Primer to the R package ‘Luminescence’

Christoph Burow

2018-02-16
Contents

Preface 5

1 General 7
  1.1 Before you start ......................................................... 7
  1.2 Loading libraries ...................................................... 7
  1.3 Help files ................................................................. 8
  1.4 Citation ................................................................ 8
  1.5 Dealing with paths ...................................................... 12
  1.6 Accessing Luminescence data ........................................ 12

2 Luminescence package 17
  2.1 Data import ................................................................. 17
  2.2 Signal analyses ............................................................. 24
  2.3 Equivalent dose calculation ............................................ 31
  2.4 Dose models ................................................................. 35
  2.5 Visualisation of DE distributions ....................................... 36

3 Scripting 43
  3.1 From .BIN(X) to AGE ..................................................... 44
  3.2 D_e(t) plot ................................................................. 45
  3.3 Minimum Dose Model .................................................... 46

Session Info 49
Preface

This document was written in preparation for an internal workshop of the Cologne Luminescence Laboratory (CLL). The main objective of this workshop was the introduction to the R package Luminescence (Kreutzer et al. 2012), to provide basic answers to commonly encountered problems in R and to give an impression on how the package can be leveraged to analyse luminescence data by using an exemplary data set. In contrast, this document is not intended to be an exhaustive guide on all the many included functions in the package. Neither is it a reference book to be used when facing a particular problem when working with the Luminescence, even though it may provide the correct answer at least implicitly.

The code in this document requires at least basic knowledge of R. There is a vast amount of commercially and freely available material to learn the basics of R. If you do not know what the <- operator does or what the differences between a data.frame, matrix or list are, it is highly recommended to catch up on the basics first before reading this document.

The intended outcome of this document is to achieve a basic understanding of data analyses with the Luminescence package. To facilitate this goal the document is structured in a (hopefully) relatable order, starting with importing raw measurement data to R and ending with calculating an age with DRAC (Durcan et al. 2015). The individual sections build on each other and should thus be read in correct order from start to finish. This is except for the last section on scripting, where code blocks are self-contained and do not rely on any previous code.
Chapter 1

General

1.1 Before you start

If you use R and the Luminescence only occasionally, ask yourself the following questions before any coding:

- When was the last time I updated R?
- When was the last time I updated the RStudio IDE?
- When was the last time I updated the Luminescence package?

If you don’t remember the answer to all of these questions, it is probably for the best to update everything!

This is the highly suggested order of doing so:

1. Download and install R from https://cran.r-project.org/bin/windows/base/
2. Download and install RStudio from https://www.rstudio.com/products/rstudio/download/
3. Install/update the Luminescence package by running the following command in the R console:
    `install.packages("Luminescence")`

Depending on how outdated your previous installation of R was the installation of the Luminescence package might take a little longer than usual. If there was a major R update (e.g., from v3.4.3 to 3.5) all packages need to be re-installed, but the running the above command takes care of downloading and installing all required dependencies.

**WARNING:** Do not update the Luminescence package before updating R itself! Each version of the Luminescence package requires a minimum version of R, so if you update the package while still running an outdated R version, it will also only install an older version of the Luminescence package!

1.2 Loading libraries

Functions in R are bundled in and so-called “packages”. A basic installation of R already comes with a certain collection of base packages, which are separately stored in the “System Library”. No matter how fundamental or basic the function, it is always part of a package, even though often not immediately realised. For example, the common function to print something to the R console `print()` is part of the base package (see topmost line in `?print`, which states “print {base}”).

When starting R (or rather RStudio) all these base functions like `print()` or `sum()` are immediately available to the user. This is because all packages in the System Library are automatically loaded when starting R. A great advantage of R, however, is that its functionality can be greatly expanded by sharing and including custom packages. After installing a package (like e.g. the Luminescence package) its functions are not
immediately available to the user when starting R. When we call a function of a package that is not yet loaded we get an error message like the following:

calc_CentralDose()

## Error in is.data.frame(x): argument "data" is missing, with no default

Instead, we have to explicitly load the package first so that R can actually find the function. The common way to do this is by using the function library(), which only requires the name of the package to be loaded.

library(Luminescence)

Explicitly loading a package is always necessary after starting R, unless the package was installed in the System Library (which is not the default behaviour and another topic for another time).

Occasionally, you will also see the function require() used to load a library. While both functions essentially do the same thing, there are fine nuances to the differences between both functions that particularly favor the use of the library() function.

### 1.3 Help files

It is nearly impossible to remember all the function names and all their arguments by heart. Fortunately enough, every function and argument in R is required to be documented in dedicated help files. The user is highly encouraged to always refer to the documentation in case of doubt on how to use this function.

There are multiple ways to access a specific help file. The safest way to do so is by calling a function with a preceding question mark ?.

?mean()

If you are using the RStudio IDE (most likely), the is a more convenient way of accessing a specific help file. When typing the function’s name in the R console or in a script file (.R, .Rmd, …) just left click on the name and press F1.

Sometimes it might happen that you do not know the exact name of a function though. To quickly navigate to an R package’s content, ie. a list of its included functions, it is easiest to use the help() function.

help(package = "Luminescence")

Be aware to explicitly use the package argument and not call help("Luminescence"), which would be interpreted as help(topic = "Luminescence") as topic is the first argument of this function. This will then open the help file to the package itself, which, actually, is not a great help (unless you wanted to have a look at the list of authors…).

### 1.4 Citation

Why should we cite the used software at all? If you just asked yourself this question this guide kindly recommends reading the following article:

1.4. CITATION

1.4.1 Citing the package

Base R already provides a function to facilitate easy citation of packages. The function citation() generally only requires the argument package, which must be a character string with the name of the package. The function automatically creates a citation for the package itself and, if provided by the package authors, also lists all additional articles related to the package.

citation(package = "Luminescence")

##
## To cite the R package 'Luminescence' please use the first two
## entries, and apply the rest if applicable:
##
## Sebastian Kreutzer, Christoph Burow, Michael Dietze, Margret C.
## Fuchs, Christoph Schmidt, Manfred Fischer and Johannes Friedrich
## (2018). Luminescence: Comprehensive Luminescence Dating Data
## Analysis. R package version 0.8.1.
## https://CRAN.R-project.org/package=Luminescence
##
## Kreutzer S, Schmidt C, Fuchs MC, Dietze M, Fischer M and Fuchs M
## (2012). "Introducing an R package for luminescence dating
##
## Dietze M, Kreutzer S, Fuchs MC, Burow C, Fischer M and Schmidt C
## _Ancient TL_, *31*(1), pp. 11-18.
##
## Fuchs MC, Kreutzer S, Burow C, Dietze M, Fischer M, Schmidt C and
## Fuchs M (2015). "Data processing in luminescence dating analysis:
## An exemplary workflow using the R package 'Luminescence'."
## _Quaternary International_, *362*, pp. 8-13. doi:
## 10.1016/j.quaint.2014.06.034 (URL:
## http://doi.org/10.1016/j.quaint.2014.06.034).
##
## Uncertainty (IEU) model." _Ancient TL_, *33*(1), pp. 16-21.
##
## Dietze M, Kreutzer S, Burow C, Fuchs MC, Fischer M and Schmidt C
## (2016). "The abanico plot: visualising chronometric data with
## individual standard errors." _Quaternary Geochronology_, *31*, pp.
## 12-18. doi: 10.1016/j.quageo.2015.09.003 (URL:
##
## Mercier N, Kreutzer S, Christophe C, Guerin G, Guibert P, Lahaye
## in luminescence dating: The baSAR-model and its implementation in
##
## Kreutzer S, Burow C, Dietze M, Fuchs MC, Fischer M and Schmidt C
## (2017). "Software in the context of luminescence dating: status,
## concepts and suggestions exemplified by the R package
## 'Luminescence'." _Ancient TL_, *35*(2), pp. 1-11.
##
1 Actually, it can also be called empty, which then provides the citation for R itself
For the Luminescence package the first entry should be used to cite the package itself, at best in conjunction with the Kreutzer et al. (2012) article. All other articles usually refer to a specific function of a package and should only be cited if that function was explicitly used.

