Visualizing MAHLERAN Results in R

John Wainwright

11 June 2025

Getting Started

Looking at the MAHLERAN manual, there are two sorts of output file: time series, which have the extension .dat, and maps which have the extension .asc. We'll look at the time-series data first:

File Name	Description
hydro***.dat	Hydrological results for outflow from the plot / catchment
sedtr***.dat	Sediment results for outflow from the plot / catchment
nutri***.dat	Dissolved nutrient concentrations
p_nut***.dat	Particulate-bound nutrient concentrations
seddetach***.dat	Total detached sediment by size class
seddepos***.dat	Total deposited sediment by size class
seddepth***.dat	Total sediment depth by size class
sedveloc***.dat	Total average sediment velocity by size class
seddisch***.dat	Total sediment discharge by size class
sedpropn***.dat	Total sediment proportion by size class
sedconcn***.dat	Average sediment concentration by size class

Where the *** in each case is a number from 001 to 999 representing the particular run number (for which you can check the inputs used in the respective param***.dat file).

The first two files on this list are the ones you're most likely to use, in that they give the main hydrological outputs (and also the rainfall input for convenience) and the main sediment-transport outputs.

Make sure this Rmd file is in your project folder as defined when you ran the MAHLERAN model before, so that the "Output" folder is in the same folder as this file. Use Session –> Set Working Directory –> To Source File Location to set the working directory to the correct location. Click on the little arrow next to the path name at the top of the Console pane and looking at the list of files in the Files pane in RStudio. If you can't see the Output folder at this stage, the remaining steps are unlikely to work properly.

Earlier, you opened the file "hydro001.dat" in a text editor to see what it looks like. If you can't remember, do so again and you should see it is in space-delimited format, with no metadata and no header line. Thus, we can use the read.table command to get the values into R:

```
mahleranOutputFolder <- "./Output/"
hydroFile <- "hydro001.dat"

hydrograph <- read.table (paste0 (mahleranOutputFolder, hydroFile),
                          header = FALSE, sep = "")

head (hydrograph)
```

```
##   V1       V2      V3     V4     V5     V6        V7        V8        V9    V10
```

```
## 1  1 0.004233 0.09693 0.1412 0.2753 0.4697 0.0001191 0.0001795 0.0002024 0.1234
## 2  2 0.004233 0.27420 0.3994 0.7785 1.3280 0.0002383 0.0003589 0.0004048 0.1746
## 3  3 0.004233 0.50380 0.7339 1.4300 2.4390 0.0003575 0.0005383 0.0006071 0.2138
## 4  4 0.004233 0.77580 1.1300 2.2000 3.7540 0.0004767 0.0007175 0.0008093 0.2469
## 5  5 0.004233 1.08400 1.5800 3.0740 5.2440 0.0005960 0.0008967 0.0010110 0.2761
## 6  6 0.004233 1.42600 2.0770 4.0390 6.8890 0.0007154 0.0010760 0.0012130 0.3025
##      V11    V12
## 1 0.1408 0.1859
## 2 0.1991 0.2629
## 3 0.2438 0.3220
## 4 0.2815 0.3718
## 5 0.3147 0.4156
## 6 0.3447 0.4553
```

There are 12 variables in the file. As there wasn't a header, R has defaulted to calling them V1 ... V12, but looking at the Manual, we can see they represent: time, rainfall, plot discharge, 3 cross-section discharges, 3 cross-section depths, 3 cross-section velocities (time in seconds, rainfall in mm s^{-1} discharges in mm^3 s^{-1} , depths in mm, velocities in mms–1. Cross sections are at 25%, 50% and 75% from the top of the plot, though we'll ignore them for the time being.) Let's give the variables more meaningful names to make our scripting easier to follow:

```
names (hydrograph) <- c ("time", "rain", "qOutflow",
                         "qSect1", "qSect2", "qSect3",
                         "dSect1", "dSect2", "dSect3",
                         "vSect1", "vSect2", "vSect3")
 #
 # names for variables:
 #   time = time since start of simulation in seconds
 #   rain = rainfall intensity in mm/s in input file, converted to mm/h below
 #   qOutflow = outflow from plot/slope/catchment in mm3/s in input file, converted to litres below
 #   qSect1, qSect2, qSect3 - flows at three cross sections in mm3/s in input file, converted to litre
 #   dSect1, dSect2, dSect3 - mean flow depths at three cross sections in mm in input file
 #   vSect1, vSect2, vSect3 - mean flow depths at three cross sections in mm in input file
```

And then we'll do some unit conversions to avoid repetition later:

```
# unit conversions
hydrograph$rain <- hydrograph$rain * 3600.  ## to mm/h
hydrograph$qOutflow <- hydrograph$qOutflow / 1000000.  ## to litres
hydrograph$qSect1 <- hydrograph$qSect1 / 1000000.  ## to litres
hydrograph$qSect2 <- hydrograph$qSect2 / 1000000.  ## to litres
hydrograph$qSect3 <- hydrograph$qSect3 / 1000000.  ## to litres
```

So our data now look like this:

```
head (hydrograph)
```

```
##   time    rain  qOutflow    qSect1    qSect2    qSect3    dSect1    dSect2
## 1    1 15.2388 9.693e-08 1.412e-07 2.753e-07 4.697e-07 0.0001191 0.0001795
## 2    2 15.2388 2.742e-07 3.994e-07 7.785e-07 1.328e-06 0.0002383 0.0003589
## 3    3 15.2388 5.038e-07 7.339e-07 1.430e-06 2.439e-06 0.0003575 0.0005383
## 4    4 15.2388 7.758e-07 1.130e-06 2.200e-06 3.754e-06 0.0004767 0.0007175
## 5    5 15.2388 1.084e-06 1.580e-06 3.074e-06 5.244e-06 0.0005960 0.0008967
## 6    6 15.2388 1.426e-06 2.077e-06 4.039e-06 6.889e-06 0.0007154 0.0010760
##      dSect3 vSect1 vSect2 vSect3
## 1 0.0002024 0.1234 0.1408 0.1859
## 2 0.0004048 0.1746 0.1991 0.2629
```

```
## 3 0.0006071 0.2138 0.2438 0.3220
## 4 0.0008093 0.2469 0.2815 0.3718
## 5 0.0010110 0.2761 0.3147 0.4156
## 6 0.0012130 0.3025 0.3447 0.4553
```

Repeating the process with the file "sedtr001.dat" you should see it is also in space-delimited format, with no metadata and no header line. Again, we can use the read.table command to get the values into R:

```
sediFile <- "sedtr001.dat"

sedigraph <- read.table (paste0 (mahleranOutputFolder, sediFile),
                         header = FALSE, sep = "")

head (sedigraph)
```

```
##   V1        V2        V3        V4        V5        V6        V7        V8
## 1  1 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## 2  2 1.408e-16 2.816e-17 6.633e-16 6.951e-16 1.075e-15 4.616e-19 1.020e-16
## 3  3 3.392e-16 6.784e-17 1.598e-15 1.674e-15 2.589e-15 1.112e-18 2.458e-16
## 4  4 5.819e-16 1.164e-16 2.740e-15 2.868e-15 4.440e-15 1.908e-18 4.215e-16
## 5  5 8.616e-16 1.723e-16 4.055e-15 4.242e-15 6.573e-15 2.825e-18 6.238e-16
## 6  6 1.174e-15 2.348e-16 5.522e-15 5.773e-15 8.954e-15 3.849e-18 8.496e-16
##          V9       V10    V11   V12      V13       V14       V15 V16
## 1 0.000e+00 0.000e+00 0.0000 0.000 0.000000 0.000e+00 0.000e+00   0
## 2 4.965e-17 4.999e-17 0.2793 0.717 0.003682 1.525e-13 4.226e-10   0
## 3 1.195e-16 1.204e-16 0.2793 0.717 0.003682 6.717e-13 4.225e-10   0
## 4 2.049e-16 2.065e-16 0.2793 0.717 0.003682 1.666e-12 4.222e-10   0
## 5 3.030e-16 3.057e-16 0.2793 0.717 0.003682 3.221e-12 4.220e-10   0
## 6 4.123e-16 4.165e-16 0.2793 0.717 0.003683 5.407e-12 4.216e-10   0
```

This time we have 16 columns of data (V1 ... 16), which the Manual tells us are time [in seconds], plot sediment output [kg s^{-1}], plot sediment flux [kg m^{-1} s^{-1}], 3 cross-section fluxes [kg m^{-1} s^{-1}], plot yield per unit area [kg m^{-2} s^{-1}], 3 cross-section yields [kg m^{-2} s^{-1}], and plot particle size out (proportions of the total for each of the six size classes).

