Package 'Luminescence'

September 9, 2016

Type Package

Title Comprehensive Luminescence Dating Data Analysis

Version 0.6.4

Date 2016-09-09

Author Sebastian Kreutzer [aut, trl, cre],
Michael Dietze [aut],
Christoph Burow [aut, trl, dtc],
Margret C. Fuchs [aut],
Christoph Schmidt [aut],
Manfred Fischer [aut, trl],
Johannes Friedrich [aut],
Norbert Mercier [aut],
Rachel K. Smedley [aut],
Julie Durcan [aut],
Georgina King [aut],
Markus Fuchs [ths]

Maintainer Sebastian Kreutzer <sebastian.kreutzer@u-bordeaux-montaigne.fr>

Contact Package Developer Team <team@r-luminescence.de>

Description A collection of various R functions for the purpose of Luminescence dating data analysis. This includes, amongst others, data import, export, application of age models, curve deconvolution, sequence analysis and plotting of equivalent dose distributions.

License GPL-3

Depends R (>= 3.3.0), utils

LinkingTo Rcpp (>= 0.12.5), RcppArmadillo (>= 0.6.700.6.0)

Imports bbmle (>= 1.0.18), data.table (>= 1.9.6), httr (>= 1.1.0), matrixStats (>= 0.50.2), methods, Rcpp (>= 0.12.5), minpack.lm (>= 1.2-0), raster (>= 2.5-2), readxl (>= 0.1.1), shape (>= 1.4.2), parallel, XML (>= 3.98-1.4), zoo (>= 1.7-13)

Suggests RLumShiny (>= 0.1.0), RLumModel (>= 0.1.1), plotly (>= 3.4.13), rmarkdown (>= 0.9.6), rjags (>= 4-6), coda (>= 0.18-1), pander (>= 0.6.0), rstudioapi (>= 0.5)

URL https://CRAN.R-project.org/package=Luminescence

Collate 'Analyse_SAR.OSLdata.R' 'CW2pHMi.R' 'CW2pLM.R' 'CW2pLMi.R' 'CW2pPMi.R' 'Luminescence-package.R' 'RcppExports.R'

'replicate_RLum.R' 'RLum-class.R' 'names_RLum.R'
'structure_RLum.R' 'length_RLum.R' 'set_RLum.R' 'get_RLum.R'
'RLum.Analysis-class.R' 'RLum.Data-class.R' 'bin_RLum.Data.R'
'RLum.Data.Curve-class.R' 'RLum.Data.Image-class.R'
'RLum.Data.Spectrum-class.R' 'RLum.Results-class.R'
'Risoe.BINfileData2RLum.Analysis.R'
'Risoe.BINfileData2RLum.Data.Curve.R' 'set_Risoe.BINfileData.R'
'get_Risoe.BINfileData.R' 'RisoeBINfileData-class.R'
'Second2Gray.R' 'analyse_IRSAR.RF.R' 'analyse_SAR.CWOSL.R'
'analyse_SAR.TL.R' 'analyse_baSAR.R' 'analyse_pIRIRSequence.R'
'app_RLum.R' 'apply_CosmicRayRemoval.R'
'apply_EfficiencyCorrection.R' 'calc_AliquotSize.R'
'calc_CentralDose.R' 'calc_CommonDose.R'
'calc_CosmicDoseRate.R' 'calc_FadingCorr.R' 'calc_FastRatio.R'
'calc_FiniteMixture.R' 'calc_FuchsLang2001.R'
'calc_HomogeneityTest.R' 'calc_IEU.R' 'calc_MaxDose.R'
'calc_MinDose.R' 'calc_OSLLxTxRatio.R' 'calc_SourceDoseRate.R'
'calc_Statistics.R' 'calc_TLLxTxRatio.R'
'calc_ThermalLifetime.R' 'calc_gSGC.R'
'extract_IrradiationTimes.R' 'fit_CWCurve.R' 'fit_LMCurve.R'
'get_Layout.R' 'get_Quote.R' 'get_rightAnswer.R'
'internal_as.latex.table.R' 'internals_RLum.R'
'merge_RLum.Analysis.R' 'merge_RLum.Data.Curve.R'
'merge_RLum.R' 'merge_RLum.Results.R'
'merge_Risoe.BINfileData.R' 'methods_DRAC.R' 'methods_RLum.R'
'model_LuminescenceSignals.R' 'plot_AbanicoPlot.R'
'plot_DRTResults.R' 'plot_DetPlot.R'
'plot_FilterCombinations.R' 'plot_GrowthCurve.R'
'plot_Histogram.R' 'plot_KDE.R' 'plot_NRt.R'
'plot_RLum.Analysis.R' 'plot_RLum.Data.Curve.R'
'plot_RLum.Data.Image.R' 'plot_RLum.Data.Spectrum.R'
'plot_RLum.R' 'plot_RLum.Results.R' 'plot_RadialPlot.R'
'plot_Risoe.BINfileData.R' 'plot_ViolinPlot.R' 'read_BIN2R.R'
'read_Daybreak2R.R' 'read_SPE2R.R' 'read_XSYG2R.R'
'report_RLum.R' 'template_DRAC.R' 'tune_Data.R' 'use_DRAC.R'
'verify_SingleGrainData.R' 'write_R2BIN.R' 'zzz.R'

RoxygenNote 5.0.1

NeedsCompilation yes

R topics documented:

Luminescence-package 4
analyse_baSAR 6
analyse_IRSAR.RF 13
analyse_pIRIRSequence 19
analyse_SAR.CWOSL 22
Analyse_SAR.OSLdata 25
analyse_SAR.TL 28
apply_CosmicRayRemoval 30
apply_EfficiencyCorrection 32
app_RLum 34

as . . . 34
BaseDataSet.CosmicDoseRate . . . 35
bin_RLum.Data . . . 37
calc_AliquotSize . . . 39
calc_CentralDose . . . 41
calc_CommonDose . . . 43
calc_CosmicDoseRate . . . 45
calc_FadingCorr . . . 49
calc_FastRatio . . . 52
calc_FiniteMixture . . . 54
calc_FuchsLang2001 . . . 57
calc_gSGC . . . 59
calc_HomogeneityTest . . . 61
calc_IEU . . . 62
calc_MaxDose . . . 64
calc_MinDose . . . 66
calc_OSLLxTxRatio . . . 71
calc_SourceDoseRate . . . 75
calc_Statistics . . . 77
calc_ThermalLifetime . . . 79
calc_TLLxTxRatio . . . 81
CW2pHMi . . . 83
CW2pLM . . . 87
CW2pLMi . . . 88
CW2pPMi . . . 91
ExampleData.BINfileData . . . 94
ExampleData.CW_OSL_Curve . . . 96
ExampleData.DeValues . . . 97
ExampleData.FittingLM . . . 98
ExampleData.LxTxData . . . 99
ExampleData.LxTxOSLData . . . 99
ExampleData.RLum.Analysis . . . 100
ExampleData.RLum.Data.Image . . . 101
ExampleData.XSYG . . . 102
extract_IrradiationTimes . . . 103
fit_CWCurve . . . 106
fit_LMCurve . . . 109
get_Layout . . . 113
get_Quote . . . 114
get_rightAnswer . . . 115
get_Risoe.BINfileData . . . 116
get_RLum . . . 117
length_RLum . . . 118
merge_Risoe.BINfileData . . . 119
merge_RLum . . . 120
merge_RLum.Analysis . . . 122
merge_RLum.Data.Curve . . . 123
merge_RLum.Results . . . 125
methods_RLum . . . 126
model_LuminescenceSignals . . . 130
names_RLum . . . 131
plot_AbanicoPlot . . . 132

plot_DetPlot . . . 139
plot_DRTResults . . . 141
plot_FilterCombinations . . . 144
plot_GrowthCurve . . . 147
plot_Histogram . . . 150
plot_KDE . . . 152
plot_NRt . . . 155
plot_RadialPlot . . . 158
plot_Risoe.BINfileData . . . 162
plot_RLum . . . 164
plot_RLum.Analysis . . . 165
plot_RLum.Data.Curve . . . 167
plot_RLum.Data.Image . . . 169
plot_RLum.Data.Spectrum . . . 171
plot_RLum.Results . . . 174
plot_ViolinPlot . . . 176
read_BIN2R . . . 177
read_Daybreak2R . . . 180
read_SPE2R . . . 181
read_XSYG2R . . . 183
replicate_RLum . . . 186
report_RLum . . . 187
Risoe.BINfileData-class . . . 190
Risoe.BINfileData2RLum.Analysis . . . 194
RLum-class . . . 196
RLum.Analysis-class . . . 197
RLum.Data-class . . . 200
RLum.Data.Curve-class . . . 201
RLum.Data.Image-class . . . 203
RLum.Data.Spectrum-class . . . 205
RLum.Results-class . . . 207
Second2Gray . . . 210
set_Risoe.BINfileData . . . 212
set_RLum . . . 213
sTeve . . . 214
structure_RLum . . . 215
template_DRAC . . . 216
tune_Data . . . 217
use_DRAC . . . 219
verify_SingleGrainData . . . 221
write_R2BIN . . . 223

Index **226**

`Luminescence-package` *Comprehensive Luminescence Dating Data Analysis*

Description

A collection of various R functions for the purpose of Luminescence dating data analysis. This includes, amongst others, data import, export, application of age models, curve deconvolution, sequence analysis and plotting of equivalent dose distributions.

Details

Package:	Luminescence
Type:	Package
Version:	0.6.4
Date:	2016-XX-XX
License:	GPL-3

Author(s)

Authors (alphabetic order)

Christoph Burow	University of Cologne, Germany
Michael Dietze	GFZ Helmholtz Centre Potsdam, Germany
Julie Durcan	University of Oxford, United Kingdom
Manfred Fischer	University of Bayreuth, Germany
Margret C. Fuchs	Helmholtz-Zentrum Dresden-Rossendorf, Helmholtz-Institute Freiberg for Resource Technology, Fr
Johannes Friedrich	University of Bayreuth, Germany
Georgina King	University of Cologne, Germany
Sebastian Kreutzer	IRAMAT-CRP2A, Universite Bordeaux Montaigne, France
Norbert Mercier	IRAMAT-CRP2A, Universite Bordeaux Montaigne, France
Christoph Schmidt	University of Bayreuth, Germany
Rachel K. Smedley	Aberystwyth University, United Kingdom

Beta-Tester

Thomas Kolb, University of Bayreuth, Germany

Supervisor

Markus Fuchs, Justus-Liebig-University Giessen, Germany

Support contact

<developers@r-luminescence.de>

We may further encourage the usage of our support forum. For this please visit our project website (link below).

Bug reporting

<bugtracker@r-luminescence.de>

Project website

http://www.r-luminescence.de

Project source code repository

https://github.com/R-Lum/Luminescence

Related package projects

https://cran.r-project.org/package=RLumShiny

http://shiny.r-luminescence.de
https://cran.r-project.org/package=RLumModel
http://model.r-luminescence.de

Package maintainer

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne, Pessac, France, <sebastian.kreutzer@u-bordeaux-montaigne.fr>

Acknowledgement

Cooperation and personal exchange between the developers is gratefully funded by the DFG (SCHM 3051/3-1) in the framework of the program "Scientific Networks". Project title: "Lum.Network: Ein Wissenschaftsnetzwerk zur Analyse von Lumineszenzdaten mit R" (2014-2017)

References

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

Dietze, M., Kreutzer, S., Burow, C., Fuchs, M.C., Fischer, M., Schmidt, C., 2016. The abanico plot: visualising chronometric data with individual standard errors. Quaternary Geochronology 31, 1-7. http://dx.doi.org/10.1016/j.quageo.2015.09.003

Fuchs, M.C., Kreutzer, S., Burow, C., Dietze, M., Fischer, M., Schmidt, C., Fuchs, M., 2015. Data processing in luminescence dating analysis: An exemplary workflow using the R package 'Luminescence'. Quaternary International, 362,8-13. http://dx.doi.org/10.1016/j.quaint.2014.06.034

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

Smedley, R.K., 2015. A new R function for the Internal External Uncertainty (IEU) model. Ancient TL 33, 16-21.

---

analyse_baSAR *Bayesian models (baSAR) applied on luminescence data*

Description

This function allows the application of Bayesian models on luminescence data, measured with the single-aliquot regenerative-dose (SAR, Murray and Wintle, 2000) protocol. In particular, it follows the idea proposed by Combes et al., 2015 of using an hierarchical model for estimating a central equivalent dose from a set of luminescence measurements. This function is (I) the adaption of this approach for the R environment and (II) an extension and a technical refinement of the published code.

Usage

```
analyse_baSAR(object, XLS_file = NULL, aliquot_range = NULL,
  source_doserate = NULL, signal.integral, signal.integral.Tx = NULL,
  background.integral, background.integral.Tx = NULL, sigmab = 0,
  sig0 = 0.025, distribution = "cauchy", baSAR_model = NULL,
  n.MCMC = 1e+05, fit.method = "EXP", fit.force_through_origin = TRUE,
  fit.includingRepeatedRegPoints = TRUE, method_control = list(),
  digits = 3L, plot = TRUE, plot_reduced = TRUE, plot.single = FALSE,
  verbose = TRUE, ...)
```

Arguments

object	Risoe.BINfileData or RLum.Results or character or list (**required**): input object used for the Bayesian analysis. If a character is provided the function assumes a file connection and tries to import a BIN-file using the provided path. If a list is provided the list can only contain either Risoe.BINfileData objects or characters providing a file connection. Mixing of both types is not allowed. If an RLum.Results is provided the function directly starts with the Bayesian Analysis (see details)
XLS_file	character (optional): XLS_file with data for the analysis. This file must contain 3 columns: the name of the file, the disc position and the grain position (the last being 0 for multi-grain measurements)
aliquot_range	numeric (optional): allows to limit the range of the aliquots used for the analysis. This argument has only an effect if the argument XLS_file is used or the input is the previous output (i.e. is RLum.Results). In this case the new selection will add the aliquots to the removed aliquots table.
source_doserate	numeric (**required**): source dose rate of beta-source used for the measuremnt and its uncertainty in Gy/s, e.g., source_doserate = c(0.12, 0.04). Paramater can be provided as list, for the case that more than one BIN-file is provided, e.g., source_doserate = list(c(0.04, 0.004), c(0.05, 0.004)).
signal.integral	vector (**required**): vector with the limits for the signal integral used for the calculation, e.g., signal.integral = c(1:5) Ignored if object is an RLum.Results object. The parameter can be provided as list, source_doserate.
signal.integral.Tx	vector (optional): vector with the limits for the signal integral for the Tx curve. If nothing is provided the value from signal.integral is used and it is ignored if object is an RLum.Results object. The parameter can be provided as list, see source_doserate.
background.integral	vector (**required**): vector with the bounds for the background integral. Ignored if object is an RLum.Results object. The parameter can be provided as list, see source_doserate.
background.integral.Tx	vector (optional): vector with the limits for the background integral for the Tx curve. If nothing is provided the value from background.integral is used. Ignored if object is an RLum.Results object. The parameter can be provided as list, see source_doserate.
sigmab	numeric (with default): option to set a manual value for the overdispersion (for LnTx and TnTx), used for the Lx/Tx error calculation. The value should be provided as absolute squared count values, cf. calc_OSLLxTxRatio. The parameter can be provided as list, see source_doserate.
sig0	numeric (with default): allow adding an extra component of error to the final Lx/Tx error value (e.g., instrumental errror, see details is calc_OSLLxTxRatio). The parameter can be provided as list, see source_doserate.
distribution	character (with default): type of distribution that is used during Bayesian calculations for determining the Central dose and overdispersion values. Allowed inputs are "cauchy", "normal" and "log_normal".
baSAR_model	character (optional): option to provide an own modified or new model for the Bayesian calculation (see details). If an own model is provided the argument distribution is ignored and set to 'user_defined'

n.MCMC	integer (with default): number of iterations for the Markov chain Monte Carlo (MCMC) simulations
fit.method	character (with default): fit method used for fitting the growth curve using the function plot_GrowthCurve. Here supported methods: EXP, EXP+LIN and LIN
fit.force_through_origin	logical (with default): force fitting through origin
fit.includingRepeatedRegPoints	logical (with default): includes the recycling point (assumed to be measured during the last cycle)
method_control	list (optional): named list of control parameters that can be directly passed to the Bayesian analysis, e.g., method_control = list(n.chains = 4). See details for further information
digits	integer (with default): round output to the number of given digits
plot	logical (with default): enables or disables plot output
plot_reduced	logical (with default): enables or disables the advanced plot output
plot.single	logical (with default): enables or disables single plots or plots arranged by analyse_baSAR
verbose	logical (with default): enables or disables verbose mode
...	parameters that can be passed to the function calc_OSLLxTxRatio (almost full support) read_excel (full support), read_BIN2R (n.records, position, duplicated.rm), see details.

Details

Internally the function consists of two parts: (I) The Bayesian core for the Bayesian calculations and applying the hierchical model and (II) a data pre-processing part. The Bayesian core can be run independently, if the input data are sufficient (see below). The data pre-processing part was implemented to simplify the analysis for the user as all needed data pre-processing is done by the function, i.e. in theory it is enough to provide a BIN/BINX-file with the SAR measurement data. For the Bayesian analysis for each aliquot the following information are needed from the SAR analysis. LxTx, the LxTx error and the dose values for all regeneration points.

How the systematic error contribution is calculated?

Standard errors (so far) provided with the source dose rate are considered as systematic uncertainties and added to final central dose by:

$$systematic.error = 1/n \sum SE(source.doserate)$$

$$SE(central.dose.final) = \sqrt{SE(central.dose)^2 + systematic.error^2}$$

Please note that this approach is rather rough and can only be valid if the source dose rate errors, in case different readers had been used, are similar. In cases where more than one source dose rate is provided a warning is given.

Input / output scenarios

Various inputs are allowed for this function. Unfortunately this makes the function handling rather complex, but at the same time very powerful. Available scenarios:

(1) - `object` **is BIN-file or link to a BIN-file**

Finally it does not matter how the information of the BIN/BINX file are provided. The function supports (a) either a path to a file or directory or a `list` of file names or paths or (b) a `Risoe.BINfileData` object or a list of these objects. The latter one can be produced by using the function `read_BIN2R`, but this function is called automatically if only a filename and/or a path is provided. In both cases it will become the data that can be used for the analysis.