For the tech-savvy the citation list can also be printed with BibTeX markup:

```r
# To see these entries in BibTeX format, use 'print(<citation>,
# bibtex=TRUE)', 'toBibtex(.)', or set
# 'options(citation.bibtex.max=999)'.

For the Luminescence package the first entry should be used to cite the package itself, at best in conjunction with the Kreutzer et al. (2012) article. All other articles usually refer to a specific function of a package and should only be cited if that function was explicitly used.

For the tech-savvy the citation list can also be printed with BibTeX markup:

citations <- citation(package = "Luminescence")
toBibtex(citations)
```

```bibtex
@Manual{,
  title = {Luminescence: Comprehensive Luminescence Dating Data Analysis},
  author = {Sebastian Kreutzer and Christoph Burow and Michael Dietze and Margret C. Fuchs and Christoph Schmidt and Manfred Fischer and Johannes Friedrich},
  year = {2018},
  note = {R package version 0.8.1},
  url = {https://CRAN.R-project.org/package=Luminescence},
}

@Article{,
  title = {Introducing an R package for luminescence dating analysis},
  author = {Sebastian Kreutzer and Christoph Schmidt and Margret C. Fuchs and Michael Dietze and Manfred Fischer and Markus Fuchs},
  year = {2012},
  journal = {Ancient TL},
  volume = {30},
  number = {1},
  pages = {1-8},
}

@Article{,
  title = {A practical guide to the R package Luminescence},
  author = {Michael Dietze and Sebastian Kreutzer and Margret C. Fuchs and Christoph Burow and Manfred Fischer},
  year = {2013},
  journal = {Ancient TL},
  volume = {31},
  number = {1},
  pages = {11-18},
}

@Article{,
  title = {Data processing in luminescence dating analysis: An exemplary workflow using the R package 'Luminescence'},
  author = {Margret C. Fuchs and Sebastian Kreutzer and Christoph Burow and Michael Dietze and Manfred Fischer},
  year = {2015},
  journal = {Quaternary International},
  volume = {362},
  pages = {8-13},
  doi = {10.1016/j.quaint.2014.06.034},
}

@Article{,
  title = {A new R function for the Internal External Uncertainty (IEU) model},
  author = {Rachel K Smedley},
  journal = {Ancient TL},
  year = {2015},
}```
1.4. CITATION

```r
## volume = {33},
## number = {1},
## pages = {16-21},
## }
##
## @Article{
## title = {The abanico plot: visualising chronometric data with individual standard errors},
## author = {Michael Dietze and Sebastian Kreutzer and Christoph Burow and Margret C. Fuchs and Manfred Fischer and Christoph Schmidt},
## year = {2016},
## journal = {Quaternary Geochronology},
## volume = {31},
## pages = {12-18},
## doi = {10.1016/j.quageo.2015.09.003},
## }
##
## @Article{
## title = {Bayesian statistics in luminescence dating: The baSAR-model and its implementation in the R package 'Luminescence'},
## author = {Norbert Mercier and Sebastian Kreutzer and Claire Christophe and Guillaume Guerin and Pierre Guibert and Christelle Lahaye and Philippe Lanos and Anne Philippe and Chantal Tribolo},
## year = {2016},
## journal = {Ancient TL},
## volume = {34},
## number = {2},
## pages = {14-21},
## }
##
## @Article{
## title = {Software in the context of luminescence dating: status, concepts and suggestions exemplified by the R package 'Luminescence'},
## author = {Sebastian Kreutzer and Christoph Burow and Michael Dietze and Margret C. Fuchs and Manfred Fischer and Christoph Schmidt},
## year = {2017},
## journal = {Ancient TL},
## volume = {35},
## number = {2},
## pages = {1-11},
## }
```

1.4.2 Citing a specific function

Sometimes it might also be a good idea to cite a specific function. To facilitate this each help file in the `Luminescence` package (v0.7.0) includes a citation section.

So if you e.g. decided to make heavy use of the Kars et al. (2008) model using the `calc_Kars2008()` function, then open its help file via `?calc_Kars2008`, scroll all the way down to the “How to cite”-section and copy-paste the text.

As an example, this is how the citation for this function looks like:


Note that these entries are automatically generated when the R package is published, thus they should always be up to date (at least in theory...).
1.5 Dealing with paths

A common source of error when using a function to either read or write files from the harddisk is how directory and file paths are handled in R. Assuming a file named test.txt that is stored in the folder R under partition D: of our file system, the common notation to that file would be D:\R\text.txt. However, if we were to use this path as input for a function to import this file, R returns the following error:

## Error: '\R' is an unrecognized escape in character string starting "D:\R"

This is because the symbol \ (backslash) has a special meaning in R (and basically in all other programming languages). Without going into further detail, it must suffice to say that backslashes should be generally avoided; unless you want to invoke its special meaning (if you want to do that, you will also know about the details). So what do we do with our file path then?

Instead of using backslashes, R allows (or rather wants) us to use the common slash / instead. So instead of D:\R\text.txt we would use the string D:/R/text.txt. For the sake of completeness, D:\\R\\text.txt (note the double backslashes) would also work, because here we invoke the backslash’s special meaning on itself.

1.6 Accessing Luminescence data

One of the first things to learn in R is how to properly index the various data structures such as a data.frame, matrix or list. The usual approach is either one or a combination of the common operators $, [ or [[. A quick reminder:

```r
df <- data.frame(x = runif(4),
                 y = letters[1:4])
df$x
```

## [1] 0.8313321 0.5299408 0.3673145 0.7724166
df[1, 2]

## [1] a
## Levels: a b c d
df[[2]]

## [1] a b c d
## Levels: a b c d
df[["y"]]

## [1] a b c d
## Levels: a b c d

Most of the time when using the Luminescence package you will deal with a new kind of data structure called RLum objects. These objects were specifically designed to account for the structural complexity of luminescence data, which are not just a collection of signal and time values, but which also contain various meta information (e.g., temperature, heating rate, stimulation time, etc.). The advantage of implementing a new data structure in R is that all the input data, which come in all kinds of forms, can be standardised to a common format. This makes handling the data internally much more robust, but also more comprehensible for the user. The drawback, however, is that the user must learn the structure of the data object first.

---

See the documentation on `Quotes` for a list of escape sequences in R. If you so happen to have any of the listed escape sequences in your path character string, the resulting error may significantly differ from the shown error above.
One of the many example data sets shipped with the Luminescence package is an IRSAR.RF.Data named object of class RLum.Analysis. The RLum.Analysis class is sort of a container data structure designed to store the raw measurement data. The example data set contains two radiofluorescence curves, as can be seen from the console output.

The console output also tells us that these two curves are apparently objects of class RLum.Data.Curve. This is the data structure that contains the actual signal and time data.

The important thing to remember is that any RLum object should generally be accessed using the get_RLum() function rather than the common $[, ] or [[]. The get_RLum() works automatically detects the specific class of the RLum object and returns the desired values. If no specific value is given, it will always return a default value, which depends on the object itself. Let’s have a look at the internal structure of our example RLum.Analysis object.