Again, let's give the variables better names

```
names(sedigraph) <- c ("time", "sedPlot", "plotFlux",
                       "sedCS1", "sedCS2", "sedCS3",
                       "plotYield", "yieldCS1", "yieldCS2", "yieldCS3",
                       "phi1", "phi2", "phi3", "phi4", "phi5", "phi6")
#
# names for variables:
#   time = time since start of simulation in seconds
#   sedPlot = total sediment coming off plot in kg/s
#   plotFlux = flux of sediment coming off plot in kg/m/s
#   sedCS1 = sediment crossing section 1 in kg/s
#   sedCS2 = sediment crossing section 2 in kg/s
#   sedCS3 = sediment crossing section 3 in kg/s
#   plotYield = plot sediment yield in kg/m^2/s
#   yieldCS1 = sediment yield crossing section 1 in kg/m2/s
#   yieldCS2 = sediment yield crossing section 2 in kg/m2/s
#   yieldCS3 = sediment yield crossing section 3 in kg/m2/s
#   phi1 = plot sediment outflow -- proportion < 0.063 mm
#   phi2 = plot sediment outflow -- proportion >= 0.063 mm - 0.25 mm
#   phi3 = plot sediment outflow -- proportion >= 0.25 mm - 0.5 mm
#   phi4 = plot sediment outflow -- proportion >= 0.5 mm - 2 mm
```

```
#   phi5 = plot sediment outflow -- proportion >= 0.2 mm - 12 mm
#   phi6 = plot sediment outflow -- proportion >= 12 mm
#

head (sedigraph)
```

```
##   time   sedPlot  plotFlux    sedCS1    sedCS2    sedCS3 plotYield  yieldCS1
## 1    1 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## 2    2 1.408e-16 2.816e-17 6.633e-16 6.951e-16 1.075e-15 4.616e-19 1.020e-16
## 3    3 3.392e-16 6.784e-17 1.598e-15 1.674e-15 2.589e-15 1.112e-18 2.458e-16
## 4    4 5.819e-16 1.164e-16 2.740e-15 2.868e-15 4.440e-15 1.908e-18 4.215e-16
## 5    5 8.616e-16 1.723e-16 4.055e-15 4.242e-15 6.573e-15 2.825e-18 6.238e-16
## 6    6 1.174e-15 2.348e-16 5.522e-15 5.773e-15 8.954e-15 3.849e-18 8.496e-16
##    yieldCS2  yieldCS3   phi1  phi2     phi3      phi4      phi5 phi6
## 1 0.000e+00 0.000e+00 0.0000 0.000 0.000000 0.000e+00 0.000e+00    0
## 2 4.965e-17 4.999e-17 0.2793 0.717 0.003682 1.525e-13 4.226e-10    0
## 3 1.195e-16 1.204e-16 0.2793 0.717 0.003682 6.717e-13 4.225e-10    0
## 4 2.049e-16 2.065e-16 0.2793 0.717 0.003682 1.666e-12 4.222e-10    0
## 5 3.030e-16 3.057e-16 0.2793 0.717 0.003682 3.221e-12 4.220e-10    0
## 6 4.123e-16 4.165e-16 0.2793 0.717 0.003683 5.407e-12 4.216e-10    0
```

We can also look at some simple summary statistics for the hydrograph data:

```
summary (hydrograph)
```

```
##       time             rain          qOutflow            qSect1
##  Min.   :   1.0   Min.   : 0.00   Min.   :9.700e-08   Min.   :1.410e-07
##  1st Qu.: 408.2   1st Qu.:15.24   1st Qu.:1.755e-03   1st Qu.:1.112e-03
##  Median : 815.5   Median :15.24   Median :3.073e-02   Median :1.300e-02
##  Mean   : 815.5   Mean   :21.32   Mean   :3.922e-02   Mean   :1.410e-02
##  3rd Qu.:1222.8   3rd Qu.:30.48   3rd Qu.:8.085e-02   3rd Qu.:2.124e-02
##  Max.   :1630.0   Max.   :60.95   Max.   :1.099e-01   Max.   :4.199e-02
##      qSect2              qSect3              dSect1
##  Min.   :2.750e-07   Min.   :4.700e-07   Min.   :0.0001191
##  1st Qu.:1.410e-03   1st Qu.:2.646e-03   1st Qu.:0.0499775
##  Median :1.600e-02   Median :2.051e-02   Median :0.2630000
##  Mean   :2.600e-02   Mean   :3.195e-02   Mean   :0.2440272
##  3rd Qu.:5.216e-02   3rd Qu.:7.017e-02   3rd Qu.:0.3669500
##  Max.   :7.096e-02   Max.   :8.299e-02   Max.   :0.5814000
##      dSect2              dSect3              vSect1           vSect2
##  Min.   :0.0001795   Min.   :0.0002024   Min.   :0.1234   Min.   :0.1408
##  1st Qu.:0.0567675   1st Qu.:0.0669225   1st Qu.:2.3525   1st Qu.:2.3792
##  Median :0.2673000   Median :0.2559000   Median :5.5385   Median :4.8370
##  Mean   :0.3204231   Mean   :0.2889875   Mean   :4.7209   Mean   :4.4493
##  3rd Qu.:0.5885000   3rd Qu.:0.5405500   3rd Qu.:6.5200   3rd Qu.:6.5113
##  Max.   :0.7812000   Max.   :0.6127000   Max.   :8.2470   Max.   :7.4360
##      vSect3
##  Min.   : 0.1859
##  1st Qu.: 3.5060
##  Median : 6.9165
##  Mean   : 6.3998
##  3rd Qu.: 9.3090
##  Max.   :10.5900
```

and for the sedigraph data:

```
summary (sedigraph)
```

```
##       time           sedPlot             plotFlux             sedCS1
##  Min.   :   1.0   Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.000e+00
##  1st Qu.: 408.2   1st Qu.:2.500e-11   1st Qu.:5.000e-12   1st Qu.:5.100e-12
##  Median : 815.5   Median :4.667e-08   Median :9.335e-09   Median :6.161e-08
##  Mean   : 815.5   Mean   :4.797e-06   Mean   :9.593e-07   Mean   :5.192e-07
##  3rd Qu.:1222.8   3rd Qu.:1.536e-06   3rd Qu.:3.072e-07   3rd Qu.:3.817e-07
##  Max.   :1630.0   Max.   :6.325e-05   Max.   :1.265e-05   Max.   :5.527e-06
##      sedCS2             sedCS3            plotYield
##  Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.000e+00
##  1st Qu.:4.500e-12   1st Qu.:3.070e-11   1st Qu.:8.100e-14
##  Median :2.274e-08   Median :2.850e-07   Median :1.531e-10
##  Mean   :6.495e-07   Mean   :2.154e-06   Mean   :1.573e-08
##  3rd Qu.:5.414e-07   3rd Qu.:2.035e-06   3rd Qu.:5.035e-09
##  Max.   :7.834e-06   Max.   :2.264e-05   Max.   :2.074e-07
##     yieldCS1            yieldCS2            yieldCS3              phi1
##  Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.0000
##  1st Qu.:7.900e-13   1st Qu.:3.200e-13   1st Qu.:1.430e-12   1st Qu.:0.3394
##  Median :9.479e-09   Median :1.624e-09   Median :1.326e-08   Median :0.7697
##  Mean   :7.988e-08   Mean   :4.639e-08   Mean   :1.002e-07   Mean   :0.6572
##  3rd Qu.:5.872e-08   3rd Qu.:3.867e-08   3rd Qu.:9.465e-08   3rd Qu.:0.9547
##  Max.   :8.504e-07   Max.   :5.596e-07   Max.   :1.053e-06   Max.   :0.9999
##       phi2               phi3                phi4                phi5
##  Min.   :0.00000   Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.000e+00
##  1st Qu.:0.01065   1st Qu.:8.500e-07   1st Qu.:5.000e-09   1st Qu.:7.100e-15
##  Median :0.07905   Median :2.840e-05   Median :1.860e-07   Median :2.462e-13
##  Mean   :0.30390   Mean   :3.578e-02   Mean   :2.518e-03   Mean   :5.752e-10
##  3rd Qu.:0.56827   3rd Qu.:3.043e-03   3rd Qu.:7.919e-05   3rd Qu.:3.075e-11
##  Max.   :0.99040   Max.   :9.473e-01   Max.   :7.818e-02   Max.   :1.652e-08
##       phi6
##  Min.   :0
##  1st Qu.:0
##  Median :0
##  Mean   :0
##  3rd Qu.:0
##  Max.   :0
```

Note that read.table is a very flexible function with lots of different options for reading tabular data into R. For reference (i.e. do NOT run this code chunk):

```
read.table (file, header = FALSE, sep = "", quote = "\"'",
            dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
            row.names, col.names, as.is = !stringsAsFactors,
            na.strings = "NA", colClasses = NA, nrows = -1,
            skip = 0, check.names = TRUE, fill = !blank.lines.skip,
            strip.white = FALSE, blank.lines.skip = TRUE,
            comment.char = "#",
            allowEscapes = FALSE, flush = FALSE,
            stringsAsFactors = FALSE,
            fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Visualizing this Output

The tidyverse has a very powerful plotting capability called ggplot (we can also use it without loading the rest of the tidyverse by opening the ggplot2 library).

To take full advantage of ggplot, we need first to do a little change to the layout of the data. At the moment the hydrograph and sedigraph data frames are in what's called wide format; in other words, it looks like a table with column headings relating to the variable names, and rows for each set of values (which in this case refer to a specific year). Although ggplot can use data in this format (as we'll see in a bit), it is far more powerful if we change our data to what's called long format by stacking the variables on top of each other. To make it simpler, we'll first create a data frame with just the three variables we're interested in:

```
combinedData <- hydrograph %>%
  select (time, rain, qOutflow) %>%
  left_join (sedigraph, by = c ("time")) %>%
  select (time, rain, qOutflow, sedPlot)
```

So in these stages, we've taken the time, rain and qOutflow values from the hydrograph data, used left_join to add the sedigraph data, matching the rows by the time variable, and then using select again to keep only sedPlot from the sedigraph data. Using left-join like this is a vary powerful way of allowing us to combine data for analysis – it would be equally easy to do in a spreadsheet for datasets like this with common time values, but imagine what you would have to do if the timesteps were different.

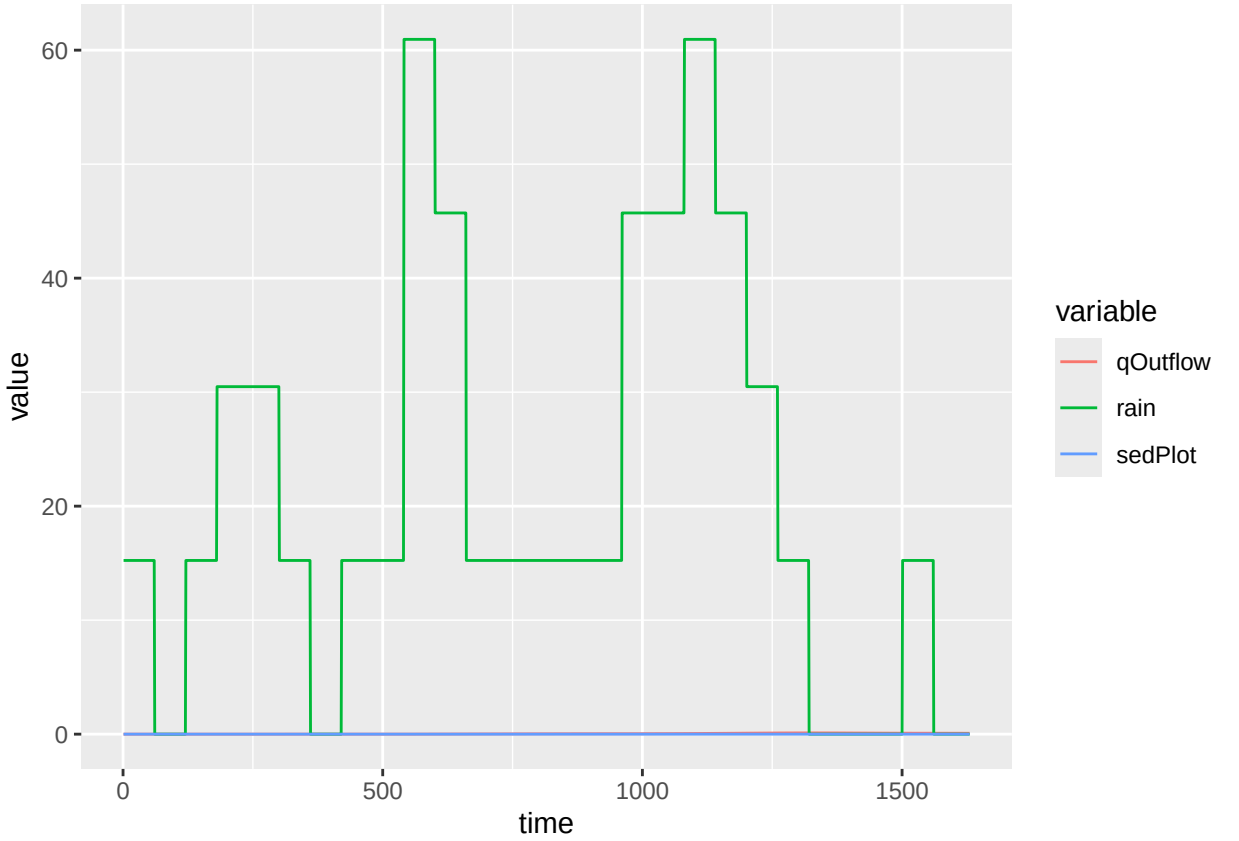
We can then use pivot_longer from tidyverse to create the long format:

```
combinedDataLong <- combinedData %>%
  pivot_longer (-time, names_to = "variable")

