RLumShiny - A graphical user interface for the R Package 'Luminescence'

Christoph Burow, Sebastian Kreutzer, Michael Dietze, Margret C Fuchs, Manfred Fischer, Christoph Schmidt, Helmut Brückner

Ancient TL

RLumShiny - A graphical user interface for the R Package 'Luminescence'

Christoph Burow[1*], Sebastian Kreutzer[2], Michael Dietze[3], Margret C. Fuchs[4], Manfred Fischer[5], Christoph Schmidt[5], Helmut Brückner[1]

[1] Institute of Geography, University of Cologne, 50923 Cologne, Germany
[2] IRAMAT-CRP2A, Université Bordeaux Montaigne, Maison de l'Archéologie, Esplanade des Antilles, 33607 Pessac Cedex, France
[3] Section 5.1 Geomorphology, GFZ German Research Centre for Geosciences, 14473 Potsdam, Germany
[4] Helmholtz-Zentrum Dresden-Rossendorf, Helmholtz Institute Freiberg for Resource Technology, Freiberg, Germany
[5] Geographical Institute, Geomorphology, University of Bayreuth, 95440 Bayreuth, Germany

[*]Corresponding Author: christoph.burow@uni-koeln.de

Received: November 28, 2016; in final form: December 5, 2016

Abstract

Since the release of the R package 'Luminescence' in 2012 the functionality of the package has been greatly enhanced by implementing further functions for measurement data processing, statistical analysis and graphical output. Along with the accompanying increase in complexity of the package, working with the command-line interface of R can be tedious, especially for users without previous experience in programming languages. Here, we present a collection of interactive web applications that provide a user-friendly graphical user interface for the 'Luminescence' package. These applications can be accessed over the internet or used on a local computer using the R package 'RLumShiny'. A short installation and usage guide is accompanied by the presentation of two exemplary applications.

Keywords: R, Software, GUI, Luminescence dating, Abanico Plot, Cosmic Dose Rate

1. Introduction

After its introduction in 1996 by Ihaka & Gentleman (1996) the programming language **R** (R Core Team, 2016) experienced a notable rise in popularity in the mid-2000s (Tippmann, 2015). This may owe to **R** being intuitive and easy to learn, open source, and available for all major computer platforms. A further major advantage of **R** is its easy extensibility by so-called packages, which are collections of pre-programmed routines and commands for all kinds of specialised purposes. To date, there are more than 9,600[1] packages available through the Comprehensive **R** Archive Network (CRAN)[2], contributed by users from various scientific fields. For the purpose of analysing luminescence data, Kreutzer et al. (2012) introduced the **R** package 'Luminescence'. The package provides a collection of functions to process luminescence data and includes, amongst others, routines for import and export of raw measurement files, statistical analysis of luminescence curves and spectra as well as plotting equivalent dose and/or age distributions. Throughout the years, the functionality of the package continuously increased, especially thanks to the helpful suggestions and comments by the users. As the field of applications with the latest release (version 0.6.4) is now larger than ever, the growth in functionality comes at the cost of increasing complexity. The practical guide by Dietze et al. (2013) or the worked example of Fuchs et al. (2015) aim at maintaining the usability of the package and giving a helping hand for users new to **R** and the 'Luminescence' package. In addition to tutorials dedicated to the use of **R** for luminescence data analysis available on the official

[1] https://mran.microsoft.com/, accessed: 2016-11-18.
[2] https://cran.r-project.org/, accessed: 2016-11-18.

website[3] of the **R** package 'Luminesecence' there is also a wide variety of excellent tutorials and books about **R** itself (e.g., Ligges, 2008; Adler, 2012; Crawley, 2012; Wickham, 2014).

While **R** is a comparatively easy-to-learn programming language, there is still a steep learning curve until a user is able to routinely achieve the desired results. In-depth knowledge of **R** fundamentals is not required when working with the 'Luminescence' package, but being familiar with the most important data structures in **R** is a must. In the simplest case, for a specific task, using the package only involves a single short function call, e.g., `Luminescence::plot_AbanicoPlot(data = de.data)` to produce an abanico plot (Dietze et al., 2016) of equivalent dose estimates. However, users may want to adjust the plot according to their requirements. While other software products such as *Origin*® or *SigmaPlot*® allow the user to comfortably click on each element of a plot to change its appearance, this is not possible in **R**. In **R** a plot cannot be changed after it has been drawn, and the user is required to re-run the function call with additional arguments that control the appearance of specific plot elements. For the `Luminescence::plot_AbanicoPlot()` function there are currently 33 such arguments, plus additional base **R** arguments that can be used to design the plot to ones desire. For more elaborate plots the function call in the **R** command-line rapidly increases in complexity. Users new to **R** may feel quickly overwhelmed and may hence not be able to exploit the full potential of the **R** command-line. But even experienced users may find it tedious to iteratively run the function until a satisfying results is produced. Considering that plotting data is also at least partly subject to personal aesthetic tastes in accordance with the information it is supposed to convey, iterating through all the possible options in the **R** command-line can be a time-consuming task. In Human-Computer Interaction an alternative approach to the command-line interface (CLI) is the graphical user interface (GUI), which allows direct, interactive manipulation and interaction with the underlying software. For users with little or no experience with command-lines a GUI offers intuitive access that counteracts the perceived steep learning curve of a CLI (Unwin & Hofmann, 1999).

Here, we present a GUI for the **R** package 'Luminescence' in the form of interactive web applications. These applications can be accessed online so that a user is not even required to have a local installation of **R**. The so-called shiny applications provide access to most of the plotting functions of the **R** package 'Luminescence' as well as to the functions for calculating the cosmic dose rate and for transforming CW-OSL curves (Table 1). We further introduce the **R** package 'RLumShiny' (Burow, 2016) that bundles all applications, is freely available through the CRAN and GitHub[4], and which can be installed and used in any local **R** environment. The general concept and basic layout of the applications are presented first. A short installation and usage guide of the **R** package 'RLumShiny' is then followed by a presentation of two applications for creating abanico plots and calculating the cosmic dose rate. For the latter, we also provide details on the underlying function `Luminescence::calc_CosmicDoseRate()` itself. Throughout the manuscript, **R** function calls and **R** related code listings are typed in `monospaced` letters. Functions of **R** packages other than 'RLumShiny' are given in the style of `package::function()`. **R** packages are given in single quotation marks and software programs are in *italics*.