```r
str(IRSAR.RF.Data)
```

```r
## Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## ..@ protocol : chr "IRSAR"
## ..@ records :List of 2
## ...$ :Formal class 'RLum.Data.Curve' [package "Luminescence"] with 7 slots
## .......@ recordType: chr "RF"
## .......@ curveType : chr NA
## .......@ data : num [1:524, 1:2] 0.175 1.773 3.314 4.574 6.311 ...
## .......- attr(*, "dimnames")=List of 2
## .......@ originator: chr NA
## .......@ info : list()
## .......@ .uid : chr "2016-01-30-10:36.0.980056973639876"
## .......@ .pid : chr "2016-01-30-10:38.0.625037926714867"
## ...$ :Formal class 'RLum.Data.Curve' [package "Luminescence"] with 7 slots
## .......@ recordType: chr "RF"
## .......@ curveType : chr NA
## .......@ data : num [1:524, 1:2] 0.377 1.696 3 4.196 5.463 ...
## .......- attr(*, "dimnames")=List of 2
## .......@ originator: chr NA
## .......@ info : list()
```
From the output we can see that this object has 6 slots, named:

- protocol
- records
- originator
- info
- .uid
- .pid

Apart from the @records slot, all slots are either empty, have a NA value or contain a single character string (e.g., @protocol contains the string "IRSAR"). The @records slot, however, is a list with two elements. These two elements are objects of class RLum.Data.Curve. The internal structure of each of these elements is very similar to that of the RLum.Analysis object. Instead of a @records slot they do have a @data slot, which contains a matrix with two columns for the stimulation time and signal intensity. If we wanted to obtain the first RF curve object while using the standard operators we would need the following code:

```r
IRSAR.RF.Data@records[[1]]
```

```
## [RLum.Data.Curve-class]
## recordType: RF
## curveType: NA
## measured values: 5
## .. range of x-values: 0.1747448 6.311132
## .. range of y-values: 1423.15 1437.8
## additional info elements: 0
```

This certainly works, but the more nested the object the more confusing it gets. With the `get_RLum()` function it looks a bit cleaner:

```r
RF <- get_RLum(IRSAR.RF.Data)
RF[[1]]
```

```
## [RLum.Data.Curve-class]
## recordType: RF
## curveType: NA
## measured values: 5
## .. range of x-values: 0.1747448 6.311132
## .. range of y-values: 1423.15 1437.8
## additional info elements: 0
```

If we want to access the raw measurement values using standard operators we need the following code:

```r
IRSAR.RF.Data@records[[1]]@data
```

```
## V1 V2
## [1,] 0.1747448 1425.940
## [2,] 1.7729883 1437.800
## [3,] 3.3144199 1427.100
## [4,] 4.5739196 1423.150
## [5,] 6.3111318 1426.233
```
Alternativaly, we can do another call to `get_RLum()`.

```r
get_RLum(RF[[1]])
```

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1747448</td>
<td>1425.940</td>
</tr>
<tr>
<td>2</td>
<td>1.7729883</td>
<td>1437.800</td>
</tr>
<tr>
<td>3</td>
<td>3.3144199</td>
<td>1427.100</td>
</tr>
<tr>
<td>4</td>
<td>4.5739196</td>
<td>1423.150</td>
</tr>
<tr>
<td>5</td>
<td>6.3111318</td>
<td>1426.233</td>
</tr>
</tbody>
</table>

While it might look like we did not save much code (if at all in this example), the power of `get_RLum()` may become more obvious when dealing with larger objects and/or in more elaborate scripts. One of the few features of `get_RLum()` is that it is vectorised, meaning that it automatically operates on a list of objects (so we do not need to bother with complicated `*apply` functions).
Chapter 2

Luminescence package

2.1 Data import

2.1.1 Importing text files

With only (very) few exceptions every function in the ‘Luminescence’ package requires some sort of data provided by the user. The easiest data structure and file format would be to provide a text file with e.g. a set of palaeodose values and their corresponding errors. A text file usually, but not necessarily has a *.txt file ending. In case of the Analyst program the exported summary file of accepted aliquots/grains has the file exention *.ANR, but is functionally no different to a common *.txt file.

There are many different functions to import text files into R and it is difficult to recommend a single function and/or a single set of parameters of a function to account for the many different shapes of text files. To avoid the associated frustration of the iterative process to import data from text files it is highly recommended to use the “Import Dataset” option provided in RStudio (Figure 1).

When clicking on “Import Dataset” it is usually recommended to select the “From text (readr)…” option, as it provides a bit more flexibility in terms of how the text file is allowed to be structured. When importing an .ANR file produced by the Analyst make sure to set “Skip” to 1, which means that the first line of the text file should not be considered for importing (here, the first line contains the filename). Then select “Comma” as delimiter and check “First Row as Names”. In the code preview window on the bottom right you will see the code that is run when finally clicking on “Import” (Figure 2).

When writing an R script it is obviously not feasible to import the data with the inbuilt dialogue option. As per the previous comment it is not possible to provide a general solution on how to import text data. As a recommendation, however, it is suggested to successfully import the data at least once with the dialogue option as shown above and then copying the preview code into the script. As long as the text file doesn’t change in structure or, if multiple files are analysed, all files follow the same internal structure it is straightforward to use the copied code. For reference a list of R internal functions to import data:

```
# text files
?read.delim
?read.delim2
?read.table

# CSV file (comma separated file)
?read.csv
?read.csv2

# import functions from external packages
```

17
CHAPTER 2. LUMINESCEENCE PACKAGE

Figure 2.1: RStudio provides a dedicated option to easily import data from text files.

Figure 2.2: Dialog of the “From text (readr)...” option in RStudio.
2.1.2 Importing a .BIN(X) file

In contrast to plain text files, *.bin(x) files produced by the SequenceEditor are stored in a binary format and are thus not human readable. This makes importing a measurement file not as straightforward and requires a custom function that is able to translate the binary data into a human readable format. Fortunately enough, the Luminescence package includes a function to properly import a *.bin(x) file according to the documentation provided by Risø.

The function that is required to import a *.bin(x) into R is named read_BIN2R(). The following sections will demonstrate how to import a single or multiple *.bin(x) files as well as extracting only a subset of the data (i.e., only a specified range of positions or only a certain luminescence curve).

2.1.2.1 Importing the whole BIN file

As indicated in the documentation (reminder: ?read_BIN2R) the function read_BIN2R() only requires on single argument, namely file. The user only has to provide a file path.

```r
file <- "D-CA1_100-150_2mm_a.BIN"
bin <- read_BIN2R(file = file, txtProgressBar = FALSE)
```

```r
## [read_BIN2R()]
## >> D-CA1_100-150_2mm_a.BIN
## >> 250 records have been read successfully!
print(bin)
```

```r
## [Risoe.BINfileData object]
## BIN/BINX version 3
## Object date: 200420, 210420
## User: alex
## System ID: 262
## Overall records: 250
## Records type: IRSL (n = 10)
## OSL (n = 140)
## RBR (n = 10)
## TL (n = 90)
## Position range: 1 : 10
## Grain range: 0 : 0
## Run range: 2 : 47
## Set range: 1 : 1
```

Note that we also used the set txtProgressBar to FALSE, which is one of the many other optional arguments of this function. The default value is TRUE and would, as the name implies, show a progress bar to inform the user about the progress of the import process.

In this example we have stored the content of the *.bin file in the variable bin. When printing the variable we are then shown a short summary of the imported data and e.g. see that the *.bin file contained 10 IRSL,
140 OSL, 10 RBR and 90 TL curves. We also see that the file contained luminescence data from positions 1 to 10. Depending on the subsequent analysis, however, we may want only a subset of the data. In the next subsections it will be demonstrated on how to extract the curves of only a couple of aliquots and how to restrict the data to OSL curves only.

### 2.1.2.2 Importing only a small subset of positions/grains

To import data of only a subset of aliquots we have to use the argument `position`, which accepts a single or a range of integer values.

