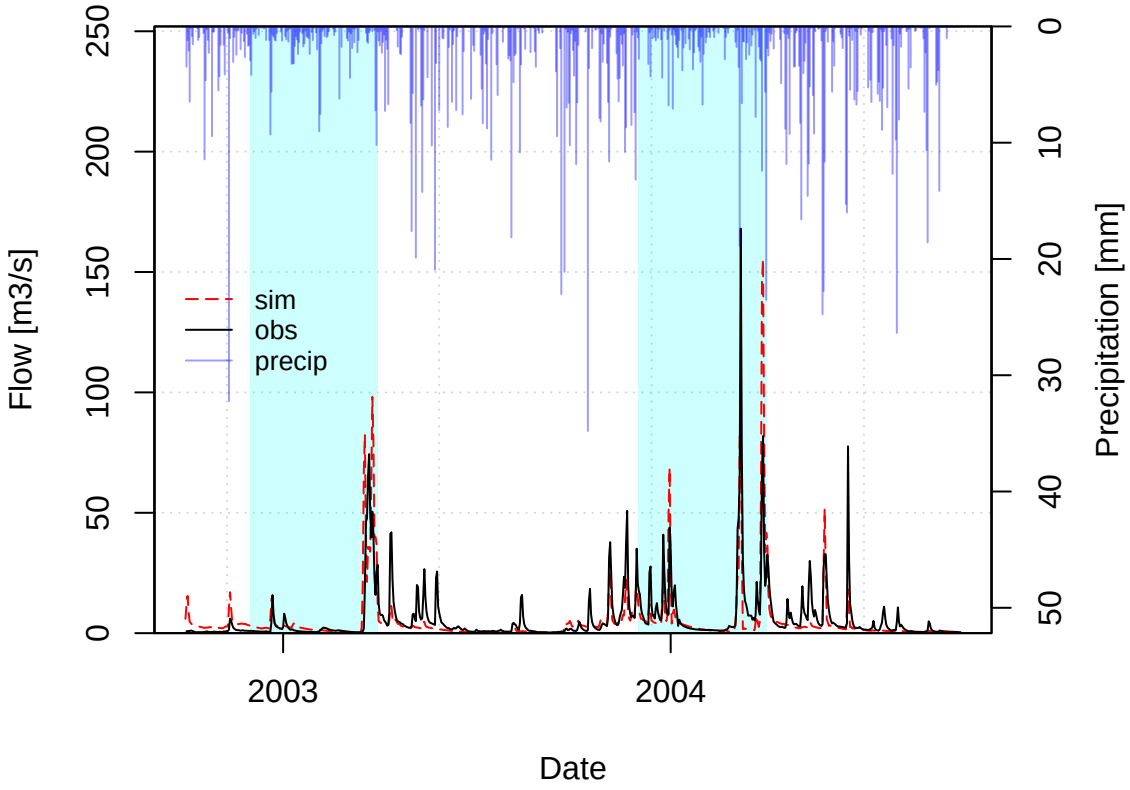
The hydrograph object flow3 now stores the simulated hydrograph (`flow36$sim`) and the observed hydrograph (`flow36$obs`), and the null subobject (`flow36$inflow`). The precip object stores the entire time series of watershed-averaged precip (`precip$sim`). We can plot the simulated and observed hydrograph with the simple commands:

```
plot(lubridate::date(flow36$sim),flow36$sim,col='red',type='l')
lines(lubridate::date(flow36$obs),flow36$obs,col='black')
```



Or using the special hydrograph plot function, which is part of the RavenR library. This function save some of the trouble of plotting the precipitation on the secondary axis.