2. Shiny applications

Even though **R** lacks native support for GUI functions, its capabilities of linking it to other programming languages allows to utilise external frameworks to build graphical user interfaces (Valero-Mora & Ledesma, 2012). Throughout the years there have been many attempts to provide the means for easier access to **R**. A non-exhaustive list of notable **R** packages linking to other languages or frameworks (given in parentheses) for building GUIs includes:

- 'rrgobi' (GGobi) (Temple Lang & Swayne, 2001; Temple Lang et al., 2016)
- 'gWidgets' (Tcl/Tk, GTK+, Java or Qt) (Verzani, 2014)
- 'cranvas' (Qt) (Xie, 2013)
- 'RGtk'/'RGtk2' (GTK+) (Robison-Cox, 2003; Lawrence & Temple Lang, 2010)
- iPlots (Java) (Urbanek & Theus, 2003; Urbanek & Wichtrey, 2013)
- 'tcltk' (Tcl/Tk) (Dalgaard, 2001a,b)

As an example, the 'tctlk' package implements an interface to the Tcl/Tk GUI toolkit and allows the user to build a Tk GUI with plain **R** code. The most prominent project making full use of the Tcl/Tk framework is the *R Commander*[5] (Fox, 2005, 2016), which provides a GUI to an exhaustive collection of statistical functions and is commonly used in teaching statistics (e.g., Konrath et al., 2013; Wagaman, 2013; Westbrooke & Rohan, 2014).

One of the more recent attempts to provide a GUI toolkit for **R** was the introduction of the 'shiny' package (Chang et al., 2016) by RStudio® in late 2012[6], which allows for building interactive web applications straight from **R**. Simple **R** code allows automatic construction of HTML, CSS and JavaScript based user interfaces. GUIs built using 'shiny' are often referred to as 'shiny applications' due to the package's name. Prior knowledge in any of these (markup-)languages is not required. The application is rendered in a web browser

[3] http://www.r-luminescence.de/, accessed: 2016-11-20.
[4] https://github.com/, accessed: 2016-11-20.
[5] *R Commander* is distributed as the **R** package 'Rcmdr' (Fox & Bouchet-Valat, 2016)
[6] https://cran.r-project.org/src/contrib/Archive/shiny/, accessed: 2016-11-18.

Table 1: Shiny applications available in the **R** package 'RLumShiny' (v0.1.1). Each application can be started using the function `app_RLum()` with the corresponding keyword as input for the parameter `app` (e.g., `app_RLum(app = 'abanico')`). * All functions are part of the 'Luminescence' package.

Application	Keyword	Function(s)*
Abanico Plot	"abanico"	`plot_AbanicoPlot()`
Radial Plot	"radialplot"	`plot_RadialPlot()`
Histogram	"histogram"	`plot_Histogram()`
Kernel Density Estimate Plot	"KDE"	`plot_KDE()`
Dose Recovery Test	"doserecovery"	`plot_DRTResults()`
Cosmic Dose Rate	"cosmicdose"	`calc_CosmicDoseRate()`
CW Curve Transformation	"transformCW"	`CW2pHMi()`, `CW2pLM()`, `CW2pLMi()`, `CW2pPMi()`

and keeps up a bidirectional communication to **R**. Any user input on the web application is automatically registered by **R**, which performs the desired action or necessary calculation and finally returns its output back to the GUI. In essence, rather than using the CLI the user operates **R** through the many pre-built and customisable input and output elements (*widgets*) for displaying plots, tables and printed output of **R** objects. One of the main advantages of 'shiny' is that the applications can be served and shared online as a web service, either by using RStudio's hosting service[7] or by installing **R** and the *Shiny Server* software on a (private) Linux server. To access the applications users only need a working internet connection and a common HTML 5 compatible browser; a local **R** environment is *not* needed. Another advantage over previous listed GUI frameworks is that 'shiny' is based on modern programming and markup languages, which allows easy integration of existing JavaScript libraries, thus greatly increasing the capabilities of 'shiny' and **R** itself.

Shiny applications generally work in any **R** environment, but we highly recommend the integrated development environment (IDE) by RStudio (RStudio Team, 2016) when the applications are run locally.

3. The R package 'RLumShiny'

While Duller (2015) acknowledges that the **R** package 'Luminescence' is capable of "extremely complex analysis", the lack of a GUI is rightfully criticised for limiting the potential user group to those with at least basic knowledge in programming. To account for the lack of a GUI and hence to make the 'Luminescence' package more accessible for users with no prior knowledge of **R** we created a collection of shiny applications (Burow et al., 2014). These applications provide a GUI to selected functions of the 'Luminescence' package, mainly, but not exclusively, focussing on its plotting capabilities (Table 1).

These shiny applications are bundled as an **R** package named 'RLumShiny' (Burow, 2016), which is distributed and freely available through the CRAN. The first version of 'RLumShiny' was released on CRAN in March 2015[8] and accumulated over 5,000 downloads[9] since then, even though it was never formally introduced to the scientific community. While it may not seem intuitive, these shiny applications were deliberately not included in the 'Luminescence' package. Much like the **R** package 'RLumModel' (Friedrich et al., 2016) for simulating luminescence in quartz, 'RLumShiny' uses the functions and object system of 'Luminescence'. But the dependency is unidirectional, meaning that 'Luminescence' does not require either of the mentioned packages in order to work. Both packages can be regarded as extensions to 'Luminescence' providing optional and particular features. For the user bundling the shiny applications in a separate package has the advantage of less overhead when installing 'Luminescence'. As 'RLumShiny' requires a couple of other **R** packages (first and foremost 'shiny' and all its sub-dependencies) installing 'Luminescence' may not only take significantly longer, but may also install packages that the user eventually does not need in case the applications are not used. Furthermore, 'RLumShiny' includes functions that extend the capabilities of 'shiny' itself (Table 2) and which should not appear in a package dedicated to the analysis of luminescence data.

From a developer's point of view, it is also easier to develop and maintain a separate **R** package as it eliminates the necessity to constantly update the code to account for changes in 'Luminescence'. Conversely, development of the 'Luminescence' package is not decelerated by the need to update the applications. Each release version of 'RLumShiny' is built and tested against a specific version of 'Luminescence'. In case of an update to a function in 'Luminescence' that breaks the corresponding shiny application in 'RLumShiny' the user is always able to revert to an earlier, compatible version of the 'Luminescence' package.

Since version 0.6.0 the 'Luminescence' package includes the homonymous function `Luminescence::app_RLum()`, a wrapper for the actual `app_RLum()` function in 'RLumShiny'. By that, users of the 'Luminescence' package are made

[7] http://www.shinyapps.io/, accessed: 2016-11-18.

[8] https://cran.r-project.org/src/contrib/Archive/RLumShiny/, accessed: 2016-11-18.

[9] Download statistics taken from https://cranlogs.r-pkg.org/, accessed: 2016-11-18.

Table 2: Functions in the **R** package 'RLumShiny' (v0.1.1). The main function is `app_RLum()`, which must be used to start any of the applications given in Table 1. All other functions are used internally and extend the functionality of the 'shiny' package.

Function	Description
`app_RLum()`	Run luminescence shiny applications.
`jscolorInput()`	Creates a JSColor widget to be used in shiny applications.
`popover()`	Create a bootstrap button with popover.
`tooltip()`	Create bootstrap tooltips for any HTML element to be used in shiny applications.

aware of the existence of a GUI, even if 'RLumShiny' is not installed. In case of the latter, running this function informs the user that 'RLumShiny' is not installed and provides instructions on how to do so if desired. Once installed it is possible to start a shiny application by either using `Luminescence::app_RLum()` or `RLumShiny::app_RLum()`.