head (combinedDataLong, 8)
```

```
## # A tibble: 8 x 3
##    time variable    value
##   <dbl> <chr>       <dbl>
## 1     1 rain     1.52e+ 1
## 2     1 qOutflow 9.69e- 8
## 3     1 sedPlot  0
## 4     2 rain     1.52e+ 1
## 5     2 qOutflow 2.74e- 7
## 6     2 sedPlot  1.41e-16
## 7     3 rain     1.52e+ 1
## 8     3 qOutflow 5.04e- 7
```

The format of ggplot will appear a little strange at first, but makes sense when you get used to it. You need to specify a data source, aesthetics (which are minimally the x and y values, but can also include colours and line types etc.), and a geometry (points, lines, bars, and many more). So let's try a simple call to plot all our data as lines:

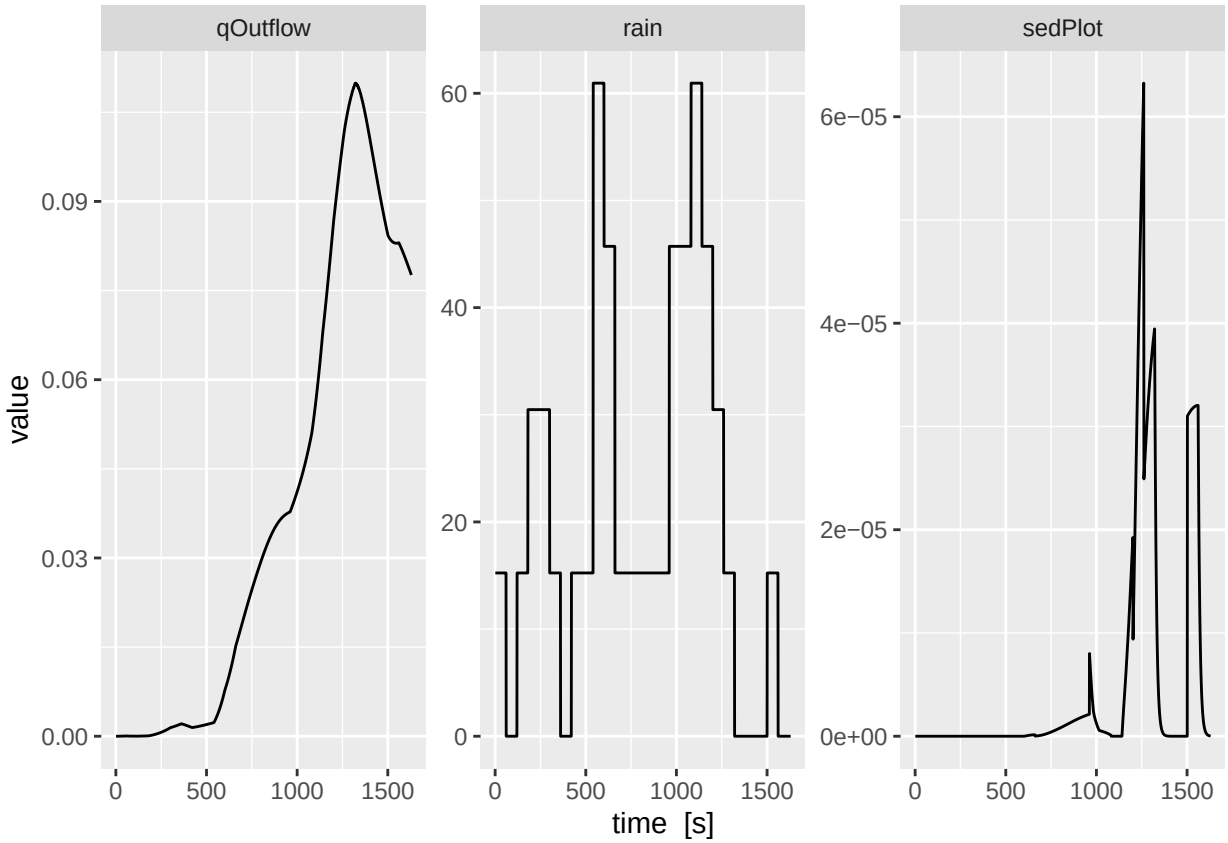
```
ggplot (data = combinedDataLong) +
  geom_line (aes (x = time, y = value, colour = variable))
```



Of course this is still rubbish because of the very different scales, but it's produced a graph with all our data on with just two lines of code, including a basic legend.

But there is an option called facet_wrap in ggplot to plot the data from the different variables into separate graphs:

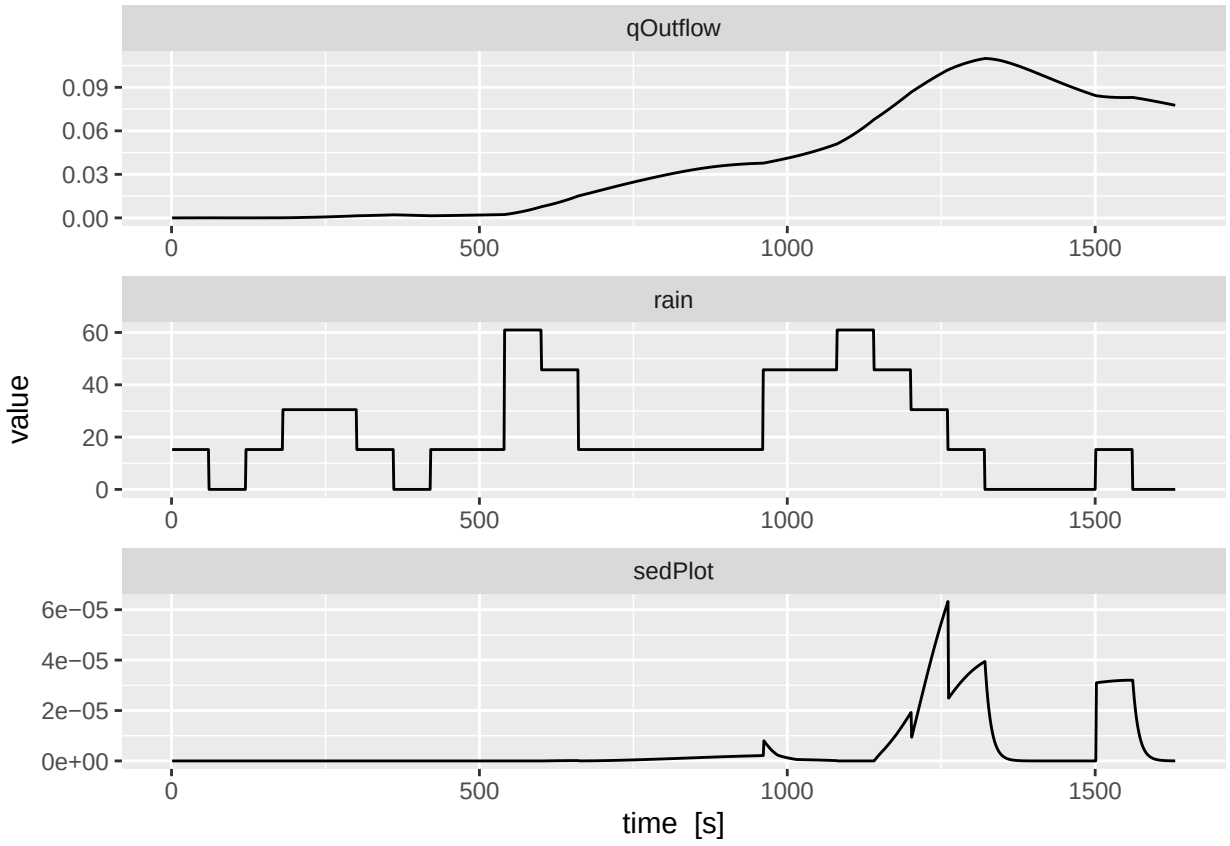
```
combinedDataLong %>%
  ggplot () +
    geom_line (aes (x = time, y = value)) +
    facet_wrap (~variable, scales = "free") +
    labs (x = "time  [s]")
```



Check what happens when you remove the scales = "free" part of the facet_wrap function call. Can you think why it might be useful to have both forms of graph when exploring your data?

And actually, rather than plotting them side-by-side, what about plotting them one above each other:

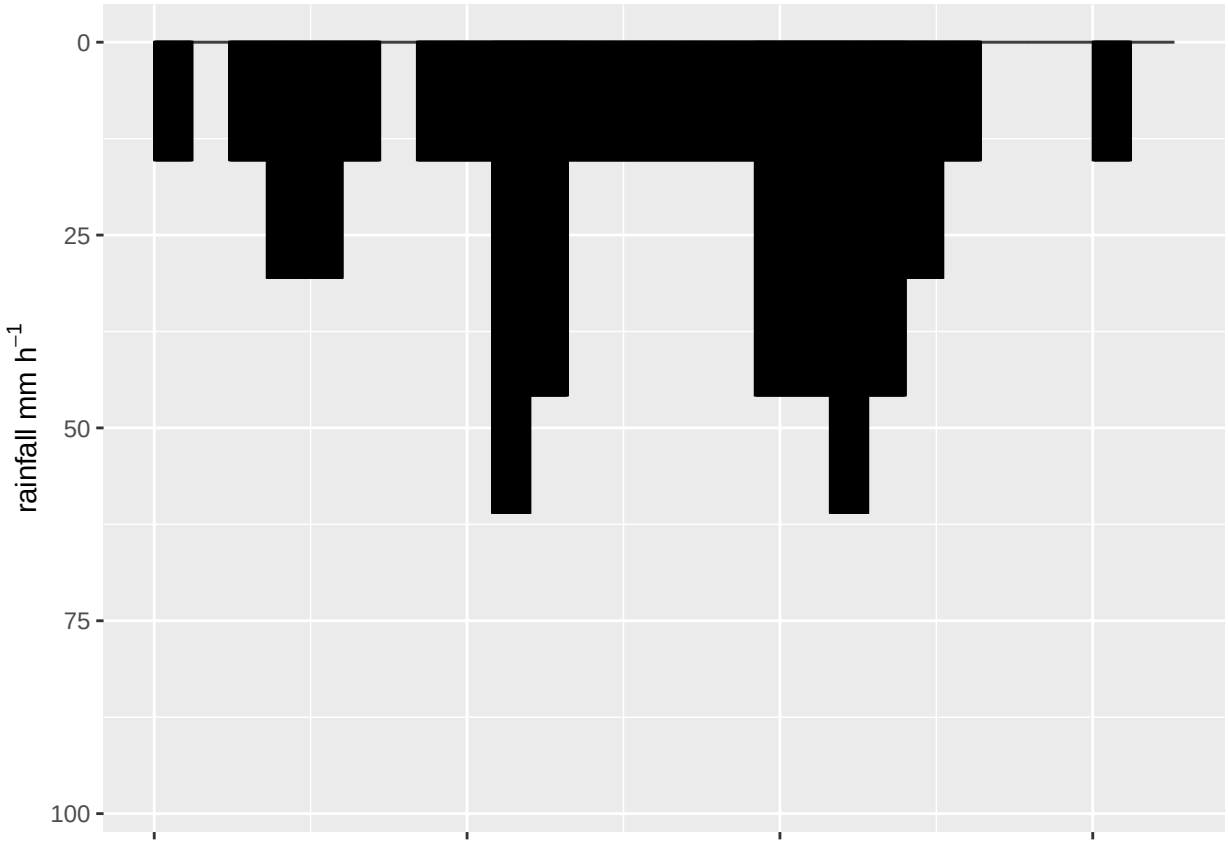
```
combinedDataLong %>%
  ggplot () +
    geom_line (aes (x = time, y = value)) +
    facet_wrap (~variable, scales = "free", ncol = 1) +
    labs (x = "time  [s]")
```



This approach is starting to look more useful for visualization. This is a very powerful way of looking at multiple variables at the same time.

We can add different labels for each of the different facets in this graph, but again it's a bit fussy. Let's return to wide format and generate the panels separately, and then add them back together. We also traditionally put rainfall at the top, whereas facet_wrap has given us the graphs in alphabetical order of the variable names. Rainfall is also usually plotted as bars pointing downwards, so let's see if we can do that using geom_col (columns) to give that format:

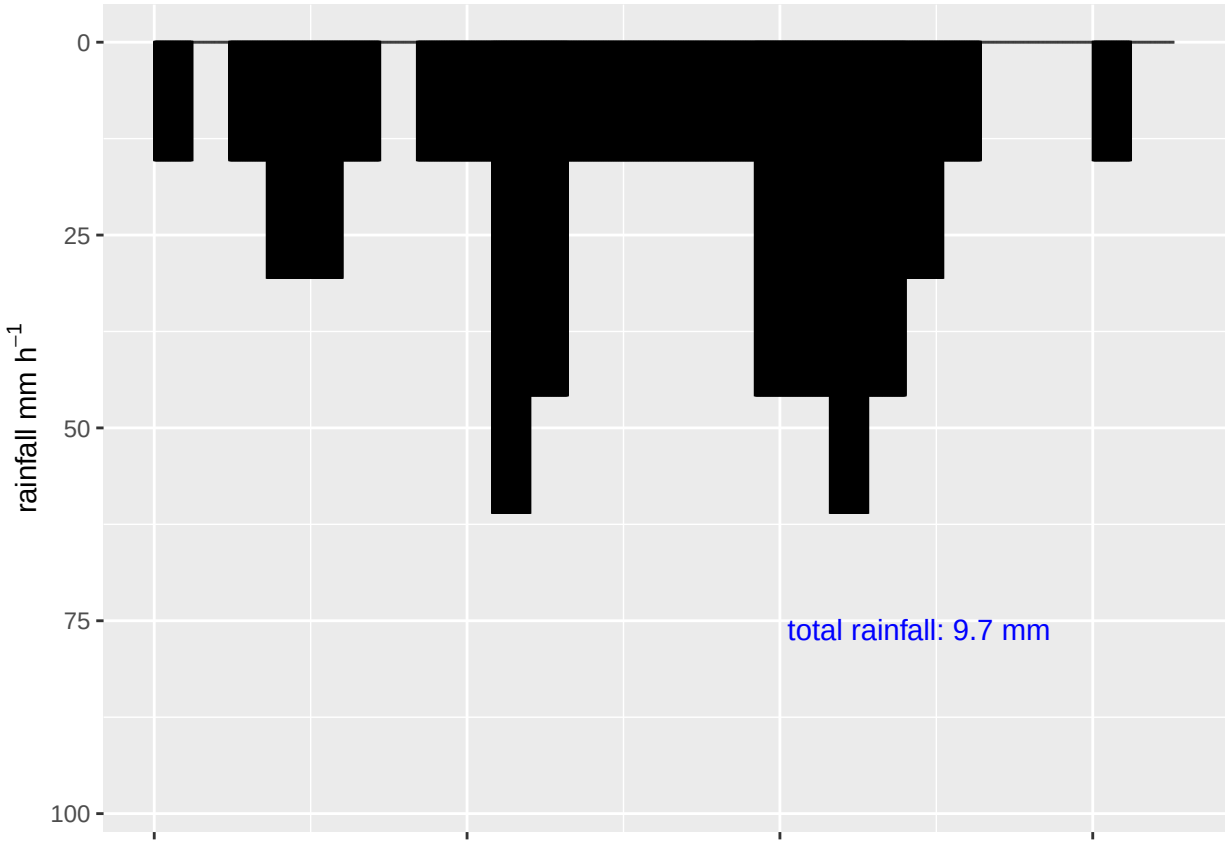
```
#calculate some useful statistics
maxTime <- max (hydrograph$time)
maxRain <- max (hydrograph$rain)
dt <- hydrograph$time[2] - hydrograph$time[1]
totalRain <- sum (hydrograph$rain / 3600.) * dt

rainPanel <- ggplot (hydrograph) +
  geom_col (aes (x = time, y = rain), width = 1, colour = "black") +
  scale_y_reverse (limits = c (maxRain * 1.6, 0)) +
  ylab (expression (paste ("rainfall mm h"^-1))) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