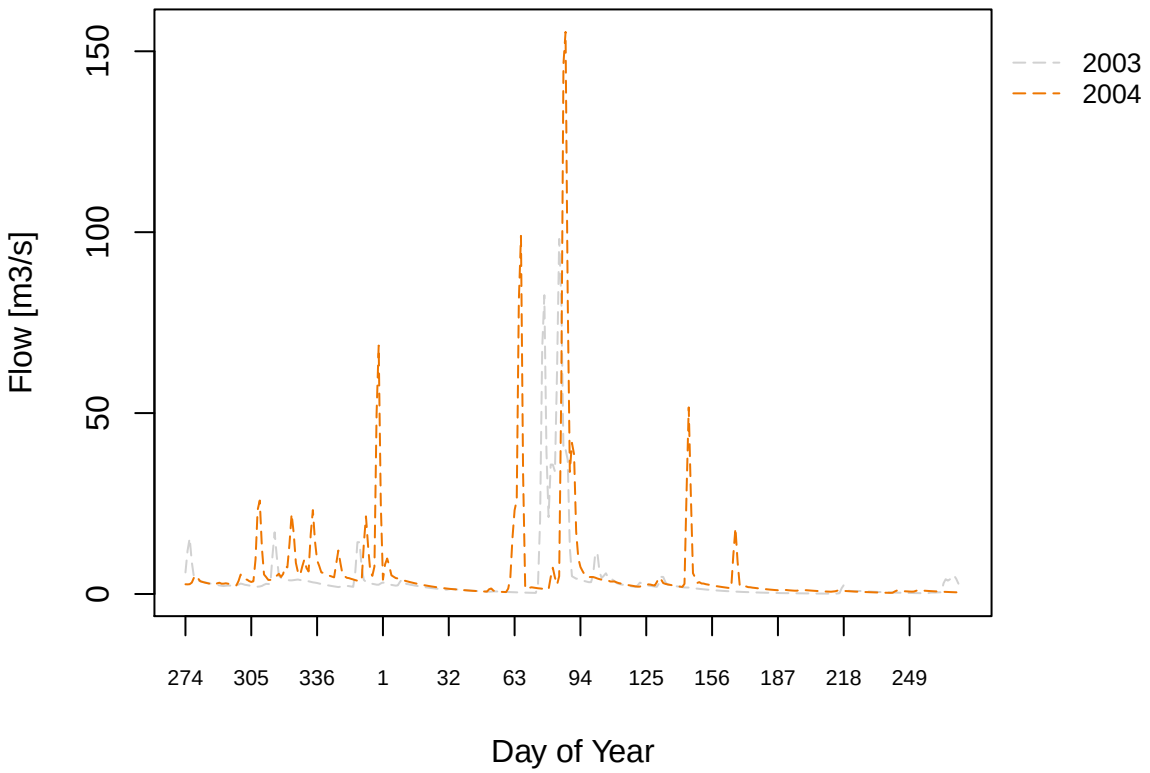
```
hyd.plot(sim=flow36$sim, obs=flow36$obs, precip=precip)
```




```
## [1] TRUE
```