`[XLS_file = NULL]`

If no XLS file (or data frame with the same format) is provided the functions runs an automatic process that consists of the following steps:

- Select all valid aliquots using the function `verify_SingleGrainData`
- Calculate Lx/Tx values using the function `calc_OSLLxTxRatio`
- Calculate De values using the function `plot_GrowthCurve`

These proceeded data are subsequently used in for the Bayesian analysis

`[XLS_file != NULL]`

If an XLS-file is provided or a `data.frame` providing similar information the pre-processing steps consists of the following steps:

- Calculate Lx/Tx values using the function `calc_OSLLxTxRatio`
- Calculate De values using the function `plot_GrowthCurve`

Means, the XLS file should contain a selection of the BIN-file names and the aliquots selected for the further analysis. This allows a manual selection of input data, as the automatic selection by `verify_SingleGrainData` might be not totally sufficient.

(2) - `object` `RLum.Results object`

If an `RLum.Results` object is provided as input and(!) this object was previously created by the function `analyse_baSAR()` itself, the pre-processing part is skipped and the function starts directly the Bayesian analysis. This option is very powerful as it allows to change parameters for the Bayesian analysis without the need to repeat the data pre-processing. If furthermore the argument `aliquot_range` is set, aliquots can be manually excluded based on previous runs.

`method_control`

These are arguments that can be passed directly to the Bayesian calculation core, supported arguments are:

Parameter	Type	Descritpion
`lower_centralD`	`numeric`	sets the lower bound for the expected De range. Change it only if you know what you
`upper_centralD`	`numeric`	sets the upper bound for the expected De range. Change it only if you know what you
`n.chains`	`integer`	sets number of parallel chains for the model (default = 3) (cf. `jags.model`)
`inits`	`list`	option to set initialisation values (cf. `jags.model`)
`thin`	`numeric`	thinning interval for monitoring the Bayesian process (cf. `jags.model`)
`variables.names`	`character`	set the variables to be monitored during the MCMC run, default: `'central_D'`, `'sigm`

User defined models

The function provides the option to modify and to define own models that can be used for the Bayesian calculation. In the case the user wants to modify a model, a new model can be piped into the funtion via the argument `baSAR_model` as `character`. The model has to be provided in the JAGS dialect of the BUGS language (cf. `jags.model`) and parameter names given with the pre-defined names have to be respected, otherwise the function will break.

FAQ

Q: How can I set the seed for the random number generator (RNG)?
A: Use the argument `method_control`, e.g., for three MCMC chains (as it is the default):

```
method_control = list( inits = list(  list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 1), list(
))
```

This sets a reproducible set for every chain separately.

Q: How can I modify the output plots?
A: You can't, but you can use the function output to create own, modified plots.

Q: Can I change the boundaries for the central_D?
A: Yes, we made it possible, but we DO NOT recommend it, except you know what you are doing!
Example: `method_control = list(lower_centralD = 10))`

Additional arguments support via the ... argument

This list summarizes the additional arguments that can be passed to the internally used functions.

Supported argument	Corresponding function	Default	Short description
threshold	verify_SingleGrainData	30	change rejection threshold for cu
sheet	read_excel	1	select XLS-sheet for import
col_names	read_excel	TRUE	first row in XLS-file is header
col_types	read_excel	NULL	limit import to specific columns
skip	read_excel	0	number of rows to be skipped du
n.records	read_BIN2R	NULL	limit records during BIN-file imp
duplicated.rm	read_BIN2R	TRUE	remove duplicated records in the
pattern	read_BIN2R	TRUE	select BIN-file by name pattern
position	read_BIN2R	NULL	limit import to a specific position
background.count.distribution	calc_OSLLxTxRatio	"non-poisson"	set assumed count distribution
fit.weights	plot_GrowthCurve	TRUE	enables / disables fit weights
fit.bounds	plot_GrowthCurve	TRUE	enables / disables fit bounds
NumberIterations.MC	plot_GrowthCurve	100	number of MC runs for error calc
output.plot	plot_GrowthCurve	TRUE	enables / disables dose response
output.plotExtended	plot_GrowthCurve	TRUE	enables / disables extended dose

Value

Function returns results numerically and graphically:

[NUMERICAL OUTPUT]

RLum.Reuslts-object

slot: `@data`

Element	Type	Description
`$summary`	`data.frame`	statistical summary, including the central dose
`$mcmc`	`mcmc`	object including raw output of `rjags`
`$models`	`character`	implemented models used in the baSAR-model core
`$input_object`	`data.frame`	summarising table (same format as the XLS-file) including, e.g., Lx/Tx values
`$removed_aliquots`	`data.frame`	table with removed aliquots (e.g., NaN, or Inf Lx/Tx values). If nothing was remov

slot: `@info`

The original function call

[PLOT OUTPUT]

- (A) Ln/Tn curves with set integration limits,
- (B) trace plots are returned by the baSAR-model, showing the convergence of the parameters (trace) and the resulting kernel density plots. If `plot_reduced = FALSE` for every(!) dose a trace and a density plot is returned (this may take a long time),
- (C) dose plots showing the dose for every aliquot as boxplots and the marked HPD in within. If boxes are coloured 'orange' or 'red' the aliquot itself should be checked,
- (D) the dose response curve resulting from the monitoring of the Bayesian modelling are provided along with the Lx/Tx values and the HPD. Note: The amount for curves displayed is limited to 1000 (random choice) for performance reasons,
- (E) the final plot is the De distribution as calculated using the conventional approach and the central dose with the HPDs marked within.

Please note: If distribution was set to `log_normal` the central dose is given as geometric mean!

Function version

0.1.25 (2016-09-09 10:32:17)

Note

If you provide more than one BIN-file, it is **strongly** recommanded to provide a `list` with the same number of elements for the following parameters:
`source_doserate`, `signal.integral`, `signal.integral.Tx`, `background.integral`, `background.integral.Tx`, `sigmab`, `sig0`.

Example for two BIN-files: `source_doserate = list(c(0.04, 0.006), c(0.05, 0.006))`

The function is currently limited to work with standard Risoe BIN-files only!

Author(s)

Norbert Mercier, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

The underlying Bayesian model based on a contribution by Combes et al., 2015.
R Luminescence Package Team

References

Combes, B., Philippe, A., Lanos, P., Mercier, N., Tribolo, C., Guerin, G., Guibert, P., Lahaye, C., 2015. A Bayesian central equivalent dose model for optically stimulated luminescence dating. Quaternary Geochronology 28, 62-70. doi:10.1016/j.quageo.2015.04.001

Further reading

Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B., 2013. Bayesian Data Analysis, Third Edition. CRC Press.

Murray, A.S., Wintle, A.G., 2000. Luminescence dating of quartz using an improved single-aliquot regenerative-dose protocol. Radiation Measurements 32, 57-73. doi:10.1016/S1350-4487(99)00253-X

See Also

read_BIN2R, calc_OSLLxTxRatio, plot_GrowthCurve, read_excel, verify_SingleGrainData, jags.model, coda.samples, boxplot.default

Examples

```
##(1) load package test data set
data(ExampleData.BINfileData, envir = environment())

##(2) selecting relevant curves, and limit dataset
CWOSL.SAR.Data <- subset(
 CWOSL.SAR.Data,
 subset = POSITION%in%c(1:3) & LTYPE == "OSL")

## Not run:
##(3) run analysis
##please not that the here selected parameters are
##choosen for performance, not for reliability
results <- analyse_baSAR(
 object = CWOSL.SAR.Data,
 source_doserate = c(0.04, 0.001),
 signal.integral = c(1:2),
 background.integral = c(80:100),
 fit.method = "LIN",
 plot = FALSE,
 n.MCMC = 200

)

print(results)


##XLS_file template
```

```
##copy and paste this the code below in the terminal
##you can further use the function write.csv() to export the example

XLS_file <-
structure(
list(
 BIN_FILE = NA_character_,
 DISC = NA_real_,
 GRAIN = NA_real_),
   .Names = c("BIN_FILE", "DISC", "GRAIN"),
   class = "data.frame",
   row.names = 1L
)


## End(Not run)
```

`analyse_IRSAR.RF` *Analyse IRSAR RF measurements*

Description

Function to analyse IRSAR RF measurements on K-feldspar samples, performed using the protocol according to Erfurt et al. (2003) and beyond.

Usage

```
analyse_IRSAR.RF(object, sequence_structure = c("NATURAL", "REGENERATED"),
  RF_nat.lim = NULL, RF_reg.lim = NULL, method = "FIT",
  method.control = NULL, test_parameters = NULL, n.MC = 10,
  txtProgressBar = TRUE, plot = TRUE, plot_reduced = FALSE, ...)
```

Arguments

`object`	`RLum.Analysis` or a `list` of `RLum.Analysis` objects (**required**): input object containing data for protocol analysis. The function expects to find at least two curves in the `RLum.Analysis` object: (1) RF_nat, (2) RF_reg. If a `list` is provided as input all other parameters can be provided as `list` as well to gain full control.
`sequence_structure`	`vector character` (with default): specifies the general sequence structure. Allowed steps are `NATURAL`, `REGENERATED`. In addition any other character is allowed in the sequence structure; such curves will be ignored during the analysis.
`RF_nat.lim`	`vector` (with default): set minimum and maximum channel range for natural signal fitting and sliding. If only one value is provided this will be treated as minimum value and the maximum limit will be added automatically.
`RF_reg.lim`	`vector` (with default): set minimum and maximum channel range for regenerated signal fitting and sliding. If only one value is provided this will be treated as minimum value and the maximum limit will be added automatically.
`method`	`character` (with default): setting method applied for the data analysis. Possible options are `"FIT"` or `"SLIDE"`.

`method.control` list (optional): parameters to control the method, that can be passed to the choosen method. These are for (1) `method = "FIT"`: 'trace', 'maxiter', 'warnOnly', 'minFactor' and for (2) `method = "SLIDE"`: 'correct_onset', 'show_density', 'show_fit', 'trace'. See details.

`test_parameters` list (with default): set test parameters. Supported parameters are: `curves_ratio`, `residuals_slope` (only for `method = "SLIDE"`), `curves_bounds`, `dynamic_ratio`, `lambda`, `beta` and `delta.phi`. All input: `numeric` values, `NA` and `NULL` (s. Details)

(see Details for further information)

`n.MC` numeric (with default): set number of Monte Carlo runs for start parameter estimation (`method = "FIT"`) or error estimation (`method = "SLIDE"`). Note: Large values will significantly increase the computation time

`txtProgressBar` logical (with default): enables `TRUE` or disables `FALSE` the progression bar during MC runs

`plot` logical (with default): plot output (`TRUE` or `FALSE`)

`plot_reduced` logical (optional): provides a reduced plot output if enabled to allow common R plot combinations, e.g., `par(mfrow(...))`. If `TRUE` no residual plot is returned; it has no effect if `plot = FALSE`

`...` further arguments that will be passed to the plot output. Currently supported arguments are `main`, `xlab`, `ylab`, `xlim`, `ylim`, `log`, `legend` (`TRUE/FALSE`), `legend.pos`, `legend.text` (passes argument to x,y in `legend`), `xaxt`

Details

The function performs an IRSAR analysis described for K-feldspar samples by Erfurt et al. (2003) assuming a negligible sensitivity change of the RF signal.

General Sequence Structure (according to Erfurt et al. (2003))

1. Measuring IR-RF intensity of the natural dose for a few seconds (RF_{nat})
2. Bleach the samples under solar conditions for at least 30 min without changing the geometry
3. Waiting for at least one hour
4. Regeneration of the IR-RF signal to at least the natural level (measuring (RF_{reg})
5. Fitting data with a stretched exponential function
6. Calculate the the palaeodose D_e using the parameters from the fitting

Actually two methods are supported to obtain the D_e: `method = "FIT"` and `method = "SLIDE"`:

`method = "FIT"`

The principle is described above and follows the original suggestions by Erfurt et al., 2003. For the fitting the mean count value of the RF_nat curve is used.

Function used for the fitting (according to Erfurt et al. (2003)):

$$\phi(D) = \phi_0 - \Delta\phi(1 - exp(-\lambda * D))^{\beta}$$

with $\phi(D)$ the dose dependent IR-RF flux, ϕ_0 the inital IR-RF flux, $\Delta\phi$ the dose dependent change of the IR-RF flux, λ the exponential parameter, D the dose and β the dispersive factor.

To obtain the palaeodose D_e the function is changed to:

$$D_e = ln(-(\phi(D) - \phi_0)/(-\lambda * \phi)^{1/\beta} + 1)/ - \lambda$$

The fitting is done using the `port` algorithm of the `nls` function.

`method = "SLIDE"`

For this method the natural curve is slided along the x-axis until congruence with the regenerated curve is reached. Instead of fitting this allows to work with the original data without the need of any physical model. This approach was introduced for RF curves by Buylaert et al., 2012 and Lapp et al., 2012.

Here the sliding is done by searching for the minimum of the squared residuals.

`method.control`

To keep the generic argument list as clear as possible, arguments to control the methods for De estimation are all preset with meaningful default parameters and can be handled using the argument `method.control` only, e.g., `method.control = list(trace = TRUE)`. Supported arguments are:

ARGUMENT	METHOD	DESCRIPTION
`trace`	`FIT`, `SLIDE`	as in `nls`; shows sum of squared residuals
`maxiter`	`FIT`	as in `nls`
`warnOnly`	`FIT`	as in `nls`
`minFactor`	`FIT`	as in `nls`
`correct_onset`	`SLIDE`	The logical argument literally spoken, shifts the curves along the x-axis by the first chan
`show_density`	`SLIDE`	`logical` (with default) enables or disables KDE plots for MC run results. If the distribut
`show_fit`	`SLIDE`	`logical` (with default) enables or disables the plot of the fitted curve rountinly obtained
`n.MC`	`SLIDE`	`integer` (wiht default): This controls the number of MC runs within the sliding (assesin

Error estimation

For `method = "FIT"` the asymmetric error range is obtained by using the 2.5 % (lower) and the 97.5 % (upper) quantiles of the RF_{nat} curve for calculating the D_e error range.

For `method = "SLIDE"` the error is obtained by bootstrapping the residuals of the slided curve to construct new natural curves for a Monte Carlo simulation. The error is returned in two ways: (a) the standard deviation of the herewith obtained D_e from the MC runs and (b) the confidence interval using the 2.5 % (lower) and the 97.5 % (upper) quantiles. The results of the MC runs are returned with the function output.

Test parameters

The argument `test_parameters` allows to pass some thresholds for several test parameters, which will be evaluated during the function run. If a threshold is set and it will be exceeded the test parameter status will be set to "FAILED". Intentionally this parameter is not termed 'rejection

criteria' as not all test parameters are evaluated for both methods and some parameters are calculated by not evaluated by default. Common for all parameters are the allowed argument options NA and NULL. If the parameter is set to NA the value is calculated but the result will not be evaluated, means it has no effect on the status ("OK" or "FAILED") of the parameter. Setting the parameter to NULL disables the parameter entirely and the parameter will be also removed from the function output. This might be useful in cases where a particular parameter asks for long computation times. Currently supported parameters are:

`curves_ratio` `numeric` (default: `1.001`):

The ratio of RF_{nat} over RF_{reg} in the range ofRF_{nat} of is calculated and should not exceed the threshold value.

`intersection_ratio` `numeric` (default: `NA`):

Calculated as absolute difference from 1 of the ratio of the integral of the normalised RF-curves, This value indicates intersection of the RF-curves and should be close to 0 if the curves have a similar shape. For this calculation first the corresponding time-count pair value on the RF_reg curve is obtained using the maximum count value of the RF_nat curve and only this segment (fitting to the RF_nat curve) on the RF_reg curve is taken for further calculating this ratio. If nothing is found at all, `Inf` is returned.

`residuals_slope` `numeric` (default: `NA`; only for `method = "SLIDE"`):

A linear function is fitted on the residuals after sliding. The corresponding slope can be used to discard values as a high (positive, negative) slope may indicate that both curves are fundamentally different and the method cannot be applied at all. Per default the value of this parameter is calculated but not evaluated.

`curves_bounds` `numeric` (default: $max(RF_{reg_counts})$):

This measure uses the maximum time (x) value of the regenerated curve. The maximum time (x) value of the natural curve cannot be larger than this value. However, although this is not recommended the value can be changed or disabled.

`dynamic_ratio` `numeric` (default: `NA`):

The dynamic ratio of the regenerated curve is calculated as ratio of the minimum and maximum count values.

`lambda`, `beta` and `delta.phi` `numeric` (default: `NA`; `method = "SLIDE"`):

The stretched exponential function suggested by Erfurt et al. (2003) describing the decay of the RF signal, comprises several parameters that might be useful to evaluate the shape of the curves. For `method = "FIT"` this parameter is obtained during the fitting, for `method = "SLIDE"` a rather rough estimation is made using the function `nlsLM` and the equation given above. Note: As this procedure requests more computation time, setting of one of these three parameters to NULL also prevents a calculation of the remaining two.

Value

A plot (optional) and an RLum.Results object is returned:

@data
$ data: data.frame table with De and corresponding values
..$ DE : numeric: the obtained equivalent dose
..$ DE.ERROR : numeric: (only method = "SLIDE") standard deviation obtained from MC runs
..$ DE.LOWER : numeric: 2.5% quantile for De values obtained by MC runs
..$ DE.UPPER : numeric: 97.5% quantile for De values obtained by MC runs
..$ DE.STATUS : character: test parameter status
..$ RF_NAT.LIM : charcter: used RF_nat curve limits
..$ RF_REG.LIM : character: used RF_reg curve limits
..$ POSITION : integer: (optional) position of the curves
..$ DATE : character: (optional) measurement date
..$ SEQUENCE_NAME : character: (optional) sequence name
..$ UID : character: unique data set ID
$ test_parameters : data.frame table test parameters
$ fit : nls nlsModel object
$ slide : list data from the sliding process, including the sliding matrix

@info
$ call : language-class: the orignal function call

The output (data) should be accessed using the function get_RLum

Function version

0.6.11 (2016-07-16 11:28:11)

Note

[THIS FUNCTION HAS BETA-STATUS]

This function assumes that there is no sensitivity change during the measurements (natural vs. regenerated signal), which is in contrast to the findings from Buylaert et al. (2012). Furthermore: In course of ongoing research this function has been almost fully re-written, but further thoughtful tests are still pending! However, as a lot new package functionality was introduced with the changes made for this function and to allow a part of such tests the re-newed code was made part of the current package.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Buylaert, J.P., Jain, M., Murray, A.S., Thomsen, K.J., Lapp, T., 2012. IR-RF dating of sand-sized K-feldspar extracts: A test of accuracy. Radiation Measurements 44 (5-6), 560-565. doi: 10.1016/j.radmeas.2012.06.021

Erfurt, G., Krbetschek, M.R., 2003. IRSAR - A single-aliquot regenerative-dose dating protocol applied to the infrared radiofluorescence (IR-RF) of coarse- grain K-feldspar. Ancient TL 21, 35-42.

