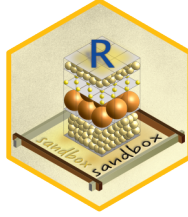


# Supplement Dietze et al.: sandbox - the grainy modelling tool



Michael Dietze<sup>1</sup>, Sebastian Kreutzer<sup>2,3</sup>, Margret C. Fuchs<sup>4</sup>, Sascha Meszner<sup>5</sup>

<sup>1</sup>*GFZ German Research Centre for Geosciences, Section 5.1 Geomorphology, Potsdam, DE*

<sup>2</sup>*Geography & Earth Sciences, Aberystwyth University, Aberystwyth, Wales, GB*

<sup>3</sup>*Archéosciences Bordeaux, UMR 6034 CNRS - Université Bordeaux-Montaigne, Pessac, FR*

<sup>4</sup>*Helmholtz-Zentrum Dresden-Rossendorf,*

*Helmholtz-Institute Freiberg for Resource Technology, Freiberg, DE*

<sup>5</sup>*JENA-GEOS-Ingenieurbüro GmbH, Jena, DE*

2022-02-10

## Contents

<b>1</b>	<b>Preparatory work</b>	<b>2</b>
1.1	Loading the package and preparing additional information . . . . .	2
1.2	Defining grain-size distributions for populations . . . . .	2
1.3	Loading the empirical data set . . . . .	6
<b>2</b>	<b>Getting started with sandbox</b>	<b>7</b>
2.1	Creating a rule book from scratch . . . . .	7
2.2	Filling the rule book with meaningful rules . . . . .	8
<b>3</b>	<b>Sampling a sediment section</b>	<b>12</b>
3.1	Adding further parameters/rules to pursue specific research questions . . . . .	13
<b>4</b>	<b>Further sample treatment routines</b>	<b>15</b>
<b>5</b>	<b>Adding physical deposition laws</b>	<b>17</b>
5.1	Depth-dependent packing density . . . . .	17
5.2	Mixing of two mineral populations with different densities . . . . .	19
<b>6</b>	<b>Optimising the depositional process – adding non-linear age-depth rules</b>	<b>22</b>
6.1	Issue 01b – Implementing pulse-pause effects for age-depth relations . . . . .	23
<b>7</b>	<b>Extension of the framework – the luminescence scope</b>	<b>25</b>
<b>8</b>	<b>Repository of the code used to generate the article figures</b>	<b>30</b>
8.1	Summary of the Gleina section parameterisation . . . . .	30
8.2	Test effect of sample container geometry on age uncertainty . . . . .	31
8.3	Test effect of sample container size and deposition rate on age uncertainty . . . . .	33
8.4	Test effect of size-dependent age inheritance . . . . .	35
	<b>References</b>	<b>42</b>

# 1 Preparatory work

## 1.1 Loading the package and preparing additional information

To make the functions of 'sandbox' [version: 0.2.1] available, the R package needs to be loaded once. The same holds for the functionality of the package 'EMMAgeo' (version: 0.9.7, Dietze & Dietze, 2019), which is only used here for auxiliary reasons, to create populations based on a real-world data set and use these in 'sandbox'.

```
library("sandbox")
library("EMMAgeo")
```

## 1.2 Defining grain-size distributions for populations

Before we explore 'sandbox', we need to prepare some information about the sediment section. Here we opted for the loess section from an outcrop near Gleina, Germany (Meszner et al., 2011). We might as well construct a fully synthetic sediment section, but prefer to go this way to stick closer to typical expectations of potential users of 'sandbox'.

We describe the grain-size composition of the data set as a linear combination of population of specific size, or end-members (EMs): EM1 has a mean grain size of 6.38 phi (12  $\mu\text{m}$ ) and a log-normal standard deviation of 0.9, EM2 a mean of 4.68 phi (39  $\mu\text{m}$ ) and a standard deviation of 0.5, and EM3 a mean of 4.29 phi (51  $\mu\text{m}$ ) and a standard deviation of 0.5. These three size population definitions result from end-member modelling analysis of the empirical grain-size distributions of samples from the Gleina section. We work with the phi-scale because it linearises the typically log-normal grain-size distributions of natural sediments. More information about the phi-scale can be found in Krumbein (1937).

To start, we load the empirical data by Meszner et al. (2011), model them and show the approximation of end-members by the log-normal distribution functions. Note that the defined distributions are scaled to 70 % to account for secondary modes of the EMMA output, which draw vol.-% values from the primary modes (Dietze & Dietze, 2019).

```
## import complete grain-size into R
X <- read.table(
  file = "gleina_grainsize_raw.txt",
  sep = "\t",
  header = TRUE)

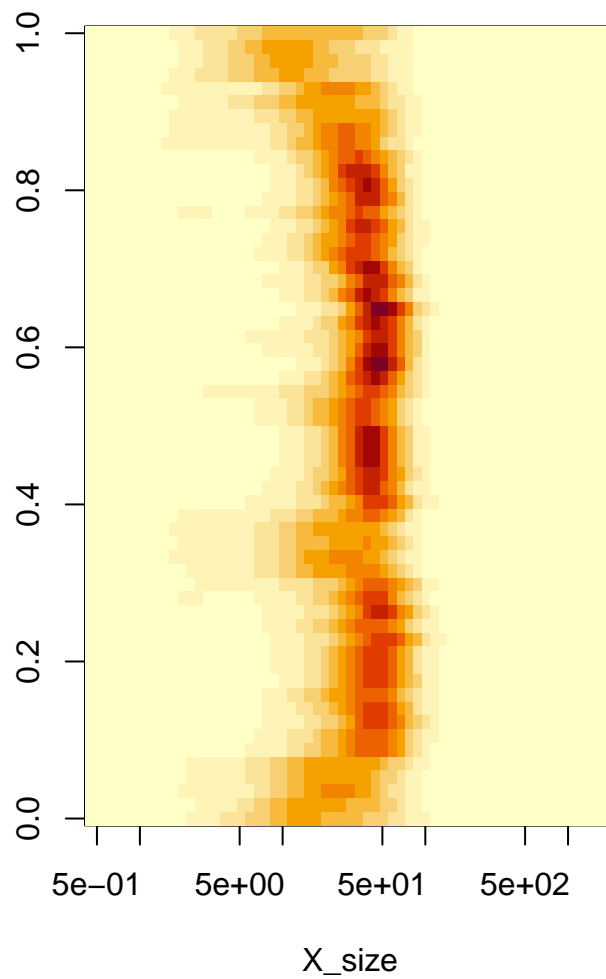
## columns with actual grain size data
## those columns start with GSD
X_gsd <- X[,grepl("GSD", colnames(X))]

## convert column header names into
## numerical values
X_size <- as.numeric(
  regmatches(colnames(X_gsd),
    regexpr("^GSD_", colnames(X_gsd))))

## remove classes with zero % throughout
i_ok <- which(colSums(X_gsd) > 0)
X_gsd <- X_gsd[,i_ok]
X_size <- X_size[i_ok]

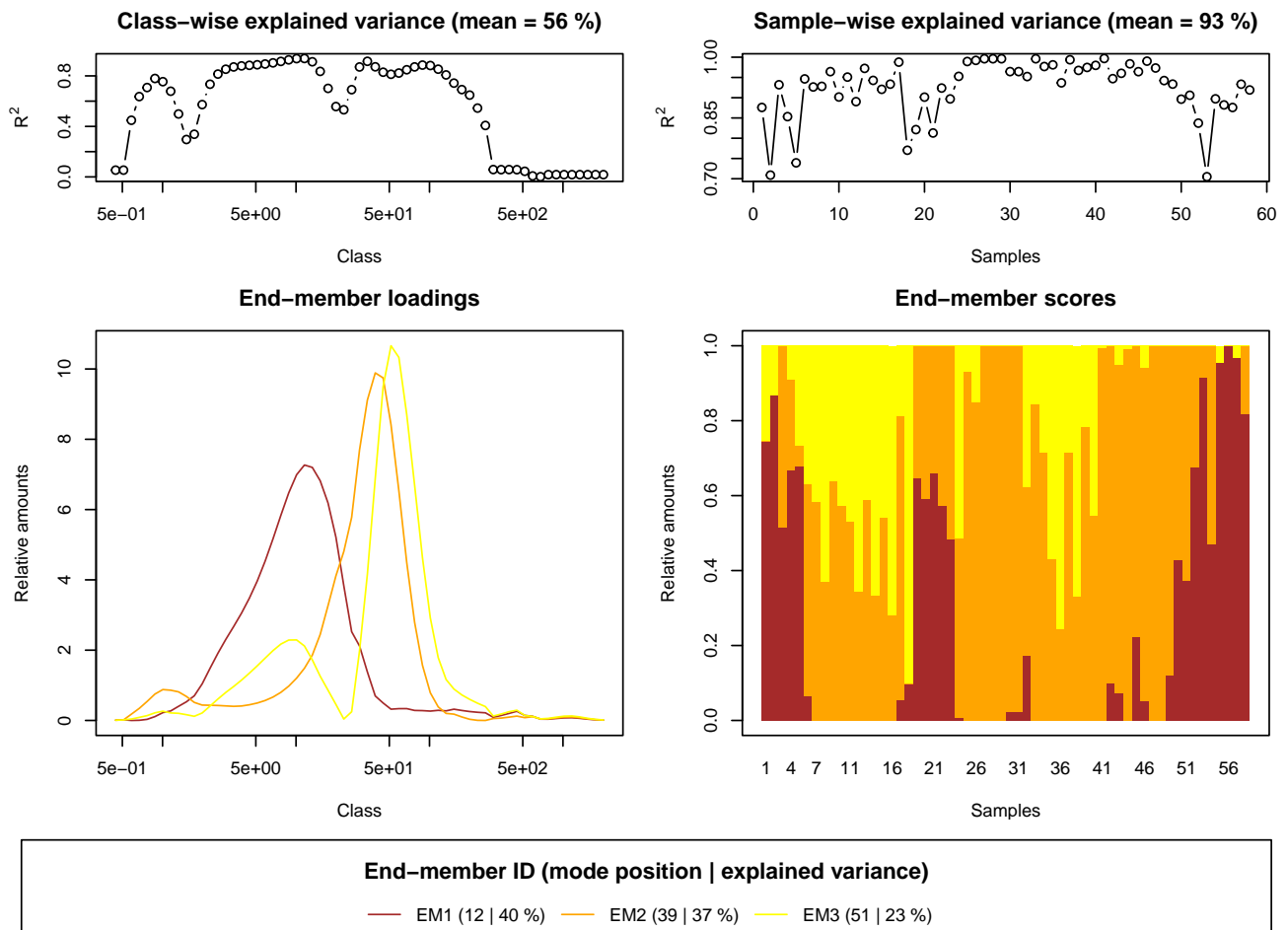
## convert units
X_size_phi <- convert.units(mu = X_size)

## plot a grain-size heat map of the data set
image(x = X_size, z = t(X_gsd), log = "x")
```



Now we run the end-member modelling using 'EMMAgeo' to identify statistically meaningful components in the dataset.

```
E_gsd <- EMMA(  
  X = X_gsd,  
  q = 3,  
  classunits = X_size,  
  plot = TRUE,  
  col = c("brown", "orange", "yellow"),  
  log = "x")
```



Now we can use the output of 'EMMAgeo' to parametrize log-normal distributions mimicking a synthetic grain-size distribution based on actual, empirical data. The modes of the end-members identified by 'EMMAgeo' converted back to the phi scale provide a good starting point.

```
convert.units(mu = E_gsd$modes)
```

```
## [1] 6.434091 4.671752 4.280096

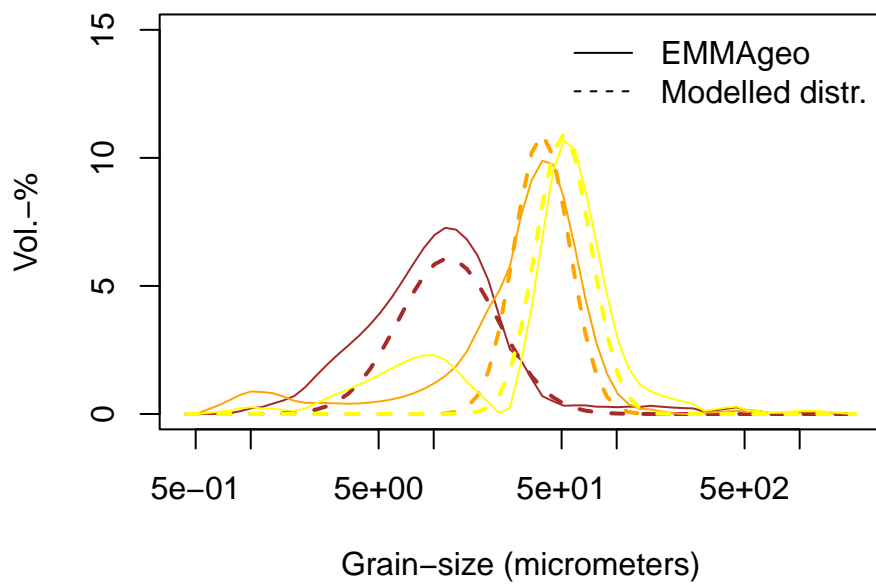
## define log-normal distributions
EM_mod_dist <- rbind(
  dnorm(x = X_size_phi, mean = 6.38, sd = 0.9),
  dnorm(x = X_size_phi, mean = 4.68, sd = 0.5),
  dnorm(x = X_size_phi, mean = 4.29, sd = 0.5))

## normalise distributions to 100 %
EM_mod_dist <- t(
  apply(X = EM_mod_dist,
    MARGIN = 1,
    FUN = function(x) {x / sum(x) * 100}))

## plot EM distributions and respective log-normal functions
plot(NA, xlim = range(X_size), ylim = c(0, 15), log = "x",
  xlab = "Grain-size (micrometers)", ylab = "Vol.-%")

for(i in 1:3) {
  lines(x = X_size, y = E_gsd$loadings[i,],
    col = c("brown", "orange", "yellow")[i])
  lines(x = X_size, y = EM_mod_dist[i,] * 0.7,
    col = c("brown", "orange", "yellow")[i], lty = 2, lwd = 2)
}
```

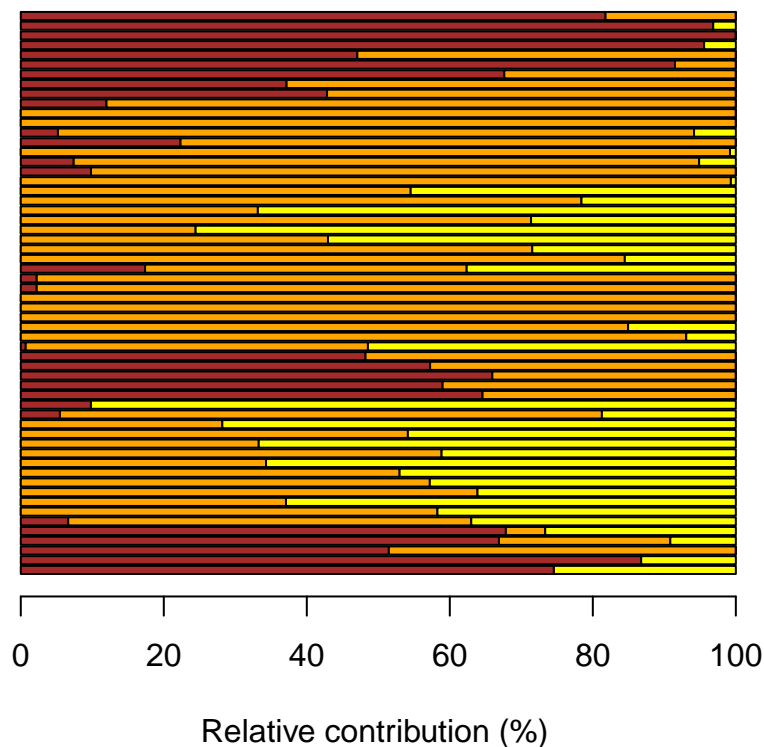
```
legend("topright", c("EMMAgeo", "Modelled distr."),
      lty = c(1,2), lwd = 1, bty = "n")
```