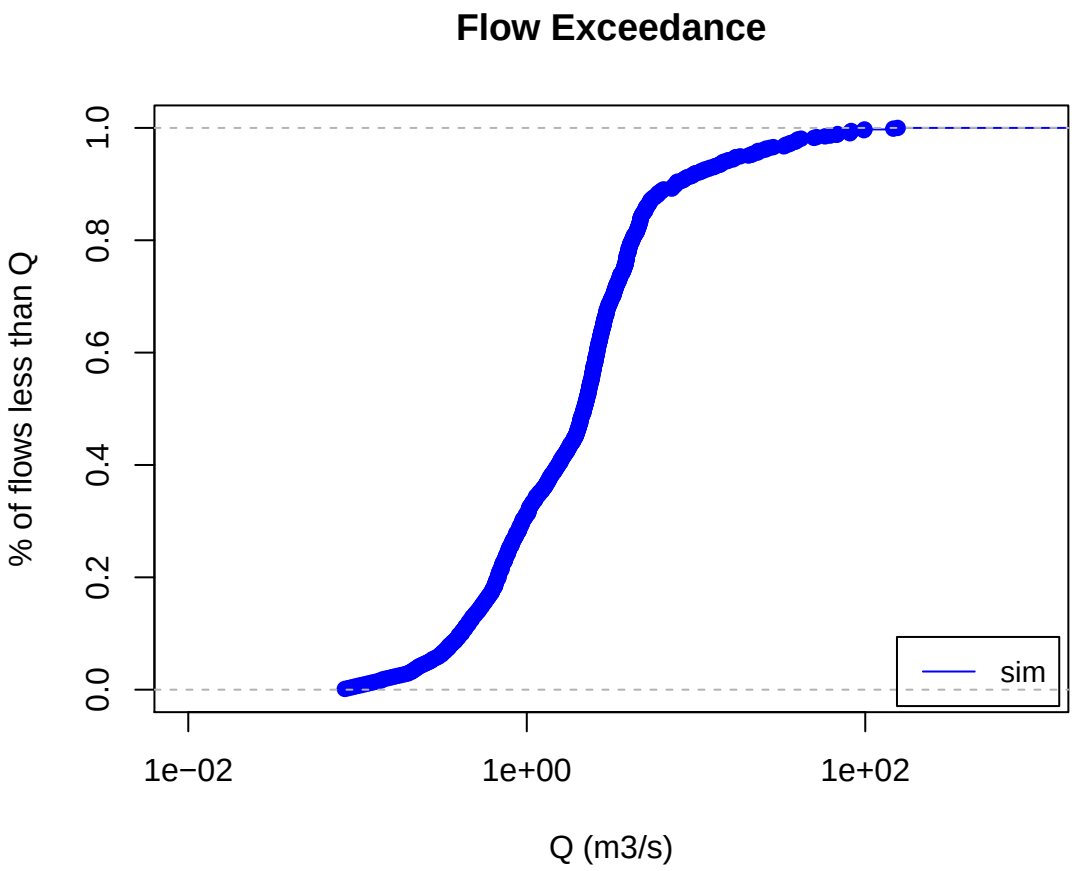
There are some other helpful functions in RavenR for understanding our hydrographs.

```
flow.spaghetti(flow36$sim)
```



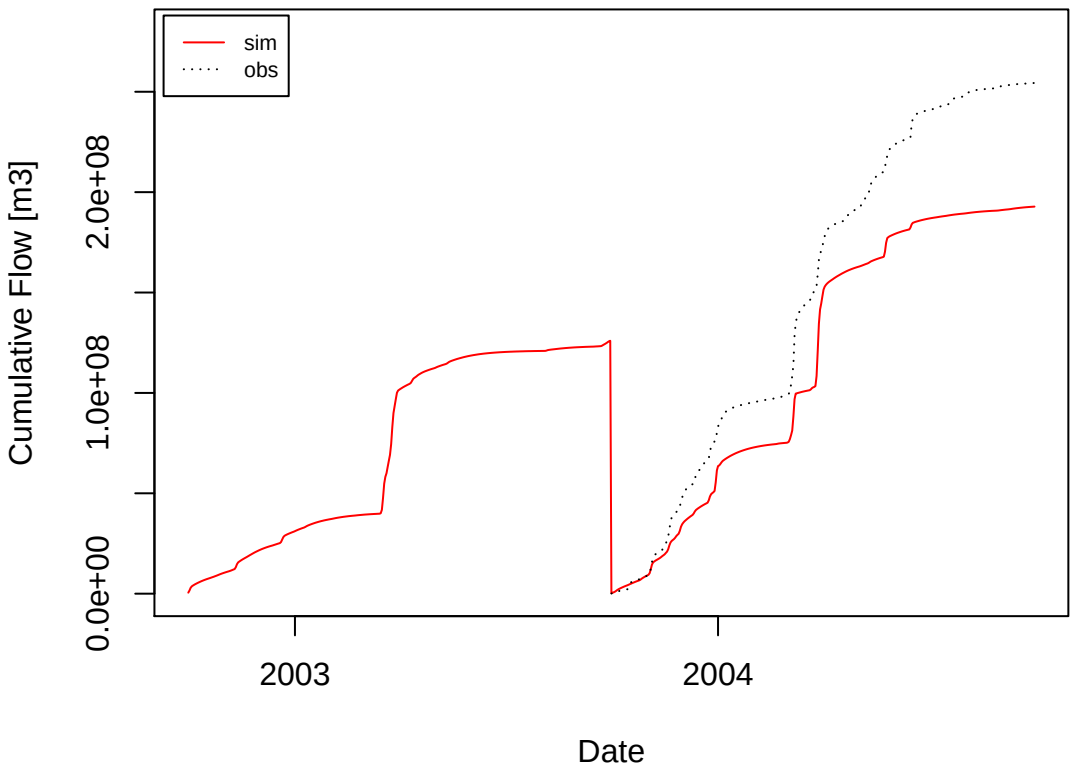

```
## [1] TRUE
```

```
flowdurcurve.plot(flow36$sim)
```