The 'RLumShiny' package is actively developed and maintained on the web-based Git[10] repository hosting service GitHub[11]. The 'RLumShiny' applications are also available as a web service hosted on a web server maintained by the corresponding author of this article[12].

3.1. Installation and usage

To install the latest stable version of 'RLumShiny' from CRAN, simply run the code given in Listing 1 in an **R** console.

Listing 1: Install the 'RLumShiny' package from the CRAN.

```
install.packages('RLumShiny')
```

Alternatively, the user can download the latest development version of 'RLumShiny' from GitHub (Listing 2). This, however, requires the 'devtools' package, which will be installed first when executing the first two code lines of Listing 2.

Listing 2: Install the development version of 'RLumShiny' through GitHub.

```
if (!require('devtools'))
    install.packages('devtools')
devtools::install_github('R-Lum/RLumShiny')
```

Both Listing 1 and Listing 2 will install the 'RLumShiny' package and all its dependencies, i.e., other **R** packages that are required to run the applications (amongst others, most notably 'shiny' and 'Luminescence'). The user only needs to make sure to have installed the most recent version of **R** to get the most recent version of the 'RLumShiny' package (but at least version $\geq$3.1.2).

To start any of the applications included in 'RLumShiny' the user only needs to run `app_RLum()` with the corresponding keyword given in Table 1. As an example, Listing 3 shows how to run the shiny application for creating abanico plots.

Listing 3: Run the shiny application for creating abanico plots.

```
library('RLumShiny')
app_RLum(app = 'abanico')
```

Note that `library('RLumShiny')` needs to be run first when starting a new **R** session, otherwise **R** cannot find the `app_RLum()` function and returns an error. `app_RLum()` only has one named argument called `app`, which accepts all keywords listed in Table 1. Additionally, the function also accepts most arguments of the `shiny::runApp()` function (see `?shiny::runApp`). Thereby it is possible to, e.g., start an application in the so-called showcase mode[13], which presents the application along with the **R** files in the application's directory in a shared tabset.

An alternative to installing and using the 'RLumShiny' package on a local computer is to host the applications as a web service using the *Shiny Server*[14] software. This enables sharing the applications with a wider user base, whether it be an organisation, a working group or anyone interested in using it by making it freely accessible on the internet. Some of the advantages include that, amongst all potential users of the service, only one person is required to set up and maintain the Shiny Server. It has to be considered, however, that setting up a Shiny Server requires a server (or web space), which may need to be purchased or rented first, and a person with sufficient knowledge in administrating a Linux server. Furthermore, the open source version of *Shiny Server* only has a limited amount of features compared to the Pro version that is subject to fee. Nonetheless, the advantages of running a freely accessible, local or access limited Shiny Server can far outweigh these drawbacks and once set up, can provide unlimited and platform independent access to the shiny applications (cf. Fig. 1).

Due to the complexity it is, however, not within the scope of the article to provide a *Shiny Server* installation guide. The reader is referred to RStudio's offi-

[10] A version control system used in software development.

[11] https://github.com/R-Lum/RLumShiny, accessed: 2016-11-18.

[12] http://shiny.r-luminescence.de, accessed: 2016-11-18.

[13] For reference see http://shiny.rstudio.com/articles/display-modes.html, accessed: 2016-11-18.

[14] https://www.rstudio.com/products/shiny/shiny-server/, accessed: 2016-11-18.

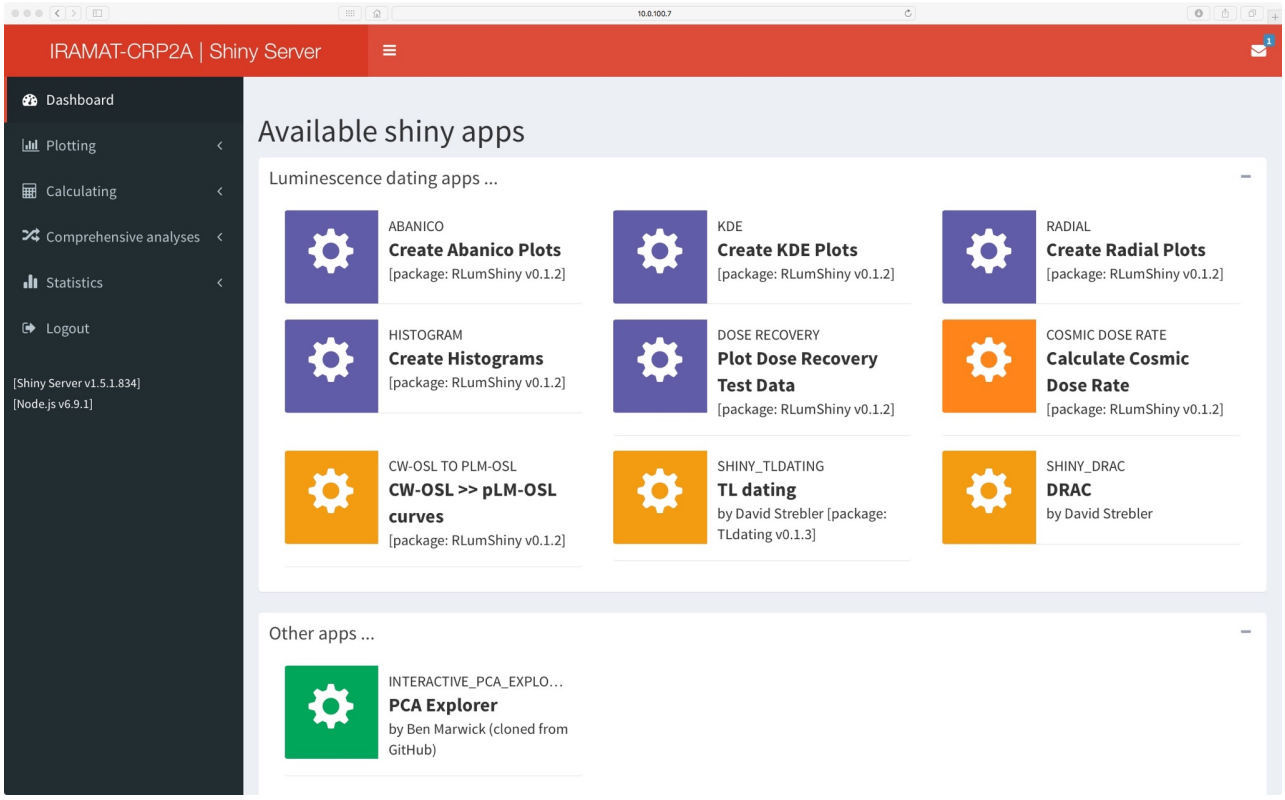

Figure 1: Example for a locally set up Shiny Server, here at the IRAMAT-CRP2A in Bordeaux. Installed are all applications available through the package 'RLumShiny' and additional applications freely available via GitHub or CRAN. The server is accessible within the local network of the IRAMAT-CRP2A only.

cial administrator's guide[15] instead. In some cases it may also be viable to use RStudio's self-service platform http://shinyapps.io, a hosting environment where users can easily upload, run and share their shiny applications. The service offers different subscription plans depending on the desired number of allowed applications, service availability and feature content.

