

STRATIGRAPHER: MAKING AND USING LITHOLOGS IN R

Devleeschouwer X. 1, Da Silva A.-C. 2, Boulvain F. 2 & Wouters S. 1, 2*

1. O.D. Earth and History of Life, Royal Belgian Institute of Natural Sciences

2. Sedimentary Petrology, University of Liège

* Corresponding and presenting author:
sebastien.wouters@doct.uliege.be



This presentation is an introduction to R and to StratigrapherR. It was originally supposed to be a poster only on StratigrapherR presented at EGU 2020. As the current situation of confinement made the physical EGU 2020 impossible, and brought the virtual EGU, more freedom was given to provide supplementary material.

We therefore present what was supposed to be a presentation available on the web, which is made to support general discussions [what we wanted to call a R'staurant] on R software development concepts for geology purposes.



```
> print("All you can code buffet")  
[1] "All you can code buffet"  
> |
```

**This is the R'staurant supportive power point,
designed to provide a basis in R for geologists:**

- Quick bullet points for fundamental R concepts
- Starting manual for StratigrapherR
- Links to other internet resources
- In development (v. 0.0.1A, for EGU 2020)
- Open to any suggestion or correction

CHAPTERS

- ▶ Introduction.....5
- ▶ General R coding.....11
 - ▶ Basic tutorials.....12
 - ▶ Efficient R coding.....13
- ▶ Avoiding common problems.....17
 - ▶ numeric values.....18
 - ▶ text and characters.....20
- ▶ StratigrapherR.....22

- ▶ **PLEASE HELP ME, my code does not work and I am desperately stuck !!!!.....41**

INTRODUCTION

Getting the necessary software

WHY R ?

- ▶ It is **free**
- ▶ It is awesome
- ▶ It will change your life
- ▶ It can make software evolve by a community effort

HOW TO GET R AND RSTUDIO ?

- ▶ Download **R** on the Comprehensive R Archive Network (CRAN)

<https://cran.r-project.org/> (or type 'download R' in your browser)

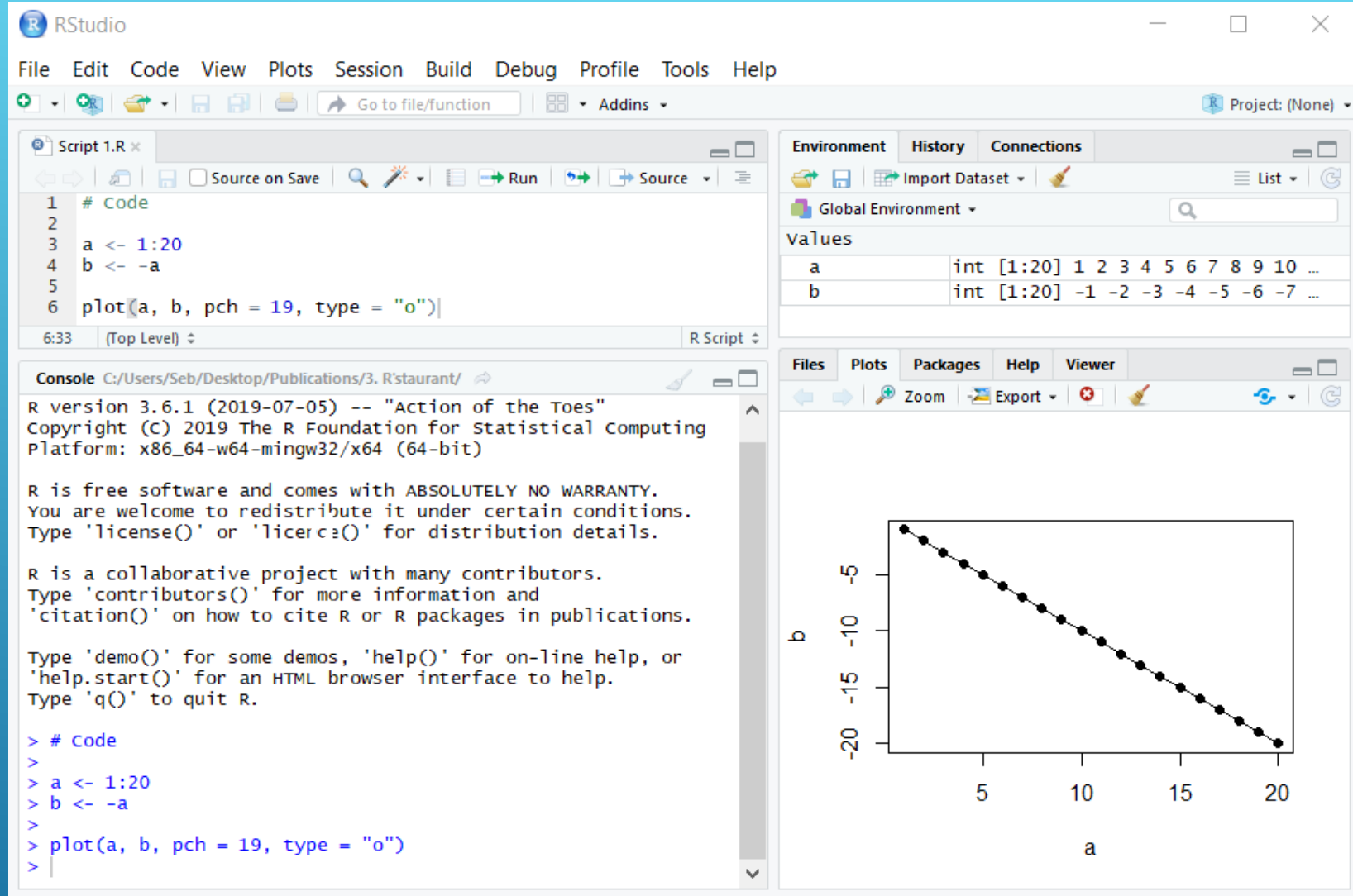
- ▶ Download **RStudio** on the RStudio website

<https://rstudio.com/products/rstudio/#rstudio-desktop>

(or type 'rstudio' in your browser)

WHY RSTUDIO ?

- ▶ RStudio is a **free** scripting interface for R
- ▶ It is widely accepted as the norm for R interfacing
- ▶ It will make your life easier



The screenshot displays the RStudio environment with the following components:

- Script Editor:** Contains the following R code:

```
1 # Code
2
3 a <- 1:20
4 b <- -a
5
6 plot(a, b, pch = 19, type = "o")
```
- Console:** Shows the R startup message and the execution of the script:

```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licenc3()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> # Code
>
> a <- 1:20
> b <- -a
>
> plot(a, b, pch = 19, type = "o")
>
```
- Environment:** Shows the Global Environment with variables:

Variable	Class	Value
a	int [1:20]	1 2 3 4 5 6 7 8 9 10 ...
b	int [1:20]	-1 -2 -3 -4 -5 -6 -7 ...
- Plots:** Displays a scatter plot of b versus a. The x-axis (a) ranges from 0 to 20, and the y-axis (b) ranges from -20 to 0. The data points form a clear downward linear trend.

HOW TO GET NEW PACKAGES ?