rainPanel
```



Notice two changes here: the x axis labelling has been switched off for reasons that will become clearer shortly, and I've stored the graph as a data structure, and then used its name to plot it out (comment out the line with the name in to see what happens).

We can add other information to an existing plot if we store it this way. Say I want to add an annotation showing the total event rainfall, I can simply take the existing stored ggplot object and add to it:

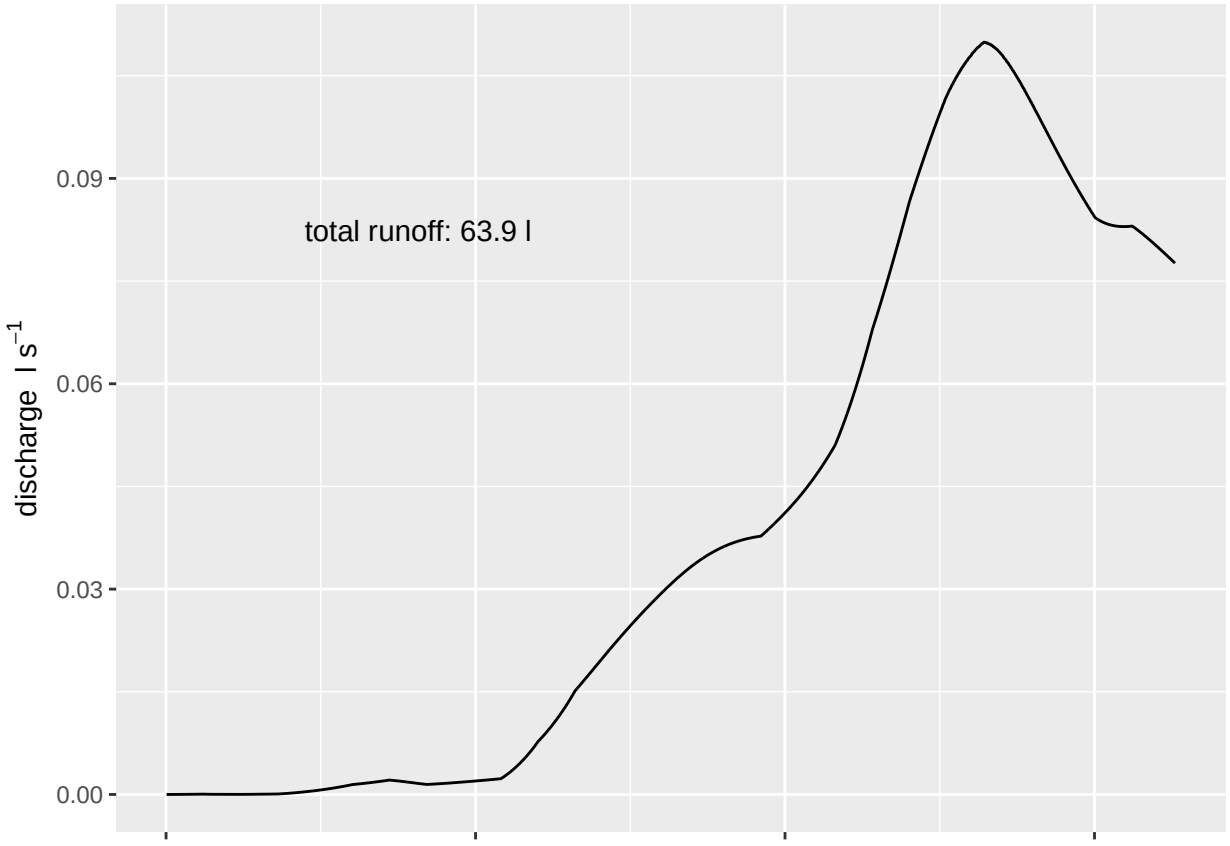
```
rainPanel <- rainPanel +
  annotate ("text",
            x = 0.75 * maxTime,
            y = 1.25 * maxRain,
            label = paste ("total rainfall:",
                           round (totalRain, digits = 1),
                           "mm", sep = " "),
            colour = "blue")

rainPanel
```



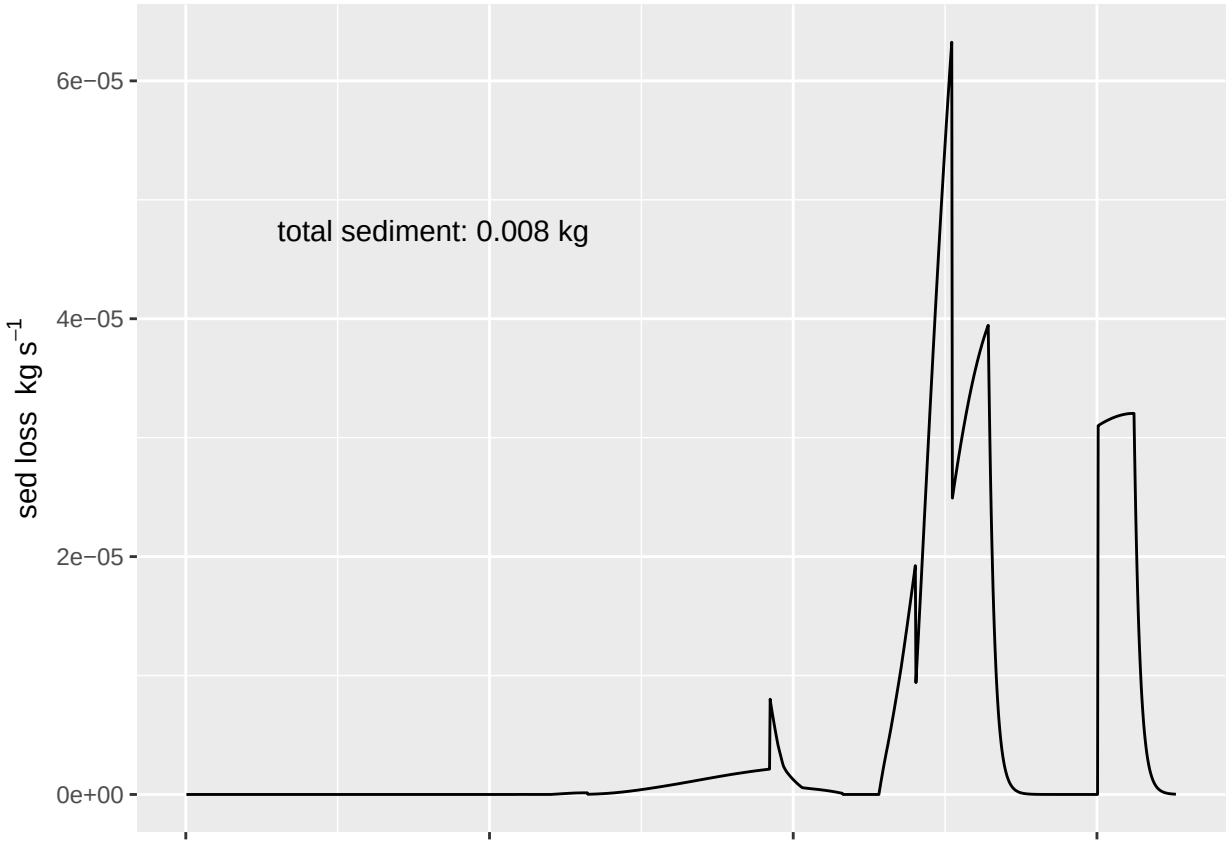
Of course, we could have added the annotate call as part of the original definition, but you now see the flexibility of this approach.

We can then follow through the same process to produce panels for the other variables:

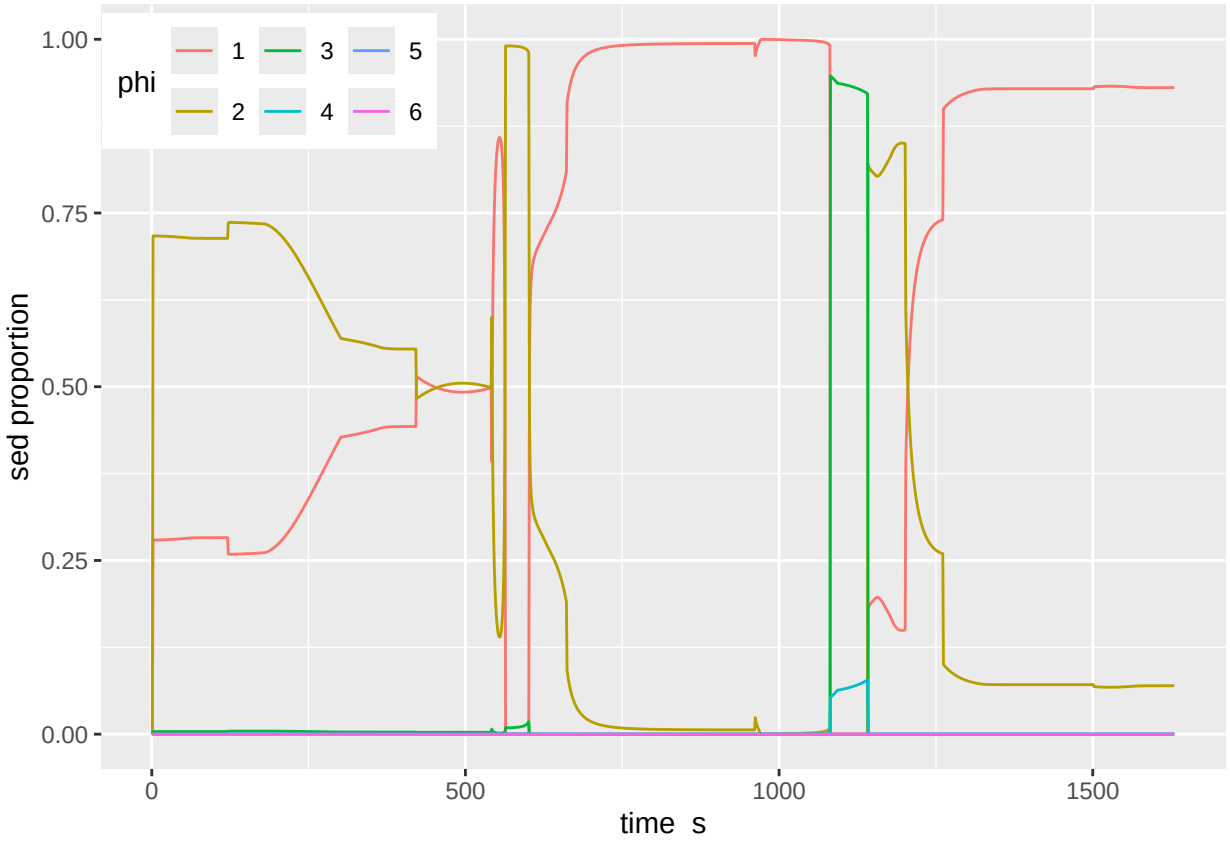
```
#
# calculate stats
#
maxFlow <- max (hydrograph$qOutflow)
totalFlow <- sum (hydrograph$qOutflow) * dt
maxSed <- max (sedigraph$sedPlot)
totalSed <- sum (sedigraph$sedPlot) * dt