3.2. Application layout and capabilities

Almost all shiny applications included in the 'RLumShiny' package follow a common layout style (Fig. 2). The exception to the rule is the application for calculating the cosmic dose rate, which will be presented separately in Section 3.3.2. The following layout descriptions hence refer to all applications other than the one for calculating the cosmic dose rate. A general characteristic all shiny applications share, however, is the responsive design, meaning that the layout adjusts dynamically while taking into account the characteristics of the device used. Shiny applications are thus always correctly rendered and perfectly usable on desktop computers, mobile phones and anything in between.

Currently, each application in the 'RLumShiny' package usually consists of two separate panels: an input panel on the left-hand side and an output panel on right-hand side. The top of each panel contains a varying amount of tabs (depending on the app) and a context-dependent content area below. In case of the input panel the content areas include various input widgets by which the parameters of the underlying function can be manipulated. Depending on the required data type of the manipulated function parameter these widgets include buttons, checkboxes, sliders, numeric and text input fields and others. Each time the user interacts with these elements the output is automatically updated, i.e., plots are redrawn and numeric output is recalculated.

The first tab of the input panel is always the "Data"-tab, where the user is able to provide the input data. In some cases the user is also able to provide a second data set, e.g., in the application for creating abanico plots (Section 3.3.1). Input data, usually equivalent doses and their individual errors, can be provided as plain text files. Additional options allow specifying the column separator or if the first line should be treated as column headers.

With respect to the output panel the first tab is always a plot, followed by one or two tabs showing an interactive table of the input data. In case of the plotting applications the last output tab shows a dynamically generated **R** script that can be copied to a text editor or *RStudio* and used to reproduce the current plot as seen in the "Plot"-tab. We regard

[15] http://docs.rstudio.com/shiny-server/, accessed: 2016-11-18.

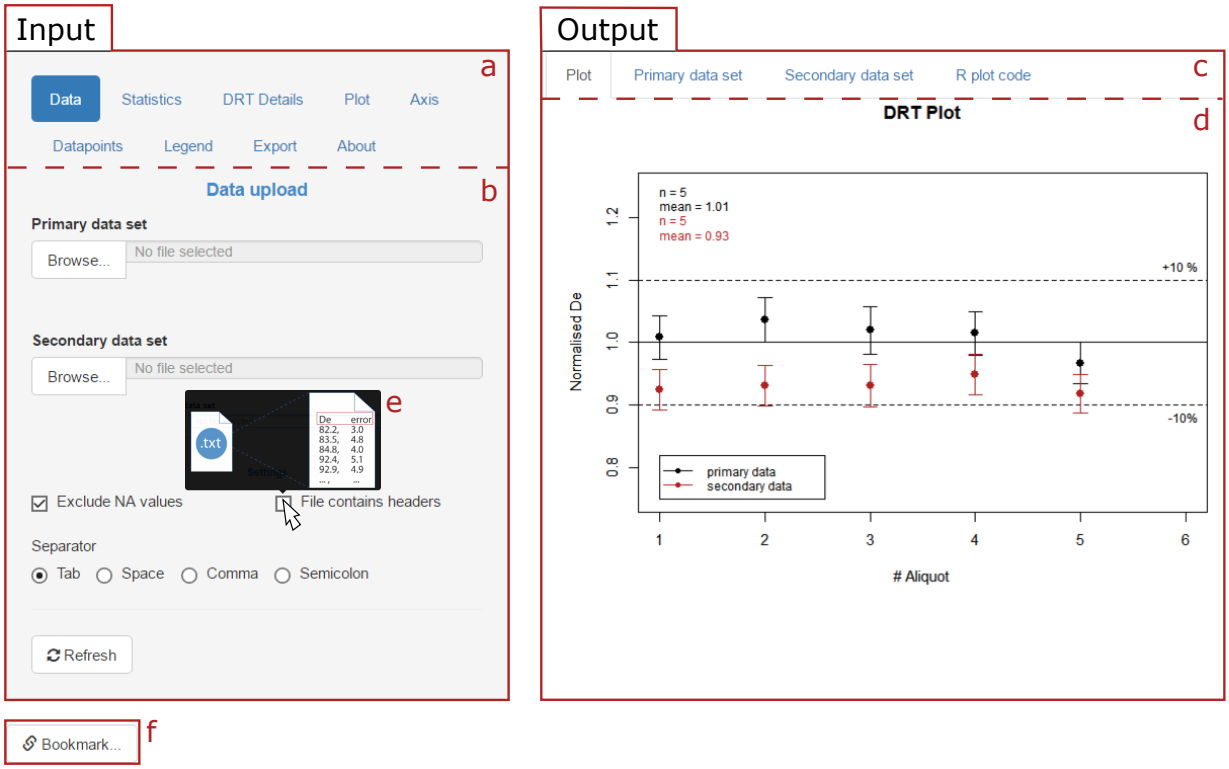

Figure 2: General layout of shiny applications in the 'RLumShiny' package. The applications follow a common GUI layout with two separate panels for input (left) and output (right). Both panels consist of a header with a varying amount of tabs (a, c) and a context-depend content area (b, d). In the example shown here (`app_RLum(app = 'doserecovery')`) (b) shows the "Data" tab content where the user is allowed to provide up to two data sets as an ASCII text file. Additional check boxes and radio buttons allow for providing the files in various style formats. Some of the input elements provide custom tooltips with graphical or text information (e). The "Bookmark" button (f) below the input panel allows saving the current state of the application. The user is provided an URL, which can be used to restore the session, i.e. all previous settings and provided data are automatically restored.

this as a valuable addition as (i) users may use this as a help to understand all the arguments of a particular function and are able to see how they should be used, and (ii) it provides the means to fully reproduce the plot from the CLI or in an existing **R** script.

Naturally, all applications for generating plots offer an export section, which is accessed by the second to last tab on the input panel (Fig. 3). There, the user is able to save the generated plot in a vector graphics format (PDF, SVG or EPS). Note that the plot dimensions in the exported file usually differ from those seen in the "Plot"-tab, as the latter is dynamically rescaled depending on the current size of the viewport. The height and width of the exported image can be specified separately. Additionally, the user can download an **R** script that includes the code required to reproduce the plot from the CLI.

3.3. Example applications

In the current version of 'RLumShiny' (v0.1.1) more than half of all included applications are exclusively there for creating graphical output. The remaining applications are to calculate the cosmic dose rate and to transform continuous-wave OSL curves to a pseudo hyperbolic, linearly or parabolic modulated curve (Table 1). In the following, specific capabilities of the 'RLumShiny' package are exemplified by the applications for creating an abanico plot and for calculating the cosmic dose rate.