The screenshot shows the RStudio interface. The console displays the R version 3.6.1 (2019-07-05) and the R Foundation copyright notice. The 'Install Packages' dialog box is open, showing the 'Repository (CRAN)' dropdown and the 'Packages' field containing 'dplyr'. A red arrow points to the 'dplyr' entry in the list. The 'Packages' pane on the right shows a list of installed packages, including 'boot', 'class', 'cluster', 'codetools', 'compiler', 'datasets', 'foreign', 'graphics', 'grDevices', 'grid', and 'KernSmooth'. A yellow arrow points to the 'Packages' tab in the top menu bar, and a blue arrow points to the 'Install' button in the 'Packages' pane.

R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for statistical computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to distribute copies of R under the terms of the
Type 'license()' on the command line to see the full text of the
license.

R is a collaborative project. You are welcome to contribute to the
Type 'contributors()' on the command line for more information.
Type 'demo()' for some demos. Type 'help.start()' for an
HTML help interface. Type 'q()' to quit R.

> |

Install Packages

Install from: [Configuring Repositories](#)
Repository (CRAN)

Packages (separate multiple with space or comma):
dplyr
dplyr.teradata
dplyrAssist

Documents/R/win-library/3.6 [Default]

Install dependencies

Install Cancel

Environment History Connections

Global Environment

Environment is empty

Files Plots Packages Help Viewer

Install Update

Name	Description	Version
<input type="checkbox"/> boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-22
<input type="checkbox"/> class	Functions for Classification	7.3-15
<input type="checkbox"/> cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.1.0
<input type="checkbox"/> codetools	Code Analysis Tools for R	0.2-16
<input type="checkbox"/> compiler	The R Compiler Package	3.6.1
<input type="checkbox"/> datasets	The R Datasets Package	3.6.1
<input type="checkbox"/> foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...	0.8-71
<input type="checkbox"/> graphics	The R Graphics Package	3.6.1
<input type="checkbox"/> grDevices	The R Graphics Devices and Support for Colours and Fonts	3.6.1
<input type="checkbox"/> grid	The Grid Graphics Package	3.6.1
<input type="checkbox"/> KernSmooth	Functions for Kernel Smoothing Supporting Wand & Jones (1995)	2.23-15

- ▶ Go to **Packages**
- ▶ Click on **Install**
- ▶ Type the **name** of the package you want to install

HOW TO GET NEW PACKAGES ?

- ▶ To get packages you can also run the function `install.packages()`, with the name of the package:

```
install.packages("StratigrapherR")
```

- ▶ To upload the package content and be able to work with it, you need to invoke it every time you open R:

```
library(StratigrapherR)
```

GENERAL R CODING

A few general things to keep in mind

- ▶ Great R basic tutorials exist, for instance:

<https://cyclismo.org/tutorial/R/>

- ▶ You can usually find good tutorials by googling them, for instance I got the following by googling 'R basic plots':

<https://sites.harding.edu/fmccown/r/>

BASIC TUTORIALS

- ▶ R is an interpreted language.

What does that mean ?

- ▶ The code is not compiled: it runs directly

How is that done ?

- ▶ The base R functions are pre-compiled, and any script ultimately redirects towards these pre-compiled functions

Why should I care ?

- ▶ This affects the speed of computations

EFFICIENT R CODING

- ▶ How do I code R scripts to make computations fast ?

The general rule is to avoid loops: 'for' loops and 'repeat' loops. Such loops are efficient in compiled languages such as C or Fortran, however they take a lot of time in R.

- ▶ How do I avoid loops ?

Find basic functions in R that do what you want

- ▶ How can I find such functions ?

Google can offer a good start, Stack overflow is also a good way to find answers to questions that were already asked by other users;

<https://stackoverflow.com/questions/tagged/r>

EFFICIENT R CODING

- ▶ Could you show me an example of a function that allows to avoid loops ?

Sure thing, here is how to compute the cumulative sum of the sequence of 1 to 10:

```
t <- c(1)
for(i in 2:10) t <- c(t, i + t[i-1])
t # this is obtained by the 'for' loop
#> [1] 1 3 6 10 15 21 28 36 45 55

cumsum(1:10) # this is obtained using the appropriate function
#> [1] 1 3 6 10 15 21 28 36 45 55
```

EFFICIENT R CODING

► What if I cannot avoid a loop ?

Try to confine the looped part of the code to the strict minimum. You can further isolate the loops using the `apply()` family of functions, that loops functions more efficiently. For a tutorial on these functions:

<https://www.guru99.com/r-apply-sapply-tapply.html>

EFFICIENT R CODING

AVOIDING COMMON PROBLEMS

A few things I wish I knew before getting into R, and that allow you to avoid common problems that beginners will eventually run into, and that drive you **INSANE** trying to understand what is happening !

Numeric values (numbers having decimals) in R can pose a problem, as they are stored in the computer as fractions. This is called **floating-point arithmetic**. Roughly put, this means that the number has an imprecision of a scale of 10^{-15} of the unit (e.g. for a value of 1.00×10^{25} , the imprecision is of a scale of 10^{10}). This is generally a minor problem for general calculations. However this can be challenging when trying to identify specific values, e.g. 0.1 will be understood in the computer in another way, which should look like this: 0.100000000000000561. One way to avoid this problem (see the code in the next slide) is to round the values to an acceptable level, and to compare them using the `all.equal()` function, which checks for equality within numerical tolerance (at the opposite of the `==` comparison relational operator and of the `identical()` function). This offers a double security to insure that the possible accumulation of computational error will not affect the identification of equal values (see the code in the next slide).

AVOIDING COMMON PROBLEMS: NUMERIC VALUES

```
a <- 1 + 1e-15 # this represents a value supposed to be 1,  
              # but deviating from it due to computational error  
  
a == 1        # failure  
#> [1] FALSE  
identical(a, 1) # failure  
#> [1] FALSE  
all.equal(a, 1) # acceptable  
#> [1] TRUE  
  
round(a, 3) == 1 # acceptable  
#> [1] TRUE  
identical(round(a, 3), 1) # acceptable  
#> [1] TRUE  
all.equal(round(a, 3), 1) # SUGGESTED OPTION  
#> [1] TRUE
```

AVOIDING COMMON PROBLEMS: NUMERIC VALUES

- ▶ How do I read text files ?

Use the `readLines()` function for pure text, `read.table()`, `read.csv()`, `read.fwf()` to open respectively; tables where columns are separated by specific characters, csv files, and fixed-width columns tables.



The `stringsAsFactors` parameter:

This parameter is present in most functions reading tables and creating data frames (i.e. the format for tables in R). By default it is generally set to `TRUE`. This means that each string (i.e. each element made up of text characters) will be set in another format, as a **factor**. This can be problematic: factors are not reacting similarly than characters in most functions. To avoid any problem it is here advised to set `stringsAsFactors` as `FALSE` when creating any data frame to avoid such problems.

AVOIDING COMMON PROBLEMS: TEXT AND CHARACTERS

- ▶ How do I identify specific characters sequences, for instance in a text file where the information is under the form 'data.1: 147'

Use pattern match and replacement, using for instance the `grep()`, `grepl()` and `sub()` functions, or the `stringr` package. Character pattern matching use characters having a specific meaning (such as `^`, `[`, `]`, `-`, `+` and `$` in the example below). A sequence of characters used to identify a text pattern is called a regular expression. For more information on regular expression in R follow this link:

<http://biostat.mc.vanderbilt.edu/wiki/pub/Main/SvetlanaEdenRFiles/regExprTalk.pdf>

```
a <- c("data.1: 147", "data.2: 983")

a
#> [1] "data.1: 147" "data.2: 983"

a[grepl("^data.1: [0-9]+$", a)]
#> [1] "data.1: 147"

as.numeric(sub("^data.1: ", "", a[grepl("^data.1: [0-9]+$", a)]))
#> [1] 147
```

**AVOIDING COMMON PROBLEMS:
TEXT AND CHARACTERS**

StratigraphheR

A package to make lithologs

WHY StratigrapherR ?