```r
bin <- read_BIN2R(file = file, fastForward = FALSE, txtProgressBar = FALSE, position = 1:3)
```

From the summary we can quickly see that subsetting the *.bin file was successful.

### 2.1.2.3 Subsetting specific record types

Usually we are interested in only a specific type of record type (e.g., OSL curves for equivalent dose calculation). While it is not necessarily required to filter the records at this stage of the analyses already, it is certainly the cleanest approach.

To extract only a further subset of our imported *.bin file we can use the base R function `subset()`. Subsetting is straightforward, but we need to know the correct terms by which to subset, however. If you are not familiar with the metadata fields in the *.bin file it is highly recommended to look at the documentation of Risoe.BINfileData-class. The luminescence type is stored in the `LTYPE` field in the *.bin file. With a simple boolean expression we can then extract the records of the desired luminescence type.
2.1. DATA IMPORT

```r
bin_OSL <- subset(bin, LTYPE == "OSL")
bin_IRSL <- subset(bin, LTYPE == "IRSL")
bin_TL <- subset(bin, LTYPE == "TL")
```

If you used an invalid field name the function will return an error that lists all valid terms, which we can use to subset.

2.1.2.4 Convert .BIN(X) object to RLum object

Before we can use our imported *.bin file records, however, we need to convert the data into another data type. The function `read_BIN2R()` returns an object of class `Risoe.BINfileData` by default, which is a lossless 1:1 representation of the *.bin file's content. Without going too much into the intricacies of data type representations, consider the following two aspects to understand why another data type may be useful: a) data produced by a Freiberg Instruments Lexsyg device follow a different data format pattern, b) the record-by-record representation in *.bin files rather than a grain-by-grain or aliquot-by-aliquot grouping makes further processing of the data rather difficult.

The Luminescence package implements a common data type to represent luminescence data, defined by the so-called RLum-class. Fortunately enough, converting our *.bin file object to an RLum object is easy.

```r
rlum <- Risoe.BINfileData2RLum.Analysis(object = bin, txtProgressBar = FALSE)
print(rlum)
```

```text
[[1]]
[[[1]]]
[[[2]]]
[[[3]]]
```
2.1.3 Fast forward

Importing a .bin(x) file and converting it to an RLum.Analysis object can be achieved in a quicker way than described above. The aforementioned function read_BIN2R() provides the argument fastForward, which, when TRUE, directly returns an RLum.Analysis object instead of the intermediate Risoe.BINfileData object.

```r
rlum <- read_BIN2R(file = file, fastForward = TRUE, txtProgressBar = FALSE)
```

## number of records: 25
## .. : RLum.Data.Curve : 25
## .. .. : #1 OSL | #2 TL | #3 OSL | #4 OSL | #5 TL | #6 OSL | #7 OSL
## .. .. : #8 TL | #9 OSL | #10 OSL | #11 TL | #12 OSL | #13 OSL | #14 TL
## .. .. : #15 OSL | #16 OSL | #17 TL | #18 OSL | #19 OSL | #20 TL | #21 OSL
## .. .. : #22 TL | #23 IRSL | #24 TL | #25 RBR

We can see that the summary printed to the console changed, but arguably for the better. Now, all records are grouped by measurement positions and we can much more easily assess the structure of our measurement sequence. In more detail, the rlum object is a list of three RLum.Analysis-objects, which is the common type for all raw measurement data objects within the Luminescence package.
In the previous sections we learned how to extract only certain data with the `subset()` function, which we applied to a `Risoe.BINfileData` object. This function works for `RLum.Analysis` just as fine, however, so that there is only very little reason not to use the `fastForward` option. We do have to consider, however, that our `RLum.Analysis` is a list of x elements (number of aliquots), so we need to subset each element individually. This is best achieved using the `lapply()` function.

```r
rbr <- lapply(rlum, function(x) subset(x, LTYPE == "RBR"))[1]
```

# LM-OSL curve of the first aliquot

```r
plot(rbr)
```

![Graph showing LM-OSL curve of the first aliquot](image-url)
2.1.4 Combining .BIN(X) files

More often than not equivalent dose measurements for a single sample are spread across multiple *.bin files. In the previous examples we only considered one BIN(X) file, but the following example will show how to import a series of related BIN(X) files and how to combine them before further analyses.

In the example below, instead of hard-coding the file names we will use a more dynamic approach by using the function `list.files()`. As its name implies, `list.files()` returns a list of file names in a directory. Since we usually do have different types of files in a directory, we can (or rather have to) filter the data by providing a matching pattern. If we wanted to e.g. import all .bin(x) files, we could use `pattern = ".bin"`. Here, we only want to import the files of sample CA-2, which is why we use `pattern = "CA2"`. The important part to remember is that the pattern must not be ambiguous.

```r
## Dynamic list of filenames that match a certain pattern
files <- list.files(path = ".", pattern = "CA2", full.names = TRUE, include.dirs = FALSE, ignore.case = TRUE)

## Import all files at once
rlums <- lapply(files, function(file) read_BIN2R(file, fastForward = TRUE, txtProgressBar = FALSE, verbose = FALSE))
str(rlums, max.level = 1)
## List of 10
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots
## $: Formal class 'RLum.Analysis' [package "Luminescence"] with 6 slots

## Flatten the list
rlums <- unlist(rlums)
str(rlums, max.level = 0)
## List of 10
```

We now have a list of 10 elements, i.e., a list of 10 aliquots imported from two separate BIN files.

2.2 Signal analyses

2.2.1 Curve Deconvolution

Before calculating the equivalent doses for each aliquot it may be useful to estimate the photoionisation cross sections of some of the OSL curves. The example *.bin file includes standard continuous-wave OSL curves as well as linearly modulated OSL curves. This allows us to use both the functions `fit_CWCurve()` and `fit_LMCurve()` and to compare their results.

First, we will begin with the LM-OSL curves, which are named “RBR” in the *.bin file. For the sake of brevity we focus on one aliquot and one curve.
# LM-OSL curve of the second aliquot

```r
lm_curve <- get_RLum(subset(rlum[[3]], LTYPE == "RBR"))

lm_fit <- fit_LMCurve(values = lm_curve,
    n.components = 3,
    fit.method = "LM",
    LED.power = 30,
    LED.wavelength = 470)
```

## [fit_LMCurve()]

## Fitting was done using a 3-component function:

```
#
#   xm.1   xm.2   xm.3   Im.1   Im.2   Im.3
# 18.342407 146.914692 701.057433 162.076721 9.926817 60.606545
#
# (equation used for fitting according Kitis & Pagonis, 2008)
# ------------------------------------------------------------------------------
# (1) Corresponding values according the equation in Bulur, 1996 for b and n0:
#   #
#   # b1 = 2.972264e+00 +/- NA
#   # n01 = 4.901446e+03 +/- NA
#   # b2 = 4.633077e-02 +/- NA
#   # n02 = 2.404487e+03 +/- NA
#   # b3 = 2.034664e-03 +/- NA
#   # n03 = 7.005197e+04 +/- NA
#   #
#   # cs from component.1 = 4.187e-17 cm^2 >> relative: 1
#   # cs from component.2 = 6.527e-19 cm^2 >> relative: 0.0156
#   # cs from component.3 = 2.866e-20 cm^2 >> relative: 7e-04
#   #
#   # (stimulation intensity value used for calculation: 7.098105e+16 1/s 1/cm^2)
#   # (errors quoted as 1-sigma uncertainties)
#   #
#   # pseudo-R^2 = 0.8532
```
We can see the photoionisation cross-sections in the console output or get from the output object.

```r
lm_df <- get_RLum(lm_fit)
cat(" cs1: ", lm_df$cs1, "\n",
"cs2: ", lm_df$cs2, "\n",
"cs3: ", lm_df$cs3)
```

```r
## cs1: 4.187405e-17
## cs2: 6.527203e-19
## cs3: 2.86649e-20
```

We now want to compare these values with the cross sections obtained from fitting CW-OSL curves. Again, we will only focus on one randomly chosen OSL curve here.