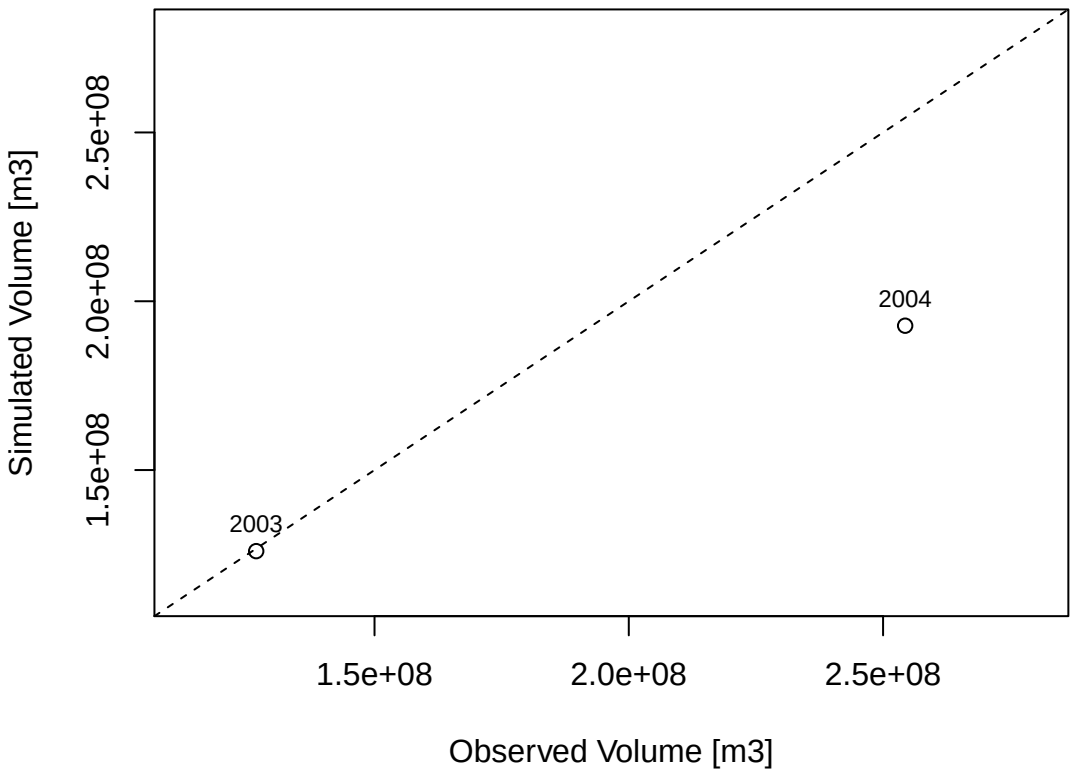
We can also use some of the Raven plots to get some diagnostics and comparisons on the simulated and observed hydrographs.

```
cum.plot.flow(flow36$sim, obs=flow36$obs)
```




```
## [1] TRUE
```