- ▶ It allows to **automate** repetitive drawings of lithologs
- ▶ It generates a basis for lithologs that can be **improved** in vector drawing software
- ▶ It allows to **integrate** lithologs to other data processing in R
- ▶ It can be made to **evolve** by anyone willing to learn R

- ▶ Working with stratigraphic data: the 'lim' object
- ▶ The lim object stores information of intervals: lower and upper boundaries ('l' and 'r', for left and right boundary), a rule for the inclusion of the boundaries ('b'), and an id ('id')

```
interval <- as.lim(l = c(0,1,2), r = c(0.5,2,2.5), id = c("Int. 1", "Int.2", "Int.3"))
```

```
interval
```

```
#> $l
```

```
#> [1] 0 1 2
```

```
#>
```

```
#> $r
```

```
#> [1] 0.5 2.0 2.5
```

```
#>
```

```
#> $id
```

```
#> [1] "Int. 1" "Int.2" "Int.3"
```

```
#>
```

```
#> $b
```

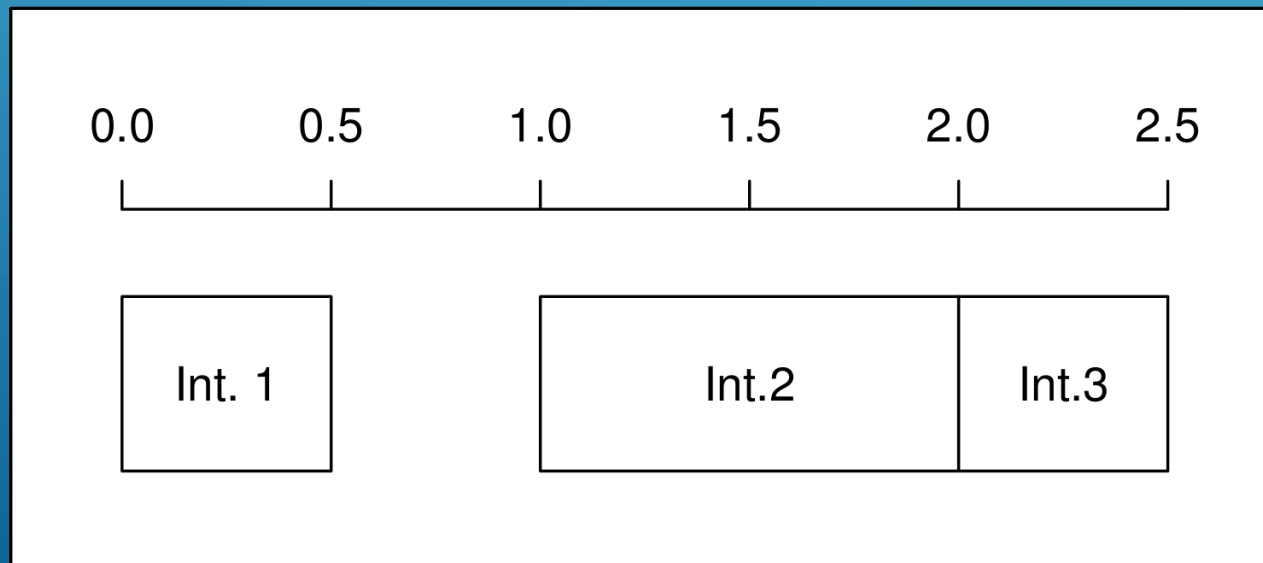
```
#> [1] "[]" "[]" "[]"
```


- ▶ The lim objects can be used for visualising intervals

```
interval <- as.lim(l = c(0,1,2), r = c(0.5,2,2.5), id = c("Int. 1","Int.2","Int.3"))

plot.new()
plot.window(ylim = c(-0.5, 2.5), xlim = c(0, 2.5))
axis(3, pos = 1.5, las = 1)

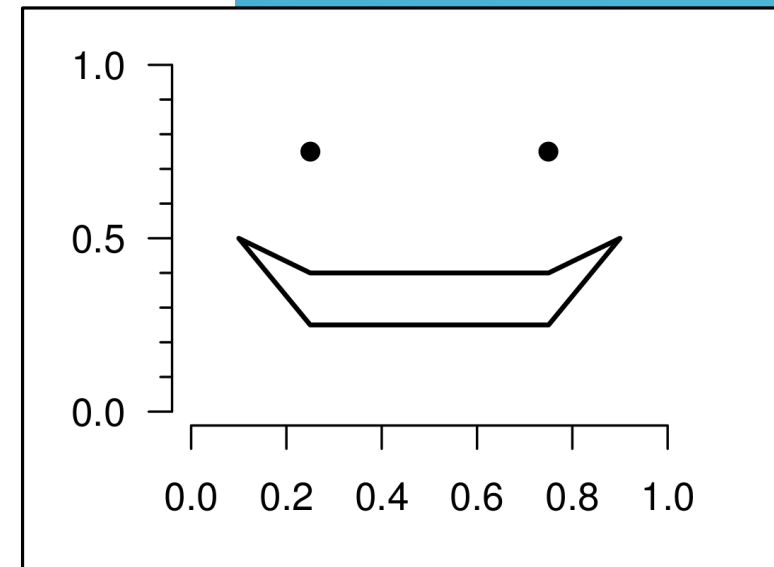
infoabar(ymin = 0, ymax = 1, xmin = interval$l, xmax = interval$r,
         labels = c(interval$id), srt = 0)
```



- ▶ The formalisation of intervals with the `lim` objects allows to define several functions:
 - `are.lim.nonunique()` checks whether intervals are duplicated
 - `are.lim.nonadjacent()` checks if the intervals share adjacent boundaries
 - `are.lim.distinct()` checks whether the intervals are overlapping
 - `simp.lim()` merges overlapping intervals having identical id
 - `flip.lim()` finds the complementary intervals of a set of intervals (i.e. the gaps)
 - `mid.lim()` defines intervals in between data points
 - `in.lim()` finds which values belong to which intervals

- ▶ The `pdfDisplay()` function can be used to generate and open plots of any size
- ▶ The `plot.new()` and `plot.window()` functions are used to introduce an empty plot
- ▶ The `minorAxis()` function allows to have an axis with minor ticks