Erfurt, G., 2003. Infrared luminescence of Pb+ centres in potassium-rich feldspars. physica status solidi (a) 200, 429-438.

Erfurt, G., Krbetschek, M.R., 2003. Studies on the physics of the infrared radioluminescence of potassium feldspar and on the methodology of its application to sediment dating. Radiation Measurements 37, 505-510.

Erfurt, G., Krbetschek, M.R., Bortolot, V.J., Preusser, F., 2003. A fully automated multi-spectral radioluminescence reading system for geochronometry and dosimetry. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 207, 487-499.

Lapp, T., Jain, M., Thomsen, K.J., Murray, A.S., Buylaert, J.P., 2012. New luminescence measurement facilities in retrospective dosimetry. Radiation Measurements 47, 803-808. doi:10.1016/j.radmeas.2012.02.006

Trautmann, T., 2000. A study of radioluminescence kinetics of natural feldspar dosimeters: experiments and simulations. Journal of Physics D: Applied Physics 33, 2304-2310.

Trautmann, T., Krbetschek, M.R., Dietrich, A., Stolz, W., 1998. Investigations of feldspar radioluminescence: potential for a new dating technique. Radiation Measurements 29, 421-425.

Trautmann, T., Krbetschek, M.R., Dietrich, A., Stolz, W., 1999. Feldspar radioluminescence: a new dating method and its physical background. Journal of Luminescence 85, 45-58.

Trautmann, T., Krbetschek, M.R., Stolz, W., 2000. A systematic study of the radioluminescence properties of single feldspar grains. Radiation Measurements 32, 685-690.

See Also

RLum.Analysis, RLum.Results, get_RLum, nls, nlsLM

Examples

```
##load data
data(ExampleData.RLum.Analysis, envir = environment())

##(1) perform analysis using the method 'FIT'
results <- analyse_IRSAR.RF(object = IRSAR.RF.Data)

##show De results and test paramter results
get_RLum(results, data.object = "data")
get_RLum(results, data.object = "test_parameters")

##(2) perform analysis using the method 'SLIDE'
results <- analyse_IRSAR.RF(object = IRSAR.RF.Data, method = "SLIDE", n.MC = 1)

## Not run:
##(3) perform analysis using the method 'SLIDE' and method control option
## 'trace
results <- analyse_IRSAR.RF(
 object = IRSAR.RF.Data,
 method = "SLIDE",
 method.control = list(trace = TRUE))
```

```
## End(Not run)
```

`analyse_pIRIRSequence` *Analyse post-IR IRSL sequences*

Description

The function performs an analysis of post-IR IRSL sequences including curve fitting on `RLum.Analysis` objects.

Usage

```
analyse_pIRIRSequence(object, signal.integral.min, signal.integral.max,
  background.integral.min, background.integral.max, dose.points = NULL,
  sequence.structure = c("TL", "IR50", "pIRIR225"), plot = TRUE,
  plot.single = FALSE, ...)
```

Arguments

`object`	`RLum.Analysis` (**required**) or `list` of `RLum.Analysis` objects: input object containing data for analysis. If a `list` is provided the functions tries to iteratre over the list.
`signal.integral.min`	`integer` (**required**): lower bound of the signal integral. Provide this value as vector for different integration limits for the different IRSL curves.
`signal.integral.max`	`integer` (**required**): upper bound of the signal integral. Provide this value as vector for different integration limits for the different IRSL curves.
`background.integral.min`	`integer` (**required**): lower bound of the background integral. Provide this value as vector for different integration limits for the different IRSL curves.
`background.integral.max`	`integer` (**required**): upper bound of the background integral. Provide this value as vector for different integration limits for the different IRSL curves.
`dose.points`	`numeric` (optional): a numeric vector containing the dose points values. Using this argument overwrites dose point values in the signal curves.
`sequence.structure`	vector character (with default): specifies the general sequence structure. Allowed values are `"TL"` and any `"IR"` combination (e.g., `"IR50"`,`"pIRIR225"`). Additionally a parameter `"EXCLUDE"` is allowed to exclude curves from the analysis (Note: If a preheat without PMT measurement is used, i.e. preheat as non TL, remove the TL step.)
`plot`	`logical` (with default): enables or disables plot output.
`plot.single`	`logical` (with default): single plot output (`TRUE/FALSE`) to allow for plotting the results in single plot windows. Requires `plot = TRUE`.
`...`	further arguments that will be passed to the function `analyse_SAR.CWOSL` and `plot_GrowthCurve`

Details

To allow post-IR IRSL protocol (Thomsen et al., 2008) measurement analyses this function has been written as extended wrapper function for the function `analyse_SAR.CWOSL`, facilitating an entire sequence analysis in one run. With this, its functionality is strictly limited by the functionality of the function `analyse_SAR.CWOSL`.

If the input is a `list`

If the input is a list of RLum.Analysis-objects, every argument can be provided as list to allow for different sets of parameters for every single input element. For further information see `analyse_SAR.CWOSL`.

Value

Plots (optional) and an `RLum.Results` object is returned containing the following elements:

DATA.OBJECT	TYPE	DESCRIPTION
`..$data :`	`data.frame`	Table with De values
`..$LnLxTnTx.table :`	`data.frame`	with the LnLxTnTx values
`..$rejection.criteria :`	`data.frame`	rejection criteria
`..$Formula :`	`list`	Function used for fitting of the dose response curve
`..$call :`	`call`	the original function call

The output should be accessed using the function `get_RLum`.

Function version

0.2.2 (2016-09-07 11:37:34)

Note

Best graphical output can be achieved by using the function `pdf` with the following options:
`pdf(file = "...", height = 15, width = 15)`

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Murray, A.S., Wintle, A.G., 2000. Luminescence dating of quartz using an improved single-aliquot regenerative-dose protocol. Radiation Measurements 32, 57-73. doi:10.1016/S1350-4487(99)00253-X

Thomsen, K.J., Murray, A.S., Jain, M., Boetter-Jensen, L., 2008. Laboratory fading rates of various luminescence signals from feldspar-rich sediment extracts. Radiation Measurements 43, 1474-1486. doi:10.1016/j.radmeas.2008.06.002

See Also

`analyse_SAR.CWOSL`, `calc_OSLLxTxRatio`, `plot_GrowthCurve`, `RLum.Analysis`, `RLum.Results` `get_RLum`

Examples

```
### NOTE: For this example existing example data are used. These data are non pIRIR data.
###
##(1) Compile example data set based on existing example data (SAR quartz measurement)
##(a) Load example data
data(ExampleData.BINfileData, envir = environment())

##(b) Transform the values from the first position in a RLum.Analysis object
object <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)

##(c) Grep curves and exclude the last two (one TL and one IRSL)
object <- get_RLum(object, record.id = c(-29,-30))

##(d) Define new sequence structure and set new RLum.Analysis object
sequence.structure  <- c(1,2,2,3,4,4)
sequence.structure <- as.vector(sapply(seq(0,length(object)-1,by = 4),
                                       function(x){sequence.structure + x}))

object <-  sapply(1:length(sequence.structure), function(x){

  object[[sequence.structure[x]]]

})

object <- set_RLum(class = "RLum.Analysis", records = object, protocol = "pIRIR")

##(2) Perform pIRIR analysis (for this example with quartz OSL data!)
## Note: output as single plots to avoid problems with this example
results <- analyse_pIRIRSequence(object,
     signal.integral.min = 1,
     signal.integral.max = 2,
     background.integral.min = 900,
     background.integral.max = 1000,
     fit.method = "EXP",
     sequence.structure = c("TL", "pseudoIRSL1", "pseudoIRSL2"),
     main = "Pseudo pIRIR data set based on quartz OSL",
     plot.single = TRUE)


##(3) Perform pIRIR analysis (for this example with quartz OSL data!)
## Alternative for PDF output, uncomment and complete for usage
## Not run:
pdf(file = "...", height = 15, width = 15)
  results <- analyse_pIRIRSequence(object,
         signal.integral.min = 1,
         signal.integral.max = 2,
         background.integral.min = 900,
         background.integral.max = 1000,
         fit.method = "EXP",
         main = "Pseudo pIRIR data set based on quartz OSL")

  dev.off()

## End(Not run)
```

analyse_SAR.CWOSL *Analyse SAR CW-OSL measurements*

Description

The function performs a SAR CW-OSL analysis on an RLum.Analysis object including growth curve fitting.

Usage

```
analyse_SAR.CWOSL(object, signal.integral.min, signal.integral.max,
  background.integral.min, background.integral.max, rejection.criteria = NULL,
  dose.points = NULL, mtext.outer, plot = TRUE, plot.single = FALSE, ...)
```

Arguments

object	RLum.Analysis (**required**): input object containing data for analysis, alternatively a list of RLum.Analysis objects can be provided.
signal.integral.min	integer (**required**): lower bound of the signal integral. Can be a list of integers, if object is of type list. If the input is vector (e.g., c(1,2)) the 2nd value will be interpreted as the minimum signal integral for the Tx curve.
signal.integral.max	integer (**required**): upper bound of the signal integral. Can be a list of integers, if object is of type list. If the input is vector (e.g., c(1,2)) the 2nd value will be interpreted as the maximum signal integral for the Tx curve.
background.integral.min	integer (**required**): lower bound of the background integral. Can be a list of integers, if object is of type list. If the input is vector (e.g., c(1,2)) the 2nd value will be interpreted as the minimum background integral for the Tx curve.
background.integral.max	integer (**required**): upper bound of the background integral. Can be a list of integers, if object is of type list. If the input is vector (e.g., c(1,2)) the 2nd value will be interpreted as the maximum background integral for the Tx curve.
rejection.criteria	list (with default): provide a named list and set rejection criteria in percentage for further calculation. Can be a list in a list, if object is of type list Allowed arguments are recycling.ratio, recuperation.rate, palaeodose.error, testdose.error and exceed.max.regpoint = TRUE/FALSE. Example: rejection.criteria = li Per default all numerical values are set to 10, exceed.max.regpoint = TRUE. Every criterium can be set to NA. In this value are calculated, but not considered, i.e. the RC.Status becomes always 'OK'
dose.points	numeric (optional): a numeric vector containg the dose points values Using this argument overwrites dose point values in the signal curves. Can be a list of numeric vectors, if object is of type list
mtext.outer	character (optional): option to provide an outer margin mtext. Can be a list of characters, if object is of type list

plot	logical (with default): enables or disables plot output.
plot.single	logical (with default) or numeric (optional): single plot output (TRUE/FALSE) to allow for plotting the results in single plot windows. If a numerice vector is provided the plots can be selected individually, i.e. plot.single = c(1,2,3,4) will plot the TL and Lx, Tx curves but not the legend (5) or the growth curve (6), (7) and (8) belong to rejection criteria plots. Requires plot = TRUE.
...	further arguments that will be passed to the function plot_GrowthCurve or calc_OSLLxTxRatio (supported: background.count.distribution, sigmab, sig0). **Please note** that if you consider to use the early light subtraction method you should provide your own sigmab value!

Details

The function performs an analysis for a standard SAR protocol measurements introduced by Murray and Wintle (2000) with CW-OSL curves. For the calculation of the Lx/Tx value the function calc_OSLLxTxRatio is used. For **changing the way the Lx/Tx error is calculated** use the argument background.count.distribution and sigmab, which will be passed to the function calc_OSLLxTxRatio.

Argument object is of type list

If the argument object is of type list containing **only** RLum.Analysis objects, the function recalls itself as often as elements are in the list. This is usefull if an entire measurement wanted to be analysed without writing separate for-loops. To gain in full control of the parameters (e.g., dose.points) for every aliquot (corresponding to one RLum.Analysis object in the list), in this case the arguments can be provided as list. This list should be of similar length as the list provided with the argument object, otherwise the function will create an own list of the requested lenght. Function output will be just one single RLum.Results object.

Please be careful when using this option. It may allow a fast an efficient data analysis, but the function may also break with an unclear error message, due to wrong input data.

Working with IRSL data

The function was originally designed to work just for 'OSL' curves, following the principles of the SAR protocol. An IRSL measurement protocol may follow this procedure, e.g., post-IR IRSL protocol (Thomsen et al., 2008). Therefore this functions has been enhanced to work with IRSL data, however, the function is only capable of analysing curves that follow the SAR protocol structure, i.e., to analyse a post-IR IRSL protocol, curve data have to be pre-selected by the user to fit the standards of the SAR protocol, i.e., Lx,Tx,Lx,Tx and so on.

Example: Imagine the measurement contains pIRIR50 and pIRIR225 IRSL curves. Only one curve type can be analysed at the same time: The pIRIR50 curves or the pIRIR225 curves.

Supported rejection criteria

'recycling.ratio': calculated for every repeated regeneration dose point.

'recuperation.rate': recuperation rate calculated by comparing the Lx/Tx values of the zero regeneration point with the Ln/Tn value (the Lx/Tx ratio of the natural signal). For methodological background see Aitken and Smith (1988).

'testdose.error': set the allowed error for the testdose, which per default should not exceed 10%. The testdose error is calculated as Tx_net.error/Tx_net.

'palaeodose.error': set the allowed error for the De value, which per default should not exceed 10%.

Value

A plot (optional) and an RLum.Results object is returned containing the following elements:

De.values	data.frame containing De-values, De-error and further parameters
LnLxTnTx.values	data.frame of all calculated Lx/Tx values including signal, background counts and the dose points
rejection.criteria	data.frame with values that might by used as rejection criteria. NA is produced if no R0 dose point exists.
Formula	formula formula that have been used for the growth curve fitting

The output should be accessed using the function get_RLum.

Function version

0.7.5 (2016-07-16 11:28:11)

Note

This function must not be mixed up with the function Analyse_SAR.OSLdata, which works with Risoe.BINfileData-class objects.

The function currently does only support 'OSL' or 'IRSL' data!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Aitken, M.J. and Smith, B.W., 1988. Optical dating: recuperation after bleaching. Quaternary Science Reviews 7, 387-393.

Duller, G., 2003. Distinguishing quartz and feldspar in single grain luminescence measurements. Radiation Measurements, 37 (2), 161-165.

Murray, A.S. and Wintle, A.G., 2000. Luminescence dating of quartz using an improved single-aliquot regenerative-dose protocol. Radiation Measurements 32, 57-73.

Thomsen, K.J., Murray, A.S., Jain, M., Boetter-Jensen, L., 2008. Laboratory fading rates of various luminescence signals from feldspar-rich sediment extracts. Radiation Measurements 43, 1474-1486. doi:10.1016/j.radmeas.2008.06.002

See Also

calc_OSLLxTxRatio, plot_GrowthCurve, RLum.Analysis, RLum.Results get_RLum

Examples

```
##load data
##ExampleData.BINfileData contains two BINfileData objects
##CWOSL.SAR.Data and TL.SAR.Data
data(ExampleData.BINfileData, envir = environment())

##transform the values from the first position in a RLum.Analysis object
object <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)

##perform SAR analysis and set rejection criteria
results <- analyse_SAR.CWOSL(
object = object,
signal.integral.min = 1,
signal.integral.max = 2,
background.integral.min = 900,
background.integral.max = 1000,
log = "x",
fit.method = "EXP",
rejection.criteria = list(
  recycling.ratio = 10,
  recuperation.rate = 10,
  testdose.error = 10,
  palaeodose.error = 10,
  exceed.max.regpoint = TRUE)
)

##show De results
get_RLum(results)

##show LnTnLxTx table
get_RLum(results, data.object = "LnLxTnTx.table")
```

Analyse_SAR.OSLdata *Analyse SAR CW-OSL measurements.*

Description

The function analyses SAR CW-OSL curve data and provides a summary of the measured data for every position. The output of the function is optimised for SAR OSL measurements on quartz.

Usage

```
Analyse_SAR.OSLdata(input.data, signal.integral, background.integral, position,
  run, set, dtype, keep.SEL = FALSE,
  info.measurement = "unkown measurement", output.plot = FALSE,
  output.plot.single = FALSE, cex.global = 1, ...)
```

Arguments

input.data	Risoe.BINfileData-class (**required**): input data from a Risoe BIN file, produced by the function read_BIN2R.
signal.integral	vector (**required**): channels used for the signal integral, e.g. signal.integral=c(1:2)
background.integral	vector (**required**): channels used for the background integral, e.g. background.integral=c(85:100
position	vector (optional): reader positions that want to be analysed (e.g. position=c(1:48). Empty positions are automatically omitted. If no value is given all positions are analysed by default.
run	vector (optional): range of runs used for the analysis. If no value is given the range of the runs in the sequence is deduced from the Risoe.BINfileData object.
set	vector (optional): range of sets used for the analysis. If no value is given the range of the sets in the sequence is deduced from the Risoe.BINfileData object.
dtype	character (optional): allows to further limit the curves by their data type (DTYPE), e.g., dtype = c("Natural","Dose") limits the curves to this two data types. By default all values are allowed. See Risoe.BINfileData-class for allowed data types.
keep.SEL	logical (default): option allowing to use the SEL element of the Risoe.BINfileData-class manually. NOTE: In this case any limitation provided by run, set and dtype are ignored!
info.measurement	character (with default): option to provide information about the measurement on the plot output (e.g. name of the BIN or BINX file).
output.plot	logical (with default): plot output (TRUE/FALSE)
output.plot.single	logical (with default): single plot output (TRUE/FALSE) to allow for plotting the results in single plot windows. Requires output.plot = TRUE.
cex.global	numeric (with default): global scaling factor.
...	further arguments that will be passed to the function calc_OSLLxTxRatio (supported: background.count.distribution, sigmab, sig0; e.g., for instrumental error) and can be used to adjust the plot. Supported" mtext, log

Details

The function works only for standard SAR protocol measurements introduced by Murray and Wintle (2000) with CW-OSL curves. For the calculation of the Lx/Tx value the function calc_OSLLxTxRatio is used.

Provided rejection criteria

'recyling ratio': calculated for every repeated regeneration dose point.
'recuperation': recuperation rate calculated by comparing the Lx/Tx values of the zero regeneration point with the Ln/Tn value (the Lx/Tx ratio of the natural signal). For methodological background see Aitken and Smith (1988)

'IRSL/BOSL': the integrated counts (`signal.integral`) of an IRSL curve are compared to the integrated counts of the first regenerated dose point. It is assumed that IRSL curves got the same dose as the first regenerated dose point. **Note:** This is not the IR depletation ratio described by Duller (2003).

Value

A plot (optional) and list is returned containing the following elements:

`LnLxTnTx`	data.frame of all calculated Lx/Tx values including signal, background counts and the dose points.
`RejectionCriteria`	data.frame with values that might by used as rejection criteria. NA is produced if no R0 dose point exists.
`SARParameters`	data.frame of additional measurement parameters obtained from the BIN file, e.g. preheat or read temperature (not valid for all types of measurements).