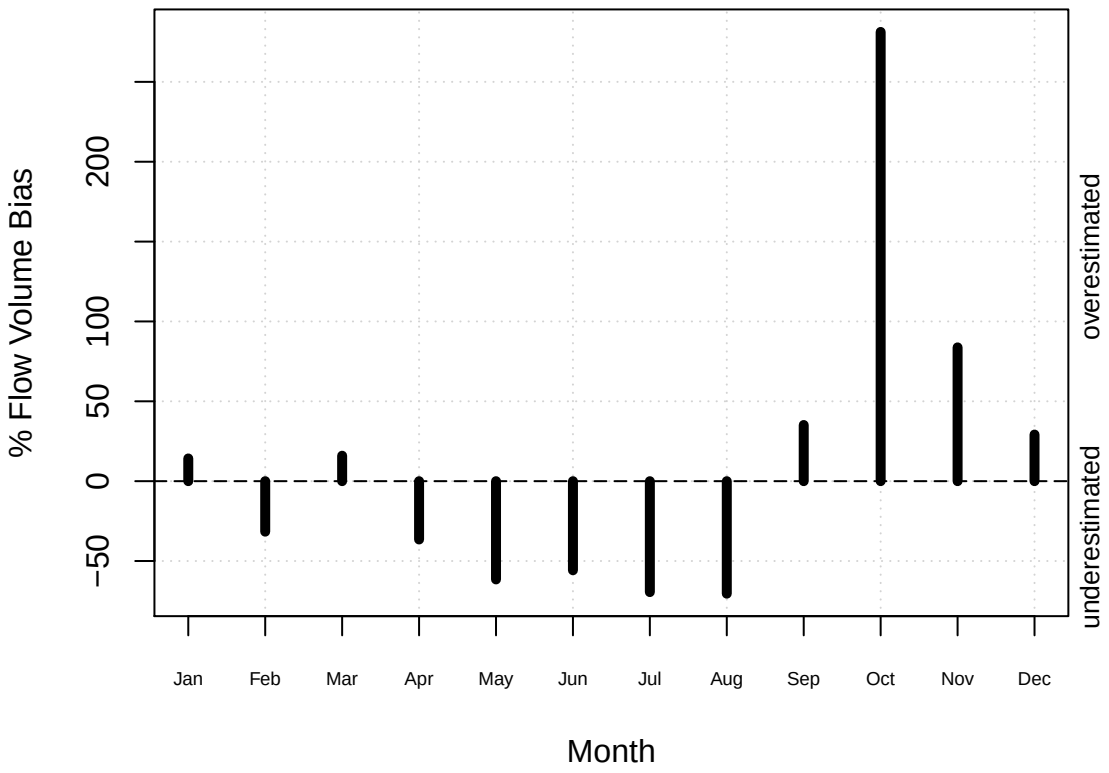
```
annual.volume(flow36$sim, flow36$obs)
```




```
##     date.end   sim.vol   obs.vol
## 1 2003-10-01 125978298 126733075
## 2 2004-09-30 192760345 254344147
```

A helpful plot for understanding the volume bias on a monthly breakdown is the **monthly.vbias** function, which plots out the volume bias between the simulated and observed hydrographs by month.

```
monthly.vbias(flow36$sim, obs=flow36$obs)
```




```
##        mvbias
## Jan  14.21804
## Feb -31.63489
## Mar  15.93412
## Apr -36.52680
## May -61.52621
## Jun -55.78284
## Jul -69.45787
## Aug -70.45509
## Sep  35.16982
## Oct 281.28584
## Nov  83.76182
## Dec  29.17106
```

A fun new addition to the RavenR package is the addition of dygraphs, which produces a dynamic hydrograph plot with a slider on the time scale. This is particularly helpful for viewing subsections of a hydrograph dyanmically, and comparing the simulated and observed hydrographs in an interactive environment.

```
hyd.dyGraphs(hy, basins="Sub36")
```