```r
# Fifth CW-OSL curve of the third aliquot
cw_curve <- get_RLum(subset(rlum[[3]], LTYPE == "OSL"))[6]
```
2.2. SIGNAL ANALYSES

cw_fit <- fit_CWCurve(values = cw_curve,
                   n.components.max = 3,
                   fit.method = "LM",
                   LED.power = 30,
                   LED.wavelength = 470)

## [fit_CWCurve()]
##
## Fitting was finally done using a 3-component function (max=3):
##
## y ~ I0.1 * lambda.1 * exp(-lambda.1 * x) + I0.2 * lambda.2 * exp(-lambda.2 * x) + I0.3 * lambda.3 * exp(-lambda.3 * x)
##
## I0 I0.error lambda lambda.error cs cs.rel
## c1 3751.3428 NA 4.75949137 NA 6.705299e-17 1.0000
## c2 805.6833 NA 1.27070518 NA 1.790203e-17 0.2670
## c3 6856.1207 NA 0.01171049 NA 1.649805e-19 0.0025
##
## pseudo-R^2 = 0.9999

To compare the results side-by-side we do the following:

cw_df <- get_RLum(cw_fit)

fit_cmp <- data.frame(lm = c(lm_df$cs1, lm_df$cs2, lm_df$cs3),
                      cw = c(cw_df$cs1, cw_df$cs2, cw_df$cs3))
We can see that the estimated photoionisation cross-section differ significantly, but at least to a certain degree this is to be expected. We will conclude this section by calculating the cross-sections for all curves of the third aliquot to get an idea of the variation in cross-sections in relation to which OSL curve in the SAR sequence we choose.

cw_curves <- get_RLum(subset(rlum[[3]], LTYPE == "OSL"))

cw_fits <- lapply(cw_curves, function(x) {
  fit_CWCurve(values = x,
               n.components.max = 3,
               LED.power = 30,
               LED.wavelength = 470,
               plot = FALSE,
               output.terminal = FALSE)
})

cw_df <- do.call(rbind, lapply(cw_fits, get_RLum))

print(data.frame(cs1 = cw_df$cs1, cs2 = cw_df$cs2, cs3 = cw_df$cs3))
2.2. Fast ratio

Another way to characterise OSL curves is by calculating the fast ratio after Durcan & Duller (2011), which can easily be done using the function `calc_FastRatio()`.

```r
osl_curves <- subset(rlum[[3]], LTYPE == "OSL")
fr <- lapply(osl_curves, function(x) {
  calc_FastRatio(object = x,
                stimulation.power = 37,
                wavelength = 470,
                fitCW.sigma = TRUE,
                plot = FALSE,
                verbose = FALSE)
})
fr_df <- do.call(rbind, get_RLum(fr))

plot(x = 1:nrow(fr_df), y = fr_df$fast.ratio, type = "l")
```
2.2.3 NR(t) plots

A rarely used method to graphically visualise the relative rates of signal decay is to produce natural/regenerated signal vs time (NR(t)) plot after Steffen et al. 2009. Before using this function, however, we need to remove all the OSL curves of the test dose signals. Subsetting the RLum.Analysis is not a viable option this time, however, as a *.bin file does not contain the relevant information to properly identify the test dose curves (at least if not explicitly set in the Sequence Editor). The easier approach would be to just remove every second curve from the RLum.Analysis object.

```r
natreg <- get_RLum(osl_curves)[seq(from = 1, to = length(osl_curves), by = 2)]
str(natreg, max.level = 1)
```

With `get_RLum()` we first extract all of the OSL curves in the RLum.Analysis object. Inside of the `[]` operator we select only the odd numbered curves by creating a sequence (`seq()`) of odd numbers from 1 to the max number of stored OSL curves.

Note that the new variable `natreg` is no longer an object of class RLum.Analysis, but a list of
2.3. EQUIVALENT DOSE CALCULATION

RLum.Data.Curve\(^1\). This is because we extracted \(\text{get\_RLum()}\) the curves rather than modifying the object in place. This is ok for us, as the details section in \text{plot\_NRt()} states that it will also accept a list of \text{RLum.Data.Curve} objects.

\text{plot\_NRt(natreg, log = "x", smooth = "rmean", k = 3)}

\begin{center}
\includegraphics[width=\textwidth]{NR(t) Plot}
\end{center}

In a NR(t) plot the signal of each channel of the natural curve is divided by the corresponding signal in the same channel of a regenerative dose. To account for the obvious difference in brightness (dose dependency) each curve is normalised to its initial signal, so the deviation from unity when dividing the signals per channel is a direct reflection of the differences in the decay rate. In this example each curve is follow the area below unity, which means that the natural signal decays faster than the regenerated signals.

2.3 Equivalent dose calculation

2.3.1 Minimalist approach

In most cases the analysed data are from quartz SAR measurements, which requires the \text{analyse\_SAR\_CWOSL()} to be used for equivalent dose calculation. Even though \text{analyse\_SAR\_CWOSL()} offers many different options (i.e., arguments) to modify the fitting procedure, there are only few arguments that are actually required to run this function.

The \text{object} argument takes an \text{RLum\_Analysis} object as input (or a list of them). Other than the input data themselves we only need to define the signal integration intervals.

\(^1\)An \text{RLum\_Analysis} object is basically only a container for a number of \text{RLum\_Data\_Curve} objects.
de <- analyse_SAR.CWOSL(object = rlum[[3]],
  signal.integral.min = 1,
  signal.integral.max = 3,
  background.integral.min = 200,
  background.integral.max = 250,
  plot.single = 6)

## [plot_GrowthCurve()] Fit: EXP (interpolation) | De = 495.45 | D01 = 447.38

Growth curve

\[
D_e = 495.45 \pm 12.83 \mid \text{fit: EXP}
\]

Note that we also used `plot.single = 6` in the function call, which restricts the graphical output to the dose response curve. By default, the function would also plot the TL (if available), \(T_x\) and \(L_x\) curves as well as a series of plots for the rejection criteria.

### 2.3.2 Applying further criteria

When you have a look at the documentation of `?analyse_SAR.CWOSL` you may notice that there is no obvious argument for setting the applied function to be fitted (single saturating exponential, linear, etc.). This is because this and related options are part of the `plot_GrowthCurve()` function, the actual workhorse of `analyse_SAR.CWOSL()`. Arguments provided by `plot_GrowthCurve()` can also be used for `analyse_SAR.CWOSL()`. This works because of the magical `...` argument, which always indicates that a function accepts an indeterminate number of further arguments that are internally passed on to other functions. From `?plot_GrowthCurve` we can learn that we must use the `fit.method` argument to change the fitting function. When we now use `fit.method = "EXP+LIN"` as input in our `analyse_SAR.CWOSL()` call, the argument will be automatically passed to the internally called `plot_GrowthCurve()` function.

In the following code block we want to further customise our fitting procedure and also adjust the rejection
criteria. From the documentation we can learn that the argument `rejection.criteria` takes a named list with numerical values. You can use the following keywords as names for the list elements:

- `recycling.ratio`
- `recuperation.rate`
- `testdose.error`
- `palaeodose.error`
- `exceed.max.regpoint`

Before adjusting the threshold values we first have a look at the performance of our previous fit.

```r
print(get_RLum(de, "rejection.criteria"))
```

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
<th>Threshold</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Recycling ratio (R5/R1)</td>
<td>1.0337000</td>
<td>0.1</td>
<td>OK</td>
</tr>
<tr>
<td>2 Recuperation rate 1</td>
<td>0.0165000</td>
<td>0.1</td>
<td>OK</td>
</tr>
<tr>
<td>3 Testdose error</td>
<td>0.0119357</td>
<td>0.1</td>
<td>OK</td>
</tr>
<tr>
<td>4 Palaeodose error</td>
<td>0.0258000</td>
<td>0.1</td>
<td>OK</td>
</tr>
<tr>
<td>5 De &gt; max. dose point</td>
<td>495.4500000</td>
<td>850.0</td>
<td>OK</td>
</tr>
</tbody>
</table>

The `data.frame` tells us that the aliquot passed all rejection criteria (“OK” status). For the purpose of demonstrating we will apply much stricter threshold values so that our aliquot fails at least one of the criteria.