Function version

0.2.17 (2016-05-02 09:36:06)

Note

Rejection criteria are calculated but not considered during the analysis to discard values.

The analysis of IRSL data is not directly supported. You may want to consider using the functions `analyse_SAR.CWOSL` or `analyse_pIRIRSequence` instead.

The development of this function will not be continued. We recommend to use the function analyse_SAR.CWOSL or instead.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Margret C. Fuchs, HZDR, Freiberg (Germany)
R Luminescence Package Team

References

Aitken, M.J. and Smith, B.W., 1988. Optical dating: recuperation after bleaching. Quaternary Science Reviews 7, 387-393.

Duller, G., 2003. Distinguishing quartz and feldspar in single grain luminescence measurements. Radiation Measurements, 37 (2), 161-165.

Murray, A.S. and Wintle, A.G., 2000. Luminescence dating of quartz using an improved single-aliquot regenerative-dose protocol. Radiation Measurements 32, 57-73.

See Also

calc_OSLLxTxRatio, Risoe.BINfileData-class, read_BIN2R

and for further analysis plot_GrowthCurve

Examples

```
##load data
data(ExampleData.BINfileData, envir = environment())

##analyse data
output <- Analyse_SAR.OSLdata(input.data = CWOSL.SAR.Data,
                              signal.integral = c(1:5),
                              background.integral = c(900:1000),
                              position = c(1:1),
                              output.plot = TRUE)

##combine results relevant for further analysis
output.SAR <- data.frame(Dose = output$LnLxTnTx[[1]]$Dose,
                         LxTx = output$LnLxTnTx[[1]]$LxTx,
                         LxTx.Error = output$LnLxTnTx[[1]]$LxTx.Error)
output.SAR
```

---

analyse_SAR.TL *Analyse SAR TL measurements*

Description

The function performs a SAR TL analysis on a RLum.Analysis object including growth curve fitting.

Usage

```
analyse_SAR.TL(object, object.background, signal.integral.min,
  signal.integral.max, sequence.structure = c("PREHEAT", "SIGNAL",
  "BACKGROUND"), rejection.criteria = list(recycling.ratio = 10,
  recuperation.rate = 10), dose.points, log = "", ...)
```

Arguments

object	RLum.Analysis(**required**): input object containing data for analysis
object.background	currently not used
signal.integral.min	integer (**required**): requires the channel number for the lower signal integral bound (e.g. signal.integral.min = 100)
signal.integral.max	integer (**required**): requires the channel number for the upper signal integral bound (e.g. signal.integral.max = 200)
sequence.structure	vector character (with default): specifies the general sequence structure. Three steps are allowed ("PREHEAT", "SIGNAL", "BACKGROUND"), in addition a parameter "EXCLUDE". This allows excluding TL curves which are not relevant for the protocol analysis. (Note: None TL are removed by default)

`rejection.criteria`
: list (with default): list containing rejection criteria in percentage for the calculation.

`dose.points`
: `numeric` (optional): option set dose points manually

`log`
: character (with default): a character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic. See plot.default).

`...`
: further arguments that will be passed to the function `plot_GrowthCurve`

Details

This function performs a SAR TL analysis on a set of curves. The SAR procedure in general is given by Murray and Wintle (2000). For the calculation of the Lx/Tx value the function calc_TLLxTxRatio is used.

Provided rejection criteria

'recyling.ratio': calculated for every repeated regeneration dose point.
'recuperation.rate': recuperation rate calculated by comparing the Lx/Tx values of the zero regeneration point with the Ln/Tn value (the Lx/Tx ratio of the natural signal). For methodological background see Aitken and Smith (1988)

Value

A plot (optional) and an `RLum.Results` object is returned containing the following elements:

`De.values`
: data.frame containing De-values and further parameters

`LnLxTnTx.values`
: data.frame of all calculated Lx/Tx values including signal, background counts and the dose points.

`rejection.criteria`
: data.frame with values that might by used as rejection criteria. NA is produced if no R0 dose point exists.

note: the output should be accessed using the function `get_RLum`

Function version

0.1.5 (2016-07-16 11:28:11)

Note

THIS IS A BETA VERSION

None TL curves will be removed from the input object without further warning.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Aitken, M.J. and Smith, B.W., 1988. Optical dating: recuperation after bleaching. Quaternary Science Reviews 7, 387-393.

Murray, A.S. and Wintle, A.G., 2000. Luminescence dating of quartz using an improved single-aliquot regenerative-dose protocol. Radiation Measurements 32, 57-73.

See Also

calc_TLLxTxRatio, plot_GrowthCurve, RLum.Analysis, RLum.Results get_RLum

Examples

```
##load data
data(ExampleData.BINfileData, envir = environment())

##transform the values from the first position in a RLum.Analysis object
object <- Risoe.BINfileData2RLum.Analysis(TL.SAR.Data, pos=3)

##perform analysis
analyse_SAR.TL(object,
               signal.integral.min = 210,
               signal.integral.max = 220,
               log = "y",
               fit.method = "EXP OR LIN",
               sequence.structure = c("SIGNAL", "BACKGROUND"))
```

---

apply_CosmicRayRemoval

Function to remove cosmic rays from an RLum.Data.Spectrum S4 class object

---

Description

The function provides several methods for cosmic ray removal and spectrum smoothing for an RLum.Data.Spectrum S4 class object

Usage

```
apply_CosmicRayRemoval(object, method = "smooth", method.Pych.smoothing = 2,
  method.Pych.threshold_factor = 3, MARGIN = 2, verbose = FALSE,
  plot = FALSE, ...)
```

Arguments

object	RLum.Data.Spectrum (**required**): S4 object of class RLum.Data.Spectrum
method	character (with default): Defines method that is applied for cosmic ray removal. Allowed methods are smooth, the default, (smooth), smooth.spline (smooth.spline) and Pych. See details for further information.

`method.Pych.smoothing`
: integer (with default): Smoothing parameter for cosmic ray removal according to Pych (2003). The value defines how many neighboring values in each frame are used for smoothing (e.g., 2 means that the two previous and two following values are used).

`method.Pych.threshold_factor`
: numeric (with default): Threshold for zero-bins in the histogram. Small values mean that more peaks are removed, but signal might be also affected by this removal.

`MARGIN`
: integer (with default): on which part the function cosmic ray removal should be applied on: 1 = along the time axis (line by line), 2 = along the wavelength axis (column by column). Note: This argument currently only affects the methods `smooth` and `smooth.spline`

`verbose`
: logical (with default): Option to suppress terminal output.,

`plot`
: logical (with default): If `TRUE` the histograms used for the cosmic-ray removal are returned as plot including the used threshold. Note: A separat plot is returned for each frame! Currently only for `method = "Pych"` a graphical output is provided.

`...`
: further arguments and graphical parameters that will be passed to the `smooth` function.

Details

`method = "Pych"`

This method applies the cosmic-ray removal algorithm described by Pych (2003). Some aspects that are different to the publication:

- For interpolation between neighbouring values the median and not the mean is used.
- The number of breaks to construct the histogram is set to: `length(number.of.input.values)/2`

For further details see references below.

`method = "smooth"`

Method uses the function `smooth` to remove cosmic rays.

Arguments that can be passed are: `kind`, `twiceit`

`method = "smooth.spline"`
Method uses the function `smooth.spline` to remove cosmic rays.
Arguments that can be passed are: `spar`

How to combine methods?

Different methods can be combined by applying the method repeatedly to the dataset (see example).

Value

Returns same object as input (`RLum.Data.Spectrum`)

Function version

0.2.1 (2016-05-02 09:36:06)

Note

-

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Pych, W., 2003. A Fast Algorithm for Cosmic-Ray Removal from Single Images. Astrophysics 116, 148-153. http://arxiv.org/pdf/astro-ph/0311290.pdf?origin=publication_detail

See Also

RLum.Data.Spectrum, smooth, smooth.spline, apply_CosmicRayRemoval

Examples

```
##(1) - use with your own data and combine (uncomment for usage)
## run two times the default method and smooth with another method
## your.spectrum <- apply_CosmicRayRemoval(your.spectrum, method = "Pych")
## your.spectrum <- apply_CosmicRayRemoval(your.spectrum, method = "Pych")
## your.spectrum <- apply_CosmicRayRemoval(your.spectrum, method = "smooth")
```

---

apply_EfficiencyCorrection

Function to apply spectral efficiency correction to RLum.Data.Spectrum S4 class objects

---

Description

The function allows spectral efficiency corrections for RLum.Data.Spectrum S4 class objects

Usage

```
apply_EfficiencyCorrection(object, spectral.efficiency)
```

Arguments

object	RLum.Data.Spectrum (**required**): S4 object of class RLum.Data.Spectrum
spectral.efficiency	data.frame (**required**): Data set containing wavelengths (x-column) and relative spectral response values (y-column) in percentage

Details

The efficiency correction is based on a spectral response dataset provided by the user. Usually the data set for the quantum efficiency is of lower resolution and values are interpolated for the required spectral resolution using the function approx

If the energy calibration differes for both data set NA values are produces that will be removed from the matrix.

Value

Returns same object as input (RLum.Data.Spectrum)

Function version

0.1.1 (2016-05-02 09:36:06)

Note

Please note that the spectral efficiency data from the camera alone may not sufficiently correct for spectral efficiency of the entire optical system (e.g., spectrometer, camera ...).

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France),
Johannes Friedrich, University of Bayreuth (Germany)
R Luminescence Package Team

References

-

See Also

RLum.Data.Spectrum

Examples

```
##(1) - use with your own data (uncomment for usage)
## spectral.efficiency <- read.csv("your data")
##
## your.spectrum <- apply_EfficiencyCorrection(your.spectrum, )
```

app_RLum *Run Luminescence shiny apps (wrapper)*

Description

Wrapper for the function app_RLum from the package RLumShiny-package. For further details and examples please see the manual of this package.

Usage

```
app_RLum(app, ...)
```

Arguments

app	character (required): name of the application to start. See details for a list of available apps.
...	further arguments to pass to runApp

Function version

0.1.0 (2016-05-02 09:36:06)

Author(s)

Christoph Burow, University of Cologne (Germany)
R Luminescence Package Team

as *as() - RLum-object coercion*

Description

for [RLum.Analysis]

for [RLum.Data.Curve]

for [RLum.Data.Image]

for [RLum.Data.Spectrum]

for [RLum.Results]

Arguments

from	RLum or list, data.frame, matrix (**required**): object to be coerced from
to	character (**required**): class name to be coerced to

Details

[RLum.Analysis]

from	to
list	list

Given that the `list` consits of `RLum.Analysis` objects.

[RLum.Data.Curve]

from	to
list	list
data.frame	data.frame
matrix	matrix

[RLum.Data.Image]

from	to
data.frame	data.frame
matrix	matrix

[RLum.Data.Spectrum]

from	to
data.frame	data.frame
matrix	matrix

[RLum.Results]

from	to
list	list

Given that the `list` consits of `RLum.Results` objects.

Note

Due to the complex structure of the `RLum` objects itself a coercing to standard R data structures will be always loosely!

See Also

`as`

`BaseDataSet.CosmicDoseRate`

Base data set for cosmic dose rate calculation

Description

Collection of data from various sources needed for cosmic dose rate calculation

Format

`values.cosmic.Softcomp`:	data frame containing cosmic dose rates for shallow depths (< 167 g cm^-2) obtained using
`values.factor.Altitude`:	data frame containing altitude factors for adjusting geomagnetic field-change factors. Value
`values.par.FJH`:	data frame containing values for parameters F, J and H (read from Fig. 2 in Prescott & Hutt

$$Dc = D0 * (F + J * exp((altitude/1000)/H))$$

Version

0.1

Source

The following data were carefully read from figures in mentioned sources and used for fitting procedures. The derived expressions are used in the function `calc_CosmicDoseRate`.

values.cosmic.Softcomp

Program:	"AGE"
Reference:	Gruen (2009)
Fit:	Polynomials in the form of

For depths between 40-167 g cm^-2:

$$y = 2 * 10^-6 * x^2 - 0.0008 * x + 0.2535$$

(For depths <40 g cm^-2)

$$y = -6 * 10^-8 * x^3 + 2 * 10^-5 * x^2 - 0.0025 * x + 0.2969$$

values.factor.Altitude

Reference:	Prescott & Hutton (1994)
Page:	499
Figure:	1
Fit:	2-degree polynomial in the form of

$$y = -0.026 * x^2 + 0.6628 * x + 1.0435$$

values.par.FJH

Reference:	Prescott & Hutton (1994)
Page:	500
Figure:	2
Fits:	3-degree polynomials and linear fits

F (non-linear part, λ < 36.5 deg.):

$$y = -7 * 10^-7 * x^3 - 8 * 10^-5 * x^2 - 0.0009 * x + 0.3988$$

F (linear part, λ > 36.5 deg.):

$$y = -0.0001 * x + 0.2347$$

J (non-linear part, λ < 34 deg.):

$$y = 5 * 10^-6 * x^3 - 5 * 10^-5 * x^2 + 0.0026 * x + 0.5177$$

J (linear part, λ > 34 deg.):

$$y = 0.0005 * x + 0.7388$$

H (non-linear part, λ < 36 deg.):

$$y = -3 * 10^-6 * x^3 - 5 * 10^-5 * x^2 - 0.0031 * x + 4.398$$

H (linear part, λ > 36 deg.):

$$y = 0.0002 * x + 4.0914$$

References

Gruen, R., 2009. The "AGE" program for the calculation of luminescence age estimates. Ancient TL, 27, pp. 45-46.

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

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

Examples

```
##load data
data(BaseDataSet.CosmicDoseRate)
```

---

bin_RLum.Data *Channel binning - method dispatchter*

Description

Function calls the object-specific bin functions for RLum.Data S4 class objects.

Usage

```
bin_RLum.Data(object, ...)
```

Arguments

object	RLum.Data (**required**): S4 object of class RLum.Data
...	further arguments passed to the specifc class method

Details

The function provides a generalised access point for specific RLum.Data objects.
Depending on the input object, the corresponding function will be selected. Allowed arguments can be found in the documentations of the corresponding RLum.Data class.

Value

An object of the same type as the input object is provided

Function version

0.1.0 (2016-05-02 09:36:06)

Note

Currenlty only RLum.Data objects of class RLum.Data.Curve are supported!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

RLum.Data.Curve

Examples

```
##load example data
data(ExampleData.CW_OSL_Curve, envir = environment())

##create RLum.Data.Curve object from this example
curve <-
  set_RLum(
      class = "RLum.Data.Curve",
      recordType = "OSL",
      data = as.matrix(ExampleData.CW_OSL_Curve)
  )

##plot data without and with 2 and 4 channel binning
plot_RLum(curve)
plot_RLum(bin_RLum.Data(curve, bin_size = 2))
plot_RLum(bin_RLum.Data(curve, bin_size = 4))
```

`calc_AliquotSize` *Estimate the amount of grains on an aliquot*

Description

Estimate the number of grains on an aliquot. Alternatively, the packing density of an aliquot is computed.

Usage

```
calc_AliquotSize(grain.size, sample.diameter, packing.density = 0.65,
  MC = TRUE, grains.counted, plot = TRUE, ...)
```

Arguments

`grain.size`	`numeric` (**required**): mean grain size (microns) or a range of grain sizes from which the mean grain size is computed (e.g. `c(100,200)`).
`sample.diameter`	`numeric` (**required**): diameter (mm) of the targeted area on the sample carrier.
`packing.density`	`numeric` (with default) empirical value for mean packing density. If `packing.density = "inf"` a hexagonal structure on an infinite plane with a packing density of $0.906\ldots$ is assumed.
`MC`	`logical` (optional): if `TRUE` the function performs a monte carlo simulation for estimating the amount of grains on the sample carrier and assumes random errors in grain size distribution and packing density. Requires a vector with min and max grain size for `grain.size`. For more information see details.
`grains.counted`	`numeric` (optional) grains counted on a sample carrier. If a non-zero positive integer is provided this function will calculate the packing density of the aliquot. If more than one value is provided the mean packing density and its standard deviation is calculated. Note that this overrides `packing.density`.
`plot`	`logical` (with default): plot output (`TRUE`/`FALSE`)
`...`	further arguments to pass (`main, xlab, MC.iter`).

Details

This function can be used to either estimate the number of grains on an aliquot or to compute the packing density depending on the the arguments provided.
The following function is used to estimate the number of grains `n`:

$$n = (\pi * x^2)/(\pi * y^2) * d$$

where `x` is the radius of the aliquot size (microns), `y` is the mean radius of the mineral grains (mm) and `d` is the packing density (value between 0 and 1).

Packing density

The default value for `packing.density` is 0.65, which is the mean of empirical values determined by Heer et al. (2012) and unpublished data from the Cologne luminescence laboratory. If `packing.density = "inf"` a maximum density of $\pi/\sqrt{12} = 0.9068\ldots$ is used. However, note

that this value is not appropriate as the standard preparation procedure of aliquots resembles a PECC ("Packing Equal Circles in a Circle") problem where the maximum packing density is asymptotic to about 0.87.

Monte Carlo simulation

The number of grains on an aliquot can be estimated by Monte Carlo simulation when setting `MC = TRUE`. Each of the parameters necessary to calculate `n` (`x`, `y`, `d`) are assumed to be normally distributed with means μ_x, μ_y, μ_d and standard deviations $\sigma_x, \sigma_y, \sigma_d$.

For the mean grain size random samples are taken first from $N(\mu_y, \sigma_y)$, where $\mu_y = mean.grain.size$ and $\sigma_y = (max.grain.size - min.grain.size)/4$ so that 95% of all grains are within the provided the grain size range. This effectively takes into account that after sieving the sample there is still a small chance of having grains smaller or larger than the used mesh sizes. For each random sample the mean grain size is calculated, from which random subsamples are drawn for the Monte Carlo simulation.

The packing density is assumed to be normally distributed with an empirically determined $\mu = 0.65$ (or provided value) and $\sigma = 0.18$. The normal distribution is truncated at `d = 0.87` as this is approximately the maximum packing density that can be achieved in PECC problem.

The sample diameter has $\mu = sample.diameter$ and $\sigma = 0.2$ to take into account variations in sample disc preparation (i.e. applying silicon spray to the disc). A lower truncation point at `x = 0.5` is used, which assumes that aliqouts with smaller sample diameters of 0.5 mm are discarded. Likewise, the normal distribution is truncated at 9.8 mm, which is the diameter of the sample disc.