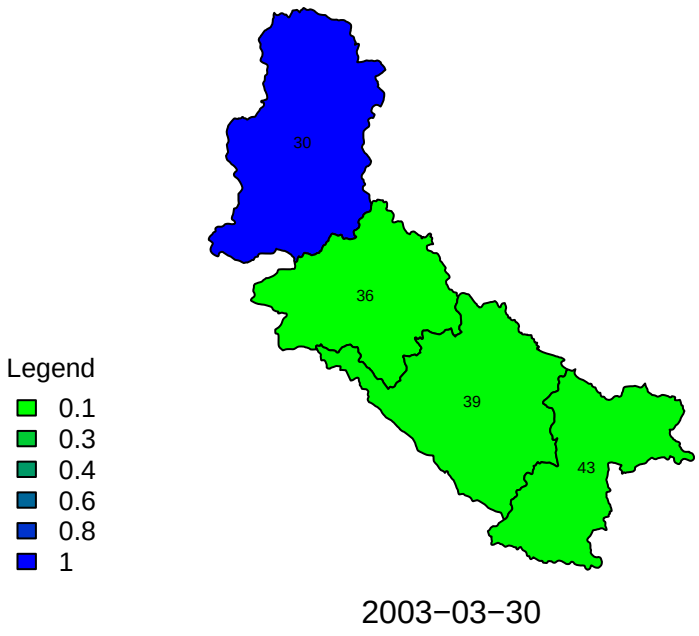
```
## [[1]]
```

Spatial Plotting

The RavenR package also has some functionality for spatial plots. This section will use some of the sample spatial data for the Nith basin to build a subbasin plot from custom output data, which can be modified to show any custom output data.

Begin by loading in the appropriate sample data files, and locating them on the local machine. This includes a custom output of daily average precipitation by subbasin for the Nith subbasin.

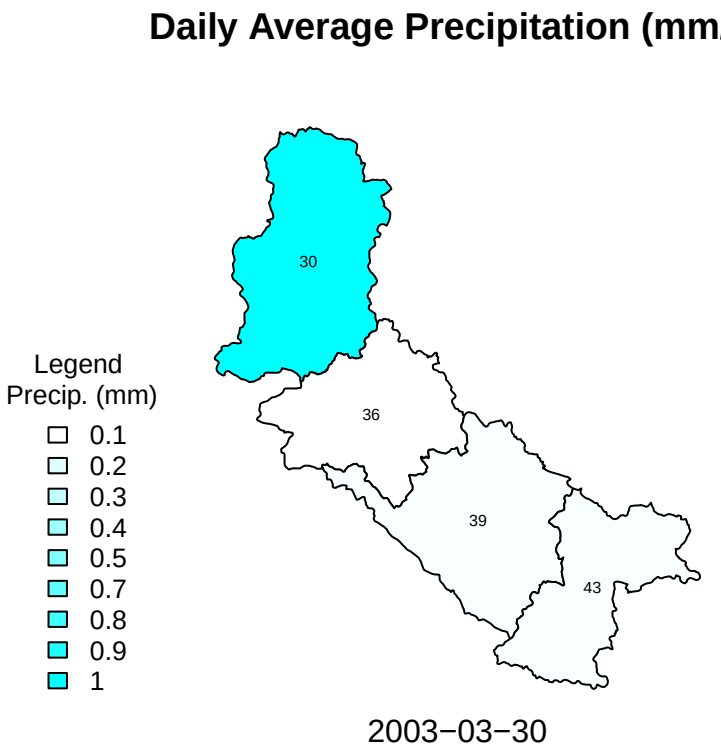
```
# Raw sample data
shpfilename <- system.file("extdata","Nith_shapefile_sample.shp",package="RavenR")

# Custom Output data from Raven for Nith basin
cust.data <- custom.read(system.file("extdata","run1_PRECIP_Daily_Average_BySubbasin.csv",
                                     package="RavenR"))

subIDcol <- 'subID' # attriute in shapefile with subbasin IDs
plot.date <- "2003-03-30" # date for which to plot custom data

# function call
SBMap.plot(shpfilename,subIDcol,plot.date,cust.data)
```




```
## [1] TRUE
```

This produces a basic static map of the Nith subbasins provided in the sample file, with the precipitation data from the Custom Output data provided at the date specified. Neat! We can add a few more features from here to cutomize our plot if we wish, such as changing the legend title to match the custom output data, the number of class breaks, the plot title, and the colour scheme.