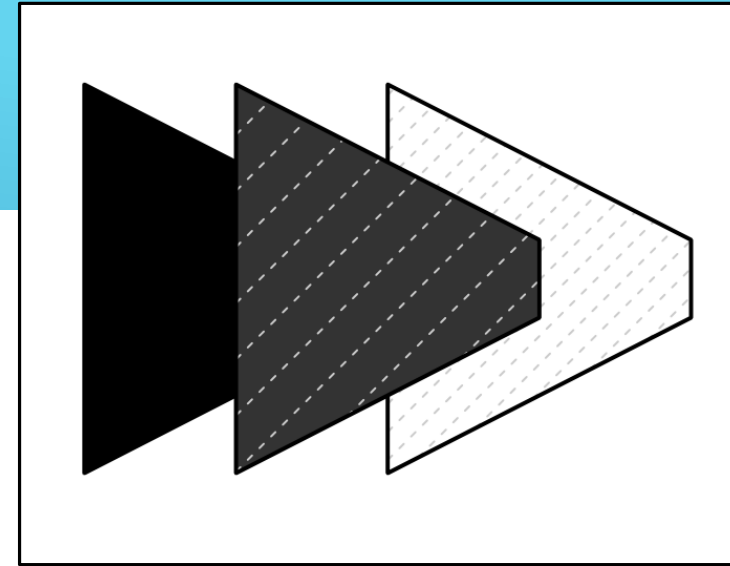
```
graphical_function <- function()
{
  opar <- par()$mar # Save initial graphical parameters
  par(mar = c(2,3,0,1))
  plot.new()
  plot.window(xlim = c(0,1), ylim = c(0,1))
  axis(1)
  minorAxis(2, at.maj = seq(0, 1, 0.5), n = 5, las = 1)
  points(c(0.25, 0.75), c(0.75, 0.75), pch = 19)
  polygon(c(0.1, 0.25, 0.75, 0.9, 0.75, 0.25),
          c(0.5, 0.25, 0.25, 0.5, 0.4, 0.4), lwd = 2)
  par(mar = opar) # Restore initial graphical parameters
}
pdfDisplay(graphical_function(), "graphical_function", width = 3,
           height = 2)
```



- ▶ The `multigons()` function plots several polygons at once

```
i <- c(rep("A1", 6), rep("A2", 6), rep("A3", 6)) # Polygon ids
x <- c(1, 2, 3, 3, 2, 1, 2, 3, 4, 4, 3, 2, 3, 4, 5, 5, 4, 3) # x coordinates
y <- c(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6) # y coordinates

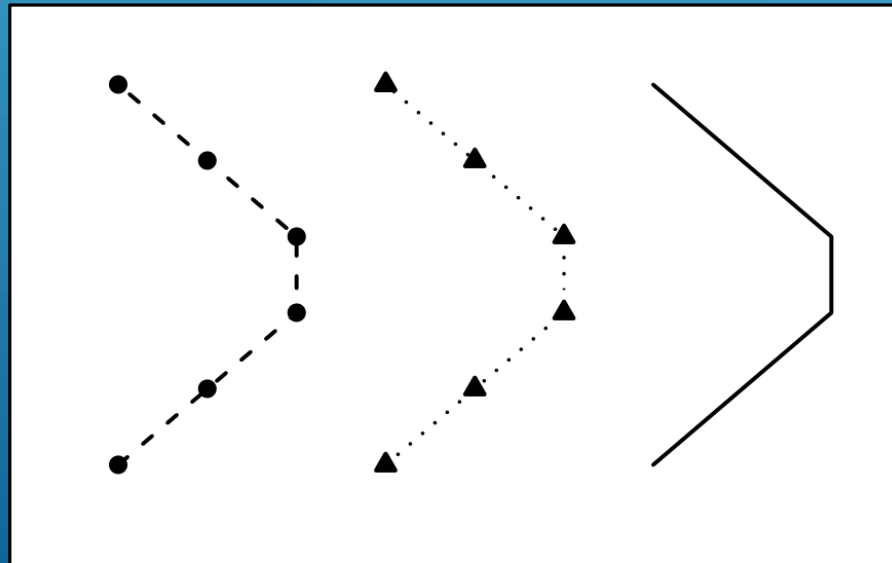
plot.new()
plot.window(xlim = c(0, 6), ylim = c(0, 7))
multigons(i, x, y,
  front = "A2", # This gets the polygon A2 in front of all others
  density = c(NA, 5, 10), # Different shading density
  scol = "grey80", # Same shading color
  col = c("black", "grey20", "white"), # Different background color
  lwd = 2, # Width of border lines for all polygons
  slty = 2, # Shading lines type, same for all polygons
  slwd = 1) # Shading lines width, same for all polygons
```



- ▶ The `multilines()` function plots several polylines at once

```
i <- c(rep("A1", 6), rep("A2", 6), rep("A3", 6))
x <- c(1, 2, 3, 3, 2, 1, 4, 5, 6, 6, 5, 4, 7, 8, 9, 9, 8, 7)
y <- c(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)

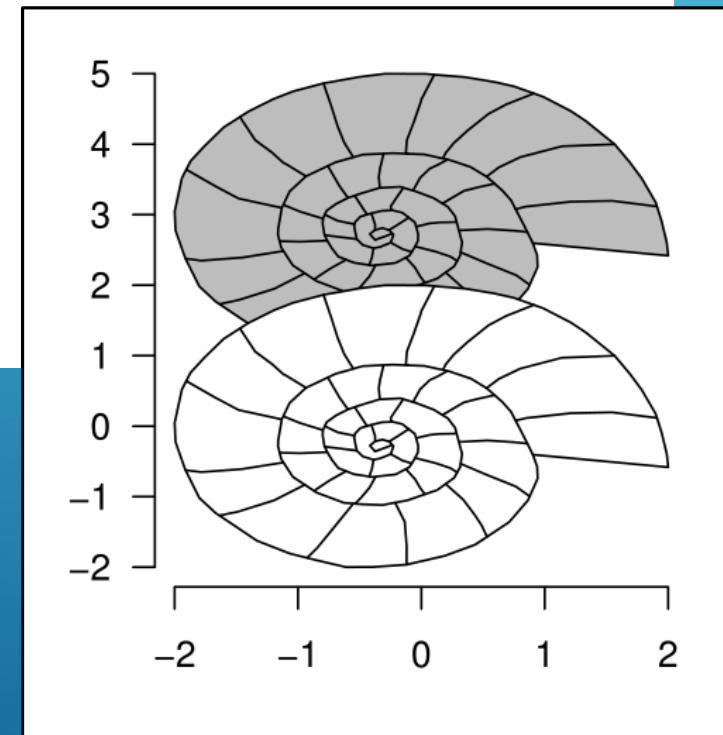
plot.new()
plot.window(xlim = c(0, 10), ylim = c(0, 7))
multilines(i, x, y, j = c("A3", "A1", "A2"), lty = c(1, 2, 3), lwd = 2,
           type = c("l", "o", "o"), pch = c(NA, 21, 24), cex = 1, bg = "black")
```



- ▶ Pre-drawn SVG objects can be imported using the `pointsvg()` function
- ▶ These SVG objects can be directly drawn using `centresvg()` or `framesvg()`

```
svg.file.directory <- tempfile(fileext = ".svg")           # Creates temporary file
writeLines(example.ammonite.svg, svg.file.directory)      # Writes svg in the file
ammonite.drawing <- pointsvg(file = svg.file.directory)  # Provides file
```

```
plot.new()
plot.window(xlim = c(-2, 2), ylim = c(-2, 5))
axis(1)
axis(2, las = 2)
centresvg(ammonite.drawing, 0, c(3,0), xfac = 2, yfac = 2,
          col = c("grey", "white"))
```