3.3.1 Abanico Plot

The abanico plot was introduced by Dietze et al. (2016), a novel plot type for showing chronometric data with individual standard errors. In essence, it is a combination of a radial plot (Galbraith, 1988) and a kernel density estimate (KDE) plot (cf. Galbraith & Roberts, 2012), which can be created using the **R** function `Luminescence::plot_AbanicoPlot()`. To produce a ready-to-use plot the user only needs to provide some input data. Yet, `Luminescence::plot_AbanicoPlot()` offers 33 arguments and an uncounted number of base **R** arguments that can be used to style the plot to ones desire. As plots generated in **R** cannot be changed after they have been drawn the user is required to repeatedly run the function call

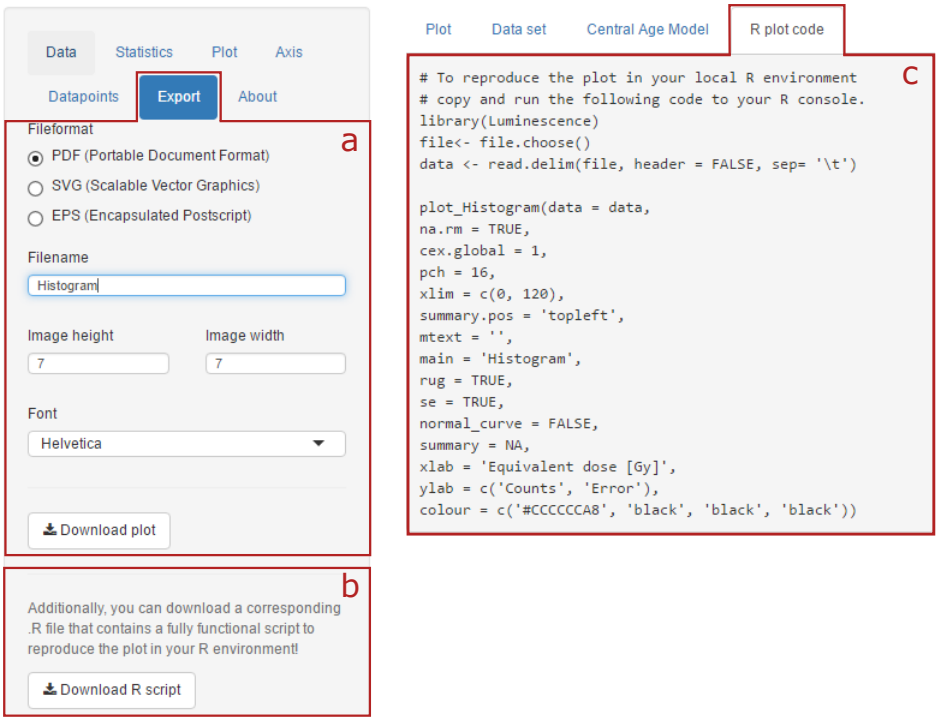

Figure 3: File export options available for the histogram application (`app_RLum(app = 'histogram')`). All plotting applications of the **R** package 'RLumShiny' include an export tab in the input panel, which (a) offers the possibility to save the generated plot as a vector graphics file (file types: PDF, SVG, EPS). Additionally, (b) the user can download an **R** script file that includes the code shown in in the output panel (c), which can be used to reproduce the generated plot in any other **R** environment that has the 'Luminescence' package installed. Alternatively, the user can also just copy and paste the code in (c) and execute it in an **R** console.

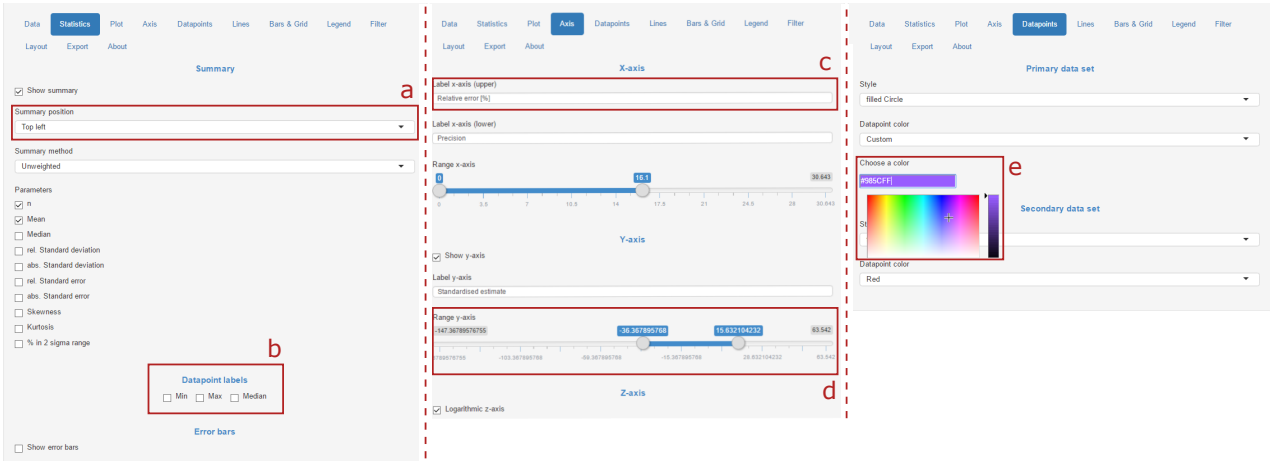

Figure 4: A selection of input options available in the shiny application to generate abanico plots. a) Mutually exclusive options such as the summary position are often manipulated using a drop down menu. b) Binary options like showing or hiding numerical information on the plot can be controlled by checkboxes. c) Plot annotations and axis labels can be changed by text input fields. d) Function arguments requiring a single numeric value or a range of values can be controlled by regular or double-ended range sliders. e) 'RLumShiny' includes the JavaScript library *JSColor* (Odvarko, 2014) along with a custom 'shiny' binding. In this example, if the user chooses "Custom" for the datapoint colour a text input field and a colour table appears, from which a colour can be picked. Alternatively, a hexadecimal RGB value can be typed in directly.

while iteratively changing the input parameters. Even for experienced users this may be a tedious and time-consuming task.

Compared to all other shiny applications in 'RLumShiny' the GUI for generating abanico plots offers the highest number of input widgets (Fig. 4). Generally, a numeric range (e.g., axis limits) is usually controlled with a regular or double-ended range slider, binary options (e.g., showing or hiding the summary) with checkboxes and mutually exclusive options (e.g., line type) with radio buttons or drop down menus. Text fields are mostly used to manipulate plot annotations and axis labels.

The `jscolorInput()` function in 'RLumShiny' extends the 'shiny' interface by including the web colour picker *JSColor*[16] (Odvarko, 2014). When the user chooses "Custom" as input in one of the colour drop down menus (e.g., in the "Datapoint"-tab) a new text input field appears. There, the user is able to enter a hexadecimal RGB value or to pick a colour from a small colour table that appears when the user clicks in the input field.