For each random sample drawn from the normal distributions the amount of grains on the aliquot is calculated. By default, `10^5` iterations are used, but can be reduced/increased with `MC.iter` (see `...`). The results are visualised in a bar- and boxplot together with a statistical summary.

Value

Returns a terminal output. In addition an `RLum.Results` object is returned containing the following element:

`summary`	data.frame summary of all relevant calculation results.
`args`	list used arguments
`call`	call the function call
`MC`	list results of the Monte Carlo simulation

The output should be accessed using the function `get_RLum`

Function version

0.31 (2016-05-16 22:20:28)

Author(s)

Christoph Burow, University of Cologne (Germany)
R Luminescence Package Team

References

Duller, G.A.T., 2008. Single-grain optical dating of Quaternary sediments: why aliquot size matters in luminescence dating. Boreas 37, 589-612.

Heer, A.J., Adamiec, G., Moska, P., 2012. How many grains are there on a single aliquot?. Ancient TL 30, 9-16.

Further reading

Chang, H.-C., Wang, L.-C., 2010. A simple proof of Thue's Theorem on Circle Packing. http://arxiv.org/pdf/1009.4322v1.pdf, 2013-09-13.

Graham, R.L., Lubachevsky, B.D., Nurmela, K.J., Oestergard, P.R.J., 1998. Dense packings of congruent circles in a circle. Discrete Mathematics 181, 139-154.

Huang, W., Ye, T., 2011. Global optimization method for finding dense packings of equal circles in a circle. European Journal of Operational Research 210, 474-481.

Examples

```
## Estimate the amount of grains on a small aliquot
calc_AliquotSize(grain.size = c(100,150), sample.diameter = 1, MC.iter = 100)

## Calculate the mean packing density of large aliquots
calc_AliquotSize(grain.size = c(100,200), sample.diameter = 8,
                 grains.counted = c(2525,2312,2880), MC.iter = 100)
```

---

calc_CentralDose — *Apply the central age model (CAM) after Galbraith et al. (1999) to a given De distribution*

Description

This function calculates the central dose and dispersion of the De distribution, their standard errors and the profile log likelihood function for sigma.

Usage

```
calc_CentralDose(data, sigmab, log = TRUE, plot = TRUE, ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
sigmab	numeric (with default): spread in De values given as a fraction (e.g. 0.2). This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100).
log	logical (with default): fit the (un-)logged central age model to De data
plot	logical (with default): plot output
...	further arguments (trace, verbose).

Details

This function uses the equations of Galbraith & Roberts (2012). The parameters `delta` and `sigma` are estimated by numerically solving eq. 15 and 16. Their standard errors are approximated using eq. 17. In addition, the profile log-likelihood function for `sigma` is calculated using eq. 18 and presented as a plot. Numerical values of the maximum likelihood approach are **only** presented in the plot and **not** in the console. A detailed explanation on maximum likelihood estimation can be found in the appendix of Galbraith & Laslett (1993, 468-470) and Galbraith & Roberts (2012, 15)

Value

Returns a plot (optional) and terminal output. In addition an `RLum.Results` object is returned containing the following element:

`summary`	data.frame summary of all relevant model results.
`data`	data.frame original input data
`args`	list used arguments
`call`	call the function call
`profile`	data.frame the log likelihood profile for sigma

The output should be accessed using the function `get_RLum`

Function version

1.3.1 (2016-05-02 09:36:06)

Author(s)

Christoph Burow, University of Cologne (Germany)
Based on a rewritten S script of Rex Galbraith, 2010

R Luminescence Package Team

References

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks Radiation Measurements 4, 459-470.

Galbraith, R.F., Roberts, R.G., Laslett, G.M., Yoshida, H. & Olley, J.M., 1999. Optical dating of single grains of quartz from Jinmium rock shelter, northern Australia. Part I: experimental design and statistical models. Archaeometry 41, 339-364.

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

Further reading

Arnold, L.J. & Roberts, R.G., 2009. Stochastic modelling of multi-grain equivalent dose (De) distributions: Implications for OSL dating of sediment mixtures. Quaternary Geochronology 4, 204-230.

Bailey, R.M. & Arnold, L.J., 2006. Statistical modelling of single grain quartz De distributions

and an assessment of procedures for estimating burial dose. Quaternary Science Reviews 25, 2475-2502.

Cunningham, A.C. & Wallinga, J., 2012. Realizing the potential of fluvial archives using robust OSL chronologies. Quaternary Geochronology 12, 98-106.

Rodnight, H., Duller, G.A.T., Wintle, A.G. & Tooth, S., 2006. Assessing the reproducibility and accuracy of optical dating of fluvial deposits. Quaternary Geochronology, 1 109-120.

Rodnight, H., 2008. How many equivalent dose values are needed to obtain a reproducible distribution?. Ancient TL 26, 3-10.

See Also

plot, calc_CommonDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

```
##load example data
data(ExampleData.DeValues, envir = environment())

##apply the central dose model
calc_CentralDose(ExampleData.DeValues$CA1)
```

---

calc_CommonDose *Apply the (un-)logged common age model after Galbraith et al. (1999) to a given De distribution*

Description

Function to calculate the common dose of a De distribution.

Usage

```
calc_CommonDose(data, sigmab, log = TRUE, ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
sigmab	numeric (with default): spread in De values given as a fraction (e.g. 0.2). This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100).
log	logical (with default): fit the (un-)logged common age model to De data
...	currently not used.

Details

(Un-)logged model

When `log = TRUE` this function calculates the weighted mean of logarithmic De values. Each of the estimates is weighted by the inverse square of its relative standard error. The weighted mean is then transformed back to the dose scale (Galbraith & Roberts 2012, p. 14).

The log transformation is not applicable if the De estimates are close to zero or negative. In this case the un-logged model can be applied instead (`log = FALSE`). The weighted mean is then calculated using the un-logged estimates of De and their absolute standard error (Galbraith & Roberts 2012, p. 14).

Value

Returns a terminal output. In addition an `RLum.Results` object is returned containing the following element:

`summary`	data.frame summary of all relevant model results.
`data`	data.frame original input data
`args`	list used arguments
`call`	call the function call

The output should be accessed using the function `get_RLum`

Function version

0.1 (2016-05-02 09:36:06)

Author(s)

Christoph Burow, University of Cologne (Germany)
R Luminescence Package Team

References

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks Radiation Measurements 4, 459-470.

Galbraith, R.F., Roberts, R.G., Laslett, G.M., Yoshida, H. & Olley, J.M., 1999. Optical dating of single grains of quartz from Jinmium rock shelter, northern Australia. Part I: experimental design and statistical models. Archaeometry 41, 339-364.

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

Further reading

Arnold, L.J. & Roberts, R.G., 2009. Stochastic modelling of multi-grain equivalent dose (De) distributions: Implications for OSL dating of sediment mixtures. Quaternary Geochronology 4, 204-230.

Bailey, R.M. & Arnold, L.J., 2006. Statistical modelling of single grain quartz De distributions

and an assessment of procedures for estimating burial dose. Quaternary Science Reviews 25, 2475-2502.

Cunningham, A.C. & Wallinga, J., 2012. Realizing the potential of fluvial archives using robust OSL chronologies. Quaternary Geochronology 12, 98-106.

Rodnight, H., Duller, G.A.T., Wintle, A.G. & Tooth, S., 2006. Assessing the reproducibility and accuracy of optical dating of fluvial deposits. Quaternary Geochronology 1, 109-120.

Rodnight, H., 2008. How many equivalent dose values are needed to obtain a reproducible distribution?. Ancient TL 26, 3-10.

See Also

calc_CentralDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())

## apply the common dose model
calc_CommonDose(ExampleData.DeValues$CA1)
```

---

calc_CosmicDoseRate *Calculate the cosmic dose rate*

Description

This function calculates the cosmic dose rate taking into account the soft- and hard-component of the cosmic ray flux and allows corrections for geomagnetic latitude, altitude above sea-level and geomagnetic field changes.

Usage

```
calc_CosmicDoseRate(depth, density, latitude, longitude, altitude,
  corr.fieldChanges = FALSE, est.age = NA, half.depth = FALSE,
  error = 10)
```

Arguments

depth	numeric (**required**): depth of overburden (m). For more than one absorber use c(depth_1, depth_2, ...,depth_n)
density	numeric (**required**): average overburden density (g/cm^3). For more than one absorber use c(density_1,density_2, ..., density_n)
latitude	numeric (**required**): latitude (decimal degree), N positive
longitude	numeric (**required**): longitude (decimal degree), E positive
altitude	numeric (**required**): altitude (m above sea-level)

`corr.fieldChanges`
: logical (with default): correct for geomagnetic field changes after Prescott & Hutton (1994). Apply only when justified by the data.

`est.age`
: numeric (with default): estimated age range (ka) for geomagnetic field change correction (0-80 ka allowed)

`half.depth`
: logical (with default): How to overcome with varying overburden thickness. If `TRUE` only half the depth is used for calculation. Apply only when justified, i.e. when a constant sedimentation rate can safely be assumed.

`error`
: numeric (with default): general error (percentage) to be implemented on corrected cosmic dose rate estimate

Details

This function calculates the total cosmic dose rate considering both the soft- and hard-component of the cosmic ray flux.

Internal calculation steps

(1) Calculate total depth of all absorber in hg/cm^2 (1 hg/cm^2 = 100 g/cm^2)

$$absorber = depth_1 * density_1 + depth_2 * density_2 + ... + depth_n * density_n$$

(2) If `half.depth = TRUE`

$$absorber = absorber/2$$

(3) Calculate cosmic dose rate at sea-level and 55 deg. latitude

a) If absorber is > 167 g/cm^2 (only hard-component; Allkofer et al. 1975): apply equation given by Prescott & Hutton (1994) (c.f. Barbouti & Rastin 1983)

$$D0 = C/(((absorber + d)^{\alpha} + a) * (absober + H)) * exp(-B * absorber)$$

b) If absorber is < 167 g/cm^2 (soft- and hard-component): derive D0 from Fig. 1 in Prescott & Hutton (1988).

(4) Calculate geomagnetic latitude (Prescott & Stephan 1982, Prescott & Hutton 1994)

$$\lambda = arcsin(0.203 * cos(latitude) * cos(longitude - 291) + 0.979 * sin(latitude))$$

(5) Apply correction for geomagnetic latitude and altitude above sea-level. Values for F, J and H were read from Fig. 3 shown in Prescott & Stephan (1982) and fitted with 3-degree polynomials for lambda < 35 degree and a linear fit for lambda > 35 degree.

$$Dc = D0 * (F + J * exp((altitude/1000)/H))$$

(6) Optional: Apply correction for geomagnetic field changes in the last 0-80 ka (Prescott & Hutton 1994). Correction and altitude factors are given in Table 1 and Fig. 1 in Prescott & Hutton (1994). Values for altitude factor were fitted with a 2-degree polynomial. The altitude factor is operated on the decimal part of the correction factor.

$$Dc' = Dc * correctionFactor$$

Usage of `depth` **and** `density`

(1) If only one value for depth and density is provided, the cosmic dose rate is calculated for exactly one sample and one absorber as overburden (i.e. `depth*density`).

(2) In some cases it might be useful to calculate the cosmic dose rate for a sample that is overlain by more than one absorber, e.g. in a profile with soil layers of different thickness and a distinct difference in density. This can be calculated by providing a matching number of values for `depth` and `density` (e.g. `depth = c(1, 2), density = c(1.7, 2.4)`)

(3) Another possibility is to calculate the cosmic dose rate for more than one sample of the same profile. This is done by providing more than one values for `depth` and only one for `density`. For example, `depth = c(1, 2, 3), density = 1.7` will calculate the cosmic dose rate for three samples in 1, 2 and 3 m depth in a sediment of density 1.7 g/cm^3.

Value

Returns a terminal output. In addition an `RLum.Results` object is returned containing the following element:

`summary`	data.frame summary of all relevant calculation results.
`args`	list used arguments
`call`	call the function call

The output should be accessed using the function `get_RLum`

Function version

0.5.2 (2015-11-29 17:27:48)

Note

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 of standard rock. The original expression by Barbouti & Rastin (1983) only considers the muon flux (i.e. hard-component) and is by their own definition only valid for depths between 10-10^4 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 ray flux. For samples at zero depth and at sea-level the underestimation can be as large as ~0.1 Gy/ka. In a previous article, Prescott & Hutton (1988) give another approximation of Barbouti & Rastins equation in the form of

$$D = 0.21 * exp(-0.070 * absorber + 0.0005 * absorber^2)$$

which is valid for depths between 150-5000 g/cm^2. For shallower depths (< 150 g/cm^2) they provided a graph (Fig. 1) from which the dose rate can be read.

As a result, this function employs the equation of Prescott & Hutton (1994) only for depths > 167 g/cm^2, i.e. only for the hard-component of the cosmic ray flux. Cosmic dose rate values for depths < 167 g/cm^2 were obtained from the "AGE" programm (Gruen 2009) and fitted with a 6-degree polynomial curve (and hence reproduces the graph shown in Prescott & Hutton 1988). However, these values assume an average overburden density of 2 g/cm^3.

It is currently not possible to obtain more precise cosmic dose rate values for near-surface samples as there is no equation known to the author of this function at the time of writing.

Author(s)

Christoph Burow, University of Cologne (Germany)
R Luminescence Package Team

References

Allkofer, O.C., Carstensen, K., Dau, W.D., Jokisch, H., 1975. Letter to the editor. The absolute cosmic ray flux at sea level. Journal of Physics G: Nuclear and Particle Physics 1, L51-L52.

Barbouti, A.I., Rastin, B.C., 1983. 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.

Crookes, J.N., Rastin, B.C., 1972. An investigation of the absolute intensity of muons at sea-level. Nuclear Physics B 39, 493-508.

Gruen, R., 2009. The "AGE" program for the calculation of luminescence age estimates. Ancient TL 27, 45-46.

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

223-227. Prescott, J.R., Hutton, J.T., 1994. Cosmic ray contributions to dose rates for luminescence and ESR dating: large depths and long-term time variations. Radiation Measurements 23, 497-500.

Prescott, J.R., Stephan, L.G., 1982. The contribution of cosmic radiation to the environmental dose for thermoluminescence dating. Latitude, altitude and depth dependences. PACT 6, 17-25.

See Also

BaseDataSet.CosmicDoseRate

Examples

```
##(1) calculate cosmic dose rate (one absorber)
calc_CosmicDoseRate(depth = 2.78, density = 1.7,
                    latitude = 38.06451, longitude = 1.49646,
                    altitude = 364, error = 10)

##(2a) calculate cosmic dose rate (two absorber)
calc_CosmicDoseRate(depth = c(5.0, 2.78), density = c(2.65, 1.7),
                    latitude = 38.06451, longitude = 1.49646,
                    altitude = 364, error = 10)

##(2b) calculate cosmic dose rate (two absorber) and
##correct for geomagnetic field changes
calc_CosmicDoseRate(depth = c(5.0, 2.78), density = c(2.65, 1.7),
                    latitude = 12.04332, longitude = 4.43243,
                    altitude = 364, corr.fieldChanges = TRUE,
                    est.age = 67, error = 15)
```

```
##(3) calculate cosmic dose rate and export results to .csv file
#calculate cosmic dose rate and save to variable
results<- calc_CosmicDoseRate(depth = 2.78, density = 1.7,
                              latitude = 38.06451, longitude = 1.49646,
                              altitude = 364, error = 10)

# the results can be accessed by
get_RLum(results, "summary")

#export results to .csv file - uncomment for usage
#write.csv(results, file = "c:/users/public/results.csv")

##(4) calculate cosmic dose rate for 6 samples from the same profile
##    and save to .csv file
#calculate cosmic dose rate and save to variable
results<- calc_CosmicDoseRate(depth = c(0.1, 0.5 , 2.1, 2.7, 4.2, 6.3),
                              density = 1.7, latitude = 38.06451,
                              longitude = 1.49646, altitude = 364,
                              error = 10)

#export results to .csv file - uncomment for usage
#write.csv(results, file = "c:/users/public/results_profile.csv")
```

`calc_FadingCorr` *Apply a fading correction according to Huntley & Lamothe (2001) for a given g-value and a given tc*

Description

This function solves the equation used for correcting the fading affected age including the error for a given g-value according to Huntley & Lamothe (2001).

Usage

```
calc_FadingCorr(age.faded, g_value, tc = NULL, tc.g_value = tc,
  n.MC = 10000, seed = NULL, txtProgressBar = TRUE, verbose = TRUE)
```

Arguments

`age.faded`	`numeric` `vector` (**required**): uncorrected age with error in ka (see example)
`g_value`	`vector` (**required**): g-value and error obtained from separate fading measurements (see example). Alternatively an `RLum.Results` object can be provided produced by the function `analyse_FadingMeasurement`, in this case tc is set automatically
`tc`	`numeric` (**required**): time in seconds between irradiation and the prompt measurement (cf. Huntley & Lamothe 2001). Argument will be ignored if `g_value` was an `RLum.Results` object
`tc.g_value`	`numeric` (with default): the time in seconds between irradiation and the prompt measurement used for estimating the g-value. If the g-value was normalised to, e.g., 2 days, this time in seconds (i.e., 172800) should be given here. If nothing is provided the time is set to tc, which is usual case for g-values obtained using the SAR method and g-values that had been not normalised to 2 days.

n.MC	integer (with default): number of Monte Carlo simulation runs for error estimation. If n.MC = 'auto' is used the function tries to find a 'stable' error for the age. Note: This may take a while!
seed	integer (optional): sets the seed for the random number generator in R using set.seed
txtProgressBar	logical (with default): enables or disables txtProgressBar
verbose	logical (with default): enables or disables terminal output

Details

As the g-value sligthly depends on the time between irradiation and the prompt measurement, this is tc, always a tc value needs to be provided. If the g-value was normalised to a distinct time or evaluated with a different tc value (e.g., external irradiation), also the tc value for the g-value needs to be provided (argument tc.g_value and then the g-value is recalcualted to tc of the measurement used for estimating the age applying the following equation:

$$\kappa_{tc} = \kappa_{tc.g}/(1 - \kappa_{tc.g} * log(tc/tc.g))$$

where

$$\kappa_{tc.g} = g/100/log(10)$$

with log the natural logarithm.

The error of the fading-corrected age is determined using a Monte Carlo simulation approach. Solving of the equation is realised using uniroot. Large values for n.MC will significantly increase the computation time.

n.MC = 'auto'

The error estimation based on a stochastic process, i.e. for a small number of MC runs the calculated error varies considerably every time the function is called, even with the same input values. The argument option n.MC = 'auto' tries to find a stable value for the standard error, i.e. the standard deviation of values calculated during the MC runs (age.corr.MC), within a given precision (2 digits) by increasing the number of MC runs stepwise and calculating the corresponding error.

If the determined error does not differ from the 9 values calculated previously within a precision of (here) 3 digits the calculation is stopped as it is assumed that the error is stable. Please note that (a) the duration depends on the input values as well as on the provided computation ressources and it may take a while, (b) the length (size) of the output vector age.corr.MC, where all the single values produced during the MC runs are stored, equals the number of MC runs (here termed observations).

To avoid an endless loop the calculation is stopped if the number of observations exceeds 10^7. This limitation can be overwritten by setting the number of MC runs manually, e.g. n.MC = 10000001. Note: For this case the function is not checking whether the calculated error is stable.

seed

This option allows to recreate previously calculated results by setting the seed for the R random number generator (see set.seed for details). This option should not be mixed up with the option n.MC = 'auto'. The results may appear similar, but they are not comparable!

FAQ

Q: Which tc value is expected?

A: tc is the time in seconds between irradiation and the prompt measurement applied during your De measurement. However, this tc might differ from the tc used for estimating the g-value. In the case of an SAR measurement tc should be similar, however, if it differs, you have to provide this tc value (the one used for estimating the g-value) using the argument `tc.g_value`.

Value

Returns an S4 object of type `RLum.Results`.

Slot: **@data**

Object	Type	Comment
`age.corr`	`data.frame`	Corrected age
`age.corr.MC`	`numeric`	MC simulation results with all possible ages from that simulation

Slot: **@info**

Object	Type	Comment
`info`	`character`	the original function call

Function version

0.4.1 (2016-07-21 10:36:31)

Note

The upper age limit is set to 500 ka!
Special thanks to Sebastien Huot for his support and clarification via e-mail.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Huntley, D.J., Lamothe, M., 2001. Ubiquity of anomalous fading in K-feldspars and the measurement and correction for it in optical dating. Canadian Journal of Earth Sciences, 38, 1093-1106.

See Also

`RLum.Results`, `get_RLum`, `uniroot`

Examples