In a next step, we can use the parametric descriptions of the three grain-size end-members to describe the populations of our virtual sediment section, noting that these resemble a set of real-world measurements quite closely. We impose that both means and standard deviations of the grain-size distributions shall be constant with depth. In addition to the parameter description, we also need the rule description, i.e. how the relative contributions of the three populations change with depth. For this constraint, we can use the modelled end-member scores (bar chart in the EMMA output)

```
## assign EMMA scores to output object
EM_mod_cont <- E_gsd$scores

## plot relative contribution as bar chart
barplot(
  t(EM_mod_cont) * 100,
  beside = FALSE, horiz = TRUE,
  col = c("brown", "orange", "yellow"),
  xlab = "Relative contribution (%)")
```



### 1.3 Loading the empirical data set

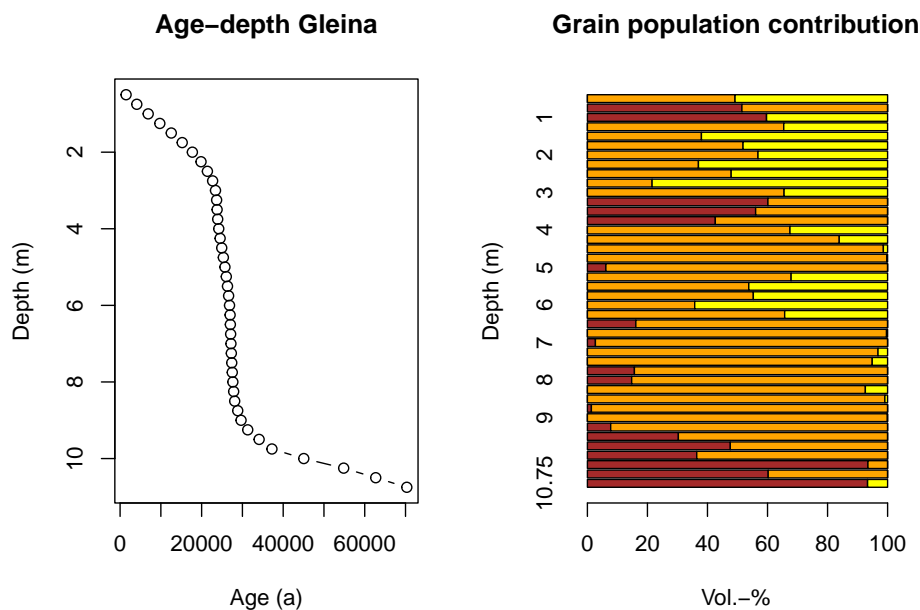
Now we are ready to import the data set and then visualise essential parts of it. Note that we have already interpolated (downscaled) the relative end-member contributions (not shown here) and added a column with luminescence ages (Zech et al., 2017). We have interpolated all measured parameters to equal depth intervals of 25 cm, ranging from 50 cm to 1075 cm depth. This interpolation was simply done to have all empirical data at the same depth intervals. Note that for simplicity, we take the luminescence ages of the dated depths for granted, knowing that this would imply some circular reasoning under normal circumstances. Feel free to plot other (geo)chemical parameters of the data set with depth.

```
## load the measurement data of the Gleina loess section
X <- read.table(
  file = "gleina_interpolated.txt",
  sep = "\t",
  header = TRUE)

## convert depth from cm to m
X$depth_int <- X$depth_int / 100

## plot age-depth relationship
par(mfrow = c(1,2))
plot(x = X$age_int,
     y = X$depth_int,
     ylim = rev(range(X$depth_int)),
     type = "b",
     main = "Age-depth Gleina",
     xlab = "Age (a)",
     ylab = "Depth (m)")

## plot relative contribution of grain-size populations
barplot(rbind(X$EM_1, X$EM_2, X$EM_3)[,nrow(X):1] * 100,
        beside = FALSE,
        horiz = TRUE,
        col = c("brown", "orange", "yellow"),
        names.arg = rev(X$depth_int),
        main = "Grain population contribution",
        xlab = "Vol.-%",
        ylab = "Depth (m)")
```



## 2 Getting started with sandbox

### 2.1 Creating a rule book from scratch

To get started with 'sandbox', we start with an empty rule book using the minimal number of required parameters by running the function `get_RuleBook` with the predefined key word `book = "empty"`. This book can then be renamed by providing the name for the list element `book$book`.

```
## create empty rule book
gleina <- get_RuleBook(book = "empty")

## assign name
gleina$book <- "gleina"
```

Note that the information about ages, populations, grain sizes, packing, and specific densities are only empty containers. To work with more than one population, adding one or more other populations is necessary, using the function `add_Population` and providing the rule book of interest and the number of populations to add. Note that automatically, all other rule book entries will receive an additional number of populations.

```
## add two further populations
gleina <- add_Population(
  book = gleina,
  populations = 2)
```

```
## show structure
str(gleina)
```

```
## List of 6
## $ book      : chr "gleina"
## $ age       :List of 2
## ..$ group: chr "general"
## ..$ age    :List of 2
## .. ..$ type : chr "exact"
## .. ..$ value:function (x, deriv = 0L)
## $ population:List of 4
## ..$ group      : chr "specific"
## ..$ population_1:List of 2
## .. ..$ type : chr "exact"
## .. ..$ value:function (x, deriv = 0L)
## ..$ population_2:List of 2
## .. ..$ type : chr "exact"
## .. ..$ value:function (x, deriv = 0L)
## ..$ population_3:List of 2
## .. ..$ type : chr "exact"
## .. ..$ value:function (x, deriv = 0L)
## $ grainsize  :List of 4
## ..$ group      : chr "specific"
## ..$ grainsize_1:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd  :function (x, deriv = 0L)
## ..$ grainsize_2:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd  :function (x, deriv = 0L)
## ..$ grainsize_3:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd  :function (x, deriv = 0L)
## $ packing    :List of 4
## ..$ group      : chr "specific"
## ..$ packing_1:List of 3
## .. ..$ type: chr "normal"
```

```
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd :function (x, deriv = 0L)
## ..$ packing_2:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd :function (x, deriv = 0L)
## ..$ packing_3:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd :function (x, deriv = 0L)
## $ density :List of 4
## ..$ group : chr "specific"
## ..$ density_1:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd :function (x, deriv = 0L)
## ..$ density_2:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd :function (x, deriv = 0L)
## ..$ density_3:List of 3
## .. ..$ type: chr "normal"
## .. ..$ mean:function (x, deriv = 0L)
## .. ..$ sd :function (x, deriv = 0L)
## - attr(*, "package")= chr "sandbox"
## - attr(*, "medium")= chr "book"
```

Still, the rule book just contains empty templates of rules and parameters. In the next step, we will add rules using the Gleina section data set imported above. Note that one can build an utterly synthetic rule book, as well.

## 2.2 Filling the rule book with meaningful rules

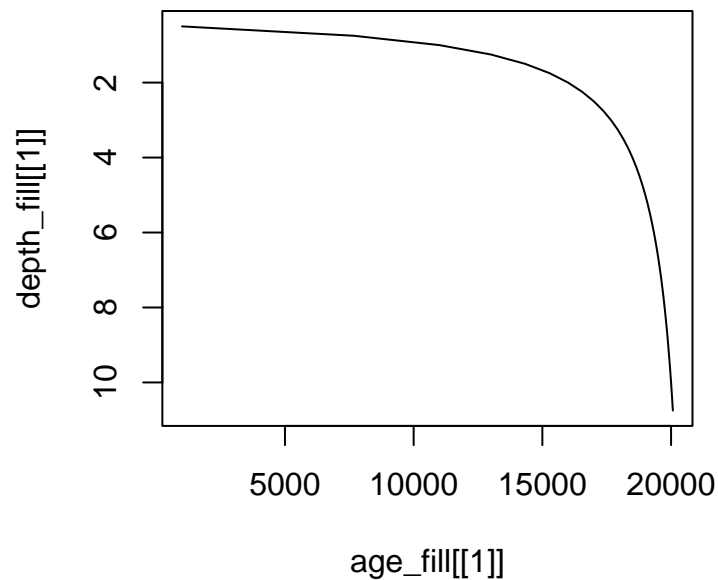
We need the function 'set\_Rule()' to set a rule (i.e., a definition of how a parameter is supposed to change with depth). In the function, we enter the rule book we want to change, the specific parameter to change, and the definition of values and their respective behaviour with depth. Finally, the interpolation function used to generalise the discrete definitions of value and depth information can be set. Currently, only spline interpolation is supported.

In the first step, we only define a completely arbitrary age-depth rule. For instance, the age should follow a  $1/\text{depth}$  relationship, specifically:  $\text{age} = 21000 - 10000/\text{depth}$ . This means that the sediment accumulation rate progressively decreases with younger ages. To implement this in the model, we need to provide lists for the true depth intervals (`depth`) and the corresponding ages (`value`). Lists are required for consistency reasons, as we will see a bit later.

```
## assign rule definitions to lists
depth_fill <- list(X$depth_int)
age_fill <- list(21000 - 10000 / X$depth_int)

## plot age-depth relationship
plot(
  x = age_fill[[1]],
  y = depth_fill[[1]],
  type = "l",
  ylim = rev(range(depth_fill[[1]])))
```



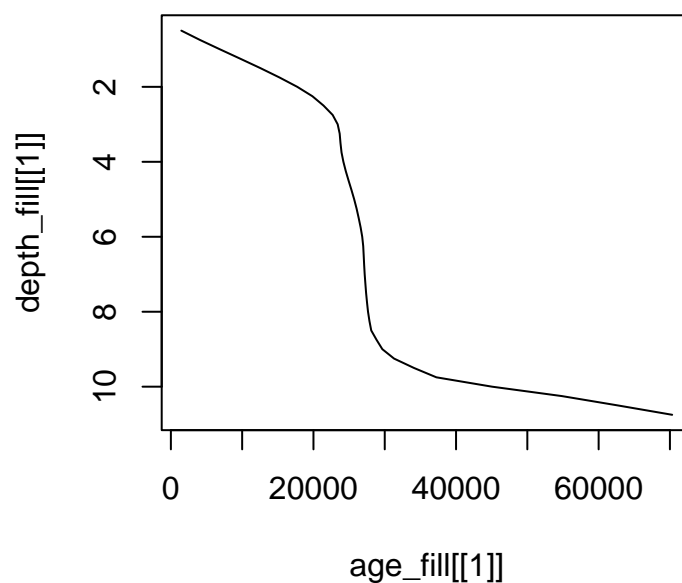


```
## add age definition
gleina <- set_Rule(
  book = gleina,
  parameter = "age",
  value = age_fill,
  depth = depth_fill)
```

Likewise, we can use the empirically determined age-depth relationship obtained from the luminescence-dating analysis of this sediment section (Zech et al., 2017). We simply overwrite the parameter `age_fill` with the real (interpolated) age data.

```
## assign rule definitions to lists
age_fill <- list(X$age_int)

## plot age-depth relationship
plot(
  x = age_fill[[1]],
  y = depth_fill[[1]],
  type = "l",
  ylim = rev(range(depth_fill[[1]])))
```



This rule definition can now be added to the rule book, to the appropriate parameter `age`:

```
## add age definition as rule
gleina <- set_Rule(
```

```

book = gleina,
parameter = "age",
value = age_fill,
depth = depth_fill)

```

In a similar way, we can now also add rules for the relative contribution of grain populations with depth, i.e., our end-member scores from EMMA above, which have already been added to the empirical Gleina data set. Note that the three end-member lists are organised in another list. This hierarchic nesting must be maintained throughout.

```

## assign rule definitions to lists
EM_contr_fill <- list(
  list(X$EM_1),
  list(X$EM_2),
  list(X$EM_3))

## add population contribution with depth
gleina <- set_Rule(
  book = gleina,
  parameter = "population",
  value = EM_contr_fill,
  depth = depth_fill)

```

Once on that road, we may also add rules of grain-size distribution with depth, this time defined by means and standard deviations for each population. Now, it becomes clear why we have used nested lists for these definitions.

Note that we define the population-specific grain-size distributions as constant with depth, i.e., we repeat the respective values for each depth value. The grain size is given in phi-units, according to the means and standard deviations as defined by fitting the EMMA results above. Hence, one may also define population grain sizes that change gradually with depth. Finally, we also define packing densities of 0.7, which are also constant with depth. In sedimentary environments, one may assume that packing density increases with depth, but we keep things simple here.

```

## get number of samples in interpolated data set
n <- nrow(X)

## assign rule definitions to lists
EM_gsd_fill <- list(
  list(mean = rep(6.38, n),
        sd = rep(0.9, n)),
  list(mean = rep(4.69, n),
        sd = rep(0.5, n)),
  list(mean = rep(4.29, n),
        sd = rep(0.5, n)))

EM_packing_fill <- list(
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)),
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)),
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)))

EM_density_fill <- list(
  list(mean = rep(2.5, n),
        sd = rep(0.01, n)),
  list(mean = rep(2.5, n),
        sd = rep(0.01, n)),
  list(mean = rep(2.5, n),
        sd = rep(0.01, n)))

```

```
## add population contribution with depth
gleina <- set_Rule(book = gleina,
  parameter = "grainsize",
  value = EM_gsd_fill,
  depth = depth_fill)

gleina <- set_Rule(book = gleina,
  parameter = "packing",
  value = EM_packing_fill,
  depth = depth_fill)

gleina <- set_Rule(book = gleina,
  parameter = "density",
  value = EM_density_fill,
  depth = depth_fill)
```

This is it. We have defined a rule book describing a synthetic sediment section, based on empirical data. Note that we need to define at least these crucial rules in almost all cases: **age**, **population**, **grainsize**, **packing**. We can now “harvest” our efforts and sample the sediment section.

### 3 Sampling a sediment section

Taking a sample in 'sandbox' is performed by the function `make_Sample()`. Like in real life, we will need to find a suitable sample container first. Currently, 'sandbox' supports cuboids (`geometry = "cuboid"`) and cylinders (`geometry = "cylinder"`) as sample containers:

- cuboids are defined by their height, width and length,
- cylinders by their radius and length.