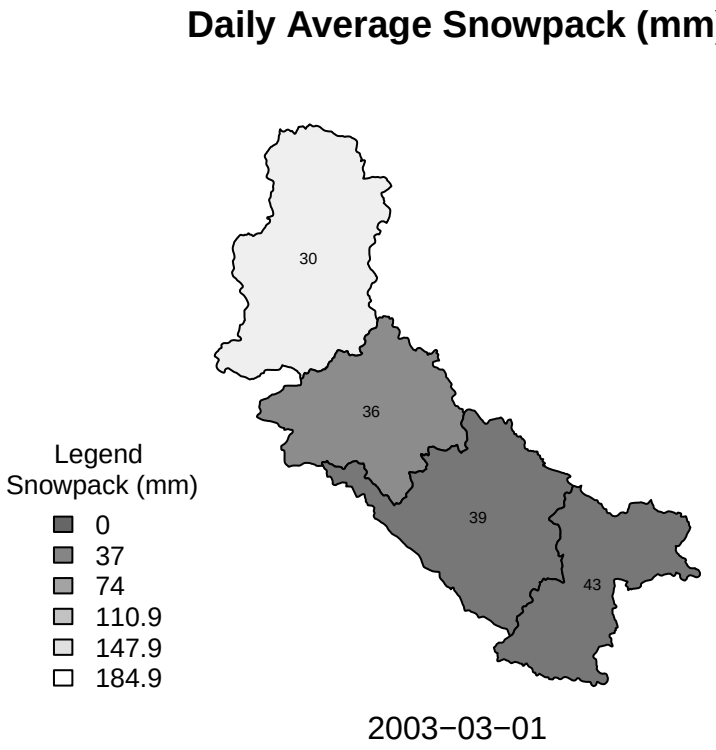
```
leg.title <- 'Legend \nPrecip. (mm)'
colour1 <- "white"
colour2 <- "cyan"
num.classes <- 8
plot.title <- 'Daily Average Precipitation (mm/d)'

# create an updated plot
SBMap.plot(shpfilename,subIDcol,plot.date,cust.data,plot.title=plot.title,
           colour1 = colour1, colour2=colour2,
           leg.title = leg.title, num.classes=num.classes)
```




```
## [1] TRUE
```

This function can accept any custom output data as the input data for a given subbasin shapefile, provided that the numbering of the custom data and the labels on the subbasin IDs in the shapefile correspond. Try to use the sample external data for snowpack in this plot as well! Use the 'run1_SNOW_Daily_Average_BySubbasin.csv' file and change the colours to grey40 and white for

snowpack. Use March 1st, 2003 as the date and 5 classes in the legend. Normalize the results, and check your plot against the one provided.




```
## [1] TRUE
```

We can also use the animated version of this function, built on the **magick** package, to produce a gif of results. This is likely much more useful than a static plot provided for a single date. Follow the example below to create an animated .gif file from the snowpack data.

```
cust.data <- custom.read(system.file("extdata","run1_SNOW_Daily_Average_BySubbasin.csv",
                                     package="RavenR"))
leg.title <- 'Legend \nSnowpack (mm)'
colour1 <- "grey40"
colour2 <- "white"
num.classes <- 5
plot.title <- 'Daily Average Snowpack (mm)'
# Feel free to adjust the date range here, although many dates make this slow to run!
plot.daterange <- '2003-02-01/2003-03-30'

gif.filename='Nith_snowpack_Feb2003_March2003.gif'
gif.speed <- 0.5
cleanup <- FALSE # see the individual plots in the scratch directory created
```

```
SBMap.animate(shpfilename,subIDcol,plot.daterange,cust.data,plot.title=plot.title,
              colour1 = colour1, colour2=colour2,
              leg.title = leg.title, normalize=T, num.classes=num.classes,
              gif.filename=gif.filename,
              gif.speed=gif.speed, cleanup=cleanup
           )
```