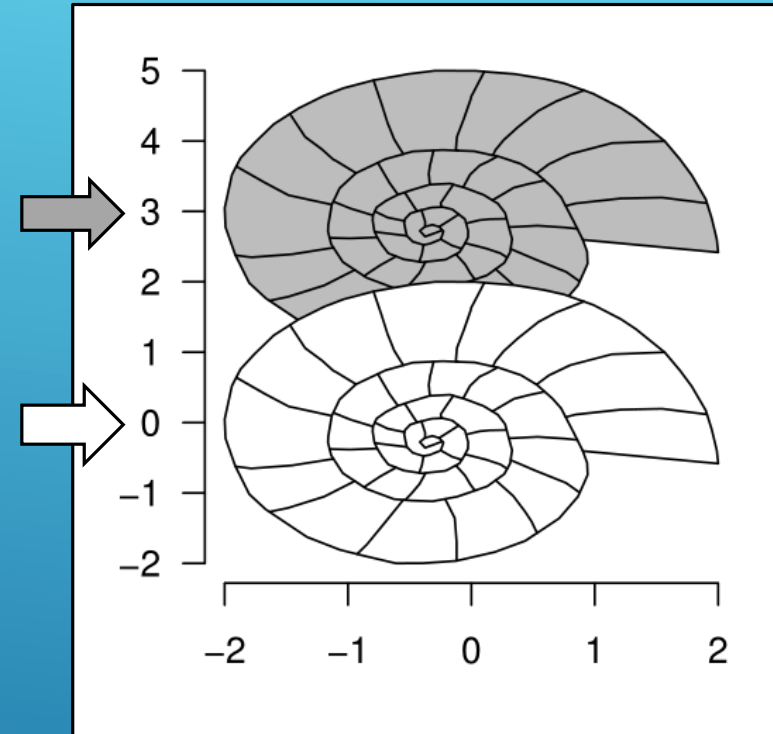


- ▶ Using `centresvg()` or `framesvg()` allows to plot the SVG object several times with one call of the function

```
centresvg(ammonite.drawing, 0, c(3,0), xfac = 2, yfac = 2,  
col = c("grey", "white"))
```



This code won't run correctly if you didn't run the code in preceding slides



- ▶ The `changesvg()` function can change the SVG object:
 - ▶ Change of the order of plotting of the polylines and polygons
 - ▶ Removal of some of the polylines or polygons
 - ▶ Inversion of the figure in x and/or y

- ▶ The information for the beds in a litholog can be provided as a table (which in R corresponds to a data frame object):

id	l	r	h	colour	litho
B1	0	1	3	grey	S
B2	1	3	4	grey	L
B3	3	4	5	black	C
B4	4	9	4	white	L
B5	9	11	4	white	L

information for each bed: id identifies each bed, l and r provide the lower and upper boundaries, h the hardness, and the colour is provided along with a code for lithology (S for shale, L for Limestone, C for chert).

- ▶ Such a table is provided in StratigrapherR as an example:

```
View (bed.example)
```


- ▶ The data frame describing the beds can be merged with a legend data frame to attribute a symbology to the beds, based on a common column (in our case, litho, standing for lithology)

```
legend <- data.frame(litho = c("S", "L", "C"),  
                    col = c("grey30", "grey90", "white"),  
                    density = c(30, 0, 10),  
                    angle = c(180, 0, 45), stringsAsFactors = FALSE)
```

View(legend)

	litho	col	density	angle
1	S	grey30	30	180
2	L	grey90	0	0
3	C	white	10	45

- ▶ The merging is done here by the `left_join()` function of the `dplyr` package

```
library(dplyr)  
bed.legend <- left_join(bed.example,  
                      legend,  
                      by = "litho")
```

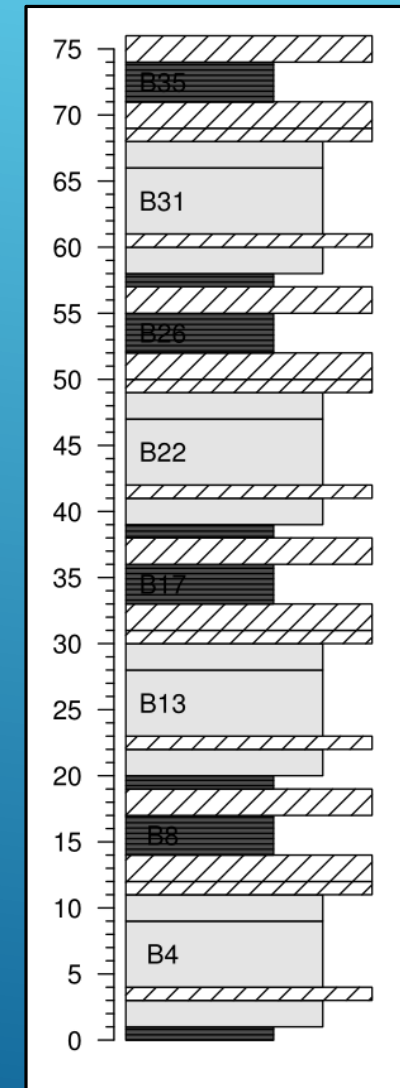
View(bed.legend)

	id	l	r	h	colour	litho	col	density	angle
1	B1	0	1	3	grey	S	grey30	30	180
2	B2	1	3	4	grey	L	grey90	0	0
3	B3	3	4	5	black	C	white	10	45
4	B4	4	9	4	white	L	grey90	0	0
5	B5	9	11	4	white	L	grey90	0	0

Showing 1 to 5 of 36 entries

- ▶ Based on the information of the provided data frames, polygons coordinates can be defined using the `litholog()` function, and its output can be plotted with the correct symbology using `multigons()`. Text can be added in the beds using `bedtext()`

```
basic.log <- litholog(l = bed.example$l, r = bed.example$r,  
                    h = bed.example$h, i = bed.example$id)  
  
plot.new()  
plot.window(xlim = c(0,6), ylim = c(-1,77))  
minorAxis(2, at.maj = seq(0, 75, 5), n = 5)  
  
multigons(basic.log$i, x = basic.log$xy, y = basic.log$dt,  
          col = bed.legend$col,  
          density = bed.legend$density,  
          angle = bed.legend$angle)  
  
bedtext(labels = bed.example$id,  
        l = bed.example$l,  
        r = bed.example$r,  
        x = 0.5, # x position where to centre the text  
        ymin = 3) # ymin defines the minimum thickness  
                 # for the beds where text can be added
```



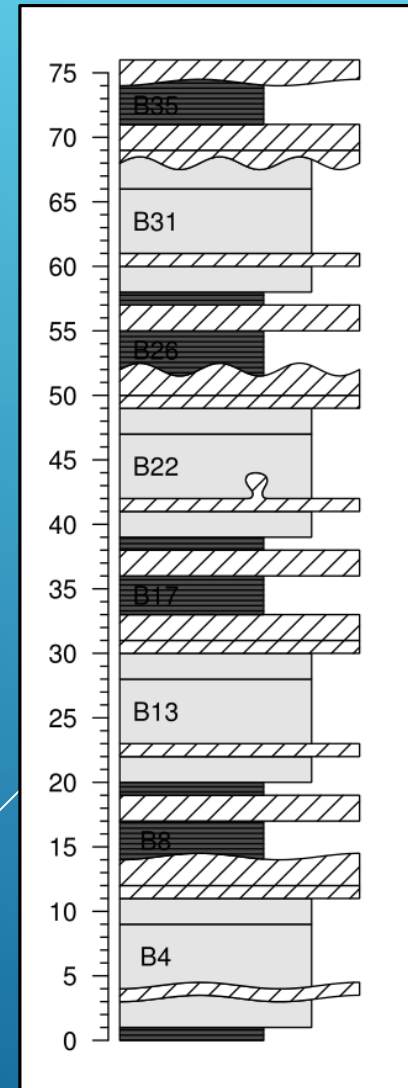
This code won't run correctly if you didn't run the code in preceding slides