In addition, we will need to specify the sampling depth with respect to the centre of the sampling container, and of course, the rule book that describes the sediment section we wish to sample. In the example below, we take one sample from 3.5 m depth, using a cuboid container of 5 cm height and width and a length of 10 micrometers. Note that sampling a virtual section is super efficient, so there is no need to take more material than we would need to get the information we want to get from it. However, note that large sample volumes for fine-grained materials will result in enormous amounts of individual grains. This may bring computers to their limits in terms of data handling. It is, therefore, wise to start with small sample volumes and increase them if needed. Overall, taking samples is the most time-consuming step in the 'sandbox' workflow routine.

*Note: 'sandbox' applies parallel processing to speed up the process of sampling our virtual sediment section; if multiple cores are available!*

```
## set random seed to keep
## this example reproducible
set.seed(2021)
```

```
## take a sample
sample_01 <- make_Sample(
  book = gleina,
  depth = 3.5,
  geometry = "cuboid",
  height = 0.05,
  width = 0.05,
  length = 0.00001)
```

```
## inspect object structure
str(sample_01)
```

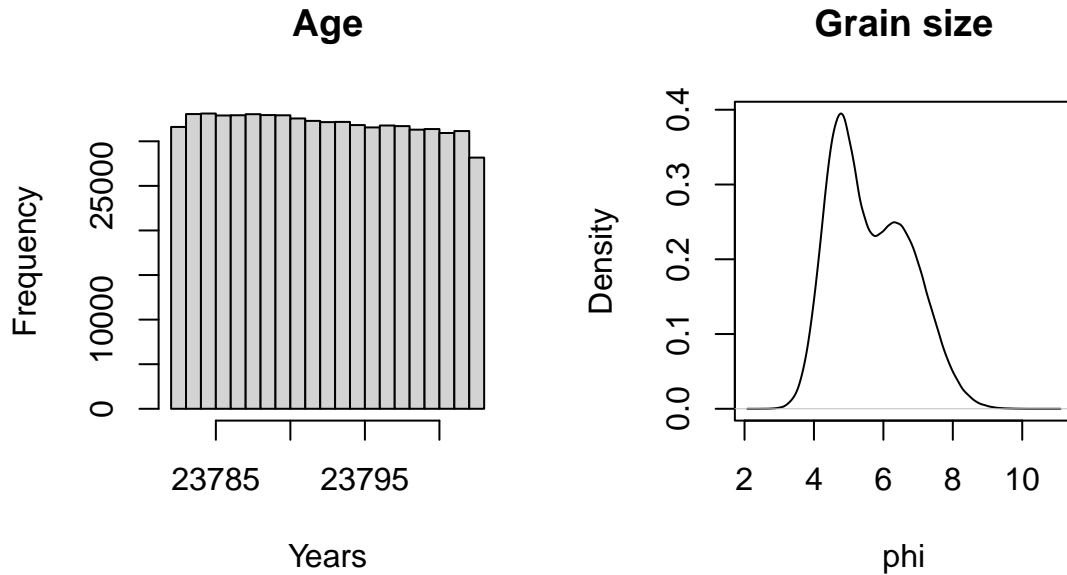
```
## 'data.frame':   671438 obs. of  8 variables:
## $ grains      : num  1 2 3 4 5 6 7 8 9 10 ...
## $ d_sample    : num  3.5 3.51 3.49 3.52 3.49 ...
## $ population: num  1 2 1 2 2 2 2 2 1 1 ...
## $ age         : num  23793 23797 23787 23802 23786 ...
## $ population: num  0.56 0.447 0.569 0.455 0.43 ...
## $ grainsize   : num  7.07 4.13 6.56 4.92 4.88 ...
## $ packing     : num  0.695 0.693 0.703 0.712 0.694 ...
## $ density     : num  2.51 2.51 2.51 2.49 2.5 ...
## - attr(*, "package")= chr "sandbox"
```

When we take a look at what we have sampled, we see that the created object contains 671438 individual grains, each described by a grain ID (`grains`), depth (`d_sample`), population ID (`population`, note the second population entry, which is a computational reference and can be ignored), true depositional age (`age`), grain diameter (`grainsize`), packing density (`packing`) and specific density (`density`). We could now go ahead and inspect for example the grain-size distribution or the age distribution:

```
par(mfrow = c(1,2))
```

```
## plot a histogram of the single grain ages
hist(x = sample_01$age, main = "Age", xlab = "Years")
```

```
## plot a density estimate plot of the grain-size distribution
plot(density(x = sample_01$grainsize),
     main = "Grain size",
     xlab = "phi")
```



As expected for a homogeneous sample covering a 5 cm depth interval with an almost constant deposition rate at that scale, the age histogram is nearly flat because of nearly uniform age distribution, biased only slightly due to the change in deposition rate picked up by the container's vertical extent. In contrast, the kernel density estimate of the grain-size distribution is bimodal. This is also as expected for a depth interval where end-members 1 (6.38 phi  $\sim$  12  $\mu$ m) and 2 (4.69 phi  $\sim$  38.7  $\mu$ m) dominate the contribution levels.

### 3.1 Adding further parameters/rules to pursue specific research questions

We can also add further rules to define grain parameters to pursue other research questions. For example, we can add major element concentrations or the pH value from the Gleina section data set. To add further rules, we need the function `add_Rule()`. There, we need to select the rule book we want to modify, provide a name (e.g., pH), decide whether the new rule is a general one (valid for all grains) or a specific one (the same only within populations). We also need to specify how the pH value shall vary, for example, uniformly, and finally provide the number of populations. We can check the changes, for example, by calling the names of the rule book entries.

```
gleina <- add_Rule(
  book = gleina,
  name = "pH",
  group = "specific",
  type = "uniform")
names(gleina)

## [1] "book"      "age"      "population" "grainsize" "packing"
## [6] "density"   "pH"
```

As with the empty rule book from the section 2.1, this freshly added rule is also just a template that must be filled. Thus, we follow the same workflow as before. We use the empirical pH values and allow for a uniform scatter around the mean of 0.2.

```
## assign rule definitions to lists
pH_fill <- list(
  list(min = X$pH - 0.2,
        max = X$pH + 0.2),
  list(min = X$pH - 0.2,
        max = X$pH + 0.2),
  list(min = X$pH - 0.2,
        max = X$pH + 0.2))

## add pH value evolution with depth
gleina <- set_Rule(
  book = gleina,
  parameter = "pH",
  value = pH_fill,
  depth = depth_fill)
```

And now, we can take a densely spaced sequence of small samples to inspect the pH value patterns with depth.

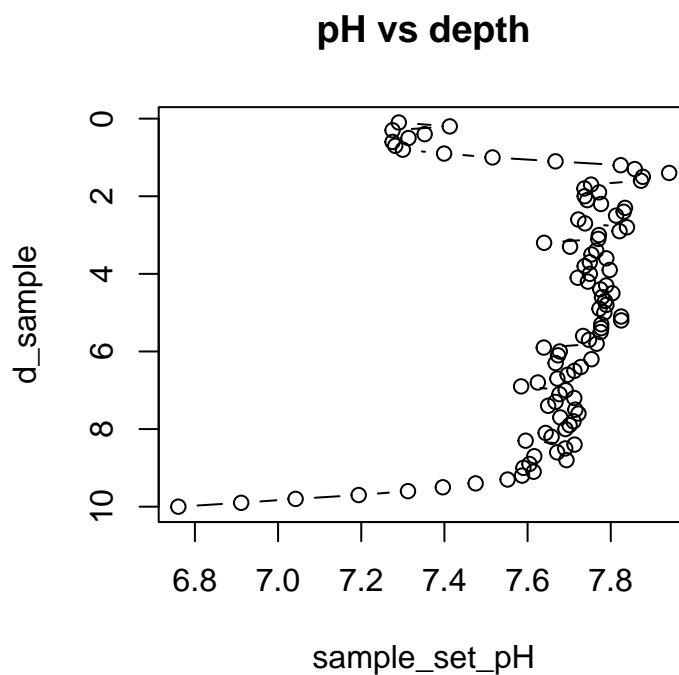
```
## set new random seed
set.seed(20212)

## define sampling interval
d_sample <- seq(from = 0.1, to = 10, by = 0.1)

## take a sample
sample_set <- lapply(X = d_sample, FUN = function(z) {
  make_Sample(
    book = gleina,
    depth = z,
    geometry = "cuboid",
    height = 0.0001,
    width = 0.0001,
    length = 0.0001)
})

## extract mean pH values
sample_set_pH <- vapply(sample_set, function(x) mean(x$pH), numeric(1))

## plot pH values with depth
plot(
  x = sample_set_pH,
  y = d_sample,
  type = "b",
  ylim = rev(range(d_sample)),
  main = "pH vs depth")
```



## 4 Further sample treatment routines

Usually, one will not work with the collected bulk sample material but instead, treat it into shape. Two typical workflow steps will be sieving and sub-sampling (preparation of aliquots). These two processes are implemented in 'sandbox', via the functions `prepare_Sieving()`, `prepare_Subsample()`, and `prepare_Aliquot()`. To illustrate their usage, let us work with the bulk sample from above (`sample_01`).

Sieving with `prepare_Sieving()` requires specifying the sample to process and the sieve intervals in phi units. As you may have noticed, a lot of the grain-size information in 'sandbox' needs to be provided in the phi scale. To make it easier for users, there is a simple conversion function between metric and phi scales: `convert_units()`, which we have already used above to convert the units for you.

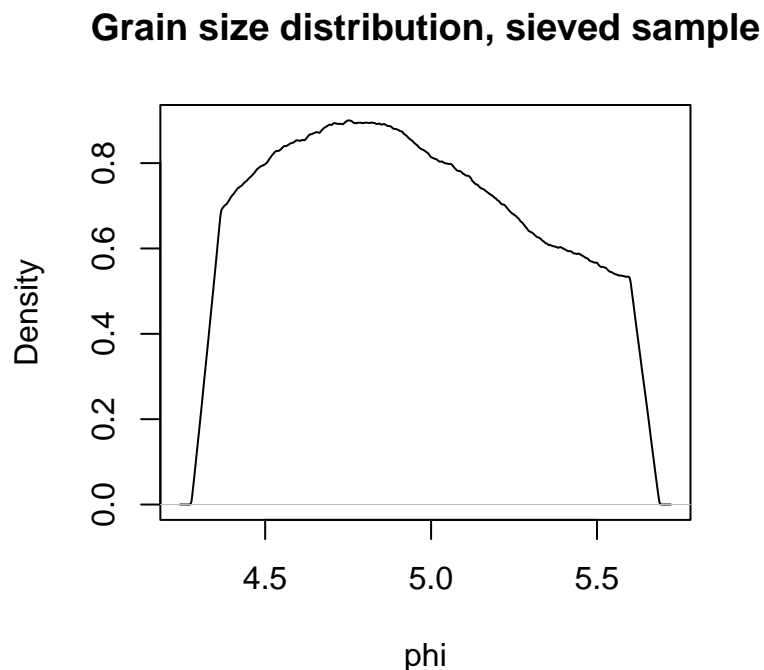
To illustrate this, we use the following code block to convert metric sieve intervals to phi units and use these for subsequent sieving. Note that we revert the order (`sort`) of the grain-size limits after converting the metric to the phi-scale, because the sieve limits must be provided in ascending order. Further, we use a rectangular density estimation kernel to avoid too many artefacts at the limits of the distribution.

```
## define grain-size limits in micrometres
lims <- c(20, 50)

## convert limits to phi scale
lims <- sort(convert_units(mu = lims))

## sieve sample
sample_01_sieved <- prepare_Sieving(
  sample = sample_01,
  interval = lims)

## plot sieved sample's grain-size distribution
plot(
  density(sample_01_sieved$grainsize, kernel = "rectangular"),
  main = "Grain size distribution, sieved sample",
  xlab = "phi")
```



Creating subsamples is a routine step in sediment analysis. Subsamples can be defined based on splitting the bulk sample into equally large subsamples to reach a defined number. Likewise, subsamples may be created based on a defined subsample volume or mass. These three approaches are available in 'sandbox'.

```
## create subsamples based on a fixed number of output samples
subsamples_number_sample_01_sieved <- prepare_Subsample(
  sample = sample_01_sieved,
  number = 10)
```

```
length(subsamples_number_sample_01_sieved)
```

```
## [1] 10
```

```
## create subsamples based on a fixed subsample volume of a 2 mm sized cube
```

```
subsamples_volume_sample_01_sieved <- prepare_Subsample(
```

```
  sample = sample_01_sieved,
```

```
  volume = 0.001^3)
```

```
length(subsamples_volume_sample_01_sieved)
```

```
## [1] 9
```

```
## create subsamples based on a fixed subsample weight of 0.05 mg
```

```
subsamples_weight_sample_01_sieved <- prepare_Subsample(
```

```
  sample = sample_01_sieved,
```

```
  weight = 5 * 10^-9)
```

```
length(subsamples_weight_sample_01_sieved)
```

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

A special kind of subsampling, primarily used in luminescence dating, is creating aliquots. Here, aliquots are defined as grains mounted as grain monolayers on tiny disks of known diameter. The diameter is expressed in mm. In addition, it is possible to set the grain packing density on the disk, which is by default set to 0.65.

```
aliquots_sample_01_sieved <- prepare_Aliquot(
```

```
  sample = sample_01_sieved,
```

```
  diameter = 10)
```



## 5 Adding physical deposition laws

By default 'sandbox' does not include any physically based rules on how the grain parameters within a modelled section shall behave. This includes both, their autigenic changes with depth such as compaction or lithification, and correlations among grain properties during transport and deposition. The main reason for this by default rather simplistic approach is that for many scientific questions that can be asked with 'sandbox', there is no need for such physically based rules, and having to add them would introduce a significant extra burden to build a rule book. In addition, there is not one universal physical rule but a range of proposed ones. However, default simplicity this does not mean that 'sandbox' cannot be used with more dedicated relationships. Here, we illustrate two scenarios. First, we show how depth dependent compaction can be parameterised. Then, we show how mixing of two mineral types, quartz and zircon, can be implemented.