```r
rc <- list(
  recycling.ratio = 2,
  recuperation.rate = 5,
  testdose.error = 5,
  palaeodose.error = 5,
  exceed.max.regpoint = TRUE)

de <- analyse_SAR.CWOSL(object = rlum[[3]],
  signal.integral.min = 1,
  signal.integral.max = 3,
  background.integral.min = 200,
  background.integral.max = 250,
  plot = FALSE,
  verbose = FALSE,
  rejection.criteria = rc,
  # arguments inherited from plot_GrowthCurve()
  fit.method = "EXP+LIN",
  mode = "interpolation",
  fit.force_through_origin = FALSE,
  fit.includingRepeatedRegPoints = TRUE,
  fit.weights = TRUE,
  NumberIterations.MC = 100)

print(get_RLum(de, "rejection.criteria"))
```

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
<th>Threshold</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Recycling ratio (R5/R1)</td>
<td>1.0337000</td>
<td>0.02</td>
<td>FAILED</td>
</tr>
</tbody>
</table>
Due to the much stricter threshold values the aliquot failed the recycling ratio criteria. While it is certainly nice to know why the aliquot failed the rejection criteria, for the subsequent analysis (calculating a mean dose) it is often enough to know that it failed in the first place. This is why the summary table that also contains the equivalent dose also provides the field RC.Status.

```
print(get_RLum(de))
```

Due to the much stricter threshold values the aliquot failed the recycling ratio criteria. While it is certainly nice to know why the aliquot failed the rejection criteria, for the subsequent analysis (calculating a mean dose) it is often enough to know that it failed in the first place. This is why the summary table that also contains the equivalent dose also provides the field RC.Status.

```
print(get_RLum(de))
```

2.3.3 Iterating over all aliquots/grains

Usually we do not want to calculate the equivalent dose of an individual aliquot, but for all of the aliquots of a bin file. `analyse_SAR.CWOSL()` allows us to also provide a list of RLum.Analysis objects, so we do not need to bother with a for-loop or the `*apply()` function family.

```
de <- analyse_SAR.CWOSL(object = rlum,  
  signal.integral.min = 1,  
  signal.integral.max = 3,  
  background.integral.min = 200,  
  background.integral.max = 250,  
  plot = FALSE,  
  verbose = FALSE)
```

```
de_df <- get_RLum(de)
```

```
print(de_df)
```

As per the previous output table RC.Status contains the character string "FAILED", which we can later use to filter the data before applying any further analysis.

```
print(de_df)
```

```
## De De.Error D01 D01.ERROR D02 D02.ERROR De.MC Fit RC.Status  
## 1 532.1 17.02 NA NA NA NA 533.68 EXP+LIN FAILED  
## 2 582.87 52.52 302.71 15.004596 NA NA 589.59 EXP OK  
## 3 411.10 4.47 710.58 15.779791 NA NA 411.47 EXP OK  
## 4 495.45 13.87 447.38 14.997224 NA NA 494.66 EXP OK  
## 5 407.79 8.12 523.85 16.369245 NA NA 407.04 EXP OK  
## 6 569.63 6.25 646.92 10.440184 NA NA 569.73 EXP OK  
## 7 425.10 16.28 341.88 9.960233 NA NA 424.60 EXP OK  
## 8 337.63 18.71 536.04 44.728951 NA NA 334.72 EXP OK  
## 9 709.25 19.96 530.91 13.977242 NA NA 708.83 EXP OK
```
2.4. DOSE MODELS

Even though we provided a list of 10 RLum.Analysis object (one for each aliquot) we are only returned a single RLum.Results object. The data table now contains 10 rows, however, with the results for each of the aliquots.

2.4 Dose models

The Luminescence package provides a lot of functions for calculating a “mean” equivalent dose. These include:

- calc_AverageDose()
- calc_CommonDose()
- calc_CentralDose()
- calc_MinDose()
- calc_MaxDose()
- calc_FiniteMixture()
- calc_FuchsLang2001()
- calc_IEU()

All of these functions more or less follow the same pattern with regards to input and output, so it suffices to apply just one the models here in this example.

The first argument is always named data and requires either an RLum.Results object, or a data.frame with two columns for the equivalent doses and their corresponding standard errors. In our example it is easiest to just pass the RLum.Analysis object to the dose model function.

cam <- calc_CentralDose(de, plot = FALSE, verbose = TRUE)

### [calc_CentralDose]

### ------------ meta data ------------
We do now, however, face the problem that the one aliquot that failed the rejection criteria was also considered when calculating the central dose. It is usually better to gather and filter the data first before applying the dose model. Here, we extract the table with the equivalent doses first, filter out only those aliquots that passed the rejection criteria and then select only the first two columns.

```r
passed <- which(de_df$RC.Status == "OK")
def <- de_df[passed, 1:2]
```

```r
cam <- calc_CentralDose(de_df, plot = FALSE, verbose = TRUE)
```

As usual, the dose model function returns an `RLum.Results` object, whose content can be accessed with the `get_RLum()` function. If we do not specify which slot of the object should be returned, we automatically receive the summary that also contains the mean equivalent dose.

```r
get_RLum(cam)
```

```
## de de_err OD OD_err Lmax
## 1 474.6642 30.28673 19.81355 4.588683 10.95064
```

### 2.5 Visualisation of DE distributions

The Luminescence package includes several functions to visualise equivalent dose distributions. These functions include:

- `plot_AbanicoPlot()`
2.5. VISUALISATION OF DE DISTRIBUTIONS

- `plot_GrowthCurve()`
- `plot_Histogram()`
- `plot_KDE()`
- `plot_RadialPlot()`
- `plot_ViolinPlot()`

As for the functions to calculate a mean dose the plotting functions also follow a common pattern. The first argument is always named `data` and accepts either an `RLum.Results` object, or a `data.frame` with two columns for the equivalent doses and their corresponding standard errors. While providing an `RLum.Results` object is certainly the more comfortable approach, it is highly suggested to always use the more simple `data.frame` as input data. ²

2.5.1 Abanico Plot

Along with the Radial Plot the Abanico Plot is arguably the most complicated and also the most customisable plot in the `Luminescence` package. With over 30 arguments to customise the plot to one’s desire, creating an Abanico Plot may appear too complicated. Luckily enough, all of those arguments already come with reasonable default values. It is only the data that we need to provide in order to obtain a plot.

`plot_AbanicoPlot(de_df)`

For standard analyses the plot will probably suffice, but if we want to create a graph for a publication there are certainly some details we may want to tweak. Instead of providing a detailed explanation of what each

²If you wondered why we recommend using a `data.frame` as input data instead of the `RLum.Results` object: as of version 0.8.0 of the `Luminescence` package, the internal structure of the `RLum.Analysis` objects returned by the various functions is not yet fully homogenised. This may cause the plotting functions to fail to obtain the original data, which is why providing the proper data via a `data.frame` is always the safer bet.
argument does or why these specific values were chosen we only refer to the documentation of the Abanico Plot (`plot_AbanicoPlot`).