- ▶ A litholog created via the `litholog()` function can be modified. To add thickness variations to beds, the `weldlog()` function can be used:

```
s1 <- sinpoint(5,0,0.5,nwave = 1.5)
s2 <- sinpoint(5,0,1,nwave = 3, phase = 0)
s3 <- framesvg(example.liquefaction, 1, 4, 0, 2, plot = FALSE, output = TRUE)

final.log <- weldlog(log = basic.log, dt = boundary.example$dt,
                    seg = list(s1 = s1, s2 = s2, s3 = s3),
                    j = c("s1", "s1", "s1", "s3", "s2", "s2", "s1"), warn = FALSE)

plot.new()
plot.window(xlim = c(0,6), ylim = c(-1,77))
minorAxis(2, at.maj = seq(0, 75, 5), n = 5, las = 1)
multigons(final.log$i, x = final.log$xy, y = final.log$dt,
          col = bed.legend$col,
          density = bed.legend$density,
          angle = bed.legend$angle)
bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
        x = 0.75, ymin = 3)
```



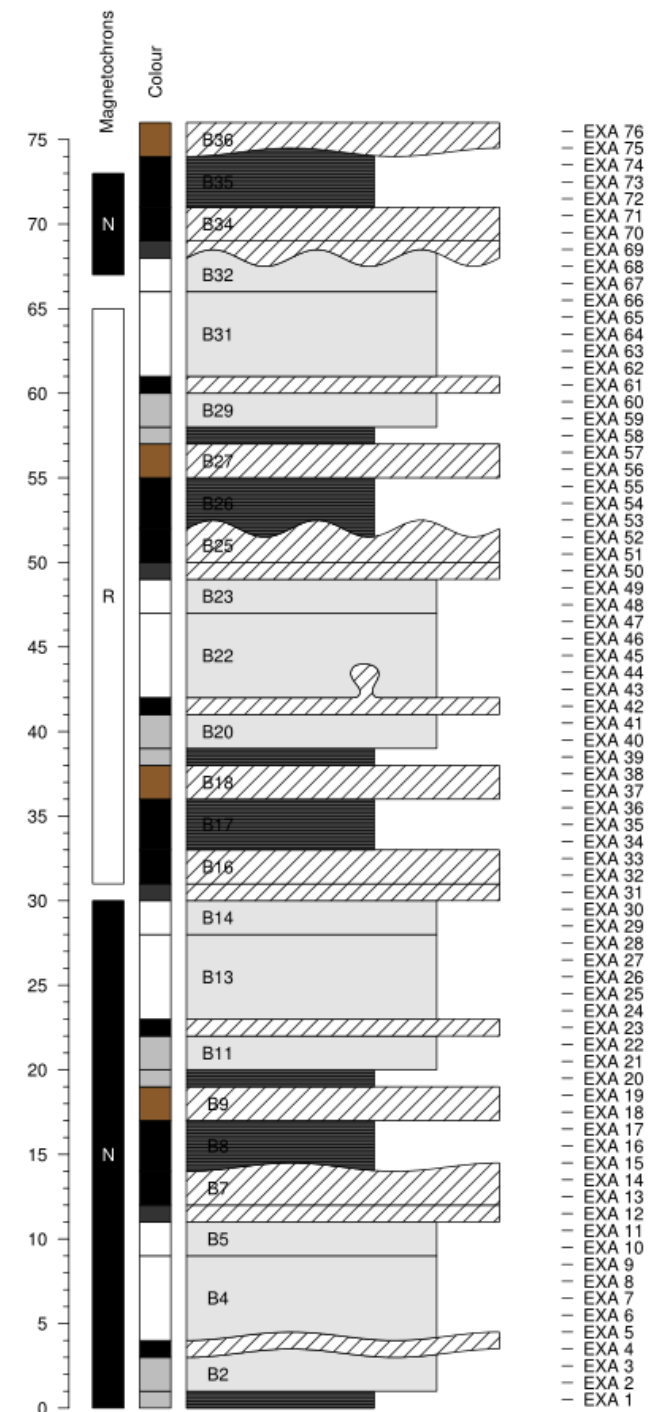
This code won't run correctly if you didn't run the code in preceding slides

Additional information can be provided using `infoBar()` for stratigraphical or lithological information, `text()` for text, and `axis()` e.g. for samples positions

```
# Code repeated from earlier examples ----
basic.log <- litholog(l = bed.example$l, r = bed.example$r,
                    h = bed.example$h, i = bed.example$id)
legend <- data.frame(litho = c("S", "L", "C"),
                    col = c("grey30", "grey90", "white"),
                    density = c(30, 0, 10),
                    angle = c(180, 0, 45), stringsAsFactors = FALSE)
bed.legend <- dplyr::left_join(bed.example, legend, by = "litho")
s1 <- sinpoint(5, 0, 0.5, nwave = 1.5)
s2 <- sinpoint(5, 0, 1, nwave = 3, phase = 0)
s3 <- framesvg(example.liquefaction, 1, 4, 0, 2, plot = FALSE, output = TRUE)
final.log <- weldlog(log = basic.log, dt = boundary.example$dt,
                    seg = list(s1 = s1, s2 = s2, s3 = s3),
                    j = c("s1", "s1", "s1", "s3", "s2", "s2", "s1"), warn = F)

# ----
plot.new()
plot.window(xlim = c(-1.5, 8), ylim = c(-1, 81))
minorAxis(2, at.maj = seq(0, 75, 5), n = 5, las = 1)
multigons(final.log$i, x = final.log$xy, y = final.log$dt,
          col = bed.legend$col,
          density = bed.legend$density,
          angle = bed.legend$angle)
bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
        x = 0.5, ymin = 2)

legend.chron <- data.frame(polarity = c("N", "R"),
                          bg.col = c("black", "white"),
                          text.col = c("white", "black"),
                          stringsAsFactors = FALSE)
chron.legend <- dplyr::left_join(chron.example, legend.chron, by = "polarity")
infoBar(-1.5, -1, chron.legend$l, chron.legend$r,
        labels = chron.legend$polarity,
        m = list(col = chron.legend$bg.col),
        t = list(col = chron.legend$text.col),
        srt = 0)
colour <- bed.example$colour
colour[colour == "darkgrey"] <- "grey20"
colour[colour == "brown"] <- "tan4"
infoBar(-0.25, -0.75, bed.example$l, bed.example$r,
        m = list(col = colour))
text(-0.5, 79, "Colour", srt = 90)
text(-1.25, 79, "Magnetochrons", srt = 90)
axis(4, at = proxy.example$dt, labels = proxy.example$name,
      pos = 6, lwd = 0, lwd.ticks = 1, las = 1)
```