```
##run the examples given in the appendix of Huntley and Lamothe, 2001

##(1) faded age: 100 a
results <- calc_FadingCorr(
```

```
   age.faded = c(0.1,0),
   g_value = c(5.0, 1.0),
   tc = 2592000,
   tc.g_value = 172800,
   n.MC = 100)

##(2) faded age: 1 ka
results <- calc_FadingCorr(
   age.faded = c(1,0),
   g_value = c(5.0, 1.0),
   tc = 2592000,
   tc.g_value = 172800,
   n.MC = 100)

##(3) faded age: 10.0 ka
results <- calc_FadingCorr(
   age.faded = c(10,0),
   g_value = c(5.0, 1.0),
   tc = 2592000,
   tc.g_value = 172800,
   n.MC = 100)

##access the last output
get_RLum(results)
```

`calc_FastRatio` *Calculate the Fast Ratio for CW-OSL curves*

Description

Function to calculate the fast ratio of quartz CW-OSL single grain or single aliquot curves after Durcan & Duller (2011).

Usage

```
calc_FastRatio(object, stimulation.power = 30.6, wavelength = 470,
  sigmaF = 2.6e-17, sigmaM = 4.28e-18, Ch_L1 = 1, x = 1, x2 = 0.1,
  dead.channels = c(0, 0), fitCW.sigma = FALSE, fitCW.curve = FALSE,
  plot = TRUE, ...)
```

Arguments

`object`	`RLum.Analysis`, `RLum.Data.Curve` or `data.frame` (**required**): x, y data of measured values (time and counts).
`stimulation.power`	`numeric` (with default): Stimulation power in mW/cm^2
`wavelength`	`numeric` (with default): Stimulation wavelength in nm
`sigmaF`	`numeric` (with default): Photoionisation cross-section (cm^2) of the fast component. Default value after Durcan & Duller (2011).
`sigmaM`	`numeric` (with default): Photoionisation cross-section (cm^2) of the medium component. Default value after Durcan & Duller (2011).

Ch_L1	numeric (with default): An integer specifying the channel for L1.
x	numeric (with default): % of signal remaining from the fast component. Used to define the location of L2 and L3 (start).
x2	numeric (with default): % of signal remaining from the medium component. Used to define the location of L3 (end).
dead.channels	numeric (with default): Vector of length 2 in the form of c(x, y). Channels that do not contain OSL data, i.e. at the start or end of measurement.
fitCW.sigma	logical (optional): fit CW-OSL curve using fit_CWCurve to calculate sigmaF and sigmaM (experimental).
fitCW.curve	logical (optional): fit CW-OSL curve using fit_CWCurve and derive the counts of L2 and L3 from the fitted OSL curve (experimental).
plot	logical (with default): plot output (TRUE/FALSE)
...	available options: verbose (logical). Further arguments passed to fit_CWCurve.

Details

This function follows the equations of Durcan & Duller (2011). The energy required to reduce the fast and medium quartz OSL components to x and x2 % respectively using eq. 3 to determine channels L2 and L3 (start and end). The fast ratio is then calculated from: $(L1 - L3)/(L2 - L3)$.

Value

Returns a plot (optional) and an S4 object of type RLum.Results. The slot data contains a list with the following elements:

summary	data.frame summary of all relevant results
data	the original input data
fit	RLum.Results object if either fitCW.sigma or fitCW.curve is TRUE
args	list of used arguments
call	call the function call

Function version

0.1.0 (2016-05-02 09:36:06)

Author(s)

Georgina King, University of Cologne (Germany)
Julie A. Durcan, University of Oxford (United Kingdom)
Christoph Burow, University of Cologne (Germany)

R Luminescence Package Team

References

Durcan, J.A. & Duller, G.A.T., 2011. The fast ratio: A rapid measure for testing the dominance of the fast component in the initial OSL signal from quartz. Radiation Measurements 46, 1065-1072.

Madsen, A.T., Duller, G.A.T., Donnelly, J.P., Roberts, H.M. & Wintle, A.G., 2009. A chronology of hurricane landfalls at Little Sippewissett Marsh, Massachusetts, USA, using optical dating. Geomorphology 109, 36-45.

Further reading

Steffen, D., Preusser, F. & Schlunegger, 2009. OSL quartz age underestimation due to unstable signal components. Quaternary Geochronology 4, 353-362.

See Also

fit_CWCurve, get_RLum, RLum.Analysis, RLum.Results, RLum.Data.Curve

Examples

```
# load example CW-OSL curve
data("ExampleData.CW_OSL_Curve")

# calculate the fast ratio w/o further adjustments
res <- calc_FastRatio(ExampleData.CW_OSL_Curve)

# show the summary table
get_RLum(res)
```

---

calc_FiniteMixture *Apply the finite mixture model (FMM) after Galbraith (2005) to a given De distribution*

Description

This function fits a k-component mixture to a De distribution with differing known standard errors. Parameters (doses and mixing proportions) are estimated by maximum likelihood assuming that the log dose estimates are from a mixture of normal distributions.

Usage

```
calc_FiniteMixture(data, sigmab, n.components, grain.probability = FALSE,
  dose.scale, pdf.weight = TRUE, pdf.sigma = "sigmab",
  pdf.colors = "gray", pdf.scale, plot.proportions = TRUE, plot = TRUE,
  ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
sigmab	numeric (**required**): spread in De values given as a fraction (e.g. 0.2). This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Wallinga 2012, p. 100).

n.components	numeric (**required**): number of components to be fitted. If a vector is provided (e.g. c(2:8)) the finite mixtures for 2, 3 ... 8 components are calculated and a plot and a statistical evaluation of the model performance (BIC score and maximum log-likelihood) is provided.
grain.probability	logical (with default): prints the estimated probabilities of which component each grain is in
dose.scale	numeric: manually set the scaling of the y-axis of the first plot with a vector in the form of c(min,max)
pdf.weight	logical (with default): weight the probability density functions by the components proportion (applies only when a vector is provided for n.components)
pdf.sigma	character (with default): if "sigmab" the components normal distributions are plotted with a common standard deviation (i.e. sigmab) as assumed by the FFM. Alternatively, "se" takes the standard error of each component for the sigma parameter of the normal distribution
pdf.colors	character (with default): color coding of the components in the the plot. Possible options are "gray", "colors" and "none"
pdf.scale	numeric: manually set the max density value for proper scaling of the x-axis of the first plot
plot.proportions	logical (with default): plot barplot showing the proportions of components
plot	logical (with default): plot output
...	further arguments to pass. See details for their usage.

Details

This model uses the maximum likelihood and Bayesian Information Criterion (BIC) approaches.

Indications of overfitting are:

- increasing BIC
- repeated dose estimates
- covariance matrix not positive definite
- covariance matrix produces NaNs
- convergence problems

Plot

If a vector (c(k.min:k.max)) is provided for n.components a plot is generated showing the the k components equivalent doses as normal distributions. By default pdf.weight is set to FALSE, so that the area under each normal distribution is always 1. If TRUE, the probability density functions are weighted by the components proportion for each iteration of k components, so the sum of areas of each component equals 1. While the density values are on the same scale when no weights are used, the y-axis are individually scaled if the probability density are weighted by the components proportion.
The standard deviation (sigma) of the normal distributions is by default determined by a common sigmab (see pdf.sigma). For pdf.sigma = "se" the standard error of each component is taken instead.
The stacked barplot shows the proportion of each component (in per cent) calculated by the FFM. The last plot shows the achieved BIC scores and maximum log-likelihood estimates for each iteration of k.

Value

Returns a plot (optional) and terminal output. In addition an RLum.Results object is returned containing the following elements:

summary	data.frame summary of all relevant model results.
data	data.frame original input data
args	list used arguments
call	call the function call
mle	covariance matrices of the log likelihoods
BIC	BIC score
llik	maximum log likelihood
grain.probability	probabilities of a grain belonging to a component
components	matrix estimates of the de, de error and proportion for each component
single.comp	data.frame single componente FFM estimate

If a vector for n.components is provided (e.g. c(2:8)), mle and grain.probability are lists containing matrices of the results for each iteration of the model.

The output should be accessed using the function get_RLum

Function version

0.4 (2016-05-02 09:36:06)

Author(s)

Christoph Burow, University of Cologne (Germany)
Based on a rewritten S script of Rex Galbraith, 2006.

R Luminescence Package Team

References

Galbraith, R.F. & Green, P.F., 1990. Estimating the component ages in a finite mixture. Nuclear Tracks and Radiation Measurements 17, 197-206.

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks Radiation Measurements 4, 459-470.

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

Roberts, R.G., Galbraith, R.F., Yoshida, H., Laslett, G.M. & Olley, J.M., 2000. Distinguishing dose populations in sediment mixtures: a test of single-grain optical dating procedures using mixtures of laboratory-dosed quartz. Radiation Measurements 32, 459-465.

Galbraith, R.F., 2005. Statistics for Fission Track Analysis, Chapman & Hall/CRC, Boca Raton.

Further reading

Arnold, L.J. & Roberts, R.G., 2009. Stochastic modelling of multi-grain equivalent dose (De) distributions: Implications for OSL dating of sediment mixtures. Quaternary Geochronology 4, 204-230.

Cunningham, A.C. & Wallinga, J., 2012. Realizing the potential of fluvial archives using robust OSL chronologies. Quaternary Geochronology 12, 98-106.

Rodnight, H., Duller, G.A.T., Wintle, A.G. & Tooth, S., 2006. Assessing the reproducibility and accuracy of optical dating of fluvial deposits. Quaternary Geochronology 1, 109-120.

Rodnight, H. 2008. How many equivalent dose values are needed to obtain a reproducible distribution?. Ancient TL 26, 3-10.

See Also

calc_CentralDose, calc_CommonDose, calc_FuchsLang2001, calc_MinDose

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())

## (1) apply the finite mixture model
## NOTE: the data set is not suitable for the finite mixture model,
## which is why a very small sigmab is necessary
calc_FiniteMixture(ExampleData.DeValues$CA1,
                   sigmab = 0.2, n.components = 2,
                   grain.probability = TRUE)

## (2) repeat the finite mixture model for 2, 3 and 4 maximum number of fitted
## components and save results
## NOTE: The following example is computationally intensive. Please un-comment
## the following lines to make the example work.
FMM<- calc_FiniteMixture(ExampleData.DeValues$CA1,
                         sigmab = 0.2, n.components = c(2:4),
                         pdf.weight = TRUE, dose.scale = c(0, 100))

## show structure of the results
FMM

## show the results on equivalent dose, standard error and proportion of
## fitted components
get_RLum(object = FMM, data.object = "components")
```

---

calc_FuchsLang2001 *Apply the model after Fuchs & Lang (2001) to a given De distribution.*

Description

This function applies the method according to Fuchs & Lang (2001) for heterogeneously bleached samples with a given coefficient of variation threshold.

Usage

```
calc_FuchsLang2001(data, cvThreshold = 5, startDeValue = 1, plot = TRUE,
  ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
cvThreshold	numeric (with default): coefficient of variation in percent, as threshold for the method, e.g. cvThreshold = 3. See details.
startDeValue	numeric (with default): number of the first aliquot that is used for the calculations
plot	logical (with default): plot output TRUE/FALSE
...	further arguments and graphical parameters passed to plot

Details

Used values

If the coefficient of variation (c[v]) of the first two values is larger than the threshold c[v_threshold], the first value is skipped. Use the startDeValue argument to define a start value for calculation (e.g. 2nd or 3rd value).

Basic steps of the approach

(1) Estimate natural relative variation of the sample using a dose recovery test
(2) Sort the input values ascendingly
(3) Calculate a running mean, starting with the lowermost two values and add values iteratively.
(4) Stop if the calculated c[v] exceeds the specified cvThreshold

Value

Returns a plot (optional) and terminal output. In addition an RLum.Results object is returned containing the following elements:

summary	data.frame summary of all relevant model results.
data	data.frame original input data
args	list used arguments
call	call the function call
usedDeValues	data.frame containing the used values for the calculation

Function version

0.4.1 (2016-05-02 09:36:06)

Note

Please consider the requirements and the constraints of this method (see Fuchs & Lang, 2001)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France) Christoph Burow, University of Cologne (Germany)
R Luminescence Package Team

References

Fuchs, M. & Lang, A., 2001. OSL dating of coarse-grain fluvial quartz using single-aliqout protocols on sediments from NE Peloponnese, Greece. In: Quaternary Science Reviews 20, 783-787.

Fuchs, M. & Wagner, G.A., 2003. Recognition of insufficient bleaching by small aliquots of quartz for reconstructing soil erosion in Greece. Quaternary Science Reviews 22, 1161-1167.

See Also

plot, calc_MinDose, calc_FiniteMixture, calc_CentralDose, calc_CommonDose, RLum.Results

Examples

```
##load example data
data(ExampleData.DeValues, envir = environment())

##calculate De according to Fuchs & Lang (2001)
temp<- calc_FuchsLang2001(ExampleData.DeValues$BT998, cvThreshold = 5)
```

---

calc_gSGC *Calculate De value based on the gSGC by Li et al., 2015*

Description

Function returns De value and De value error using the global standardised growth curve (gSGC) assumption proposed by Li et al., 2015 for OSL dating of sedimentary quartz

Usage

```
calc_gSGC(data, gSGC.type = "0-250", gSGC.parameters, n.MC = 100,
  verbose = TRUE, plot = TRUE, ...)
```

Arguments

data	data.frame (**required**): input data of providing the following columns: 'LnTn', 'LnTn.error', Lr1Tr1', 'Lr1Tr1.error', 'Dr1' Note: column names are not required. The function expect the input data in the given order
gSGC.type	character (with default): define the function parameters that should be used for the iteration procedure: Li et al., 2015 (Table 2) presented function parameters for two dose ranges: "0-450" and "0-250"

gSGC.parameters	list (optional): option to provide own function parameters used for #' fitting as named list. Nomenclature follows Li et al., 2015, i.e. list(A,A.error,D0,D0.error,c,c.error,Y range requires a vector for the range the function is considered as valid, e.g. range = c(0,250) Using this option overwrites the default parameter list of the gSGC, meaning the argument gSGC.type will be without effect
n.MC	integer (with default): number of Monte Carlo simulation runs for error estimation, s. details.
verbose	logical: enable or disable terminal output
plot	logical: enable or disable graphical feedback as plot
...	parameters will be passed to the plot output

Details

The error of the De value is determined using a Monte Carlo simulation approach. Solving of the equation is realised using uniroot. Large values for n.MC will significantly increase the computation time.

Value

Returns an S4 object of type RLum.Results.

@data
$ De.value (data.frame)
.. $ De
.. $ De.error
.. $ Eta
$ De.MC (list) contains the matricies from the error estimation.
$ uniroot (list) contains the uniroot outputs of the De estimations

@info
$ call (call) the original function call

Function version

0.1.1 (2016-09-09 10:32:17)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montagine (France)

R Luminescence Package Team

References

Li, B., Roberts, R.G., Jacobs, Z., Li, S.-H., 2015. Potential of establishing a 'global standardised growth curve' (gSGC) for optical dating of quartz from sediments. Quaternary Geochronology 27, 94-104. doi:10.1016/j.quageo.2015.02.011

See Also

RLum.Results, get_RLum, uniroot

Examples

```
results <- calc_gSGC(data = data.frame(
LnTn =  2.361, LnTn.error = 0.087,
Lr1Tr1 = 2.744, Lr1Tr1.error = 0.091,
Dr1 = 34.4))

get_RLum(results, data.object = "De")
```

---

calc_HomogeneityTest *Apply a simple homogeneity test after Galbraith (2003)*

Description

A simple homogeneity test for De estimates

Usage

```
calc_HomogeneityTest(data, log = TRUE, ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
log	logical (with default): peform the homogeniety test with (un-)logged data
...	further arguments (for internal compatibility only).

Details

For details see Galbraith (2003).

Value

Returns a terminal output. In addition an RLum.Results object is returned containing the following element:

summary	data.frame summary of all relevant model results.
data	data.frame original input data
args	list used arguments
call	call the function call

The output should be accessed using the function get_RLum

Function version

0.2 (2016-05-02 09:36:06)

Author(s)

Christoph Burow, University of Cologne (Germany)
R Luminescence Package Team

References

Galbraith, R.F., 2003. A simple homogeneity test for estimates of dose obtained using OSL. Ancient TL 21, 75-77.

See Also

pchisq

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())

## apply the homogeneity test
calc_HomogeneityTest(ExampleData.DeValues$BT998)
```

---

calc_IEU

Apply the internal-external-uncertainty (IEU) model after Thomsen et al. (2007) to a given De distribution

---

Description

Function to calculate the IEU De for a De data set.