This works well, although the overall precipitation is much more exciting to watch in a gif.

```
cust.data <- custom.read(system.file("extdata","run1_PRECIP_Daily_Maximum_BySubbasin.csv",
                                     package="RavenR"))
leg.title <- 'Legend \nDaily Max Precip (mm)'
colour1 <- "white"
colour2 <- "blue"
num.classes <- 5
plot.title <- 'Daily Max Precip. (mm)'
plot.daterange <- '2003-05-01/2003-06-30'

gif.filename='Nith_precip_May2003_June2003.gif'
gif.speed <- 0.5
cleanup <- FALSE

SBMap.animate(shpfilename,subIDcol,plot.daterange,cust.data,plot.title=plot.title,
              colour1 = colour1, colour2=colour2,
              leg.title = leg.title, normalize=T, num.classes=num.classes,
              gif.filename=gif.filename,
              gif.speed=gif.speed, cleanup=cleanup
           )
```

More Exploring and Getting Help

The RavenR library can be explored to see what other functions are available in the package.

```
ls("package:RavenR")
```

Using the ?help option (where help is the name of a RavenR command), figure out how to plot:

1. a comparison of annual peak flows, and
2. the mean and median annual observed flow using the barplot() function (hint: use the apply.wyearly function to calculate annual mean and median)

Building a model workflow script

Now we will build a simple script which will provide a bunch of visualizations that we can use to look at the Nith river model each time we run it. This can be made as complex as you want.

Start with a new script. From RStudio, go to the main menu. `Choose File -> New File -> R Script`. Populate the script with the following. You can find the Nith model files in the Raven Tutorials.

```
# Load the RavenR sample data
# ====================================================
indir <- "C:/temp/Nith/"
outdir <- "C:/temp/Nith/output/"
fileprefix <- "Nith"

if (dir.exists(outdir)==FALSE) {
  dir.create(outdir)
}

setwd(outdir)

# RUN RAVEN
# ====================================================
# writes complete command prompt command
# > Raven.exe [filename] -o [outputdir]
RavenCMD <-paste(indir,"Raven.exe ",indir,fileprefix," -o ",outdir,sep="")
system(RavenCMD) # this runs raven from the command prompt
```

Once the model is run, we can read in the output (or use the package data) and save some of the plots to file.

```
# GENERATE OUTPUT PLOTS
# ====================================================
# use the package data, or read in the model output files

# ff_data <- forcings.read("ForcingFunctions.csv")
pdf("forcings.pdf") # create a pdf file to direct plot to
forcings.plot(ff_data$forcings)
dev.off() #finishes writing plot to .pdf file

data(watershed.data)
mywshd <- watershed.data$watershed.storage
#mywshd <- RavenR::watershed.read("WatershedStorage.csv")$watershed.storage
png("snowpack.png") # create a png file to direct plot to
plot(mywshd$snow)
dev.off() #finishes writing plot to .png file
```

Modify the script

Modify the above script to generate png image plots of monthly subbasin-averaged PET in Subbasin 43 using the :CustomOutput option (you will have to add a :CustomOutput command to the Raven input rvi file). You will also want to use the RavenR `custom.read()` and `customoutput.plot()` commands.

More exercises

This short exercise is meant to serve as a brief introduction to the RavenR package. The complete RavenR Tutorial can be found on the Raven downloads page or on the RavenR Github page. If you have any comments, suggestions or bug reports, please email the authors of the package or feel free to let us know on the Raven forum.