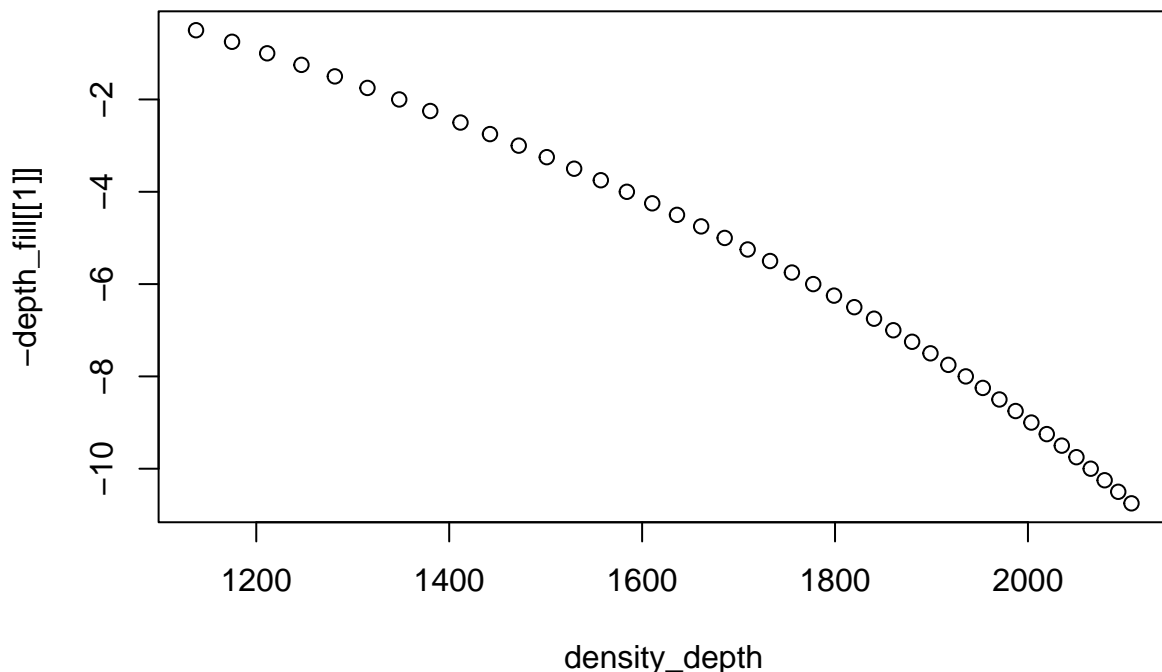
### 5.1 Depth-dependent packing density

We let the raw density be defined by depth following a classic yet simple porosity model (Sheldon & Retallack, 2001):

```
density_bulk <- function(porosity_0, depth, k, density_specific) {  
  return(density_specific - density_specific * porosity_0 * exp(-k * depth))  
}
```

Here, `porosity_0` is the initial porosity at zero depth, for loessic material most likely somewhere around 0.6, `depth` is the burial depth in metres, `k` is the compaction rate coefficient in  $(\text{Pa s})^{-1}$ , strongly material dependent and arbitrarily set to  $10^{-1}$  here, and `density_specific` is the specific density of the sediment grains, most likely somewhere around  $2650 \text{ kg/m}^3$ . Applying this equation to a depth vector ranging from 0 to 10 m gives is a simple yet physically meaningful depth-density relationship:

```
depth_seq <- seq(from = 0, to = 10, by = 0.5)  
density_depth <- density_bulk(porosity_0 = 0.6,  
                              depth = depth_fill[[1]],  
                              k = 10^-1,  
                              density_specific = 2650)  
  
plot(x = density_depth, y = -depth_fill[[1]])
```



The predicted packing density vector can now be assigned to the rule book of the sediment section to model. Note that here we assign the same packing density to all three populations, including a small random scatter. The result of that should be that with increasing depth, we will get more and more grains in our samples due to the higher packing density. Note that due to the also changing contribution of end-members of different size also influence the result, which is why the resulting plot does not resemble the above relationship one to one.

```

## assign rule definitions to lists
packing_fill <- list(
  list(min = density_depth - 10,
        max = density_depth + 10),
  list(min = density_depth - 10,
        max = density_depth + 10),
  list(min = density_depth - 10,
        max = density_depth + 10))

## add updated packing density evolution with depth
gleina_packing <- set_Rule(book = gleina,
                           parameter = "packing",
                           value = packing_fill,
                           depth = depth_fill)

## define sampling interval
d_sample <- seq(from = 0.1, to = 10, by = 2)

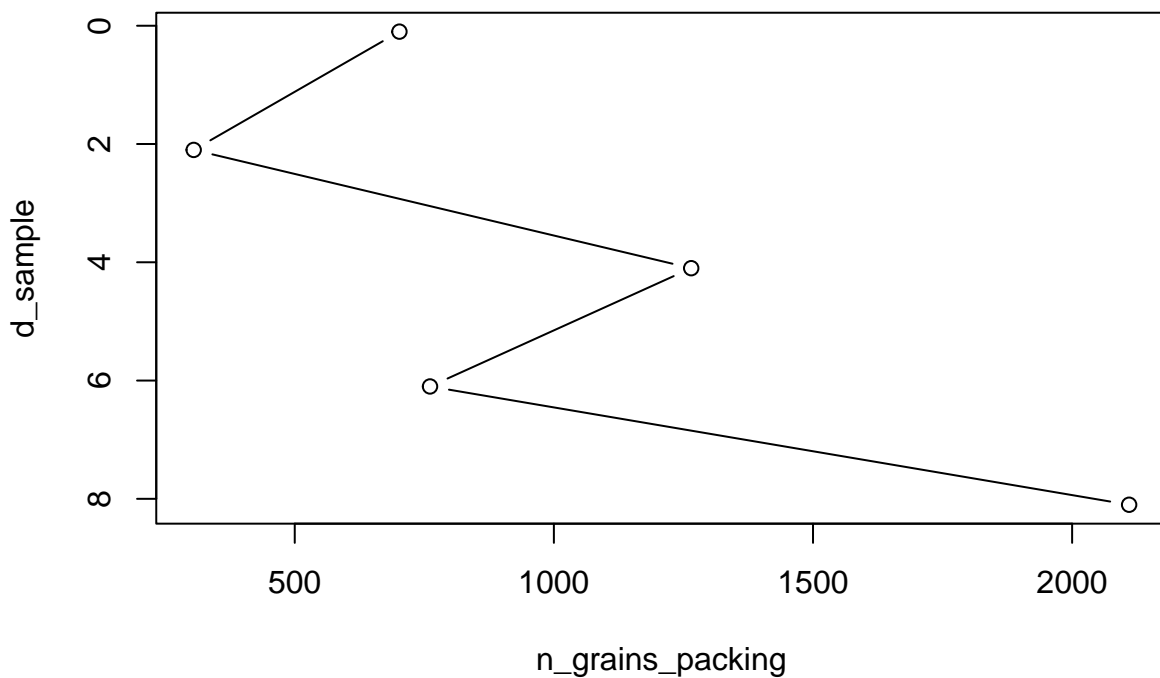
## take a sample
sample_set_packing <- lapply(X = d_sample, FUN = function(z) {
  make_Sample(
    book = gleina_packing,
    depth = z,
    geometry = "cuboid",
    height = 0.00005,
    width = 0.00005,
    length = 0.00005)
})

## collect number of grains in sample
n_grains_packing <- vapply(sample_set_packing, function(x) length(x$age), numeric(1))

## plot pH values with depth
plot(
  x = n_grains_packing,
  y = d_sample,
  type = "b",
  ylim = rev(range(d_sample)),
  main = "Sampled grains as function of depth-dependent packing density")

```

## Sampled grains as function of depth-dependent packing density



### 5.2 Mixing of two mineral populations with different densities

Aeolian transport of mineral grains follows to a significant extent Stokes law. Simply put, a fluid such as air is able to drag particles of a given threshold mass. For ideal spherical mineral grains, that mass corresponds to a characteristic volume (or diameter) which is set by the specific density of that mineral. Take for example, grains of quartz and zircon. Quartz grains ( $2.65 \text{ g/cm}^3$ ) have a lower specific density than zircon grains ( $4.6 \text{ g/cm}^3$ ). Hence, a wind gust can lift larger quartz grains than zircon grains, specifically the zircon grains will have an about 83.2 % smaller diameter.

```
## calculate mass of a 1 mm large sand grain (m = rho V = rho 4/3 pi r^3)
m_sand <- 2650 * (4/3 * pi * 0.0005^3)
```

```
## calculate diameter of an equally heavy zircon grain
d_zirc = 2 * (m_sand / (4/3 * pi * 4600))^(1/3)
```

```
## get percentage of diameter
d_zirc * 100 / 0.001
```

```
## [1] 83.20754
```

To parameterise an equal gravimetric mixture of sand and zircon grains in 'sandbox', we need to change one of the existing populations. Let us take the finest population as an example (EM 1 with a mode at 6.38 phi, which dominates the basal parts of the Gleina section). We need to describe this end-member now by two distinct 'sandbox' populations, one made of quartz and one made of zircon with otherwise the same properties and mixing ratios. The zircon population would have a diameter of `convert_units(mu = convert_units(phi = 6.38) * 0.83) = 6.65 phi`. The end-member scores as depth-dependent contribution of EM1 needs to be equally shared between the new quartz and zircon population. Hence, we need to do the following: adding a new population, update its diameter and density, update the contributions of population 1 and 4. Note that we also change the grain-size standard deviation of EM3 from 0.9 to 0.1, to make the effects of bimodality visible.

```
## add a further populations
gleina_zirc <- add_Population(
  book = gleina,
  populations = 1)

## assign rule definitions to lists
EM_contr_zirc <- list(
  list(X$EM_1 / 2),
```

```

list(X$EM_2),
list(X$EM_3),
list(X$EM_1 / 2))

EM_gsd_zirc <- list(
  list(mean = rep(6.38, n),
        sd = rep(0.1, n)),
  list(mean = rep(4.69, n),
        sd = rep(0.5, n)),
  list(mean = rep(4.29, n),
        sd = rep(0.5, n)),
  list(mean = rep(6.65, n),
        sd = rep(0.1, n)))

EM_packing_zirc <- list(
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)),
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)),
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)),
  list(mean = rep(0.7, n),
        sd = rep(0.01, n)))

EM_density_zirc <- list(
  list(mean = rep(2.65, n),
        sd = rep(0.01, n)),
  list(mean = rep(2.65, n),
        sd = rep(0.01, n)),
  list(mean = rep(2.65, n),
        sd = rep(0.01, n)),
  list(mean = rep(4.60, n),
        sd = rep(0.01, n)))

## add population contribution
gleina_zirc <- set_Rule(
  book = gleina_zirc,
  parameter = "population",
  value = EM_contr_zirc,
  depth = depth_fill)

gleina_zirc <- set_Rule(
  book = gleina_zirc,
  parameter = "grainsize",
  value = EM_gsd_zirc,
  depth = depth_fill)

gleina_zirc <- set_Rule(book = gleina_zirc,
  parameter = "packing",
  value = EM_packing_zirc,
  depth = depth_fill)

gleina_zirc <- set_Rule(book = gleina_zirc,
  parameter = "density",
  value = EM_density_zirc,
  depth = depth_fill)

```

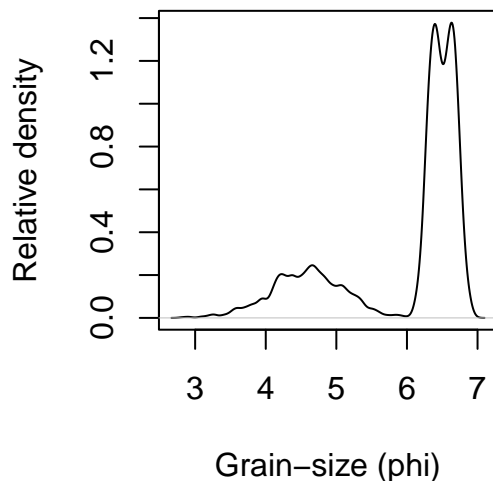
Now, we can sample the new section near its base where the modified end-member is supposed to dominate and plot its properties in terms of grain-size distribution and individual grain densities. Note that for the latter plot we need to isolate grains that are either from population 1 or 4.

```
## sample the section
sample_zirc <- make_Sample(
  book = gleina_zirc,
  depth = 10.7,
  geometry = "cuboid",
  height = 0.001,
  width = 0.001,
  length = 0.0001)

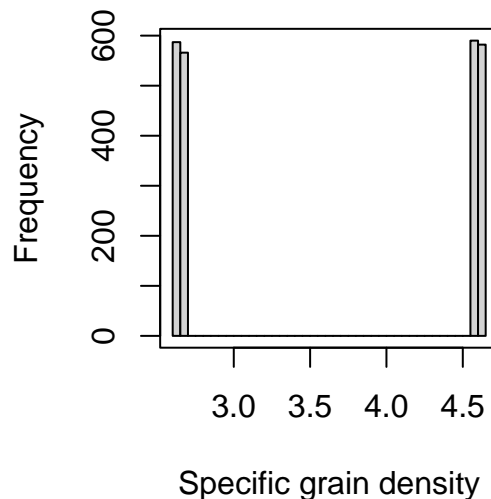
## plot results
par(mfcol = c(1, 2))
plot(density(sample_zirc$grainsize, bw = 0.05),
     xlab = "Grain-size (phi)",
     ylab = "Relative density",
     main = "Grain-size distribution")

grains_EM3 <- which(sample_zirc$population %in% c(1, 4))
hist(sample_zirc$density[grains_EM3],
     breaks = 50,
     xlab = "Specific grain density",
     main = "Grain density values of EM3 grains")
box()
```

**Grain-size distribution**



**Grain density values of EM3 grains**



The bimodality of the new EM3, now equally composed of quartz and zircon grains, is obvious but only because we set the width of the grain-size distributions that build it to unrealistically small values (0.01 instead of 0.9). Otherwise, the size effect would not be visible. In other words, from a grain-size perspective such isogravitational grain mobilisation effects may be ignored for distributions that are wide enough. In contrast, the density effects are clear and reflect the composition character of EM3 that we have imposed by our actions above. A practical implication arises especially for OSL work flows, where zircon contaminations have major effects on the measurement results and therefore those minerals are explicitly removed by, for example density separation.

## 6 Optimising the depositional process – adding non-linear age-depth rules

In sandbox, one could easily implement non-linear age depth relationships. As explained in the above example, for many applications a simple age depth model is just fine. However, there might be cases when more elaborated approaches are needed. In principle there is no such thing like continuous sediment accumulation. Rather sediment is brought in pulses intercepted by pauses. This ultimately results in a step function shape of the age depth relationship.

The example below shows how regular step functions of sedimentation and paused sedimentation can be parameterised. We define a regular spaced time series and a step change depth vector. Then, we fit a spline to the age-depth relationship. Note that we decisively use a monotonic spline (`method = "hyman"`) to avoid age inversions and that the minimum resolution of the spline should be high enough to adequately describe the imposed step function (10 versus 100 values).