Usage

```
calc_IEU(data, a, b, interval, decimal.point = 2, plot = TRUE, ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
a	numeric: slope
b	numeric: intercept
interval	numeric: fixed interval (e.g. 5 Gy) used for iteration of Dbar, from the mean to Lowest.De used to create Graph.IEU [Dbar.Fixed vs Z]
decimal.point	numeric (with default): number of decimal points for rounding calculations (e.g. 2)
plot	logical (with default): plot output
...	further arguments (trace, verbose).

Details

This function uses the equations of Thomsen et al. (2007). The parameters a and b are estimated from dose-recovery experiments.

Value

Returns a plot (optional) and terminal output. In addition an RLum.Results object is returned containing the following element:

summary	data.frame summary of all relevant model results.
data	data.frame original input data
args	list used arguments
call	call the function call
tables	list a list of data frames containing all calculation tables

The output should be accessed using the function get_RLum.

Function version

0.1.0 (2016-05-02 09:36:06)

Author(s)

Rachel Smedley, Geography & Earth Sciences, Aberystwyth University (United Kingdom)
Based on an excel spreadsheet and accompanying macro written by Kristina Thomsen.
R Luminescence Package Team

References

Smedley, R.K., 2015. A new R function for the Internal External Uncertainty (IEU) model. Ancient TL 33, 16-21.

Thomsen, K.J., Murray, A.S., Boetter-Jensen, L. & Kinahan, J., 2007. Determination of burial dose in incompletely bleached fluvial samples using single grains of quartz. Radiation Measurements 42, 370-379.

See Also

plot, calc_CommonDose, calc_CentralDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

```
## load data
data(ExampleData.DeValues, envir = environment())

## apply the IEU model
ieu <- calc_IEU(ExampleData.DeValues$CA1, a = 0.2, b = 1.9, interval = 1)
```

`calc_MaxDose` *Apply the maximum age model to a given De distribution*

Description

Function to fit the maximum age model to De data. This is a wrapper function that calls calc_MinDose() and applies a similiar approach as described in Olley et al. (2006).

Usage

```
calc_MaxDose(data, sigmab, log = TRUE, par = 3, bootstrap = FALSE,
  init.values, plot = TRUE, ...)
```

Arguments

`data`	`RLum.Results` or `data.frame` (**required**): for `data.frame`: two columns with De (`data[,1]`) and De error (`values[,2]`)
`sigmab`	`numeric` (**required**): spread in De values given as a fraction (e.g. 0.2). This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100).
`log`	`logical` (with default): fit the (un-)logged three parameter minimum dose model to De data
`par`	`numeric` (with default): apply the 3- or 4-parametric minimum age model (`par=3` or `par=4`).
`bootstrap`	`logical` (with default): apply the recycled bootstrap approach of Cunningham & Wallinga (2012).
`init.values`	`numeric` (with default): starting values for gamma, sigma, p0 and mu. Custom values need to be provided in a vector of length three in the form of `c(gamma, sigma, p0)`.
`plot`	`logical` (with default): plot output (`TRUE`/`FALSE`)
`...`	further arguments for bootstrapping (`bs.M`, `bs.N`, `bs.h`,`sigmab.sd`). See details for their usage.

Details

Data transformation

To estimate the maximum dose population and its standard error, the three parameter minimum age model of Galbraith et al. (1999) is adapted. The measured De values are transformed as follows:

1. convert De values to natural logs
2. multiply the logged data to creat a mirror image of the De distribution
3. shift De values along x-axis by the smallest x-value found to obtain only positive values
4. combine in quadrature the measurement error associated with each De value with a relative error specified by sigmab
5. apply the MAM to these data

When all calculations are done the results are then converted as follows

1. subtract the x-offset
2. multiply the natural logs by -1
3. take the exponent to obtain the maximum dose estimate in Gy

Further documentation

Please see `calc_MinDose`.

Value

Please see `calc_MinDose`.

Function version

0.3 (2015-11-29 17:27:48)

Author(s)

Christoph Burow, University of Cologne (Germany)
Based on a rewritten S script of Rex Galbraith, 2010

R Luminescence Package Team

References

Arnold, L.J., Roberts, R.G., Galbraith, R.F. & DeLong, S.B., 2009. A revised burial dose estimation procedure for optical dating of young and modern-age sediments. Quaternary Geochronology 4, 306-325.

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks Radiation Measurements 4, 459-470.

Galbraith, R.F., Roberts, R.G., Laslett, G.M., Yoshida, H. & Olley, J.M., 1999. Optical dating of single grains of quartz from Jinmium rock shelter, northern Australia. Part I: experimental design and statistical models. Archaeometry 41, 339-364.

Galbraith, R.F., 2005. Statistics for Fission Track Analysis, Chapman & Hall/CRC, Boca Raton.

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

Olley, J.M., Roberts, R.G., Yoshida, H., Bowler, J.M., 2006. Single-grain optical dating of grave-infill associated with human burials at Lake Mungo, Australia. Quaternary Science Reviews 25, 2469-2474.

Further reading

Arnold, L.J. & Roberts, R.G., 2009. Stochastic modelling of multi-grain equivalent dose (De) distributions: Implications for OSL dating of sediment mixtures. Quaternary Geochronology 4, 204-230.

Bailey, R.M. & Arnold, L.J., 2006. Statistical modelling of single grain quartz De distributions

and an assessment of procedures for estimating burial dose. Quaternary Science Reviews 25, 2475-2502.

Cunningham, A.C. & Wallinga, J., 2012. Realizing the potential of fluvial archives using robust OSL chronologies. Quaternary Geochronology 12, 98-106.

Rodnight, H., Duller, G.A.T., Wintle, A.G. & Tooth, S., 2006. Assessing the reproducibility and accuracy of optical dating of fluvial deposits. Quaternary Geochronology 1, 109-120.

Rodnight, H., 2008. How many equivalent dose values are needed to obtain a reproducible distribution?. Ancient TL 26, 3-10.

See Also

calc_CentralDose, calc_CommonDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())

# apply the maximum dose model
calc_MaxDose(ExampleData.DeValues$CA1, sigmab = 0.2, par = 3)
```

---

calc_MinDose — *Apply the (un-)logged minimum age model (MAM) after Galbraith et al. (1999) to a given De distribution*

Description

Function to fit the (un-)logged three or four parameter minimum dose model (MAM-3/4) to De data.

Usage

```
calc_MinDose(data, sigmab, log = TRUE, par = 3, bootstrap = FALSE,
  init.values, level = 0.95, plot = TRUE, multicore = FALSE, ...)
```

Arguments

data	RLum.Results or data.frame (**required**): for data.frame: two columns with De (data[,1]) and De error (values[,2])
sigmab	numeric (**required**): spread in De values given as a fraction (e.g. 0.2). This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100).
log	logical (with default): fit the (un-)logged minimum dose model to De data
par	numeric (with default): apply the 3- or 4-parametric minimum age model (par=3 or par=4). The MAM-3 is used by default.

bootstrap	logical (with default): apply the recycled bootstrap approach of Cunningham & Wallinga (2012).
init.values	numeric (optional): a named list with starting values for gamma, sigma, p0 and mu (e.g. list(gamma=100 sigma=1.5, p0=0.1, mu=100)). If no values are provided reasonable values are tried to be estimated from the data.
level	logical (with default): the confidence level required (defaults to 0.95).
plot	logical (with default): plot output (TRUE/FALSE)
multicore	logical (with default): enable parallel computation of the bootstrap by creating a multicore SNOW cluster. Depending on the number of available logical CPU cores this will drastically reduce the computation time. Note that this option is highly experimental and not work for all machines. (TRUE/FALSE)
...	(optional) further arguments for bootstrapping (bs.M,bs.N, bs.h, sigmab.sd). See details for their usage. Further arguments are verbose to de-/activate console output (logical), debug for extended console output (logical) and cores (integer) to manually specify the number of cores to be used when multicore=TRUE.

Details

Parameters

This model has four parameters:

gamma:	minimum dose on the log scale
mu:	mean of the non-truncated normal distribution
sigma:	spread in ages above the minimum
p0:	proportion of grains at gamma

If par=3 (default) the 3-parametric minimum age model is applied, where gamma=mu. For par=4 the 4-parametric model is applied instead.

(Un-)logged model

In the original version of the three-parameter minimum dose model, the basic data are the natural logarithms of the De estimates and relative standard errors of the De estimates. This model will be applied if log=TRUE.

If log=FALSE, the modified un-logged model will be applied instead. This has essentially the same form as the original version. gamma and sigma are in Gy and gamma becomes the minimum true dose in the population.

While the original (logged) version of the mimimum dose model may be appropriate for most samples (i.e. De distributions), the modified (un-logged) version is specially designed for modern-age and young samples containing negative, zero or near-zero De estimates (Arnold et al. 2009, p. 323).

Initial values & boundaries

The log likelihood calculations use the nlminb function for box-constrained optimisation using PORT routines. Accordingly, initial values for the four parameters can be specified via init.values.

If no values are provided for `init.values` reasonable starting values are estimated from the input data. If the final estimates of *gamma*, *mu*, *sigma* and *p0* are totally off target, consider providing custom starting values via `init.values`.
In contrast to previous versions of this function the boundaries for the individual model parameters are no longer required to be explicitly specified. If you want to override the default boundary values use the arguments `gamma.lower`, `gamma.upper`, `sigma.lower`, `sigma.upper`, `p0.lower`, `p0.upper`, `mu.lower` and `mu.upper`.

Bootstrap

When `bootstrap=TRUE` the function applies the bootstrapping method as described in Wallinga & Cunningham (2012). By default, the minimum age model produces 1000 first level and 3000 second level bootstrap replicates (actually, the number of second level bootstrap replicates is three times the number of first level replicates unless specified otherwise). The uncertainty on sigmab is 0.04 by default. These values can be changed by using the arguments `bs.M` (first level replicates), `bs.N` (second level replicates) and `sigmab.sd` (error on sigmab). With `bs.h` the bandwidth of the kernel density estimate can be specified. By default, `h` is calculated as

$$h = (2 * \sigma_{DE})/\sqrt{n}$$

Multicore support

This function supports parallel computing and can be activated by `multicore=TRUE`. By default, the number of available logical CPU cores is determined automatically, but can be changed with `cores`. The multicore support is only available when `bootstrap=TRUE` and spawns `n` R instances for each core to get MAM estimates for each of the N and M boostrap replicates. Note that this option is highly experimental and may or may not work for your machine. Also the performance gain increases for larger number of bootstrap replicates. Also note that with each additional core and hence R instance and depending on the number of bootstrap replicates the memory usage can significantly increase. Make sure that memory is always availabe, otherwise there will be a massive perfomance hit.

Value

Returns a plot (optional) and terminal output. In addition an `RLum.Results` object is returned containing the following elements:

`summary`	data.frame summary of all relevant model results.
`data`	data.frame original input data
`args`	list used arguments
`call`	call the function call
`mle`	mle2 object containing the maximum log likelihood functions for all parameters
`BIC`	numeric BIC score
`confint`	data.frame confidence intervals for all parameters
`profile`	profile.mle2 the log likelihood profiles
`bootstrap`	list bootstrap results

The output should be accessed using the function `get_RLum`

Function version

0.4.3 (2016-09-09 10:32:17)

Note

The default starting values for *gamma*, *mu*, *sigma* and *p0* may only be appropriate for some De data sets and may need to be changed for other data. This is especially true when the un-logged version is applied.
Also note that all R warning messages are suppressed when running this function. If the results seem odd consider re-running the model with `debug=TRUE` which provides extended console output and forwards all internal warning messages.

Author(s)

Christoph Burow, University of Cologne (Germany)
Based on a rewritten S script of Rex Galbraith, 2010
The bootstrap approach is based on a rewritten MATLAB script of Alastair Cunningham.
Alastair Cunningham is thanked for his help in implementing and cross-checking the code.
R Luminescence Package Team

References

Arnold, L.J., Roberts, R.G., Galbraith, R.F. & DeLong, S.B., 2009. A revised burial dose estimation procedure for optical dating of young and modern-age sediments. Quaternary Geochronology 4, 306-325.

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks Radiation Measurements 4, 459-470.

Galbraith, R.F., Roberts, R.G., Laslett, G.M., Yoshida, H. & Olley, J.M., 1999. Optical dating of single grains of quartz from Jinmium rock shelter, northern Australia. Part I: experimental design and statistical models. Archaeometry 41, 339-364.

Galbraith, R.F., 2005. Statistics for Fission Track Analysis, Chapman & Hall/CRC, Boca Raton.

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

Further reading

Arnold, L.J. & Roberts, R.G., 2009. Stochastic modelling of multi-grain equivalent dose (De) distributions: Implications for OSL dating of sediment mixtures. Quaternary Geochronology 4, 204-230.

Bailey, R.M. & Arnold, L.J., 2006. Statistical modelling of single grain quartz De distributions and an assessment of procedures for estimating burial dose. Quaternary Science Reviews 25, 2475-2502.

Cunningham, A.C. & Wallinga, J., 2012. Realizing the potential of fluvial archives using robust OSL chronologies. Quaternary Geochronology 12, 98-106.

Rodnight, H., Duller, G.A.T., Wintle, A.G. & Tooth, S., 2006. Assessing the reproducibility and accuracy of optical dating of fluvial deposits. Quaternary Geochronology 1, 109-120.

Rodnight, H., 2008. How many equivalent dose values are needed to obtain a reproducible distribution?. Ancient TL 26, 3-10.

See Also

calc_CentralDose, calc_CommonDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MaxDose

Examples

```
## Load example data
data(ExampleData.DeValues, envir = environment())

# (1) Apply the minimum age model with minimum required parameters.
# By default, this will apply the un-logged 3-parametric MAM.
calc_MinDose(data = ExampleData.DeValues$CA1, sigmab = 0.1)

# (2) Re-run the model, but save results to a variable and turn
# plotting of the log-likelihood profiles off.
mam <- calc_MinDose(data = ExampleData.DeValues$CA1,
                    sigmab = 0.1,
                    plot = FALSE)

# Show structure of the RLum.Results object
mam

# Show summary table that contains the most relevant results
res <- get_RLum(mam, "summary")
res

# Plot the log likelihood profiles retroactively, because before
# we set plot = FALSE
plot_RLum(mam)

# Plot the dose distribution in an abanico plot and draw a line
# at the minimum dose estimate
plot_AbanicoPlot(data = ExampleData.DeValues$CA1,
                 main = "3-parameter Minimum Age Model",
                 line = mam,polygon.col = "none",
                 hist = TRUE,
                 rug = TRUE,
                 summary = c("n", "mean", "mean.weighted", "median", "in.ci"),
                 centrality = res$de,
                 line.col = "red",
                 grid.col = "none",
                 line.label = paste0(round(res$de, 1), "\U00B1",
                                     round(res$de_err, 1), " Gy"),
                 bw = 0.1,
                 ylim = c(-25, 18),
                 summary.pos = "topleft",
                 mtext = bquote("Parameters: " ~
                                sigma[b] == .(get_RLum(mam, "args")$sigmab) ~ ", " ~
                                gamma == .(round(log(res$de), 1)) ~ ", " ~
```

```
                                    sigma == .(round(res$sig, 1)) ~ ", " ~
                                    rho == .(round(res$p0, 2))))


## Not run:
# (3) Run the minimum age model with bootstrap
# NOTE: Bootstrapping is computationally intensive
# (3.1) run the minimum age model with default values for bootstrapping
calc_MinDose(data = ExampleData.DeValues$CA1,
             sigmab = 0.15,
             bootstrap = TRUE)

# (3.2) Bootstrap control parameters
mam <- calc_MinDose(data = ExampleData.DeValues$CA1,
                    sigmab = 0.15,
                    bootstrap = TRUE,
                    bs.M = 300,
                    bs.N = 500,
                    bs.h = 4,
                    sigmab.sd = 0.06,
                    plot = FALSE)

# Plot the results
plot_RLum(mam)

# save bootstrap results in a separate variable
bs <- get_RLum(mam, "bootstrap")

# show structure of the bootstrap results
str(bs, max.level = 2, give.attr = FALSE)

# print summary of minimum dose and likelihood pairs
summary(bs$pairs$gamma)

# Show polynomial fits of the bootstrap pairs
bs$poly.fits$poly.three

# Plot various statistics of the fit using the generic plot() function
par(mfcol=c(2,2))
plot(bs$poly.fits$poly.three, ask = FALSE)

# Show the fitted values of the polynomials
summary(bs$poly.fits$poly.three$fitted.values)

## End(Not run)
```

`calc_OSLLxTxRatio` *Calculate Lx/Tx ratio for CW-OSL curves*

Description

Calculate Lx/Tx ratios from a given set of CW-OSL curves assuming late light background subtraction.

Usage

```
calc_OSLLxTxRatio(Lx.data, Tx.data, signal.integral,
  signal.integral.Tx = NULL, background.integral,
  background.integral.Tx = NULL,
  background.count.distribution = "non-poisson", sigmab = NULL, sig0 = 0,
  digits = NULL)
```

Arguments

Lx.data	RLum.Data.Curve or data.frame (**required**): requires a CW-OSL shine down curve (x = time, y = counts)
Tx.data	RLum.Data.Curve or data.frame (optional): requires a CW-OSL shine down curve (x = time, y = counts). If no input is given the Tx.data will be treated as NA and no Lx/Tx ratio is calculated.
signal.integral	vector (**required**): vector with the limits for the signal integral.
signal.integral.Tx	vector (optional): vector with the limits for the signal integral for the Tx curve. If nothing is provided the value from signal.integral is used.
background.integral	vector (**required**): vector with the bounds for the background integral.
background.integral.Tx	vector (optional): vector with the limits for the background integral for the Tx curve. If nothing is provided the value from background.integral is used.
background.count.distribution	character (with default): sets the count distribution assumed for the error calculation. Possible arguments poisson or non-poisson. See details for further information
sigmab	numeric (optional): option to set a manual value for the overdispersion (for LnTx and TnTx), used for the Lx/Tx error calculation. The value should be provided as absolute squared count values, e.g. sigmab = c(300,300). Note: If only one value is provided this value is taken for both (LnTx and TnTx) signals.
sig0	numeric (with default): allow adding an extra component of error to the final Lx/Tx error value (e.g., instrumental errror, see details).
digits	integer (with default): round numbers to the specified digits. If digits is set to NULL nothing is rounded.

Details

The integrity of the chosen values for the signal and background integral is checked by the function; the signal integral limits have to be lower than the background integral limits. If a vector is given as input instead of a data.frame, an artificial data.frame is produced. The error calculation is done according to Galbraith (2002).