flowPanel <- ggplot (hydrograph) +
  geom_line (aes (x = time, y = qOutflow)) +
  xlab ("time  s") +
  ylab (expression (paste ("discharge  l s"^-1))) +
  annotate ("text",
            x = 0.25 * maxTime,
            y = 0.75 * maxFlow,
            label = paste ("total runoff:",
                           round (totalFlow, digits = 1),
                           "l", sep = " "))  +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

flowPanel
```




```
sedPanel <- ggplot (sedigraph) +
  geom_line (aes (x = time, y = sedPlot)) +
  xlab ("time  s") +
  ylab (expression (paste ("sed loss  kg s"^-1))) +
  annotate ("text",
            x = 0.25 * maxTime,
            y = 0.75 * maxSed,
            label = paste ("total sediment:",
                           round (totalSed,
                                  digits = 3),
                           "kg", sep = " ")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

sedPanel
```

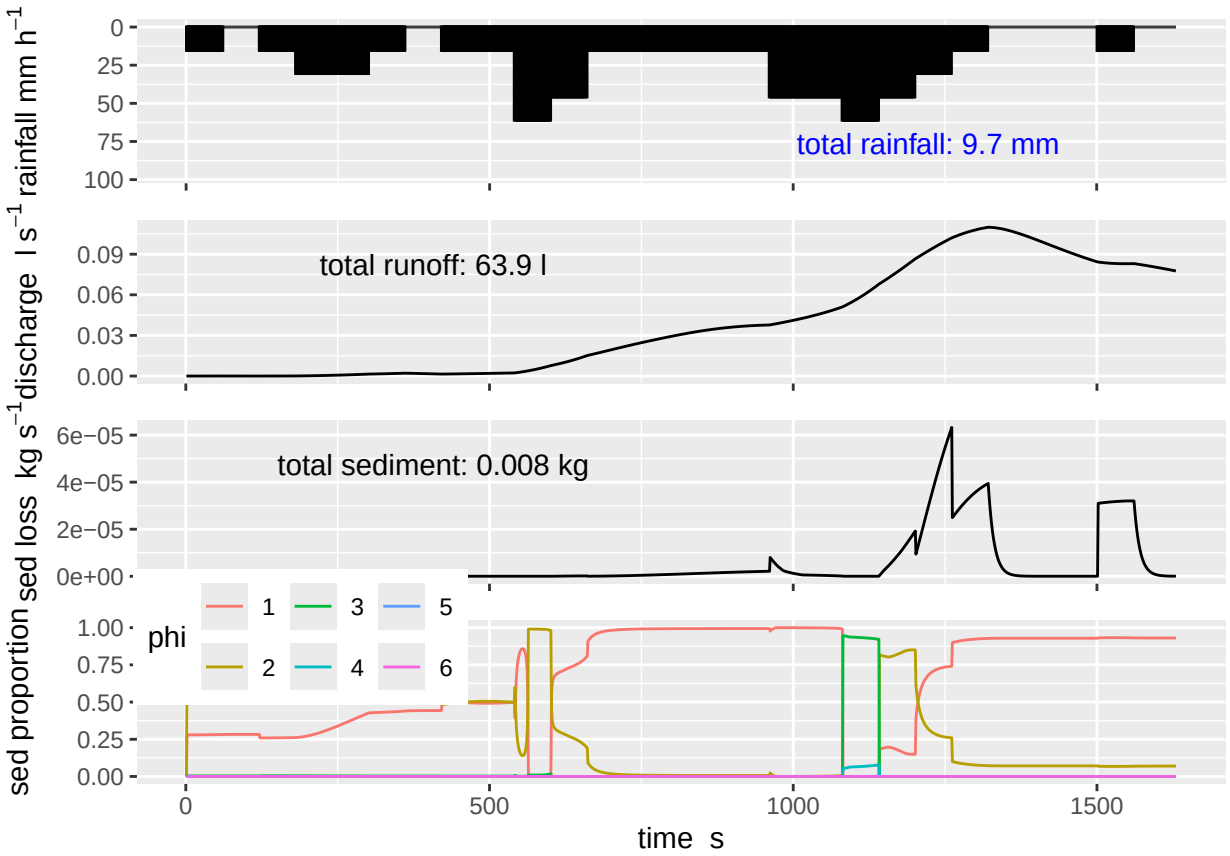



```
phiPanel <- ggplot (sedigraph, aes (x = time)) +
  geom_line (aes (y = phi1, colour = "1")) +
  geom_line (aes (y = phi2, colour = "2")) +
  geom_line (aes (y = phi3, colour = "3")) +
  geom_line (aes (y = phi4, colour = "4")) +
  geom_line (aes (y = phi5, colour = "5")) +
  geom_line (aes (y = phi6, colour = "6")) +
  labs (x = "time  s",
        y = "sed proportion",
        colour = "phi") +
  theme (legend.position = "inside",
         legend.position.inside = c (0.15, 0.9),
         legend.direction = "horizontal")

phiPanel
```



Notice for the last panel I have added the x-axis information. Now we could copy and paste all of these images into our favourite graphics editor to stack them up, but it's much easier to do it directly in R. There are two steps: first we create an object that has all of our panels as a set of graphics objects (Grobs), and then we use the grid package to plot it.

```
allPanels <- rbind (ggplotGrob (rainPanel),
                    ggplotGrob (flowPanel),
                    ggplotGrob (sedPanel),
                    ggplotGrob (phiPanel),
                    size = "max")
grid.newpage ()
grid.draw (allPanels)
```



The y axis labels overwrite and the legend is too big, but if you use the function windows() and then plot in a larger window, this issue goes away (so we don't have to tweak font sizes). Do this in the console if your window vanishes as soon as it's finished:

```
windows ()
grid.newpage ()
grid.draw (allPanels)
```

Even better, we don't have to copy and paste or save the final window, we can save the file directly from R:

```
ggsave (filename = "./mahleranOutput.png", allPanels,
        width = 210, height = 250, units = "mm")
```

You can save in different formats just by changing the filename extension ("eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (on windows only) are the different options).

We now have a publication-quality reproducible output of our results.

In terms of data management, it might be better to save our results (e.g. the image) into a separate Results folder within the project, so you have versions of the script, input data, and output results all in separate places.

At Last! A(nother) Benefit of Coding

So far we have used code that has read in data and plotted them out. We can read in data from a new file and use the same code if the data frame name is the same. But what if we want to make the process much more flexible and produce output for any input that has the right format? This is where being able to define our own functions comes into play.

We define a function as follows. This very simple example takes two input values and adds them together:

```
simpleFunction <- function (a, b){
  answer <- a + b
  return (answer)
}
```

Nothing seems to have happened – but if you look over in the Environment pane in RStudio (top right), you should see your simpleFunction now appears as being available to you. We can now call this as a function in the same way as any other:

```
simpleFunction (1, 2)
```

```
## [1] 3
```

It's not a very good function as it's easy to break:

```
# this call WILL produce an error
simpleFunction ("fish", "bananas")
```

```
## Error in a + b: non-numeric argument to binary operator
```

but it shows the principle: we pass input values in brackets to the function definition, and use return to return a value to the user. That returned value can be used in the same way as other variables:

```
addAB <- simpleFunction (1, 2)

addAB
```

```
## [1] 3
```

So let's make a more complicated function where we collect all the panel plots to automate the process, starting from reading the data to saving the file. The only difference from the code above is the part that specifies the filename using the number of the run rather than directly as the filename:

```
mahleranPlotOutflows <- function (runNumber = 1,
                                  filePath = "./",
                                  outPath = "./",
                                  outFile = "mahleranOutput.png"){
  # define file names based on path and run number
  if (runNumber < 10){
    hydroFile <- paste0 (filePath, 'hydro00', runNumber, '.dat')
    sediFile  <- paste0 (filePath, 'sedtr00', runNumber, '.dat')
  }
  if (runNumber >= 10 & runNumber < 100) {
    hydroFile <- paste0 (filePath, 'hydro0', runNumber, '.dat')
    sediFile  <- paste0 (filePath, 'sedtr0', runNumber, '.dat')
  }
  if (runNumber >= 100 & runNumber < 1000) {
    hydroFile <- paste0 (filePath, 'hydro', runNumber, '.dat')
    sediFile  <- paste0 (filePath, 'sedtr', runNumber, '.dat')
  }

  #read in files
  hydrograph <- read.table (hydroFile,
                            header = FALSE, sep = "")
  sedigraph <- read.table (sediFile,
                           header = FALSE, sep = "")
  #set up variable names
```