3.3.2 Cosmic dose rate

The shiny application for calculating the cosmic dose rate is chosen as an example (i) to take the opportunity to provide details on the underlying function `Luminescence::calc_CosmicDoseRate()`, (ii) as its layout differs from all other applications in 'RLumShiny', and (iii) as it includes a unique feature.

Despite its universal use, the equation to calculate the cosmic dose rate provided by Prescott & Hutton (1994) is falsely stated to be valid from the surface to 10^4 hg cm^{-2} (1 hg cm^{-2} = 100 g cm^{-2}) of standard rock[17]. The original expression by Barbouti & Rastin (1983) only considers the muon flux (i.e., the hard-component of the cosmic flux) and is, by their own account, only valid for depths between 10 hg cm^{-2} and 10^4 hg cm^{-2}. Thus, for near-surface samples (i.e., for depths <167 g cm^{-2}) the equation of Prescott & Hutton (1994) underestimates the total cosmic dose rate as it neglects the influence of the soft-component of the cosmic flux. For samples at zero depth and at sea-level the underestimation can be as large as ~0.1 Gy ka^{-1}. In a previous article, Prescott & Hutton (1988) give another approximation of the equations in Barbouti & Rastin (1983) in the form of

$$\dot{D}_c = 0.21\, e^{(-0.07\, x\, +\, 5\times10^{-4}\, x^2)} \qquad (1)$$

where $\dot{D}_c$ is the cosmic dose rate in Gy ka^{-1} and x is the depth in hg cm^{-2}. This expression is valid for depths between 150 g cm^{-2} and 5000 g cm^{-2}. For shallower depths (<150 g cm^{-2}) the cosmic dose rate must be read from Figure 1 in Prescott & Hutton (1988). As a result, `Luminescence::calc_CosmicDoseRate()` employs Equation 2 of Prescott & Hutton (1994) only for

[16] http://jscolor.com/, accessed: 2016-11-18.

[17] To obtain the depth in units of centimeters values given in g cm^{-2} must be divided by the material's density (in g cm^{-3}). Example: In a sediment of density 1.8 g cm^{-3}, 167 g cm^{-2} equates to a sample depth of ~93 cm.

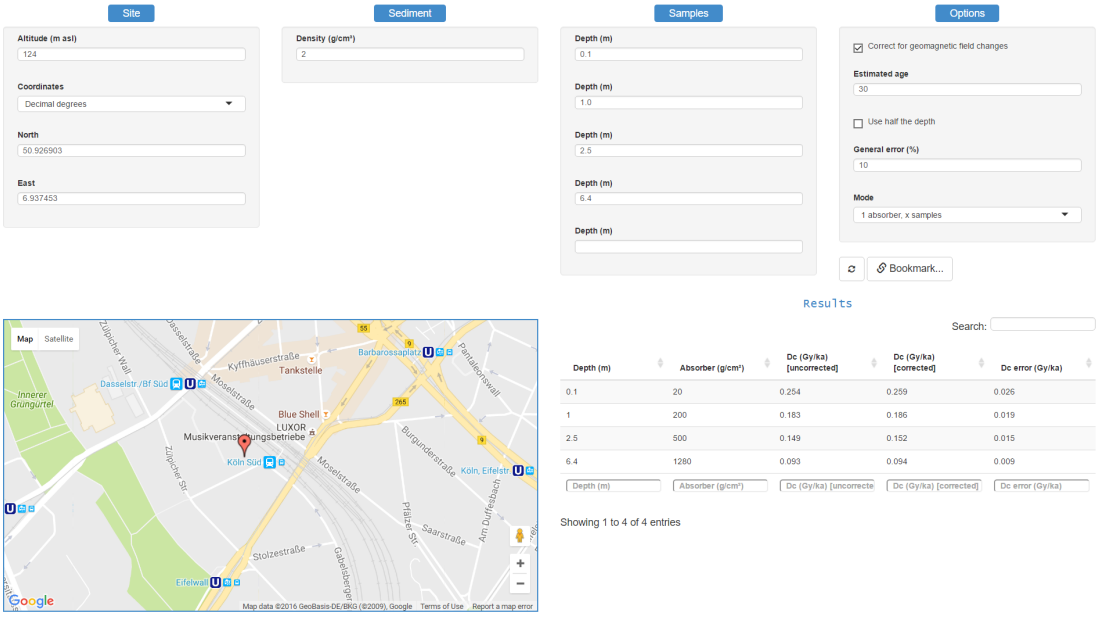

Figure 5: Shiny application for calculating the cosmic dose rate using the **R** function `Luminescence::calc_CosmicDose()`. In contrast to most other applications in 'RLumShiny' this application only provides numerical output (bottom right). A unique feature of this application is the use of Google Maps™ (using the 'googleVis' package (Gesmann & de Castillo, 2011)), which serves as a visual control to whether the provided longitude and latitude are correct.

depths >167 g cm^{-2}, i.e., only for the hard-component of the cosmic flux. Cosmic dose rate values for depths <167 g cm^{-2} were carefully reproduced from Figure 1 in Prescott & Hutton (1988) and fitted with a 6-degree polynomial curve. When the user provides a sample depth smaller than 167 g cm^{-2} the cosmic dose rate is thus estimated from the fitted curve instead.

With regards to the shiny application for `Luminescence::calc_CosmicDoseRate()` its comparatively small number of arguments (n = 9) favoured a "flatter" design, i.e., making all options available in one panel without separate tabs (Fig. 5). This resulted in a horizontally aligned interface, with the user input at the top and the output at the bottom. A unique feature of this application is the implementation of a Google Map by using the 'googleVis' package (Gesmann & de Castillo, 2011). As the latitude and longitude are compulsory for calculating the cosmic dose rate, the provided values are also used for finding the place on the Google Map. This serves as a visual control as to whether the provided values are correct. Finally, the application enhances the underlying `Luminescence::calc_CosmicDoseRate()` by allowing the user to provide the longitude and latitude in different coordinate formats, which are internally converted to decimal degrees as required by the function.

4. Discussion

This contribution introduced so-called shiny applications, which provide a graphical user interface to a selected number of functions of the **R** package 'Luminescence'. Built using the 'shiny' framework, the user is presented a scalable and intuitive GUI allowing for direct manipulation and interaction with the underlying **R** functions.

While we are confident that these applications lower the entry threshold for users new to **R** or the **R** package 'Luminescence', installing and using the 'RLumShiny' package in a local **R** environment is still not (and probably never will be) as straightforward as standalone software such as *Analyst* (Duller, 2015) or *RadialPlotter* (Vermeesch, 2009). The user is still required to install **R**, an IDE (e.g., *RStudio*) and finally all required **R** packages, notably 'Luminescence' and 'RLumShiny'. Usage of 'RLumShiny' is kept as easy as possible, however, as the user only needs to remember one single function (`app_RLum()`) and the keywords given in Table 1 to start a particular application.