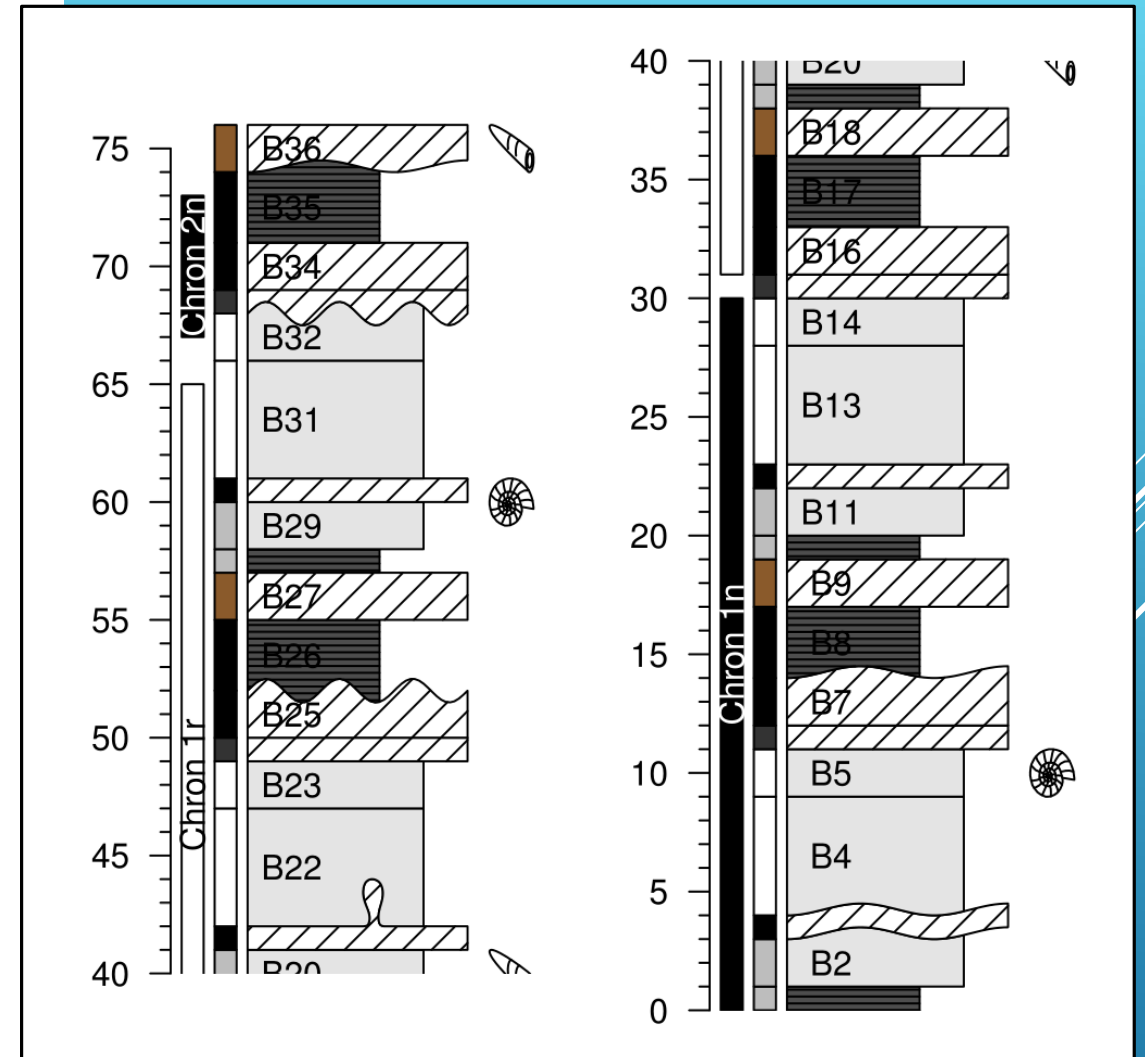
The litholog plotting code can be set into a function, and used to plot only a part of the log, to be able to latter plot it onto several pages (using for instance LaTeX)

```
log.function <- function(xlim = c(-2.5,7), ylim = c(-1,77))
{
  plot.new()
  plot.window(xlim = xlim, ylim = ylim)
  minorAxis(2, at.maj = seq(0, 75, 5), n = 5, pos = -1.75, las = 1)
  multigons(final.log$i, x = final.log$xy, y = final.log$dt,
  col = bed.legend$col,
  density = bed.legend$density,
  angle = bed.legend$angle)
  bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
  x = 1, edge = TRUE, ymin = 2)
  centresvg(example.ammonite, 6,
  fossil.example$dt[fossil.example$type == "ammonite"],
  xfac = 0.5)
  centresvg(example.belemnite, 6,
  fossil.example$dt[fossil.example$type == "belemnite"],
  xfac = 0.5)
  infobar(-1.5, -1, chron.legend$l, chron.legend$r,
  labels = chron.legend$id,
  m = list(col = chron.legend$bg.col),
  t = list(col = chron.legend$text.col))
  infobar(-0.25, -0.75, bed.example$l, bed.example$r,
  m = list(col = colour))
}

gr <- function()
{
  opar <- par() # Save initial graphical parameters
  par(mar = c(1,2,1,2), yaxs = "i")
  for(i in 1:0)
  {
    ylim <- c(0,40)
    log.function(ylim = ylim + 40*i)
  }
  par(mar = opar$mar, myaxs = opar$yaxs) # Restore initial graphical parameters
}

```

pdfDisplay(gr(), name = "divided log", width = 3, height = 5)