```r
plot_AbanicoPlot(de_df,
  z.0 = "median",
  zlim = c(200, 1000),
  ylim = c(-100, 100),
  dispersion = "qr",
  zlab = c("Equivalent dose (s)"),
  main = "Sample CA-2 (Qz 100-150 mm)",
  summary = c("n", "mean", "se.abs", "kurtosis", "skewness"),
  summary.pos = "topleft",
  rotate = FALSE,
  rug = TRUE,
  grid.col = FALSE,
  y.axis = FALSE,
  line = cam,
  line.col = "red",
  line.label = "CAM")
```

Sample CA-2 (Qz 100–150 mm)

<table>
<thead>
<tr>
<th>n</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>475.29</td>
</tr>
<tr>
<td>se</td>
<td>34.31</td>
</tr>
<tr>
<td>kurtosis</td>
<td>2.32</td>
</tr>
<tr>
<td>skewness</td>
<td>0.65</td>
</tr>
</tbody>
</table>

2.5.2 Further plots

In this subsection we want to give you an impression of all the plots available in the Luminescence package. In general, if you understood how to use and customise the `plot_AbanicoPlot()` function, using all the other plotting functions will be straightforward. Of course, the number of arguments differs from plot to plot, so it is always advisable to have a look at their corresponding documentation.
In the following, we plot the same dose distribution with some of the other plotting functions.

**Kernel Density Estimate Plot**

```r
plot_KDE(de_df,
          main = "Sample CA-2 (Qz 100-150 mm)",
          xlab = c("Equivalent dose (s)",
          summary = c("n", "mean", "se.abs", "kurtosis", "skewness"),
          summary.pos = "topleft",
          bw = 25,
          cex = 0.9)
```

**Histogram**

```r
# use custom breaks
breaks <- hist(de_df$De, breaks = 5, plot = FALSE)$breaks

plot_Histogram(de_df,
               main = "Sample CA-2 (Qz 100-150 mm)",
               xlab = c("Equivalent dose (s)",
               rug = TRUE,
               normal_curve = TRUE,
               breaks = breaks)
```
CHAPTER 2. LUMINESCENCE PACKAGE

Sample CA−2 (Qz 100−150 mm)

Radial Plot

plot_RadialPlot(de_df,
               main = "Sample CA-2 (Qz 100-150 mm)",
               zlab = c("Equivalent dose (s)")
               summary = c("n", "mean", "se.abs", "kurtosis", "skewness"),
               summary.pos = "topleft",
               bw = 25,
               cex = 0.9)
2.5. VISUALISATION OF DE DISTRIBUTIONS

Sample CA–2 (Qz 100–150 mm)

\[ \text{n} = 10 \]
\[ \text{mean} = 485.5 \]
\[ \text{kurtosis} = 2.32 \]
\[ \text{skewness} = 0.65 \]

Violin Plot

```
plot_ViolinPlot(de_df,
    main = "Sample CA-2 (Qz 100-150 mm)",
    summary = c("n", "mean", "se.abs", "kurtosis", "skewness"),
    summary.pos = "topleft",
    cex = 0.8)
```
Sample CA–2 (Qz 100–150 mm)

\[\text{n} = 10, \quad \text{mean} = 485.5, \quad \text{se.abs} = 34.31, \quad \text{kurtosis} = 2.32, \quad \text{skewness} = 0.65\]
Chapter 3

Scripting

The designated outcome of this document is for you to roughly understand the R scripts in the following subsections. The word “roughly” was explicitly chosen, as we are well aware that this document only provides a small outline of all of the included functions in the R package Luminescence. Also, whether you can follow the script or not also highly depends on the individual knowledge or basic R data structures and functions.

What is a script? A script is a sequence of R commands issued in consecutive order that build on each other and which act on a common data origin to produce a desired outcome. Something like the following can be considered a script:

```r
my_value <- 4
x <- my_value
y <- 2
z <- x/y
print(z)
```

## [1] 2

Of course, this is a very simple script, but ultimately the very essence of a “script”. As soon as you do not run each function directly from the R console, but write the code to a file and then run the whole code at once, it can be considered scripting. Alas, this very document is a script!

The following subsections each contain a self-contained script for a designated purpose (e.g., to calculate the age of a sample starting from *.bin file). There will be no line-by-line code explanations, but only short summaries on what these scripts do and what the intended outcome is.

Writing a script for the analyses of luminescence data is not about the knowledge of the functions in the Luminescence package, but about knowing about data structures and types in R and how to access and process them. To write your own script it is essential to know, e.g., the differences between data.frames and lists and how to index them properly. Since we often operate on multiple data sets (e.g., multiple *.bin files and/or aliquots/grains) it is also mandatory to be well familiar with writing for-loops and how to the *apply-function family.

There are certain good practices to consider when writing a script. First of all, scripts should be concise. Before writing a script you should spend at least some time on thinking about the required input data as well as to clearly define what the outcome of the script should be. There is always a risk of a script to become too convoluted, to a point where it is both slow and hard to comprehend, even for the author. Even if you understand your own script at the time of writing, ask yourself the question if you would also be able to understand it when re-visiting the script a week or a month later. The longer a script and the more convoluted, the harder it will be to re-use a script at a later point in time. This is especially true when a
previously working script fails and you have to fix the code first. So instead of writing a very, very long script that does everything consider writing multiple shorter scripts that each specialises in a certain task. Scripts should be as dynamically coded as possible. This is the very bane of many scripts, because they fail to

### 3.1 From .BIN(X) to AGE

This is the most condensed script to calculate the age of a sample starting from a single raw measurement file. The basic procedure of the script is as follows:

- import the data via `read_BIN2R()`
- calculate the equivalent doses with `analyse_SAR.CWOSL()`
- estimate the central dose using `calc_CentralDose()`
- calculate the age using `use_DRAC()`
- export the DRAC results via `write.csv()`

For converting the equivalent doses from seconds to Gray we need to define a variable that stores the beta dose rate of the reader. For the age calculation we obviously also need some more external data of the sample itself (e.g., the radionuclide contents).

```r
## General settings
reader_doseRate <- 0.1169

## Import BIN file
bin <- read_BIN2R(file = "D-CA1_100-150_2mm_a.BIN", fastForward = TRUE,
                  txtProgressBar = FALSE, verbose = FALSE)

## Calculate equivalent doses
de <- analyse_SAR.CWOSL(object = bin,
                        signal.integral.min = 1, signal.integral.max = 3,
                        background.integral.min = 200, background.integral.max = 250,
                        plot = FALSE, verbose = FALSE)

de_df <- get_RLum(de)
de_dist <- data.frame(de_df$De, de_df$De.Error)

## Age Model
cam <- calc_CentralDose(de_dist, plot = FALSE, verbose = FALSE)
cam_df <- get_RLum(cam)

## DRAC input table
drac_input <- template_DRAC(nrow = 1, notification = FALSE)

drac_input$"Project ID" <- "DRAC-Example"
drac_input$"Sample ID" <- "Quartz"
drac_input$"External U (ppm)" <- 1.15
drac_input$"errExternal U (ppm)" <- 0.06
drac_input$"External Th (ppm)" <- 8.19
drac_input$"errExternal Th (ppm)" <- 0.54
drac_input$"External K (%)" <- 0.40
drac_input$"errExternal K (%)" <- 0.02
drac_input$"Grain size min (microns)" <- 100
drac_input$"Grain size max (microns)" <- 150
```
## DRAC output

```
drac_output <- use_DRAC(drac_input, verbose = FALSE)
```

## Age

```
age <- get_RLum(drac_output)
print(paste("Age (ka): ", age$highlights$`Age (ka)`", ", "+-", age$highlights$`errAge (ka)`))
```

## Write DRAC highlight table to CSV file

```
write.csv(x = age$highlights, 
          file = "DRAC_highlights.csv", 
          quote = FALSE, row.names = FALSE)
```

3.2 $D_E(T)$ PLOT

The following script is a demonstration how to use the `lapply()` function to iterate the imported `RLum.Analysis` object while progressively increasing the signal interval when calculating the equivalent dose using `analyse_SAR.CWOSL()`. The resulting equivalent doses are then plotted in consecutive order in order to visualise its dependency on the chosen integral; a plot that is commonly referred to as a De(t) plot (Bailey, 2001).

Note that the below script is actually superseded by the already included `plot_DetPlot()` function in the Luminescence package, but was included for demonstrative purposes. Also, the script uses the popular `ggplot2` package for plotting, which is part of the so-called “tidyverse”, a collection of `R` packages designed for data science. It is by no means necessary to use packages of the “tidyverse” and can easily be replaced with standard `R` code.