```
  #
  # names for variables:
  #   time = time since start of simulation in seconds
  #   rain = rainfall intensity in mm/s in input file, converted to mm/h below
  #   qOutflow = outflow from plot/slope/catchment in mm3/s in input file, converted to litres below
  #   qSect1, qSect2, qSect3 - flows at three cross sections in mm3/s in input file, converted to
  #      litres below
  #   dSect1, dSect2, dSect3 - mean flow depths at three cross sections in mm in input file
  #   vSect1, vSect2, vSect3 - mean flow depths at three cross sections in mm in input file
  names (hydrograph) <- c ("time", "rain", "qOutflow",
                           "qSect1", "qSect2", "qSect3",
                           "dSect1", "dSect2", "dSect3",
                           "vSect1", "vSect2", "vSect3")
#
# names for variables:
#   time = time since start of simulation in seconds
#   sedPlot = total sediment coming off plot in kg/s
#   plotFlux = flux of sediment coming off plot in kg/m/s
#   sedCS1 = sediment crossing section 1 in kg/s
#   sedCS2 = sediment crossing section 2 in kg/s
#   sedCS3 = sediment crossing section 3 in kg/s
#   plotYield = plot sediment yield in kg/m^2/s
#   yieldCS1 = sediment yield crossing section 1 in kg/m2/s
#   yieldCS2 = sediment yield crossing section 2 in kg/m2/s
#   yieldCS3 = sediment yield crossing section 3 in kg/m2/s
#   phi1 = plot sediment outflow -- proportion < 0.063 mm
#   phi2 = plot sediment outflow -- proportion >= 0.063 mm - 0.25 mm
#   phi3 = plot sediment outflow -- proportion >= 0.25 mm - 0.5 mm
#   phi4 = plot sediment outflow -- proportion >= 0.5 mm - 2 mm
#   phi5 = plot sediment outflow -- proportion >= 0.2 mm - 12 mm
#   phi6 = plot sediment outflow -- proportion >= 12 mm
#
  names(sedigraph) <- c ("time", "sedPlot", "plotFlux",
                         "sedCS1", "sedCS2", "sedCS3",
                         "plotYield", "yieldCS1", "yieldCS2", "yieldCS3",
                         "phi1", "phi2", "phi3", "phi4", "phi5", "phi6")
  # and do unit conversions
  hydrograph$rain <- hydrograph$rain * 3600.  ## to mm/h
  hydrograph$qOutflow <- hydrograph$qOutflow / 1000000.  ## to litres
  hydrograph$qSect1 <- hydrograph$qSect1 / 1000000.  ## to litres
  hydrograph$qSect2 <- hydrograph$qSect2 / 1000000.  ## to litres
  hydrograph$qSect3 <- hydrograph$qSect3 / 1000000.  ## to litres

  # calculate some useful statistics
  maxTime <- max (hydrograph$time)
  maxRain <- max (hydrograph$rain)
  dt <- hydrograph$time[2] - hydrograph$time[1]
  totalRain <- sum (hydrograph$rain / 3600.) * dt

  # produce rainfall panel
  rainPanel <- ggplot (hydrograph) +
    geom_col (aes (x = time, y = rain), width = 1, colour = "black") +
    scale_y_reverse (limits = c (maxRain * 1.6, 0)) +
```

```
  ylab (expression (paste ("rainfall mm h"^-1))) +
  annotate ("text",
            x = 0.75 * maxTime,
            y = 1.25 * maxRain,
            label = paste ("total rainfall:",
                           round (totalRain, digits = 1),
                           "mm", sep = " "),
            colour = "blue") +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

#
# calculate even more useful stats
#
maxFlow <- max (hydrograph$qOutflow)
totalFlow <- sum (hydrograph$qOutflow) * dt
maxSed <- max (sedigraph$sedPlot)
totalSed <- sum (sedigraph$sedPlot) * dt

# produce flow panel
flowPanel <- ggplot (hydrograph) +
  geom_line (aes (x = time, y = qOutflow)) +
  xlab ("time  s") +
  ylab (expression (paste ("discharge  l s"^-1))) +
  annotate ("text",
            x = 0.25 * maxTime,
            y = 0.75 * maxFlow,
            label = paste ("total runoff:",
                           round (totalFlow, digits = 1),
                           "l", sep = " "))  +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())
# produce sediment panel
sedPanel <- ggplot (sedigraph) +
  geom_line (aes (x = time, y = sedPlot)) +
  xlab ("time  s") +
  ylab (expression (paste ("sed loss  kg s"^-1))) +
  annotate ("text",
            x = 0.25 * maxTime,
            y = 0.75 * maxSed,
            label = paste ("total sediment:",
                           round (totalSed,
                                  digits = 3),
                           "kg", sep = " ")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())
# produce particle size panel
phiPanel <- ggplot (sedigraph, aes (x = time)) +
  geom_line (aes (y = phi1, colour = "1")) +
  geom_line (aes (y = phi2, colour = "2")) +
  geom_line (aes (y = phi3, colour = "3")) +
  geom_line (aes (y = phi4, colour = "4")) +
  geom_line (aes (y = phi5, colour = "5")) +
```

```
    geom_line (aes (y = phi6, colour = "6")) +
    labs (x = "time  s",
          y = "sed proportion",
          colour = "phi") +
    theme (legend.position = "inside",
           legend.position.inside = c (0.15, 0.9),
           legend.direction = "horizontal")
  # group panels vertically
  allPanels <- rbind (ggplotGrob (rainPanel),
                      ggplotGrob (flowPanel),
                      ggplotGrob (sedPanel),
                      ggplotGrob (phiPanel),
                      size = "max")
  # open new window and plot panels
  windows ()
  grid.newpage ()
  grid.draw (allPanels)

  # save the panels directly to file
  ggsave (filename = paste0 (outPath, outFile),
          allPanels,
          width = 210, height = 250, units = "mm")
  #return some useful statistics
  summaryOutput <- data.frame ("Total Rainfall" = totalRain,
                               "Total Flow" = totalFlow,
                               "Total Sediment" = totalSed)

  return (summaryOutput)

}
```

Then call the function with the relevant parameters (note that I've cunningly used the default names specified in the function for all but the location of the data files, so that's the only thing specified here). I've also provided a simple data frame outputting the average statistics.

```
mahleranPlotOutflows (filePath = mahleranOutputFolder)
```

```
##   Total.Rainfall Total.Flow Total.Sediment
## 1        9.65142   63.92307    0.007818307
```

When calling the function, it might be better to specify which run number you're interested in so you don't get confused:

```
mahleranPlotOutflows (runNumber = 1, filePath = mahleranOutputFolder)
```

```
##   Total.Rainfall Total.Flow Total.Sediment
## 1        9.65142   63.92307    0.007818307
```

And you should see the same output as before. You don't need to specify the names of the variables in the input as long as they're in the same order as specified in the function call. So, a third way of getting the same output is:

```
mahleranPlotOutflows (1, mahleranOutputFolder)
```

```
##   Total.Rainfall Total.Flow Total.Sediment
## 1        9.65142   63.92307    0.007818307
```

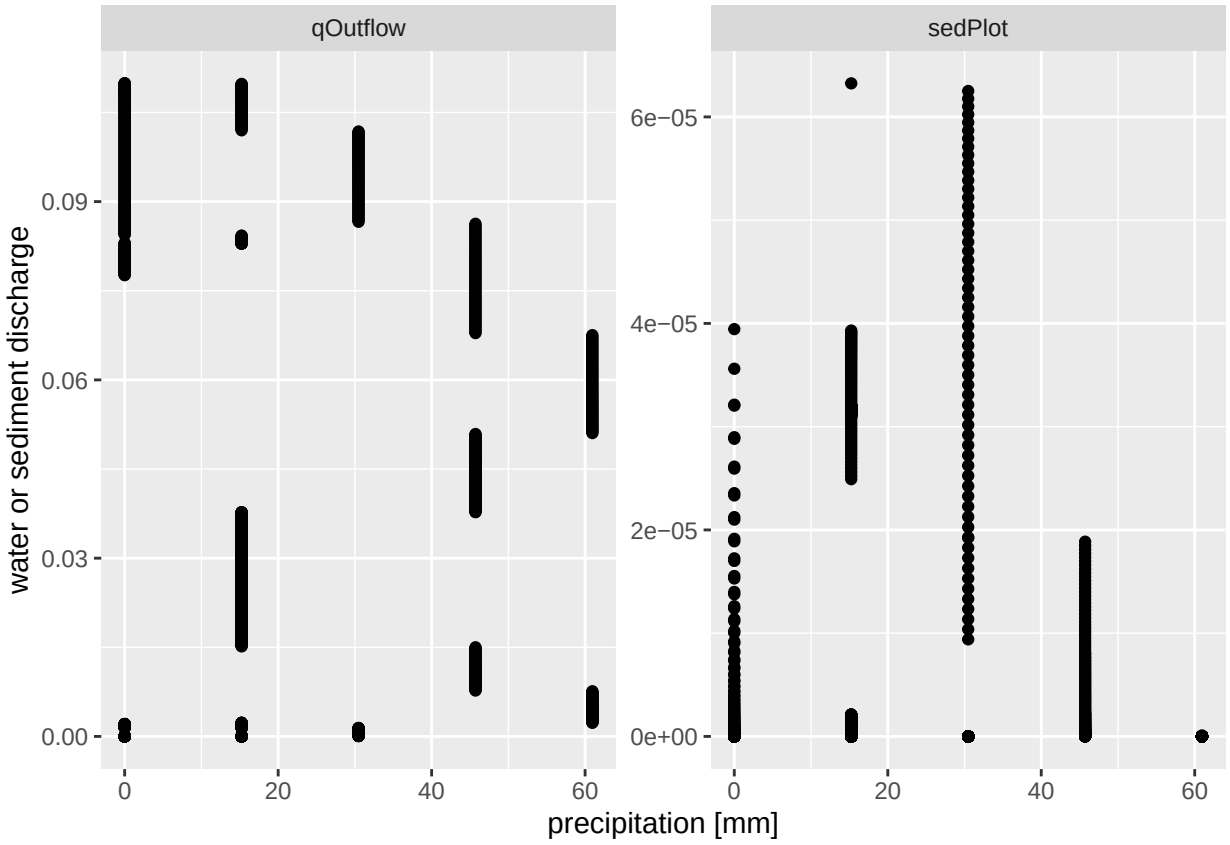
Before, you changed the input file to use the doubled rainfall amount. Use your function to see how it is different. What happens if you run the function with a run number that doesn't exist in your output folder?

Now copy the code for this function and paste it into a new R script file called MahleranVis.R (with a .R extension, NOT an Rmd file). Then all you need to do to be able to use the function in any other code is to add:

```
# DON'T run this -- it won't work as you need to specify the path name properly
#  and you need to have generated the MahleranVis.R file
source ("Path/To/Script/File/On/Your/Computer/MahleranVis.R")
```

Before we leave this dataset, let's have a quick think about what else we might want to do with it. For example, we might want to see if there are patterns in the link between precipitation and runoff and sediment transfer rather than just plotting results as time series. Again we can plot variable by variable, or use facet_wrap in ggplot to simplify things:

```
combinedData %>%
  select (-time) %>%
  pivot_longer (-rain, names_to = "variable") %>%
  ggplot () +
    geom_point (aes (x = rain, y = value)) +
    facet_wrap (~variable, scales = "free") +
    labs (x = "precipitation [mm]",
          y = "water or sediment discharge")
```



Note the approach again not to change the dataset, but we take the dataframe, pass it through a couple of formatting statements from tidyverse and then plot.

What About the Other Output Files?

You should be able to modify the script above straightforwardly to visualize the other time-series data. In the meantime, let's look at visualizing the spatial outputs given this is a fully distributed model.

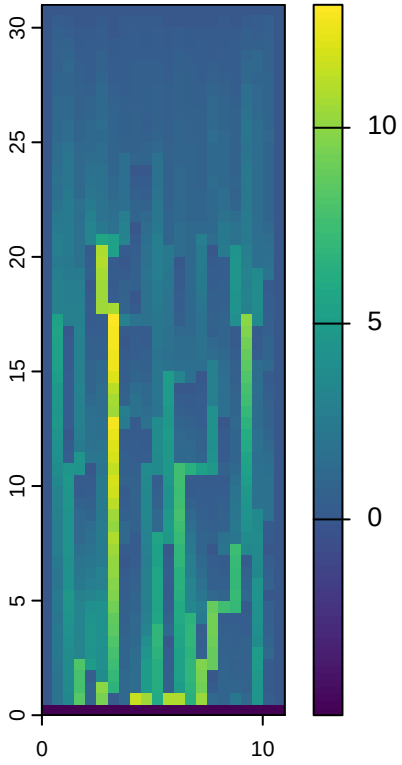
The map outputs from MAHLERAN are as follows:

File Name	Description
dschg***.asc	Total event discharge [m3]
depth***.asc	Peak flow depth [mm]
veloc***.asc	Peak flow velocity [mm s–1]
sedtr***.asc	Sediment transport [total sediment mass in kg]
ksat_***.asc	Final infiltration rate (copied from the input)
detac***.asc	Total detachment [kg]
depos***.asc	Total deposition [kg]
soild***.asc	Map of dmax_soil [m]
soilv***.asc	Map of vmax_soil [mm s–1]
neter***.asc	Net erosion [kg]
radet***.asc	Total raindrop detachment [kg]
fldet***.asc	Total flow detachment [kg]
nutri***. asc	NH4 transport [total ammonium mass in mg]
pnitr***.asc	Particulate-bound nitrate flux from each cell (g)
pammo***.asc	Particulate-bound ammonium flux from each cell (g)
pTNxx***.asc	Particulate-bound TN flux from each cell (g)
pTPxx***.asc	Particulate-bound TP flux from each cell (g)
pICxx***.asc	Particulate-bound IC flux from each cell (g)
pTCxx***.asc	Particulate-bound TC flux from each cell (g)

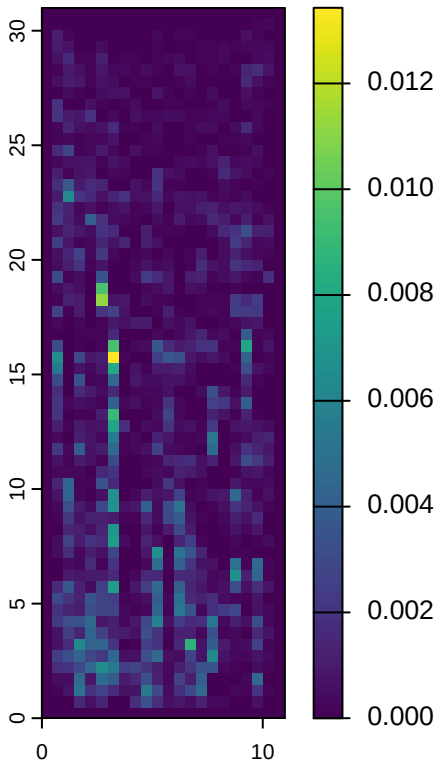
Again, the *** represents the run number as above.

You can open the .asc files to see they are human readable raster files and have simple information about grid and cell size. ArcGIS or QGIS will open these files directly. But we also saw last time how to open raster files and plot them using the terra package in R. Let's start by looking at the total event discharge:

```
dschgFile <- "dschg001.asc"

qMap <- rast (paste0 (mahleranOutputFolder, dschgFile))

# notice we can scale the discharge directly -- here converting from cubic metres to litres
plot (qMap * 1000.)
```

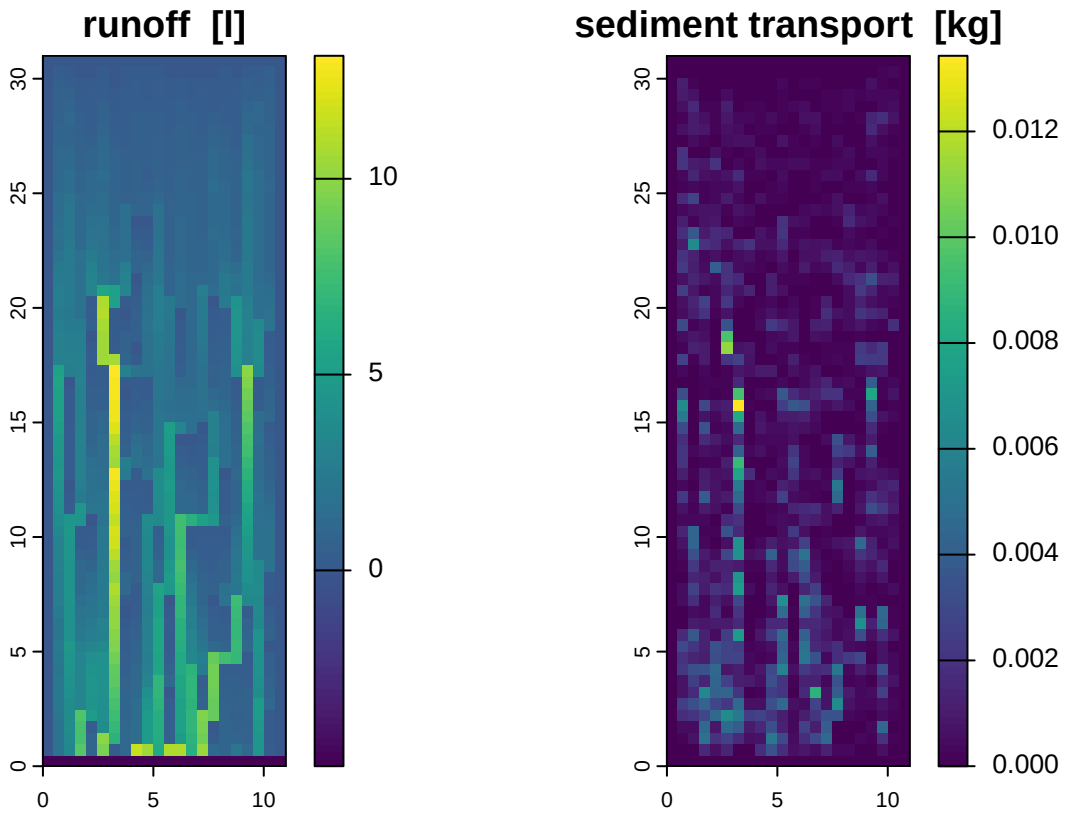



```
sedQFile <- "sedtr001.asc"

sedQMap <- rast (paste0 (mahleranOutputFolder, sedQFile))

# there are artefacts in the output on the bottom row which we need to remove from the map
#   before plotting (these artefacts are about the way outflows are calculated -- there is
#   nothing wrong with the results)
sedQMap[sedQMap < 0] <- 0  #replace all negative values with zeroes
plot (sedQMap)
```



It's useful to plot the runoff and sediment transport values side-by-side.

```
#tell R that we want one row and two columns of plots
par (mfrow = c (1, 2))
#plot the discharge map in the first column
plot (qMap * 1000., main = "runoff  [l]")
#plot the sediment transport map in the second column
plot (sedQMap, main = "sediment transport  [kg]")
```




```
#tell R to revert to a single plot at a time
par (mfrow = c (1, 1))
```

Try writing a function that will plot the runoff and sediment discharge next to each other like this, specifying the run number.

Try producing a further function by modifying the previous one to show the flow depth and velocity (i.e. flow hydraulics) as well as the water and sediment discharges. Save all the functions in your MahleranVis.R script file for when you need them for further analyses.