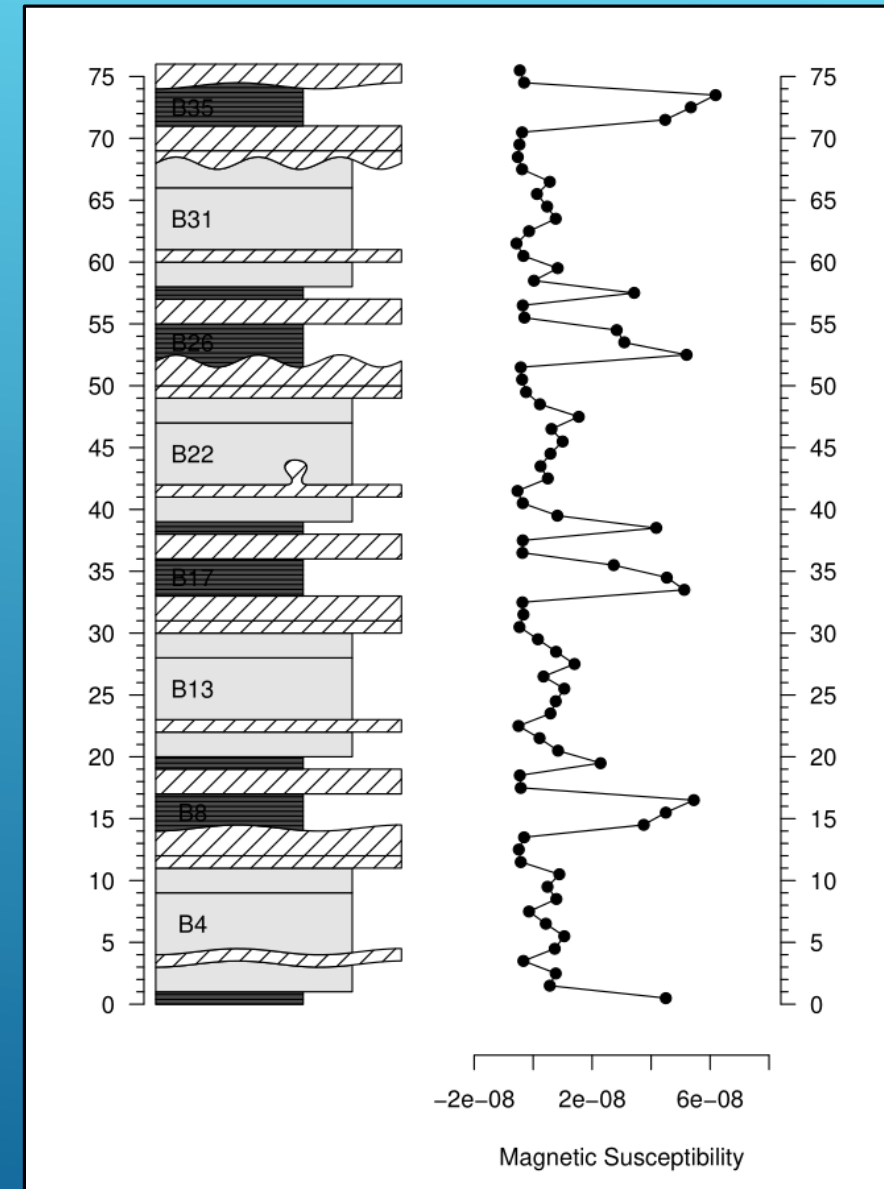


This code won't run correctly if you didn't run the code in preceding slides

Any other plot(s) can be added along the litholog, using for instance the `par()` function. Great care should be taken to insure that the depth axis is similar

```
# Code repeated from earlier examples ----
basic.log <- litholog(l = bed.example$l, r = bed.example$r,
                    h = bed.example$h, i = bed.example$id)
legend <- data.frame(litho = c("S", "L", "C"),
                    col = c("grey30", "grey90", "white"),
                    density = c(30, 0, 10),
                    angle = c(180, 0, 45), stringsAsFactors = FALSE)
bed.legend <- dplyr::left_join(bed.example, legend, by = "litho")
s1 <- sinpoint(5, 0, 0.5, nwave = 1.5)
s2 <- sinpoint(5, 0, 1, nwave = 3, phase = 0)
s3 <- framesvg(example.liquefaction, 1, 4, 0, 2, plot = FALSE, output = TRUE)
final.log <- weldlog(log = basic.log, dt = boundary.example$dt,
                    seg = list(s1 = s1, s2 = s2, s3 = s3),
                    j = c("s1", "s1", "s1", "s3", "s2", "s2", "s1"), warn = F)

# ----
opar <- par()$mfrow # Save initial graphical parameters
par(mfrow = c(1, 2))
plot.new()
plot.window(xlim = c(0, 6), ylim = c(-1, 77))
minorAxis(2, at.maj = seq(0, 75, 5), n = 5, las = 1)
multigons(final.log$i, x = final.log$xy, y = final.log$dt,
          col = bed.legend$col,
          density = bed.legend$density,
          angle = bed.legend$angle)
bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
        x = 0.75, ymin = 3)
plot.new()
plot.window(xlim = c(-2*10^-8, 8*10^-8), ylim = c(-1, 77))
minorAxis(4, at.maj = seq(0, 75, 5), n = 5, las = 1)
lines(proxy.example$ms, proxy.example$dt, type = "o", pch = 19)
axis(1)
title(xlab = "Magnetic Susceptibility")
par(mfrow = opar) # Restore initial graphical parameters
```



A legend can be created using `nlegend()`. Each symbol is in another plot, and is centred on the coordinates (0,0).

```
# Code repeated from earlier examples ----
legend <- data.frame(litho = c("S", "L", "C"),
  col = c("grey30", "grey90", "white"),
  density = c(30, 0, 10),
  angle = c(180, 0, 45), stringsAsFactors = FALSE)

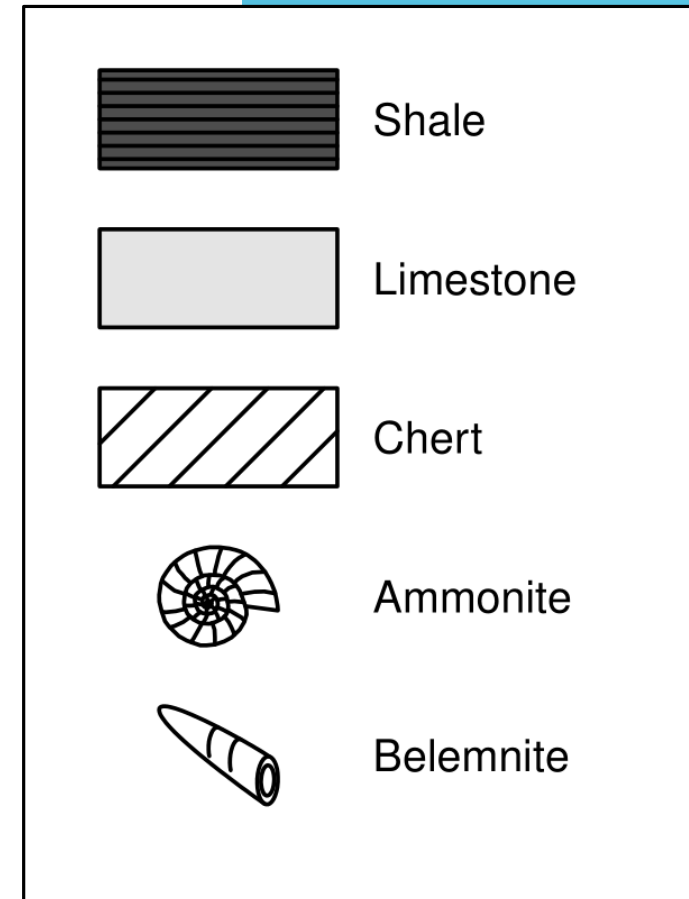
# ----
f <- function(i)
{
  multigons(i = rep(1, 4), c(-1,-1,1,1), c(-1,1,1,-1),
    col = legend$col[i],
    density = legend$density[i],
    angle = legend$angle[i])
}

opar <- par() # Save initial graphical parameters
par(mar = c(0,0,0,0), mfrow = c(5,1))

nlegend(t = "Shale")
f(1)
nlegend(t = "Limestone")
f(2)
nlegend(t = "Chert")
f(3)

nlegend(t = "Ammonite")
centresvg(example.ammonite,0,0,xfac = 0.5)
nlegend(t = "Belemnite")
centresvg(example.belemnite,0,0,xfac = 0.5)

par(mar = opar$mar, mfrow = opar$mfrow) # Restore initial graphical parameters
```



**PLEASE
HELP ME,
MY CODE DOES NOT WORK
AND I AM DESPERATELY STUCK !!!!**

Self-explanatory

PLEASE HELP ME, MY CODE DOES NOT WORK AND I AM DESPERATELY STUCK !!!!

- ▶ Starting to code in any language is difficult, and everyone rising to the challenge will sooner or later be faced with what seems to be an insurmountable task. On a personal level I can say without a doubt that I wouldn't have been able to code what I wanted to code without the help of one of my relatives, who is a trained programmer that I can annoy with my R problems every now and then. Seeking help is actually a full part of the job of programmer. Thankfully there are a few resources available out there:
 - ▶ Google (or any other search engine, it is basic but efficient)
 - ▶ Stack Overflow <https://stackoverflow.com/questions/tagged/r>
 - ▶ The maintainer of the package you are using
- ▶ To ask questions, is it best to provide a **reprex**, which stands for “**representative example**”: <https://community.rstudio.com/t/faq-whats-a-reproducible-example-reprex-and-how-do-i-do-one/5219>
- ▶ If you have any question on StratigrapherR, please contact the maintainer:
sebastien.wouters@doct.uliege.be
- ▶ You can post your question on Stack Overflow, and link the question to the maintainer, so that other users can take advantage of the answer