```
# load additional libraries
library(tidyverse)

# Import BIN file
bin <- read_BIN2R(file = "D-CA1_100-150_2mm_a.BIN", fastForward = TRUE, 
        txtProgressbar = FALSE, verbose = FALSE)

signal.integral.max <- 2:4

# Calculate equivalent doses
de <- lapply(signal.integral.max, function(x) { 
    analyse_SAR.CWOSL(object = bin, 
            signal.integral.min = 1, signal.integral.max = x, 
            background.integral.min = 200, background.integral.max = 250, 
            plot = FALSE, verbose = FALSE)
})
```
3.3 Minimum Dose Model

In the absence of a well-bleached sample of the same mineral derived from the same source and of equivalent age to derive a reasonable estimate of $\sigma_b$ from (Galbraith & Roberts, 2012), one may resort to a purely statistical reasoning for a chosen $\sigma_b$ value. The `calc_MinDose()` (and likewise, the `calc_MaxDose()` and `calc_FiniteMixture()` models) provide an estimate of the maximum likelihood as well as the Bayesian Information Criterion (BIC), which can be used to choose between several results when applying a range of $\sigma_b$ values.

The following script first calculates the equivalent doses from the provided BIN file and then uses the `lapply()` function to iteratively apply the `calc_MinDose()` while progressively increasing the $\sigma_b$ value. We then choose the model with the lowest BIC score as our final estimate.

```r
## Import BIN file
bin <- read_BIN2R(file = "D-CA1_100-150_2mm_a.BIN", fastForward = TRUE, 
txtProgressBar = FALSE, verbose = FALSE)

## Calculate equivalent doses
de <- analyse_SAR.CWOSL(object = bin, 
  signal.integral.min = 1, signal.integral.max = 3, 
  background.integral.min = 200, background.integral.max = 250, 
  plot = FALSE, verbose = FALSE)

de_df <- get_RLum(de)
de_dist <- data.frame(de_df$De, de_df$De.Error)

## Minimum dose model
sigmab_range <- seq(from = 0.1, to = 0.2, by = 0.01)

mam <- lapply(sigmab_range, function(x) {
  calc_MinDose(de_dist, sigmab = x, log = TRUE, par = 3, plot = FALSE, verbose = FALSE)
})

mam_df <- do.call(rbind, get_RLum(mam))
```
## Results

```r
mam_df

# de  de_err  ci_level ci_lower  ci_upper  par  sig  p0  mu
# 1 411.1863 38.41256 0.95 314.9007 465.4779 3 0.2735070 0.3521620 NA
# 2 415.9179 39.99267 0.95 315.9843 472.7555 3 0.2681402 0.3946327 NA
# 3 420.2823 41.60574 0.95 316.7562 479.8507 3 0.2616834 0.4307325 NA
# 4 424.5648 43.32371 0.95 317.2642 487.0932 3 0.2542049 0.4643213 NA
# 5 428.7750 45.32112 0.95 317.6654 495.3241 3 0.2454921 0.4954613 NA
# 6 432.9593 47.66160 0.95 318.0356 504.8690 3 0.2350123 0.5240223 NA
# 7 437.3469 49.79710 0.95 318.3913 513.5959 3 0.2221564 0.5522528 NA
# 8 442.3140 51.52836 0.95 318.8492 520.8403 3 0.2063903 0.5853134 NA
# 9 473.5193 NA 0.95 NA 530.9584 3 0.5466625 1.0000000 NA
# 10 473.6368 NA 0.95 NA 534.3334 3 0.5513966 1.0000000 NA
# 11 466.3117 54.94108 0.95 322.0313 537.4003 3 0.1346211 0.8519162 NA

# Lmax  BIC
# 1 2.122646 7.737638
# 2 2.132008 7.718912
# 3 2.121100 7.740730
# 4 2.094580 7.793770
# 5 2.065988 7.870953
# 6 2.008424 7.966081
# 7 1.955373 8.072183
# 8 1.900572 8.181786
# 9 1.665451 8.652026
# 10 1.735055 8.512819
# 11 1.756353 8.470223

# Lowest BIC score
mam_df[which.min(mam_df$BIC),]

# de  de_err  ci_level ci_lower  ci_upper  par  sig  p0  mu
# 2 415.9179 39.99267 0.95 315.9843 472.7555 3 0.2681402 0.3946327 NA

# Lmax  BIC
# 2 2.132008 7.718912
```
library(devtools)

devtools::session_info()

## Session info

## setting value
## version R Under development (unstable) (2018-02-07 r74236)
## system x86_64, mingw32
## ui RTerm
## language (EN)
## collate English United Kingdom.1252
## tz Europe/Berlin
## date 2018-02-16

## Packages

## package    * version  date   source
## backports   * 1.1.2    2017-12-13 CRAN (R 3.5.0)
## base        * 3.5.0    2018-02-08 local
## bbmle       1.0.20    2017-10-30 CRAN (R 3.5.0)
## bookdown    0.6.2     2018-02-09 Github (rstudio/bookdown@d3570d3)
## compiler    3.5.0     2018-02-08 local
## data.table  1.10.4-3  2017-10-27 CRAN (R 3.5.0)
## datasets    * 3.5.0    2018-02-08 local
## devtools    * 1.13.4   2017-11-09 CRAN (R 3.5.0)
## digest      0.6.15    2018-01-28 CRAN (R 3.5.0)
## evaluate    0.10.1    2017-06-24 CRAN (R 3.5.0)
## graphics    * 3.5.0    2018-02-08 local
## grDevices   * 3.5.0    2018-02-08 local
## grid        3.5.0     2018-02-08 local
## htmltools   0.3.6     2017-04-28 CRAN (R 3.5.0)
## httr        1.3.1     2017-08-20 CRAN (R 3.5.0)
## knitr       1.19      2018-01-29 CRAN (R 3.5.0)
## lattice     0.20-35   2017-03-25 CRAN (R 3.5.0)
## Luminescence * 0.8.1   2018-02-14 Github (r-lum/luminescence@311df24)
## magrittr    * 1.5      2014-11-22 CRAN (R 3.5.0)
## MASS        7.3-48    2017-12-24 CRAN (R 3.5.0)
## matrixStats 0.53.1    2018-02-11 CRAN (R 3.5.0)
## memoise     1.1.0     2017-04-21 CRAN (R 3.5.0)
## methods     * 3.5.0    2018-02-08 local
## minpack.lm   1.2-1     2016-11-20 CRAN (R 3.5.0)
## numDeriv     2016.8-1  2016-08-27 CRAN (R 3.5.0)
<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Date</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>3.5.0</td>
<td>2018-02-08</td>
<td>local</td>
</tr>
<tr>
<td>R6</td>
<td>2.2.2</td>
<td>2017-06-17</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>raster</td>
<td>2.6-7</td>
<td>2017-11-13</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>Rcpp</td>
<td>0.12.15</td>
<td>2018-01-20</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>rmarkdown</td>
<td>1.8</td>
<td>2017-11-17</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>rprojroot</td>
<td>1.3-2</td>
<td>2018-01-03</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>rstudioapi</td>
<td>0.7</td>
<td>2017-09-07</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>sp</td>
<td>1.2-7</td>
<td>2018-01-19</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>stats</td>
<td>* 3.5.0</td>
<td>2018-02-08</td>
<td>local</td>
</tr>
<tr>
<td>stats4</td>
<td>3.5.0</td>
<td>2018-02-08</td>
<td>local</td>
</tr>
<tr>
<td>stringi</td>
<td>1.1.6</td>
<td>2017-11-17</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>stringr</td>
<td>1.2.0</td>
<td>2017-02-18</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>tools</td>
<td>3.5.0</td>
<td>2018-02-08</td>
<td>local</td>
</tr>
<tr>
<td>utils</td>
<td>* 3.5.0</td>
<td>2018-02-08</td>
<td>local</td>
</tr>
<tr>
<td>withr</td>
<td>2.1.1</td>
<td>2017-12-19</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>xfun</td>
<td>0.1</td>
<td>2018-01-22</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>yaml</td>
<td>2.1.16</td>
<td>2017-12-12</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>zoo</td>
<td>1.8-1</td>
<td>2018-01-08</td>
<td>CRAN (R 3.5.0)</td>
</tr>
</tbody>
</table>