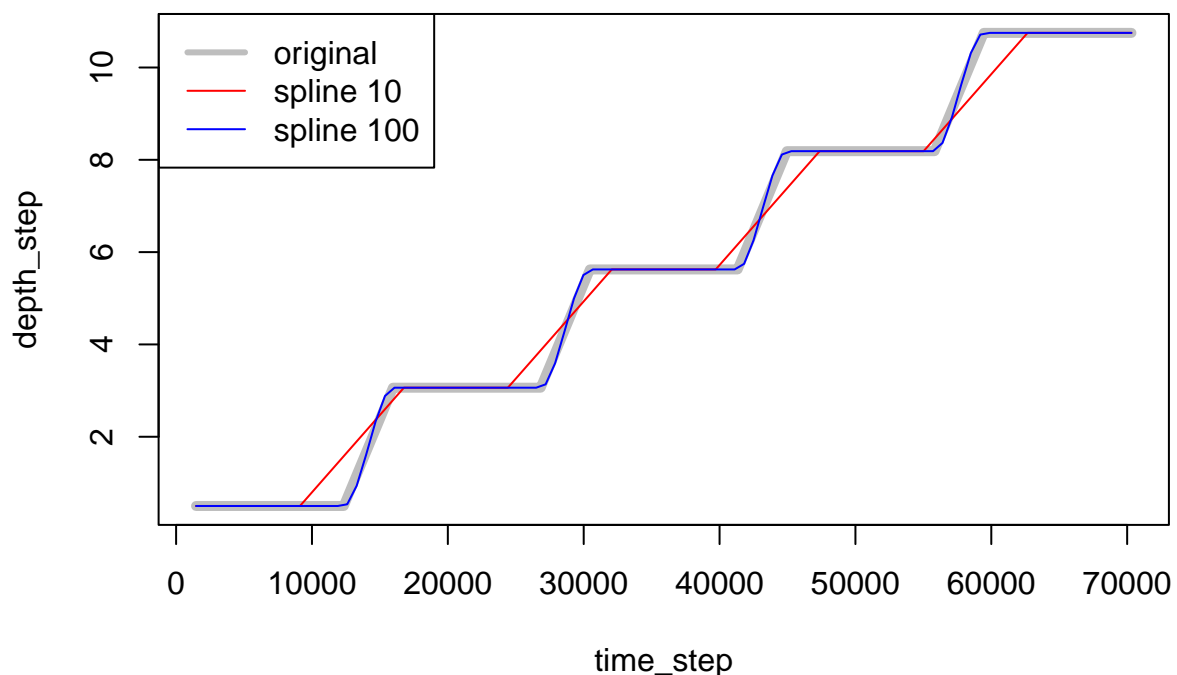
```
time_step <- seq(from = min(age_fill[[1]]),
                to = max(age_fill[[1]]),
                length.out = 20)
depth_step <- rep(seq(from = min(depth_fill[[1]]),
                    to = max(depth_fill[[1]]),
                    length.out = 5), each = 4)

time_int_10 <- seq(from = min(age_fill[[1]]),
                  to = max(age_fill[[1]]),
                  length.out = 10)

time_int_100 <- seq(from = min(age_fill[[1]]),
                   to = max(age_fill[[1]]),
                   length.out = 100)

depth_int_10 <- spline(x = time_step, y = depth_step, xout = time_int_10,
                     method = "hyman")
depth_int_100 <- spline(x = time_step, y = depth_step, xout = time_int_100,
                      method = "hyman")

plot(x = time_step, y = depth_step, type = "l", lwd = 5, col = "grey")
lines(x = depth_int_10$x, y = depth_int_10$y, col = "red")
lines(x = depth_int_100$x, y = depth_int_100$y, col = "blue")
legend(x = "topleft", legend = c("original", "spline 10", "spline 100"),
      col = c("grey", "red", "blue"), lty = 1, lwd = c(3, 1, 1))
```



It is a little more effort to define such a behaviour of an age depth relationship, but it is easily possible. However, having non-linear deposition as a default feature of 'sandbox' would impose quite some hard constraints to the modelling framework.

## 6.1 Issue 01b – Implementing pulse-pause effects for age-depth relations

Usually the sequences of deposition pulses and pauses are not regular. It is further assumed that both pulses and pauses as well as previous and subsequent pulses are autocorrelated. Hence, taking the above issue a step further means that both the deposition values and pause values can change, to some extent independent or dependent from each other. To illustrate how such a behaviour can be realised in 'sandbox' we create a perfectly linear age-depth relationship in this case, as well. Then, we create random deposition pauses (very clumsy but just for illustration) by three random numbers between zero and the maximum of time minus the maximum pause duration. We also impose that pauses are not allowed to overlap, thus we add to each pause the maximum duration time afterwards. Now, we change the depth values during the three pauses to constant and lower all later depth values to the value at the end of the pause. Again, the code is clumsy but does its job. Then, we fit the spline again and have a representation of a non-linear age depth relationship for which we can control the number of possible breaks and their duration. We could also add more parameters to define their clustering in time (autocorrelation) and so on but let us not make things more complicated than needed, here.

```
## define initial time and depth vectors
time <- seq(from = 0, to = 10, length.out = 100)
depth <- seq(from = 1, to = 5, length.out = length(time))

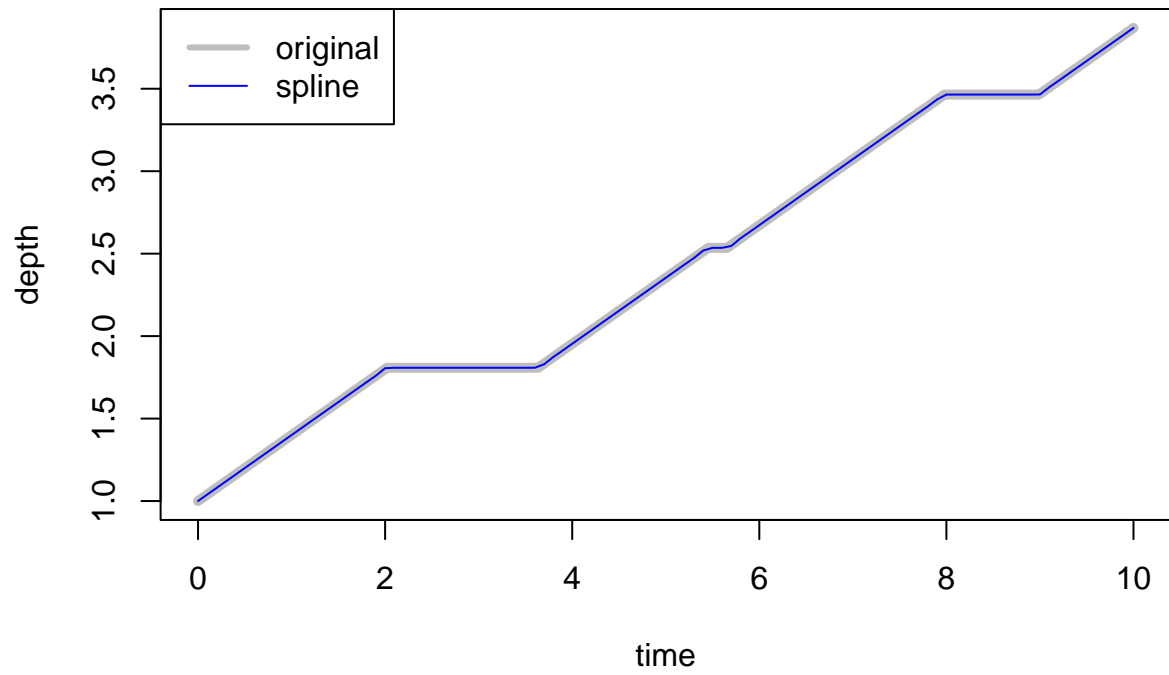
## create sedimentation breaks and their durations
i_breaks <- round(sort(runif(3, 0, length(time) - 80)))
i_breaks <- i_breaks + (1:3 * 20)
d_breaks <- round(runif(3, 1, 20))

## implement breaks in depth vector
for(i in 1:length(d_breaks)) {

  depth[i_breaks[i]:(i_breaks[i] + d_breaks[i])] <- depth[i_breaks[i]]
  depth[(i_breaks[i] + d_breaks[i] + 1):length(depth)] <-
    depth[(i_breaks[i] + d_breaks[i] + 1):length(depth)] -
    (depth[(i_breaks[i] + d_breaks[i] + 1)] -
     depth[(i_breaks[i] + d_breaks[i])])
}

## fit spline to relationship
s_01 <- spline(x = time, y = depth,
               xout = seq(from = min(time), to = max(time), by = 0.1),
               method = "hyman")

plot(x = time, y = depth, type = "l", lwd = 5, col = "grey")
lines(x = s_01$x, y = s_01$y, col = "blue")
legend(x = "topleft", legend = c("original", "spline"),
       col = c("grey", "blue"), lty = 1, lwd = c(3, 1))
```



The real problem arises when we want to have ensembles of such non-linear age depth relationships implemented. We see no other useful approach than Monte Carlo methods. This means, that if one wishes to study the effect of 1000 possible non-linear age depth relationships then the above code must be run 1000 times. If one wishes to propagate these effects into sandbox, then the entire sandbox analysis code snippet needs to be evaluated 1000 times. All this is possible and can be implemented, albeit at the cost of extra coding and repetitive computation runs.



## 7 Extension of the framework – the luminescence scope

The 'sandbox' framework is deliberately kept open to extending its functionality into different analytical directions, such as geochemistry, sedimentology, or geochronology. One example of the latter scientific field is luminescence dating. Scientists have contributed other R packages: 'Luminescence' (Kreutzer et al., 2012) and 'RLumModel' (Friedrich et al., 2016), for example. For a seamless interaction with these other packages, 'sandbox' has been adopted to support keywords that create grain parameters required to efficiently integrate related functions.

Here, we illustrate how 'sandbox' can be extended to create virtual sediment sections that contain all relevant parameters to simulate the luminescence behaviour of grains following established models (Friedrich, 2018). Hence, these virtual sections can be sampled. The resulting virtual luminescence measurements can be seamlessly analysed with the R package 'Luminescence', following standard routines.

To create a rule book with all grain parameters relevant for luminescence modelling already defined, we use the argument `osl`, providing it with a keyword that describes which luminescence model to follow, for example, `osl = "Bailey2001"` (Bailey, 2001). Then, we can build the rules as in the example above, but simpler by setting the depth from 0.5 m to 10.5 m in 1 m intervals. The true depositional age ranges linearly from 500 to 10,500 years. And, using the default settings for the 56 parameters of the Bailey (2001) model for bright grains, we can set up our rule book at once. Note that these 56 parameters can also be explicitly defined manually, similar to adding the pH value to the Gleina section. Likewise, the default parameter settings can be modified as needed at any time, which is what we will do below this example here.

```
## get empty rule book
book_osl <- get_RuleBook(
  book = "empty",
  osl = "Bailey2001")

## change age depth data
depth_true <- list(
  seq(from = 0.5,
    to = 10.5,
    by = 1))

## get number of depth intervals
n_depth <- length(depth_true[[1]])

## set true age
age_true <- list(
  seq(from = 0,
    to = 10500,
    length.out = n_depth))

## set grain-size distribution
gsd_osl <- list(
  list(mean = rep(2.5, n_depth),
    sd = rep(0.05, n_depth)))

## set dose rate
mean_dose_rate_natural <- 0.004
sd_dose_rate_natural <- 0.001
drate_osl <- list(
  list(mean = rep(mean_dose_rate_natural, n_depth),
    sd = rep(sd_dose_rate_natural, n_depth)))

## update rule book with true age and depth definition
book_osl <- set_Rule(
  book = book_osl,
  parameter = "age",
  value = age_true,
  depth = depth_true)
```

```
## update rule book with default luminescence model parameters
book_osl <- set_Rule(
  book = book_osl,
  parameter = "Bailey2001",
  depth = depth_true)

book_osl <- set_Rule(
  book = book_osl,
  parameter = "grainsize",
  value = gsd_osl,
  depth = depth_true)

book_osl <- set_Rule(
  book = book_osl,
  parameter = "osl_doserate",
  value = drate_osl,
  depth = depth_true)
```

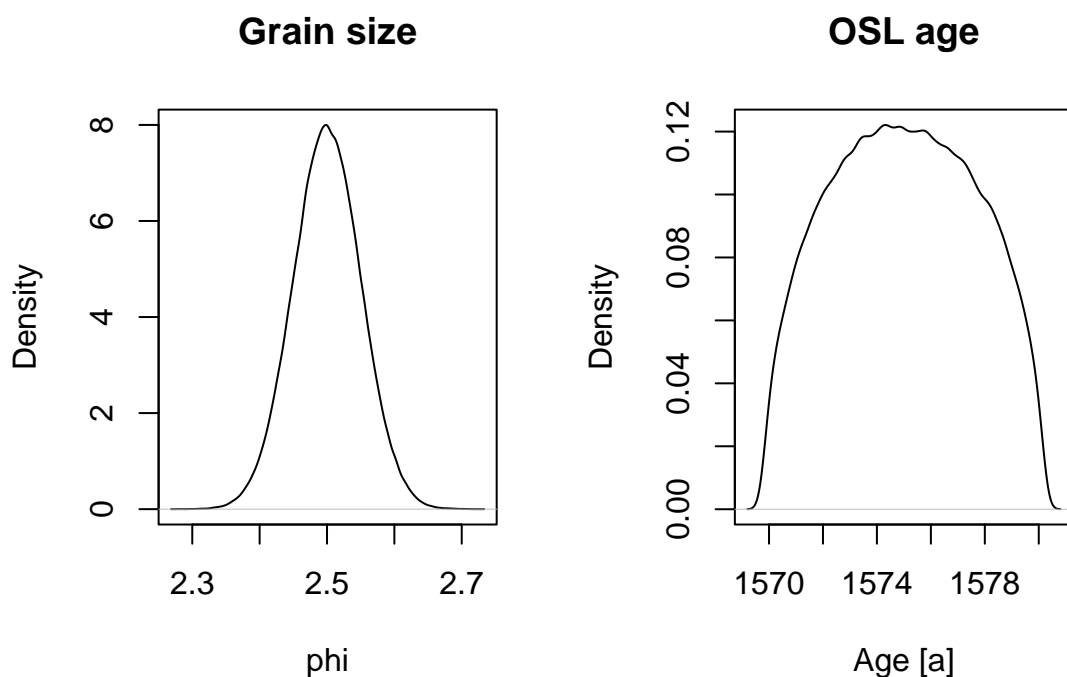
Now, we can take a sample from this sediment section at 2 m depth, plot the distribution shape of its grain size, and prepare aliquots of the grains. We assume here that the sample purely contains quartz grains of predominantly fine sand (mode around 2.5 phi).

```
## set seed for reproducibility
set.seed(2021)

## take a sample (diameter 1 cm, length 1 cm)
sample_osl <- make_Sample(
  book = book_osl,
  depth = 2,
  geometry = "cylinder",
  radius = 0.005,
  length = 0.01)
```

While the next step might be difficult for natural sediment profiles, with 'sandbox' we can have a look into the expected distribution of grain sizes and even the true deposition ages of all sampled grains.

```
## plot size distribution density of the sampled grains
par(mfrow = c(1,2))
plot(density(x = sample_osl$grainsize), main = "Grain size", xlab = "phi")
plot(density(x = sample_osl$age), main = "OSL age", xlab = "Age [a]")
```



In other words, our sample covers a grain-size range of 151-206  $\mu\text{m}$  and the expected depositional age range is (1570:1580) years.

In the next step, we can split our sample into single aliquots with multiple grains on it.

```
## make aliquots of sample
sample_osl_aliquots <- prepare_Aliquot(
  sample = sample_osl,
  diameter = 15)
```

The resulting preparation step yields 73 aliquots of 15 mm diameter. This would be a quite wealthy yield for an empirical scientist. The 15 mm diameter was chosen for illustrative reasons and to reduce the computation time, it does not necessarily match typically used aliquot discs in luminescence laboratories.

Now, we are ready to simulate the OSL measurements using the luminescence model parameters. The measurements use the capabilities of the package 'RLumModel', specifically the function `RLumModel::model_LuminescenceSignals()`. Other protocols may be added to the package during subsequent development stages. The 'RLumModel' functionality has been wrapped into a convenience function of 'sandbox' called `measure_SAR_OSL()`. For details about the workflow in 'RLumModel' see the package manual and vignettes in Friedrich et al. (2021).