Ideally, shiny applications are shared as web applications served by a dedicated server running **R** and *Shiny Server*. While this requires potentially investing in a corresponding infrastructure (e.g., renting web space) and a person experienced in setting up and maintaining a Linux server, the aforementioned drawbacks are largely eliminated. Once a Shiny Server is up and running, all the users need to do is to enter a specific URL in a HTML 5 compatible web browser; a local **R** environment is no longer needed. Furthermore, access permissions to the shiny applications can be controlled by the server administrator.

A general limitation to providing a GUI to **R** in general and the 'Luminescence' package in particular is that the user is always limited to the options provided by the GUI. For example, the function `Luminescence::plot_AbanicoPlot()` accepts a theoretically infinite number of input data, but is restricted to two data sets in the shiny application; otherwise, the GUI would become too convoluted. The user is thus required to revert to the CLI when the GUI does not provide the necessary means to fulfil the desired task. In essence, a CLI will always be more powerful than a GUI.

A specific limitation of the 'RLumShiny' package is that the currently included applications cover only a fairly restricted amount of functions of the 'Luminescence' package. Most applications provide a GUI to plotting functions and to a few functions dedicated to very specific problems (such as calculating the cosmic dose rate or transforming CW-OSL curves). There is no application dedicated to the analysis of raw luminescence data (e.g., `Luminescence::analyse_SAR.CWOSL()`) yet, which, without doubt, would also profit from a GUI. It is the aim of the authors to develop more shiny applications in the future. We may also invite other **R** users to contribute and further improve the package.

In summary, we believe the presented shiny applications, bundled in the **R** package 'RLumShiny', are a welcome contribution to the luminescence community and a useful addition to the **R** package 'Luminescence'. It is designed to be used by both users with, and without, prior knowledge of **R**.

5. Conclusion

The authors of the **R** package 'Luminescence' (Kreutzer et al., 2016) are fully aware that, despite its capabilities for complex and non-standard analysis of luminescence data, working with the command-line interface of **R** can be tedious at best and overwhelming at worst. Even though much work is put into simplifying the usage of the package to continuously lower the entry threshold, at least basic knowledge of **R** will always be required. Thus, the potential user base of the package cannot be exhausted, at least as long as the CLI is the only means of utilising the 'Luminescence' package.

As an alternative to the CLI, a graphical user interface allows for direct, interactive manipulation and interaction with the underlying software. For users with little or no experience with command-lines a GUI offers intuitive access that counteracts the perceived steep learning curve of a CLI (Unwin & Hofmann, 1999). To account for the demand of a GUI for the **R** package 'Luminescence' we presented a series of so-called shiny applications. These applications are built using the 'shiny' framework (Chang et al., 2016), which allows building a HTML, CSS and JavaScript based GUI straight from **R**. These applications are bundled in the **R** package 'RLumShiny' (Burow, 2016), which is freely available either through the CRAN (`https://CRAN.R-project.org/package=RLumShiny`) or from GitHub (`https://github.com/R-Lum/RLumShiny`).

The shiny applications included in 'RLumShiny' can be (i) used on a local computer with a working **R** environment, or (ii) shared as web applications with a wider audience (e.g., an organisation or working group) by setting up a Shiny Server. A Shiny Server run by the authors of this article can be freely accessed under http://shiny.r-luminescence.de. Note, however, that the performance of this server is fairly limited and not indicative for the general performance of shiny applications.

The current version of 'RLumShiny' (v0.1.1) includes a total of seven applications providing a GUI to ten functions of the 'Luminescence' package. Hence, there are many more functions that may greatly benefit from a GUI, and it is the aim of the authors to provide more shiny applications in the future. Finally, herewith we invite everyone to contribute to this package. 'RLumShiny' and the included JavaScript library *JSColor* (Odvarko, 2014) are licensed under the GNU General Public License version 3 (GPL-3). Code derived from the 'shinysky' package (AnalytixWare, 2014) is covered by the MIT licence.

Acknowledgments

We are thankful to the **R** Core Team for providing the **R** programming environment (R Core Team 2016) and the CRAN mirrors for open access to **R** packages. Cooperation and personal exchange between the package developers is gratefully funded by the DFG (SCHM3051/3-1) in the framework of the program 'Scientific Networks'. The work of SK is financed by a programme supported by the ANR (n°ANR-10-LABX-52).

References

Adler, J. *R in a Nutshell*. Oreilly & Associates Incorporated, 2nd edition, 2012.

AnalytixWare. *shinysky: A set of Shiny UI components/widgets.*, 2014. URL https://github.com/AnalytixWare/ShinySky. R package version 0.1.2.

Barbouti, A.I. and Rastin, B.C. *A study of the absolute intensity of muons at sea level and under various thicknesses of absorber*. Journal of Physics G: Nuclear and Particle Physics, 9: 1577–1595, 1983.

Burow, C. *RLumShiny: 'Shiny' Applications for the R Package 'Luminescence'*, 2016. URL https://CRAN.R-project.org/package=RLumShiny. R package version 0.1.1.

Burow, C., Kreutzer, S., Dietze, M., Fuchs, M.C., Fischer, M., Schmidt, C., and Brückner, H. *Shiny R.Lum - Interactive web applications for the R packages 'Luminescence' and 'ESR'*. In *German Luminescence- and ESR-Meeting 2014, Gießen.*, 2014. URL http://www.r-luminescence.de/grafik/poster_screenshots/2014_Poster_LED_Giessen_g.jpg. Poster presentation.

Chang, W., Cheng, J., Allaire, JJ., Xie, Y., and McPherson, J. *shiny: Web Application Framework for R*, 2016. URL https://CRAN.R-project.org/package=shiny. R package version 0.13.2.

Crawley, M.J. *The R Book*. Wiley, 2nd edition, 2012.

Dalgaard, P. *A Primer on the R-Tcl/Tk Package*. R News, 1(3): 27–31, 2001a.

Dalgaard, P. *The R-Tcl/Tk interface*. In Hornik, K. and Leisch, F. (eds.), *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, 2001b.

Dietze, M., Kreutzer, S., Fuchs, M.C., Burow, C., Fischer, M., and Schmidt, C. *A practical guide to the R package Luminescence*. Ancient TL, 31: 11–18, 2013.

Dietze, M., Kreutzer, S., Burow, C., Fuchs, M.C., Fischer, M., and Schmidt, C. *The abanico plot: visualising chronometric data with individual errors*. Quaternary Geochronology, 31: 12–18, 2016.

Duller, G.A.T. *The Analyst software package for luminescence data: overview and recent improvements*. Ancient TL, 33(1): 35–42, 2015.

Fox, J. *The R Commander: A Basic-Statistics Graphical User Interface to R*. Journal of Statistical Software, 14(9): 1–42, 2005.

Fox, J. *Using the R Commander: A Point-and-Click Interface for R*. Chapman and Hall/CRC, 2016.