*Note: The here chosen protocol parameters are arbitrary, and for illustrative reasons only. It is up to the reader, you, to play with more parameters to simulate various effects.*

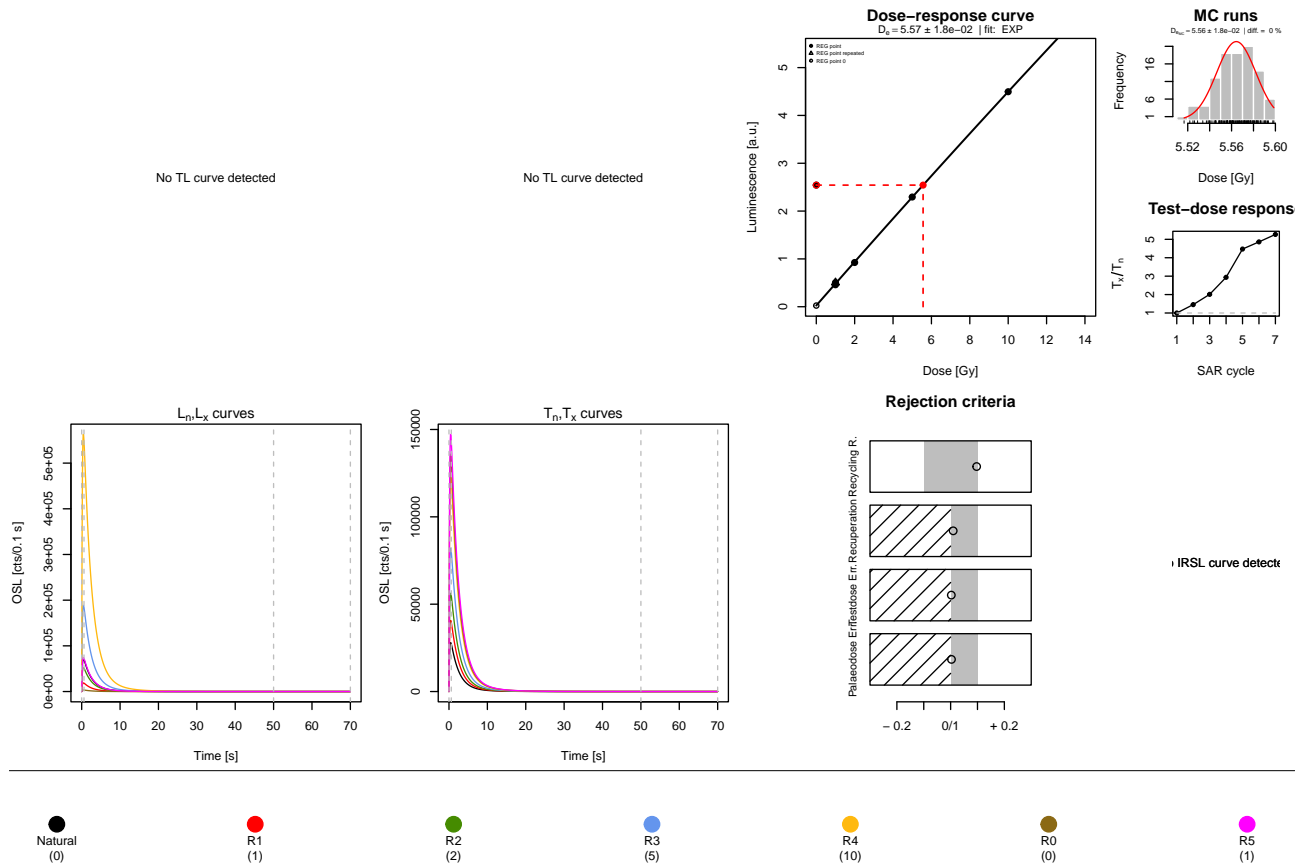
```
## define SAR measurement sequence
## the first reg dose is zero, the 'natural' dose
sequence <- list(
  RegDose = c(0, 1, 2, 5, 10, 0, 1),
  TestDose = 2,
  PH = 220,
  CH = 200,
  OSL_temp = 125,
  OSL_duration = 70)

## measure all aliquots in a row
sar_all <- measure_SAR_OSL(
  aliquot = sample_osl_aliquots,
  sequence = sequence,
  dose_rate = 0.1)
```

So far, so good. We have virtually measured many quartz grain aliquots, generated the necessary shine-down curves, and have created output that can be directly used in the R package 'Luminescence', just as if we had imported a BINX-file from a real luminescence reader. These imported data can now be analysed with standard procedures. The SAR-CWOSL function and the resulting OSL age distribution can be visualised.

Below, as example, we show the resulting shine-down curves for the first aliquot, to ensure that integration limits and dose points were set appropriately.

*Note: We did not simulate TL curves for the preheat, hence such curves can not be plotted.*



Now, after having confirmed the settings, we can run the analysis for the full dataset without plot and terminal output for all 73 aliquots.

```
# process measurement file
D_e <- Luminescence::analyse_SAR.CWOSL(
  object = sar_all,
  signal.integral.min = 1,
  signal.integral.max = 7,
  background.integral.min = 301,
  background.integral.max = 401,
  dose.points = dose_points,
  verbose = FALSE,
  plot = FALSE)
```

The luminescence age is calculated in the conventional way by dividing the equivalent dose ( $D_e$ , in Gy) by the natural dose rate ( $\dot{D}$  in Gy/a). For the example and for convenience reasons we assume the uncertainty of the source rate to be 0.

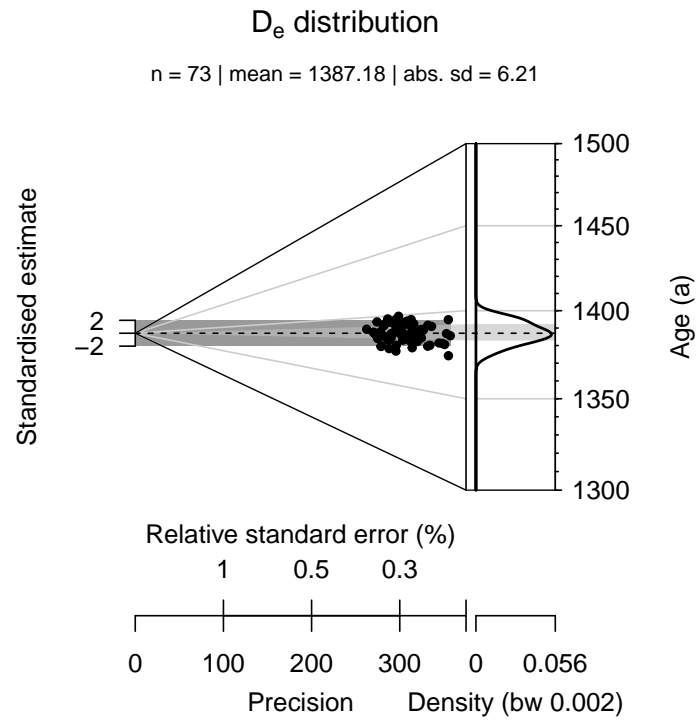
```
dose_rate_natural <- 0.004

age <- D_e$data[,c("De", "De.Error")] / dose_rate_natural
```

```
## Loading required package: Luminescence
```

```
## Welcome to the R package Luminescence version 0.9.19.9000-6 [Built: 2022-02-02 12:04:45 UTC]
## Fast component to slow component: 'Life is short!'
```

```
## plot age distribution
Luminescence::plot_AbanicoPlot(
  data = age,
  summary = c("n", "mean", "sd.abs"),
  zlab = "Age (a)",
  zlim = c(1300, 1500))
```



The results show that we ended up with an age range as somewhat expected. However, (1) the ages are younger than expected, perhaps the selected model and its parameters was not the best, and (2) the individual uncertainties are unrealistically small. Both issues leave a lot of room for further simulations, which are beyond the scope of this document.

## 8 Repository of the code used to generate the article figures

The R code in the following sections is just a one to one copy of the material used to produce the figures in the main article. The code is provided here for the sake of transparency and reproducibility. It is not intended to be a tutorial-like document. In addition, some of the figures produced in this section have been modified later, to homogenise its layout.

### 8.1 Summary of the Gleina section parameterisation

```
## load the measurement data of the Gleina loess section
```

```
X <- read.table(  
  file = "gleina_interpolated.txt",  
  sep = "\t",  
  header = TRUE)
```

```
## convert depth from cm to m
```

```
X$depth_int <- X$depth_int / 100
```

```
## get number of samples in interpolated data set
```

```
n <- nrow(X)
```

```
## create empty rule book
```

```
gleina <- get_RuleBook(book = "empty")
```

```
## add a further population
```

```
gleina <- add_Population(  
  book = gleina,  
  populations = 2)
```

```
## assign rule definitions to lists
```

```
depth <- list(X$depth_int)
```

```
age <- list(X$age_int)
```

```
## add age definition
```

```
gleina <- set_Rule(  
  book = gleina,  
  parameter = "age",  
  value = age,  
  depth = depth)
```

```
## assign rule definitions to lists
```

```
em_scores <- list(  
  list(X$EM_1),  
  list(X$EM_2),  
  list(X$EM_3))
```

```
## add population contribution with depth
```

```
gleina <- set_Rule(  
  book = gleina,  
  parameter = "population",  
  value = em_scores,  
  depth = depth)
```

```
## define population's grain-size distributions
```

```
EM_gsd <- list(  
  list(mean = rep(6.38, n),  
    sd = rep(0.9, n)),  
  list(mean = rep(4.69, n),  
    sd = rep(0.5, n)),  
  list(mean = rep(4.29, n),
```

```

    sd = rep(0.5, n)))

gleina <- set_Rule(
  book = gleina,
  parameter = "grainsize",
  value = EM_gsd,
  depth = depth)

## define packing density distributions
EM_packing <- list(
  list(mean = rep(0.7, n),
    sd = rep(0.01, n)),
  list(mean = rep(0.6, n),
    sd = rep(0.01, n)),
  list(mean = rep(0.5, n),
    sd = rep(0.01, n)))

gleina <- set_Rule(
  book = gleina,
  parameter = "packing",
  value = EM_packing,
  depth = depth)

## define specific grain density distributions
EM_density <- list(
  list(mean = rep(2.65, n),
    sd = rep(0.01, n)),
  list(mean = rep(2.65, n),
    sd = rep(0.01, n)),
  list(mean = rep(2.65, n),
    sd = rep(0.01, n)))

gleina <- set_Rule(
  book = gleina,
  parameter = "density",
  value = EM_density,
  depth = depth)

```

## 8.2 Test effect of sample container geometry on age uncertainty

```

## set random seed for reproducibility
set.seed(2021)

## cylinder
sample_cylinder <- make_Sample(
  book = gleina,
  depth = 5,
  geometry = "cylinder",
  radius = 0.005,
  length = 0.001)

## cubiod
sample_cube <- make_Sample(
  book = gleina,
  depth = 5,
  geometry = "cuboid",
  height = 0.01,
  width = 0.01,
  length = 0.0008)

```

```

## larger cubiod with
sample_cuboid_2 <- make_Sample(
  book = gleina,
  depth = 5,
  geometry = "cuboid",
  height = 0.005,
  width = 0.02,
  length = 0.0008)

sample_cuboid_4 <- make_Sample(
  book = gleina,
  depth = 5,
  geometry = "cuboid",
  height = 0.0025,
  width = 0.04,
  length = 0.0008)

sample_cuboid_8 <- make_Sample(
  book = gleina,
  depth = 5,
  geometry = "cuboid",
  height = 0.00125,
  width = 0.08,
  length = 0.0008)

par(mfrow = c(3,2))

## set list with objects to plot
o <- list(
  cylinder = sample_cylinder,
  cube = sample_cube,
  cuboid_2 = sample_cuboid_2,
  cuboid_4 = sample_cuboid_4,
  cuboid_8 = sample_cuboid_8)

for (i in names(o))
  hist(x = o[[i]]$age, main = i)

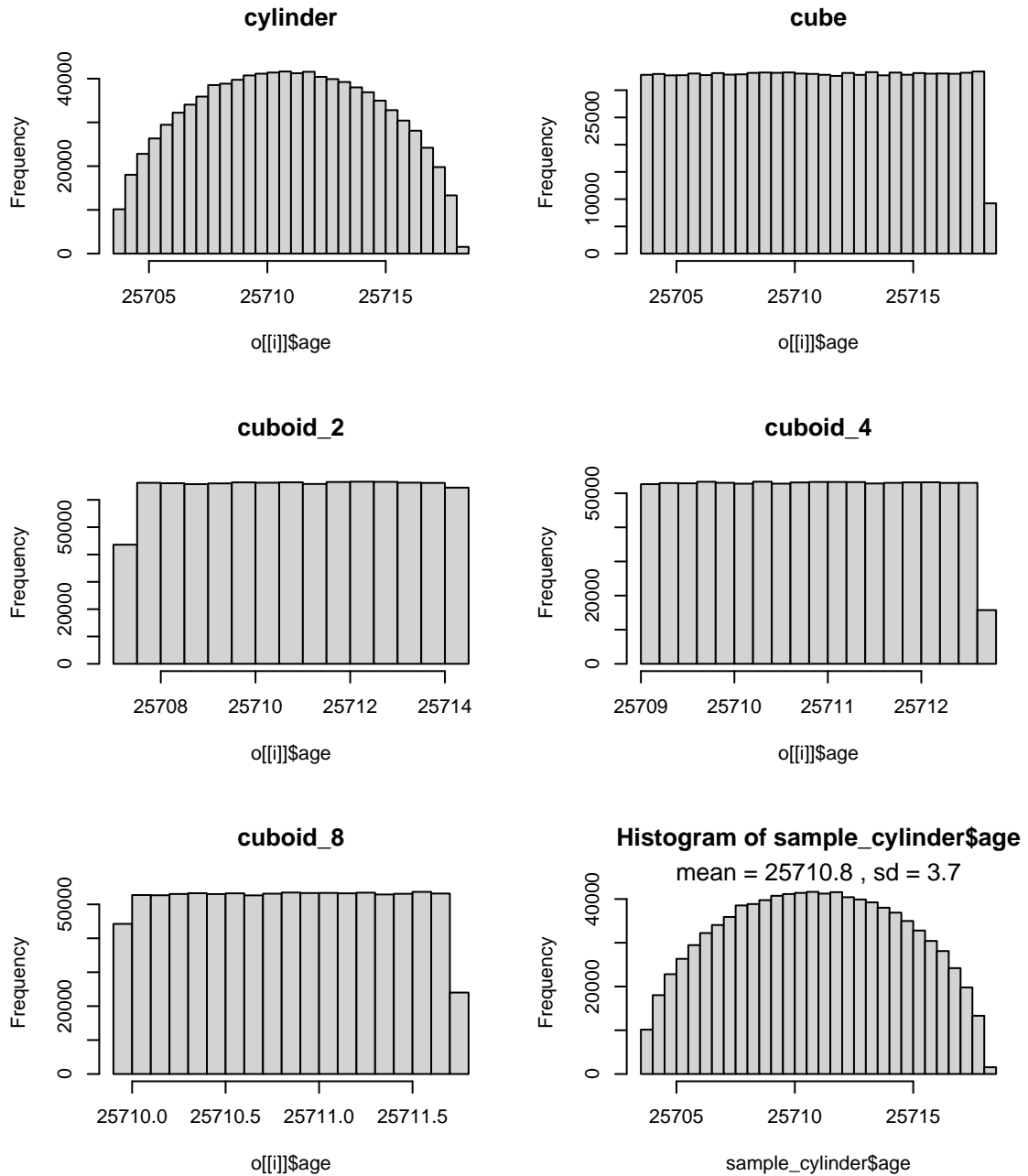
## calculate statistical values
df <- data.frame(
  SAMPLE = names(o),
  MEAN = vapply(o, function(x) mean(x$age), numeric(1)),
  SD = vapply(o, function(x) sd(x$age), numeric(1)))

hist(x = sample_cylinder$age)

mtext(side = 3, paste(
  "mean =", round(mean(sample_cylinder$age), 1),
  ", sd =", round(sd(sample_cylinder$age), 1)
), cex = 0.8)