Fox, J. and Bouchet-Valat, M. *Rcmdr: R Commander*, 2016. URL https://CRAN.R-project.org/package=Rcmdr. R package version 2.3-1.

Friedrich, J., Kreutzer, S., and Schmidt, C. *Solving ordinary differential equations to understand luminescence: RLumModel, an advanced research tool for simulating luminescence in quartz using R*. Quaternary Geochronology, 35: 88–100, 2016.

Fuchs, M.C., Kreutzer, S., Burow, C., Dietze, M., Fischer, M., Schmidt, C., and Fuchs, M. *Data processing in luminescence dating analysis: An exemplary workflow using the R package 'Luminescence'*. Quaternary International, 362: 8–13, 2015.

Galbraith, R. and Roberts, R.G. *Statistical aspects of equivalent dose and error calculation and display in OSL dating: An overview and some recommendations*. Quaternary Geochronology, 11: 1–27, 2012.

Galbraith, R.F. *Graphical Display of Estimates Having Differing Standard Errors*. Technometrics, 30: 271–281, 1988.

Gesmann, M. and de Castillo, D. *Using the Google Visualisation API with R*. The R Journal, 3(2): 40–44, December 2011.

Ihaka, R. and Gentleman, R. *R: A Language for Data Analysis*. Journal of Computational and Graphical Statistics, 5(3): 299–314, 1996.

Konrath, A.C., Henning, E., Walter, O.M.F.C, da Cunha Alves, C., and Samohyl, R.W. *Applications in teaching Statistical Quality Control with different R interfaces*. In *Global Engineering Education Conference (EDUCON), 2013 IEEE*, pp. 146–155, March 2013. doi: 10.1109/EduCon.2013.6530099.

Kreutzer, S., Schmidt, C., Fuchs, M.C., Dietze, M., Fischer, M., and Fuchs, M. *Introducing an R package for luminescence dating analysis*. Ancient TL, 30: 1–8, 2012.

Kreutzer, S., Dietze, M., Burow, C., Fuchs, M.C., Schmidt, C., Fischer, M., Friedrich, J., Mercier, N., Smedley, R.K., Durcan, J., and King, G. *Luminescence: Comprehensive Luminescence Dating Data Analysis*, 2016. URL https://CRAN.R-project.org/package=Luminescence. R package version 0.6.4.

Lawrence, M. and Temple Lang, D. *RGtk2: A Graphical User Interface Toolkit for R*. Journal of Statistical Software, 37(8): 1–52, 2010. URL http://www.jstatsoft.org/v37/i08/.

Ligges, U. *Programmieren mit R (Statistik und ihre Anwendungen)*. Springer, 3rd edition, 2008.

Odvarko, J. *jscolor - JavaScript Color Picker (v1.4.4)*. https://github.com/EastDesire/jscolor, 2014. GitHub repository.

Prescott, J.R. and Hutton, J.T. *Cosmic ray and gamma ray dosimetry for TL and ESR*. Nuclear Tracks and Radiation Measurements, 14: 223–227, 1988.

Prescott, J.R. and Hutton, J.T. *Cosmic ray contributions to dose rates for luminescence and ESR dating: large depths and long-term time variations*. Radiation Measurements, 23: 497–500, 1994.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2016. URL https://www.R-project.org/.

Robison-Cox, J. *Putting RGtk to Work*. In Hornik, K., Leisch, F., and Zeileis, A. (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, 2003.

RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., 2016. URL https://www.rstudio.com/.

Temple Lang, D. and Swayne, D.F. *GGobi meets R: an extensible environment for interactive dynamic data visualization*. In Hornik, K. and Leisch, F. (eds.), *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, 2001.

Temple Lang, D., Swayne, D., Wickham, H., and Lawrence, M. *rggobi: Interface Between R and 'GGobi'*, 2016. URL https://CRAN.R-project.org/package=rggobi. R package version 2.1.21.

Tippmann, S. *Programming tools: adventures with R*. Nature, 517: 109–110, 2015.

Unwin, A. and Hofmann, H. *GUI and Command-line - Conflict or Synergy?* In Berk, K. and Pourahmadi, M. (eds.), *Computing Science and Statistics*, Proceedings of the 31st Symposium on the Interface, pp. 246–253, 1999.

Urbanek, S. and Theus, M. *iPlots - High Interaction Graphics for R*. In Hornik, K., Leisch, F., and Zeileis, A. (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, 2003.

Urbanek, S. and Wichtrey, T. *iplots: iPlots - interactive graphics for R*, 2013. URL https://CRAN.R-project.org/package=iplots. R package version 1.1-7.

Valero-Mora, P. and Ledesma, R. *Graphical User Interfaces for R*. Journal of Statistical Software, 49(1): 1–8, 2012. ISSN 1548-7660. doi: 10.18637/jss.v049.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v049i01.

Vermeesch, P. *RadialPlotter: a Java application for fission track, luminescence and other radial plots*. Radiation Measurements, 44: 409–410, 2009. URL http://www.ucl.ac.uk/~ucfbpve/radialplotter/.

Verzani, J. *gWidgets: gWidgets API for building toolkit-independent, interactive GUIs*, 2014. URL https://CRAN.R-project.org/package=gWidgets. Based on the iwidgets code of S. Urbanek and suggestions by S. Urbanek and P. Grosjean and M. Lawrence. R package version 0.0-54.

Wagaman, A. S. *Meeting Student Needs for Multivariate Data Analysis: A Case Study in Teaching a Multivariate Data Analysis Course with No Pre-requisites*. ArXiv e-prints, 2013. URL https://arxiv.org/pdf/1310.7141v1.

Westbrooke, I. and Rohan, M. *Statistical Training in the Workplace*. In MacGillivray, H., Phillips, B., and Martin, M.A. (eds.), *Topics from Australian Conferences on Teaching Statistics*, pp. 311–327, New York, 2014. Springer New York. ISBN 978-1-4939-0603-1. doi: 10.1007/978-1-4939-0603-1_17.

Wickham, H. *Advanced R*. Chapman & Hall, 2014.

Xie, Y. *Interactive statistical graphics based on Qt*, 2013. URL https://github.com/ggobi/cranvas. R package version 0.8.5.

Reviewer

Shannon Mahan

Reviewer's comment

In this paper, Burow et al. present a graphical user interface (GUI) for the **R** package 'Luminescence' using the **R** 'shiny' package. **R** 'shiny' is a way to create GUIs for **R** functions and has the capability to be hosted as web applications. As many readers will already know, the **R** 'Luminescence' package is a very powerful tool for conducting statistical treatment of luminescence data and age determinations. One problem, that the authors also note, is that the command line interface of **R** is not very user friendly and can be frustrating and sometimes difficult for even users familiar with the language. The GUI helps avoid this issue and highlights many options in **R** that a user may not know or understand. Although some prior knowledge is necessary to run the GUI, and programs like *Analyst* and *Radialplotter* may always be more intuitive to use, the authors acknowledge this well and, in my opinion, do a good job of describing what 'RLumShiny' is and isn't capable of.