```





```
knitr::kable(df)
```

	SAMPLE	MEAN	SD
cylinder	cylinder	25710.83	3.6554225
cube	cube	25710.83	4.2264061
cuboid_2	cuboid_2	25710.83	2.1115943
cuboid_4	cuboid_4	25710.83	1.0556037
cuboid_8	cuboid_8	25710.83	0.5275774

### 8.3 Test effect of sample container size and deposition rate on age uncertainty

```
##set random seed for reproducibility
set.seed(2021)
```

```
## set parameters
```

```
depth_sample <- seq(from = 10, to = 1, by = -1)
diameter_sample <- c(0.005, 0.01, 0.02, 0.03, 0.05, 0.07, 0.1, 0.2, 0.3, 0.5)
length_sample <- 1e-08 / (pi * (diameter_sample / 2)^2)
```

```
deposition_rate <- diff(depth_fill[[1]]) / diff(age_fill[[1]]) * 1000
```

```

parameter_set <- data.frame(
  depth = rep(depth_sample, length(diameter_sample)),
  diam = rep(diameter_sample, each = length(depth_sample)),
  length = rep(length_sample, each = length(depth_sample)))

age_set <- lapply(1:length(sample_set), function(i) {
  sample_i <- make_Sample(
    book = gleina,
    depth = parameter_set$depth[i],
    geometry = "cylinder",
    radius = parameter_set$diam[i] / 2,
    length = parameter_set$length[i])
  sample_i$age
})

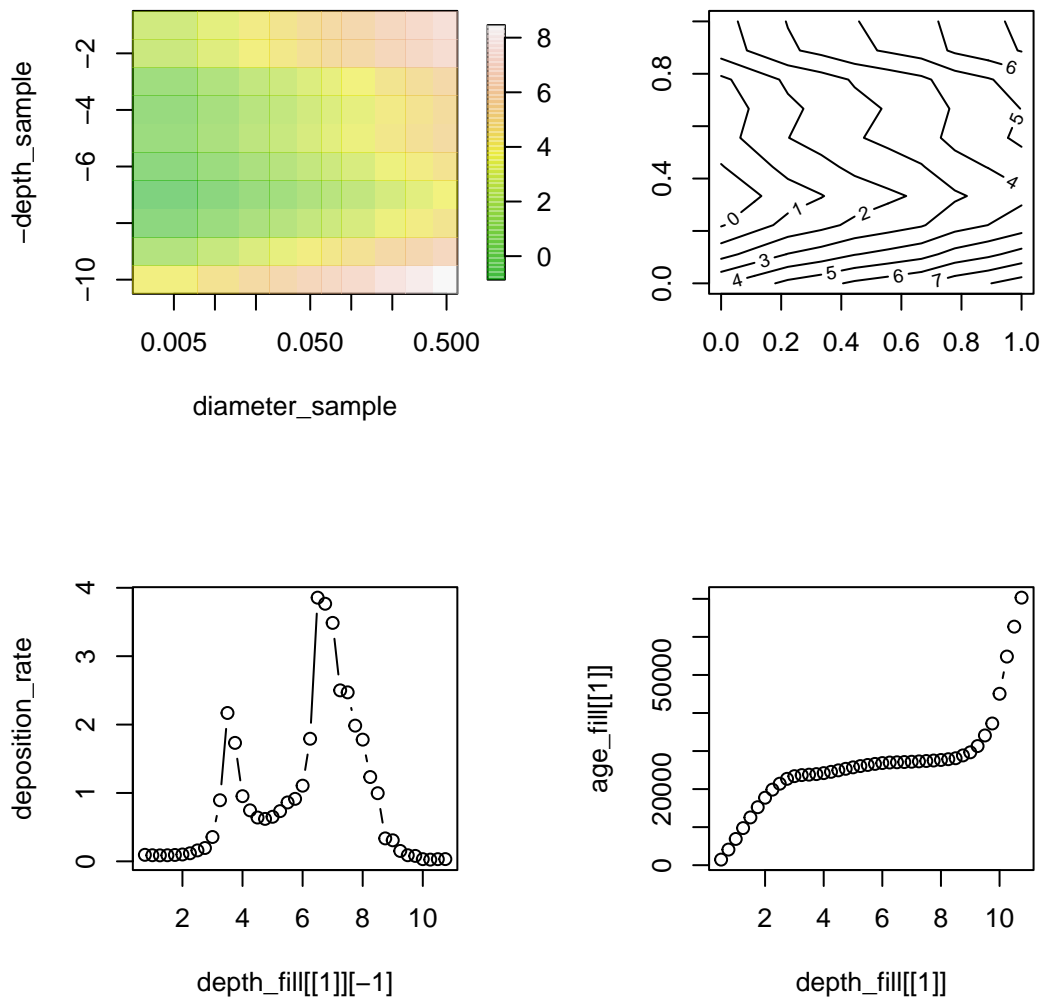
age_scatter_abs <- vapply(X = age_set, FUN = sd, numeric(1))
age_scatter_rel <- vapply(X = age_set, FUN = function(x) {
  sd(x) / mean(x) * 100}, numeric(1))

age_scatter_abs_mat <- matrix(unlist(age_scatter_abs), ncol = 10)
age_scatter_rel_mat <- matrix(unlist(age_scatter_rel), ncol = 10)

par(mfrow = c(2, 2))
fields::image.plot(
  x = diameter_sample,
  y = -depth_sample,
  z = log(t(age_scatter_abs_mat)),
  log = "x",
  col = adjustcolor(terrain.colors(200), 0.5))
contour(x = log(t(age_scatter_abs_mat)))

plot(x = depth_fill[[1]][-1], y = deposition_rate, type = "b")
plot(x = depth_fill[[1]], y = age_fill[[1]], type = "b")

```



#### 8.4 Test effect of size-dependent age inheritance

```
## copy rule book
gleina_2 <- gleina

## add inheritance rule
gleina_2 <- add_Rule(
  book = gleina_2,
  name = "inherited",
  group = "specific",
  type = "uniform")

## assign rule definitions to lists
inherited <- list(
  list(min = rep(0, n),
       max = rep(200, n)),
  list(min = rep(0, n),
       max = rep(200, n)),
  list(min = rep(0, n),
       max = rep(5000, n)))

## add population contribution with depth
gleina_2 <- set_Rule(
  book = gleina_2,
  parameter = "inherited",
  value = inherited,
  depth = depth_fill)
```

```

## sample the section
depth_2 <- seq(from = 1, to = 10, by = 0.5)
sample_set_2 <- vector(
  mode = "list",
  length = length(depth_2))

for (i in 1:length(sample_set_2)) {
  sample_set_2[[i]] <- make_Sample(
    book = gleina_2,
    depth = depth_2[i],
    geometry = "cylinder",
    radius = 0.01,
    length = 0.00005)
}

## make sieve fractions, commonly used limits
set_2_080_200 <- lapply(
  X = sample_set_2,
  FUN = prepare_Sieving,
  interval = convert_units(mu = c(80, 200)))

set_2_004_011 <- lapply(
  X = sample_set_2,
  FUN = prepare_Sieving,
  interval = convert_units(mu = c(4, 11)))

## make sieve fractions, EMMA-based limits
set_2_em1 <- lapply(
  X = sample_set_2,
  FUN = prepare_Sieving,
  interval = convert_units(mu = c(144, 89)))

set_2_em2 <- lapply(
  X = sample_set_2,
  FUN = prepare_Sieving,
  interval = convert_units(mu = c(23, 29)))

set_2_em3 <- lapply(
  X = sample_set_2,
  FUN = prepare_Sieving,
  interval = convert_units(mu = c(3, 7)))

## get mean true ages
age_080_200 <- unlist(x = lapply(X = set_2_080_200, FUN = function(x) {mean(x$age)}))
age_004_011 <- unlist(x = lapply(X = set_2_004_011, FUN = function(x) {mean(x$age)}))
age_em1 <- unlist(x = lapply(X = set_2_em1, FUN = function(x) {mean(x$age)}))
age_em2 <- unlist(x = lapply(X = set_2_em2, FUN = function(x) {mean(x$age)}))
age_em3 <- unlist(x = lapply(X = set_2_em3, FUN = function(x) {mean(x$age)}))

## get mean inherited ages
inh_080_200 <- unlist(x = lapply(X = set_2_080_200, FUN = function(x) {mean(x$inherited)}))
inh_004_011 <- unlist(x = lapply(X = set_2_004_011, FUN = function(x) {mean(x$inherited)}))
inh_em1 <- unlist(x = lapply(X = set_2_em1, FUN = function(x) {mean(x$inherited)}))
inh_em2 <- unlist(x = lapply(X = set_2_em2, FUN = function(x) {mean(x$inherited)}))
inh_em3 <- unlist(x = lapply(X = set_2_em3, FUN = function(x) {mean(x$inherited)}))

## generate plot of classic coarse grain ages
par(mfcol = c(1, 2))
plot(x = age_080_200,
     y = -depth_2,

```

```

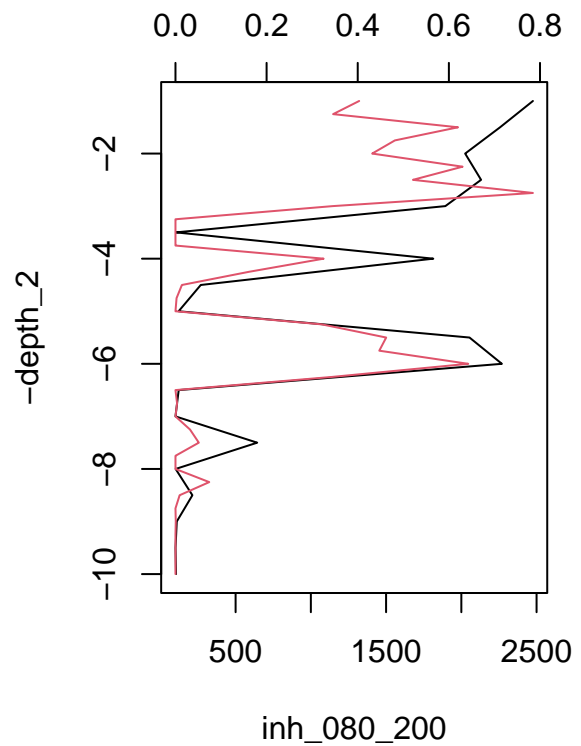
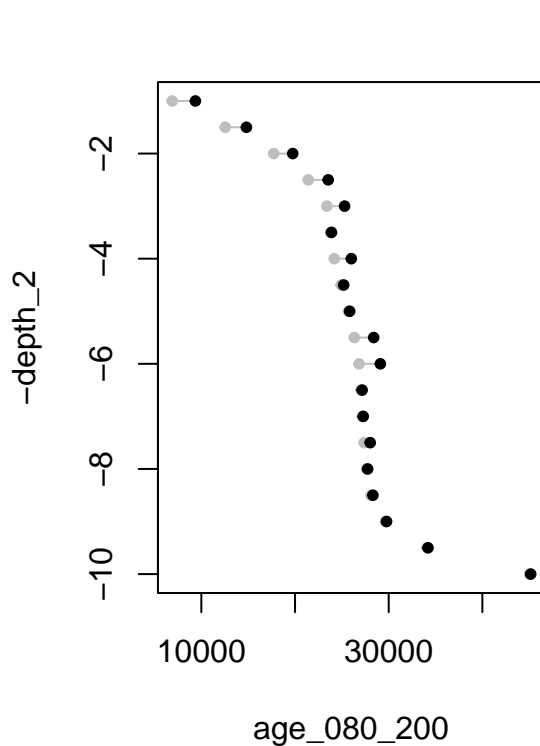
    col = "grey",
    pch = 20)
for (i in 1:length(age_080_200)) {
  lines(x = c(age_080_200[i],
              age_080_200[i] + inh_080_200[i]),
        y = c(-depth_2[i], -depth_2[i]),
        col = "grey")
}

points(
  x = age_080_200 + inh_080_200,
  y = -depth_2,
  col = 1,
  pch = 20)

plot(
  x = inh_080_200,
  y = -depth_2,
  type = "l")

par(new = TRUE)
plot(
  x = X$EM_3[3:39],
  y = -X$depth_int[3:39],
  type = "l",
  col = 2,
  ann = FALSE,
  axes = FALSE)
axis(side = 3)

```



```

## generate plot of classic fine grain ages
par(mfcol = c(1, 2))
plot(
  x = age_004_011,
  y = -depth_2,
  col = "grey",
  pch = 20)

```

```

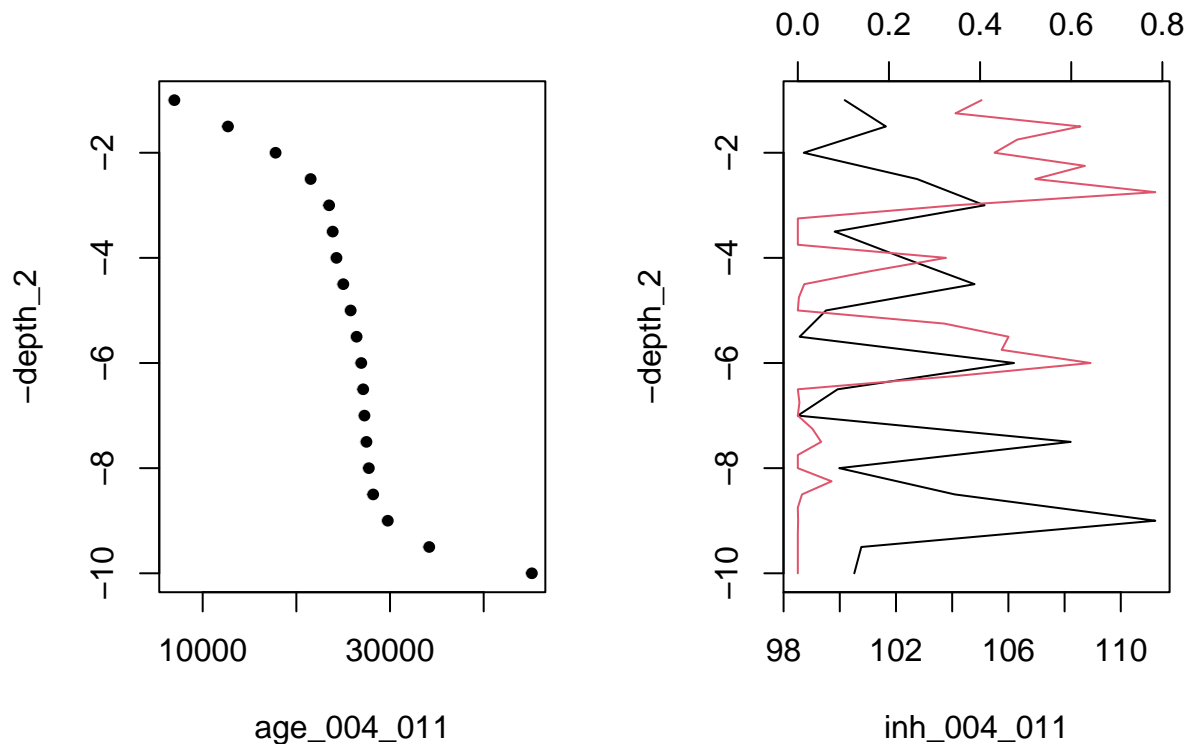
for (i in 1:length(age_004_011)) {
  lines(x = c(age_004_011[i],
              age_004_011[i] + inh_004_011[i]),
        y = c(-depth_2[i], -depth_2[i]),
        col = "grey")
}

points(
  x = age_004_011 + inh_004_011,
  y = -depth_2,
  col = 1,
  pch = 20)

plot(
  x = inh_004_011,
  y = -depth_2,
  type = "l")

par(new = TRUE)
plot(
  x = X$EM_3[3:39],
  y = -X$depth_int[3:39],
  type = "l",
  col = 2,
  ann = FALSE,
  axes = FALSE)
axis(side = 3)

```



```

## generate plot of EM1 ages
par(mfcol = c(1, 2))
plot(x = age_em1,
     y = -depth_2,
     col = "grey",
     pch = 20)
for (i in 1:length(age_em1)) {
  lines(x = c(age_em1[i], age_em1[i] + inh_em1[i]),

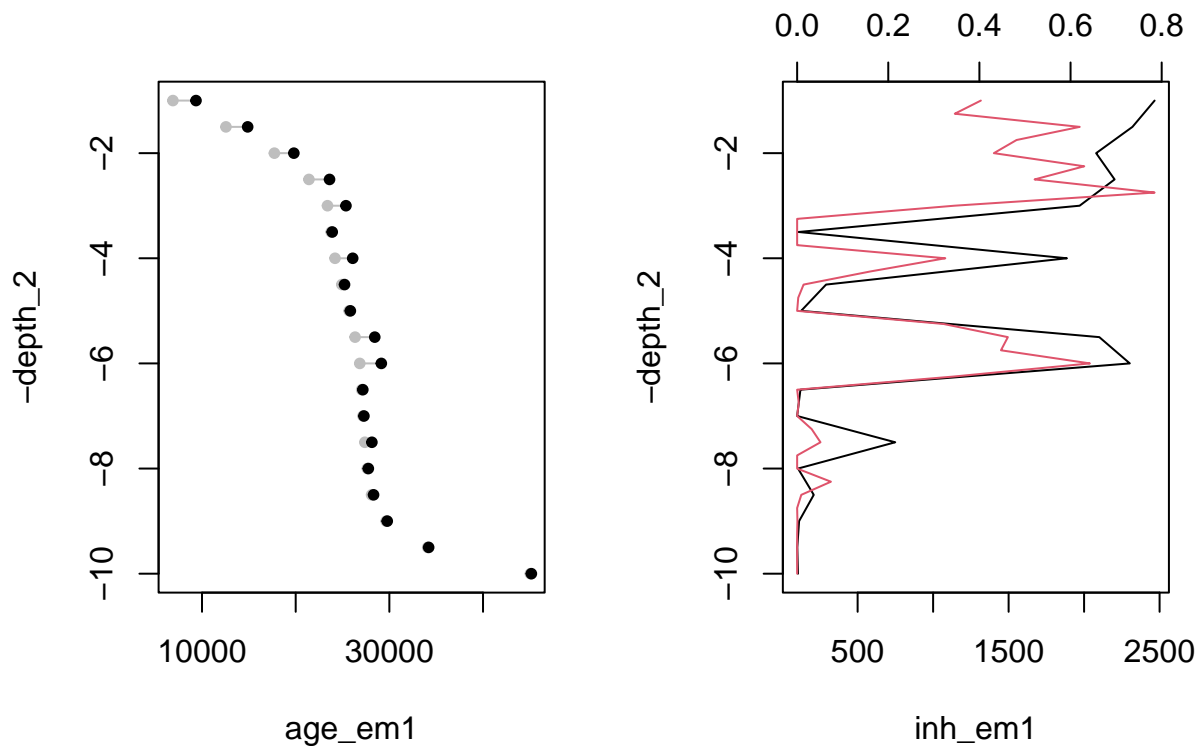
```

```

        y = c(-depth_2[i], -depth_2[i]),
        col = "grey")
}
points(x = age_em1 + inh_em1,
       y = -depth_2,
       col = 1,
       pch = 20)

plot(x = inh_em1,
     y = -depth_2,
     type = "l")
par(new = TRUE)
plot(x = X$EM_3[3:39],
     y = -X$depth_int[3:39],
     type = "l",
     col = 2,
     ann = FALSE,
     axes = FALSE)
axis(side = 3)

```



```

## generate plot of EM2 ages
par(mfcol = c(1, 2))
plot(x = age_em2,
     y = -depth_2,
     col = "grey",
     pch = 20)
for (i in 1:length(age_em2)) {
  lines(x = c(age_em2[i], age_em2[i] + inh_em2[i]),
       y = c(-depth_2[i], -depth_2[i]),
       col = "grey")
}
points(x = age_em2 + inh_em2,
       y = -depth_2,
       col = 1,
       pch = 20)

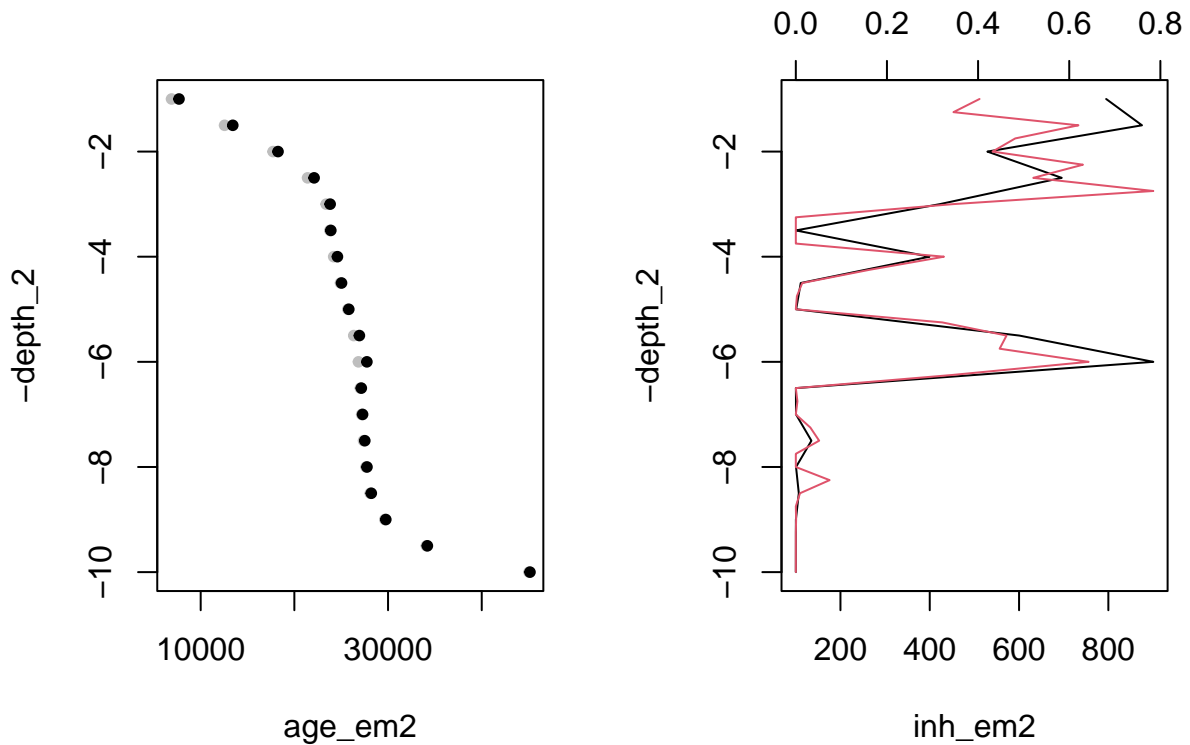
plot(x = inh_em2,

```

```

    y = -depth_2,
    type = "l")
par(new = TRUE)
plot(x = X$EM_3[3:39],
     y = -X$depth_int[3:39],
     type = "l",
     col = 2,
     ann = FALSE,
     axes = FALSE)
axis(side = 3)

```



```

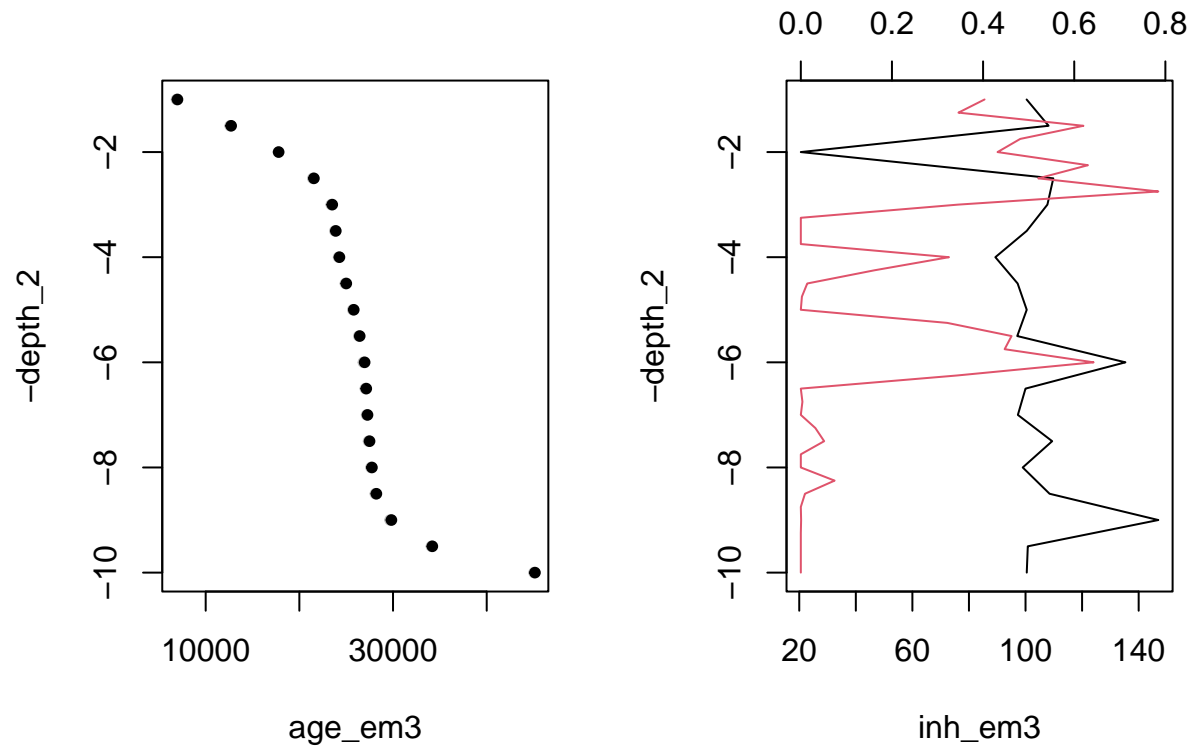
## generate plot of EM3 ages
par(mfcol = c(1, 2))
plot(x = age_em3,
     y = -depth_2,
     col = "grey",
     pch = 20)
for (i in 1:length(age_em3)) {
  lines(x = c(age_em3[i], age_em3[i] + inh_em3[i]),
       y = c(-depth_2[i], -depth_2[i]),
       col = "grey")
}
points(x = age_em3 + inh_em3,
      y = -depth_2,
      col = 1,
      pch = 20)

plot(x = inh_em3,
     y = -depth_2,
     type = "l")
par(new = TRUE)
plot(x = X$EM_3[3:39],
     y = -X$depth_int[3:39],
     type = "l",
     col = 2,
     ann = FALSE,
     axes = FALSE)

```



```
axis(side = 3)
```



## References

- Bailey, R. M. (2001). Towards a general kinetic model for optically and thermally stimulated luminescence of quartz. *Radiation Measurements*, 33(1), 17–45. [https://doi.org/10.1016/s1350-4487\(00\)00100-1](https://doi.org/10.1016/s1350-4487(00)00100-1)
- Dietze, E., & Dietze, M. (2019). Grain-size distribution unmixing using the r package EMMAgeo, 68, 29–46. <https://doi.org/10.5194/egqsj-68-29-2019>
- Friedrich, J. (2018). *Modelling quartz luminescence signal dynamics relevant for dating and dosimetry* ({PhD} thesis). University of Bayreuth, Bayreuth, Germany.
- Friedrich, J., Kreutzer, S., & Schmidt, C. (2016). Solving ordinary differential equations to understand luminescence: 'RLumModel' an advanced research tool for simulating luminescence in quartz using R. *Quaternary Geochronology*, 35(C), 88–100. <https://doi.org/10.1016/j.quageo.2016.05.004>
- Friedrich, J., Kreutzer, S., & Schmidt, C. (2021). *RLumModel: Solving ordinary differential equations to understand luminescence*. Retrieved from <https://CRAN.R-project.org/package=RLumModel>
- Kreutzer, S., Schmidt, C., Fuchs, M. C., Dietze, M., Fischer, M., & Fuchs, M. (2012). Introducing an r package for luminescence dating analysis. *Ancient TL*, 30(1), 1–8.
- Krumbein, E. A. W. C. (1937). The sediments of Barataria Bay. *Journal of Sedimentary Petrology*, 7(1), 1–15. <https://doi.org/10.1306/D4268F8B-2B26-11D7-8648000102C1865D>
- Meszner, S., Fuchs, M., & Faust, D. (2011). Loess-palaeosol-sequences from the loess area of saxony (germany). *Quaternary Science Journal*, 60, 47–65. <https://doi.org/10.3285/eg.60.1.03>
- Sheldon, N. D., & Retallack, G. J. (2001). Equation for compaction of paleosols due to burial. *Geology*, 29(3), 247–250. [https://doi.org/10.1130/0091-7613\(2001\)029%3C0247:EFCOPD%3E2.0.CO;2](https://doi.org/10.1130/0091-7613(2001)029%3C0247:EFCOPD%3E2.0.CO;2)
- Zech, M., Kreutzer, S., Zech, R., Goslar, T., Meszner, S., McIntyre, C., et al. (2017). Comparative 14-c and OSL dating of loess-paleosol sequences to evaluate post-depositional contamination of n-alkane biomarkers. *Quaternary Research*, 87(1), 180–189. <https://doi.org/10.1017/qua.2016.7>