Please note: In cases where the calculation results in NaN values (for example due to zero-signal, and therefore a division of 0 by 0), these NaN values are replaced by 0.

sigmab

The default value of `sigmab` is calculated assuming the background is constant and **would not** applicable when the background varies as, e.g., as observed for the early light substraction method.

sig0

This argument allows to add an extra component of error to the final Lx/Tx error value. The input will be treated as factor that is multiplied with the already calculated LxTx and the result is add up by:

$$se(LxTx) = \sqrt{(se(LxTx)^2 + (LxTx * sig0)^2)}$$

background.count.distribution

This argument allows selecting the distribution assumption that is used for the error calculation. According to Galbraith (2002, 2014) the background counts may be overdispersed (i.e. do not follow a poisson distribution, which is assumed for the photomultiplier counts). In that case (might be the normal case) it has to be accounted for the overdispersion by estimating σ^2 (i.e. the overdispersion value). Therefore the relative standard error is calculated as:

(a) `poisson`

$$rse(\mu_S) \approx \sqrt{(Y_0 + Y_1/k^2)}/Y_0 - Y_1/k$$

(b) `non-poisson`

$$rse(\mu_S) \approx \sqrt{(Y_0 + Y_1/k^2 + \sigma^2(1 + 1/k))}/Y_0 - Y_1/k$$

Please note that when using the early background subtraction method in combination with the 'non-poisson' distribution argument, the corresponding Lx/Tx error may considerably increase due to a high sigmab value. Please check whether this is valid for your data set and if necessary consider to provide an own sigmab value using the corresponding argument `sigmab`.

Value

Returns an S4 object of type `RLum.Results`.

Slot `data` contains a `list` with the following structure:
$LxTx.table (data.frame)
.. $ LnLx
.. $ LnLx.BG
.. $ TnTx
.. $ TnTx.BG
.. $ Net_LnLx
.. $ Net_LnLx.Error
.. $ Net_TnTx.Error
.. $ LxTx
.. $ LxTx.Error
$ calc.parameters (list)
.. $ sigmab.LnTx
.. $ sigmab.TnTx
.. $ k
$ call (original function call)

Function version

0.6.3 (2016-09-09 10:32:17)

Note

The results of this function have been cross-checked with the Analyst (vers. 3.24b). Access to the results object via `get_RLum`.

Caution: If you are using early light subtraction (EBG), please either provide your own `sigmab` value or use `background.count.distribution = "poisson"`.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Duller, G., 2007. Analyst. http://www.nutech.dtu.dk/english/~/media/Andre_Universitetsenheder/Nutech/Produkter%20og%20services/Dosimetri/radiation_measurement_instruments/tl_osl_reader/Manuals/analyst_manual_v3_22b.ashx

Galbraith, R.F., 2002. A note on the variance of a background-corrected OSL count. Ancient TL, 20 (2), 49-51.

Galbraith, R.F., 2014. A further note on the variance of a background-corrected OSL count. Ancient TL, 31 (2), 1-3.

See Also

`RLum.Data.Curve`, `Analyse_SAR.OSLdata`, `plot_GrowthCurve`, `analyse_SAR.CWOSL`

Examples

```
##load data
data(ExampleData.LxTxOSLData, envir = environment())

##calculate Lx/Tx ratio
results <- calc_OSLLxTxRatio(Lx.data, Tx.data, signal.integral = c(1:2),
                             background.integral = c(85:100))

##get results object
get_RLum(results)
```

`calc_SourceDoseRate` *Calculation of the source dose rate via the date of measurement*

Description

Calculating the dose rate of the irradiation source via the date of measurement based on: source calibration date, source dose rate, dose rate error. The function returns a data.frame that provides the input argument dose_rate for the function `Second2Gray`.

Usage

```
calc_SourceDoseRate(measurement.date, calib.date, calib.dose.rate, calib.error,
  source.type = "Sr-90", dose.rate.unit = "Gy/s", predict = NULL)
```

Arguments

`measurement.date`	`character` or `Date` (**required**): date of measurement in "YYYY-MM-DD". Exceptionally, if no value is provided, the date will be set to today. The argument can be provided as vector.
`calib.date`	`character` or `Date` (**required**): date of source calibration in "YYYY-MM-DD"
`calib.dose.rate`	`numeric` (**required**): dose rate at date of calibration in Gy/s or Gy/min
`calib.error`	`numeric` (**required**): error of dose rate at date of calibration Gy/s or Gy/min
`source.type`	`character` (with default): specify irrdiation source (`Sr-90` or `Co-60` or `Am-214`), see details for further information
`dose.rate.unit`	`character` (with default): specify dose rate unit for input (`Gy/min` or `Gy/s`), the output is given in Gy/s as valid for the function `Second2Gray`
`predict`	`integer` (with default): option allowing to predicit the dose rate of the source over time in days set by the provided value. Starting date is the value set with `measurement.date`, e.g., `calc_SourceDoseRate(...,predict = 100)` calculates the source dose rate for the next 100 days.

Details

Calculation of the source dose rate based on the time elapsed since the last calibration of the irradiation source. Decay parameters assume a Sr-90 beta source.

$$dose.rate = D0 * exp(-log(2)/T.1/2 * t)$$

with: D0 <- calibration dose rate T.1/2 <- half-life of the source nuclide (here in days) t <- time since source calibration (in days) log(2) / T.1/2 equals the decay constant lambda

Information on the date of measurements may be taken from the data's original .BIN file (using e.g., BINfile <- readBIN2R() and the slot BINfile@METADATA$DATE)

Allowed source types and related values

#	Source type	T.1/2	Reference
[1]	Sr-90	28.90 y	NNDC, Brookhaven National Laboratory
[2]	Am-214	432.6 y	NNDC, Brookhaven National Laboratory
[3]	Co-60	5.274 y	NNDC, Brookhaven National Laboratory

Value

Returns an S4 object of type RLum.Results. Slot data contains a list with the following structure:
$ dose.rate (data.frame)
.. $ dose.rate
.. $ dose.rate.error
.. $ date (corresponding measurement date)
$ parameters (list)
.. $ source.type
.. $ halflife
.. $ dose.rate.unit
$ call (the original function call)

The output should be accessed using the function get_RLum.
A plot method of the output is provided via plot_RLum

Function version

0.3.0 (2015-11-29 17:27:48)

Note

Please be careful when using the option predict, especially when a multiple set for measurement.date and calib.date is provided. For the source dose rate prediction the function takes the last value measurement.date and predicts from that the the source source dose rate for the number of days requested, means: the (multiple) orignal input will be replaced. However, the function do not change entries for the calibration dates, but mix them up. Therefore, it is not recommended to use this option when multiple calibration dates (calib.date) are provided.

Author(s)

Margret C. Fuchs, HZDR, Helmholtz-Institute Freiberg for Resource Technology (Germany),
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

NNDC, Brookhaven National Laboratory (http://www.nndc.bnl.gov/)

See Also

Second2Gray, get_RLum, plot_RLum

Examples

```
##(1) Simple function usage
##Basic calculation of the dose rate for a specific date
dose.rate <-  calc_SourceDoseRate(measurement.date = "2012-01-27",
                                  calib.date = "2014-12-19",
                                  calib.dose.rate = 0.0438,
                                  calib.error = 0.0019)
```

```
##show results
get_RLum(dose.rate)

##(2) Usage in combination with another function (e.g., Second2Gray() )
## load example data
data(ExampleData.DeValues, envir = environment())

## use the calculated variable dose.rate as input argument
## to convert De(s) to De(Gy)
Second2Gray(ExampleData.DeValues$BT998, dose.rate)

##(3) source rate prediction and plotting
dose.rate <-  calc_SourceDoseRate(measurement.date = "2012-01-27",
                                  calib.date = "2014-12-19",
                                  calib.dose.rate = 0.0438,
                                  calib.error = 0.0019,
                                  predict = 1000)
plot_RLum(dose.rate)


##(4) export output to a LaTeX table (example using the package 'xtable')
## Not run:
xtable::xtable(get_RLum(dose.rate))


## End(Not run)
```

`calc_Statistics` *Function to calculate statistic measures*

Description

This function calculates a number of descriptive statistics for De-data, most fundamentally using error-weighted approaches.

Usage

```
calc_Statistics(data, weight.calc = "square", digits = NULL, n.MCM = 1000,
  na.rm = TRUE)
```

Arguments

`data`	`data.frame` or `RLum.Results` object (required): for `data.frame` two columns: De (`data[,1]`) and De error (`data[,2]`). To plot several data sets in one plot the data sets must be provided as `list`, e.g. `list(data.1, data.2)`.
`weight.calc`	`character`: type of weight calculation. One out of `"reciprocal"` (weight is 1/error), `"square"` (weight is 1/error^2). Default is `"square"`.
`digits`	`integer` (with default): round numbers to the specified digits. If digits is set to `NULL` nothing is rounded.
`n.MCM`	`numeric` (with default): number of samples drawn for Monte Carlo-based statistics. Set to zero to disable this option.

na.rm	logical (with default): indicating whether NA values should be stripped before the computation proceeds.

Details

The option to use Monte Carlo Methods (n.MCM > 0) allows calculating all descriptive statistics based on random values. The distribution of these random values is based on the Normal distribution with De values as means and De_error values as one standard deviation. Increasing the number of MCM-samples linearly increases computation time. On a Lenovo X230 machine evaluation of 25 Aliquots with n.MCM = 1000 takes 0.01 s, with n = 100000, ca. 1.65 s. It might be useful to work with logarithms of these values. See Dietze et al. (2016, Quaternary Geochronology) and the function plot_AbanicoPlot for details.

Value

Returns a list with weighted and unweighted statistic measures.

Function version

0.1.6 (2016-05-16 22:14:31)

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())

## show a rough plot of the data to illustrate the non-normal distribution
plot_KDE(ExampleData.DeValues$BT998)

## calculate statistics and show output
str(calc_Statistics(ExampleData.DeValues$BT998))

## Not run:
## now the same for 10000 normal distributed random numbers with equal errors
x <- as.data.frame(cbind(rnorm(n = 10^5, mean = 0, sd = 1),
                         rep(0.001, 10^5)))

## note the congruent results for weighted and unweighted measures
str(calc_Statistics(x))

## End(Not run)
```

`calc_ThermalLifetime` *Calculates the Thermal Lifetime using the Arrhenius equation*

Description

The function calculates the thermal lifetime of charges for given E (in eV), s (in 1/s) and T (in deg. C.) parameters. The function can be used in two operational modes:

Usage

```
calc_ThermalLifetime(E, s, T = 20, output_unit = "Ma", profiling = FALSE,
  profiling_config = NULL, verbose = TRUE, plot = TRUE, ...)
```

Arguments

`E`	`numeric` (**required**): vector of trap depths in eV, if `profiling = TRUE` only the first two elements are considered
`s`	`numeric` (**required**): vector of frequency factor in 1/s, if `profiling = TRUE` only the first two elements are considered
`T`	`numeric` (with default): temperature in deg. C for which the lifetime(s) will be calculted. A vector can be provided.
`output_unit`	`character` (with default): output unit of the calculated lifetimes, accepted entries are: `"Ma"`, `"ka"`, `"a"`, `"d"`, `"h"`, `"min"`, `"s"`
`profiling`	`logical` (with default): this option allows to estimate uncertainties based on given E and s parameters and their corresponding standard error (cf. details and examples section)
`profiling_config`	`list` (optional): allows to set configurate parameters used for the profiling (and only have an effect here). Supported parameters are: `n` (number of MC runs), `E.distribution` (distribution used for the resampling for E) and `s.distribution` (distribution used for the resampling for s). Currently only the normal distribution is supported (e.g., `profiling_config = list(E.distribution = "norm")`
`verbose`	`logical`: enables/disables verbose mode
`plot`	`logical`: enables/disables output plot, currenlty only in combination with `profiling = TRUE`.
`...`	further arguments that can be passed in combination with the plot output. Standard plot parameters are supported (`plot.default`)

Details

Mode 1 (`profiling = FALSE`)

An arbitrary set of input parameters (E, s, T) can be provided and the function calculates the thermal lifetimes using the Arrhenius equation for all possible combinations of these input parameters. An array with 3-dimensions is returned that can be used for further analyses or graphical output (see example 1)

Mode 2 (`profiling = TRUE`)

This mode tries to profile the variation of the thermal lifetime for a chosen temperature by accounting for the provided E and s parameters and their corresponding standard errors, e.g., `E = c(1.600, 0.001)`

The calculation based on a Monte Carlo simulation, where values are sampled from a normal distribution (for E and s).

Used equation (Arrhenius equation)

$$\tau = 1/sexp(E/kT)$$

where: τ in s as the mean time an electron spends in the trap for a given T, E trap depth in eV, s the frequency factor in 1/s, T the temperature in K and k the Boltzmann constant in eV/K (cf. Furetta, 2010).

Value

A RLum.Results object is returned a along with a plot (for `profiling = TRUE`). The output object contain the following slots:

`@data`

Object	Type	Description
`lifetimes`	array or numeric	calculated lifetimes
`profiling_matrix`	matrix	profiling matrix used for the MC runs

`@info`

Object	Type	Description
`call`	`call`	the original function call

Function version

0.1.0 (2016-05-02 09:36:06)

Note

The profiling is currently based on resampling from a normal distribution, this distribution assumption might be, however, not valid for given E and s paramters.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Furetta, C., 2010. Handbook of Thermoluminescence, Second Edition. ed. World Scientific.

See Also

matplot, rnorm, get_RLum,

Examples

```
##EXAMPLE 1
##calculation for two trap-depths with similar frequency factor for different temperatures
E <- c(1.66, 1.70)
s <- 1e+13
T <- 10:20
temp <- calc_ThermalLifetime(
  E = E,
  s = s,
  T = T,
  output_unit = "Ma"
)
contour(x = E, y = T, z = temp$lifetimes[1,,],
        ylab = "Temperature [\u00B0C]",
        xlab = "Trap depth [eV]",
        main = "Thermal Lifetime Contour Plot"
)
mtext(side = 3, "(values quoted in Ma)")

##EXAMPLE 2
##profiling of thermal life time for E and s and their standard error
E <- c(1.600, 0.003)
s <- c(1e+13,1e+011)
T <- 20
calc_ThermalLifetime(
  E = E,
  s = s,
  T = T,
  profiling = TRUE,
  output_unit = "Ma"
)
```

---

calc_TLLxTxRatio *Calculate the Lx/Tx ratio for a given set of TL curves [beta version]*

Description

Calculate Lx/Tx ratio for a given set of TL curves.

Usage

```
calc_TLLxTxRatio(Lx.data.signal, Lx.data.background, Tx.data.signal,
  Tx.data.background, signal.integral.min, signal.integral.max)
```

Arguments

Lx.data.signal	RLum.Data.Curve or data.frame (**required**): TL data (x = temperature, y = counts) (TL signal)
Lx.data.background	RLum.Data.Curve or data.frame (optional): TL data (x = temperature, y = counts). If no data are provided no background subtraction is performed.

`Tx.data.signal` RLum.Data.Curve or data.frame (**required**): TL data (x = temperature, y = counts) (TL test signal)

`Tx.data.background`
RLum.Data.Curve or data.frame (optional): TL data (x = temperature, y = counts). If no data are provided no background subtraction is performed.

`signal.integral.min`
integer (**required**): channel number for the lower signal integral bound (e.g. `signal.integral.min = 100`)

`signal.integral.max`
integer (**required**): channel number for the upper signal integral bound (e.g. `signal.integral.max = 200`)

Details

-

Value

Returns an S4 object of type RLum.Results. Slot data contains a list with the following structure:

$ LxTx.table
.. $ LnLx
.. $ LnLx.BG
.. $ TnTx
.. $ TnTx.BG
.. $ Net_LnLx
.. $ Net_LnLx.Error

Function version

0.3.0 (2015-11-29 17:27:48)

Note

This function has still BETA status!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Christoph Schmidt, University of Bayreuth (Germany)
R Luminescence Package Team

References

-

See Also

RLum.Results, analyse_SAR.TL

Examples

```
##load package example data
data(ExampleData.BINfileData, envir = environment())

##convert Risoe.BINfileData into a curve object
temp <- Risoe.BINfileData2RLum.Analysis(TL.SAR.Data, pos = 3)


Lx.data.signal <- get_RLum(temp, record.id=1)
Lx.data.background <- get_RLum(temp, record.id=2)
Tx.data.signal <- get_RLum(temp, record.id=3)
Tx.data.background <- get_RLum(temp, record.id=4)
signal.integral.min <- 210
signal.integral.max <- 230

output <- calc_TLLxTxRatio(Lx.data.signal,
                           Lx.data.background,
                           Tx.data.signal, Tx.data.background,
                           signal.integral.min, signal.integral.max)
get_RLum(output)
```

CW2pHMi *Transform a CW-OSL curve into a pHM-OSL curve via interpolation under hyperbolic modulation conditions*

Description

This function transforms a conventionally measured continuous-wave (CW) OSL-curve to a pseudo hyperbolic modulated (pHM) curve under hyperbolic modulation conditions using the interpolation procedure described by Bos & Wallinga (2012).

Usage

```
CW2pHMi(values, delta)
```

Arguments

values	RLum.Data.Curve or data.frame (**required**): RLum.Data.Curve or data.frame with measured curve data of type stimulation time (t) (values[,1]) and measured counts (cts) (values[,2]).
delta	vector (optional): stimulation rate parameter, if no value is given, the optimal value is estimated automatically (see details). Smaller values of delta produce more points in the rising tail of the curve.

Details

The complete procedure of the transformation is described in Bos & Wallinga (2012). The input data.frame consists of two columns: time (t) and count values (CW(t))

Internal transformation steps

(1) log(CW-OSL) values

(2) Calculate t' which is the transformed time:

$$t' = t - (1/\delta) * log(1 + \delta * t)$$

(3) Interpolate CW(t'), i.e. use the log(CW(t)) to obtain the count values for the transformed time (t'). Values beyond `min(t)` and `max(t)` produce `NA` values.

(4) Select all values for t' < `min(t)`, i.e. values beyond the time resolution of t. Select the first two values of the transformed data set which contain no `NA` values and use these values for a linear fit using `lm`.

(5) Extrapolate values for t' < `min(t)` based on the previously obtained fit parameters.

(6) Transform values using

$$pHM(t) = (\delta * t/(1 + \delta * t)) * c * CW(t')$$

$$c = (1 + \delta * P)/\delta * P$$

$$P = length(stimulation\ period)$$

(7) Combine all values and truncate all values for t' > `max(t)`

The number of values for t' < `min(t)` *depends on the stimulation rate parameter* `delta`*. To avoid the production of too many artificial data at the raising tail of the determined pHM curve, it is recommended to use the automatic estimation routine for* `delta`*, i.e. provide no value for* `delta`*.*

Value

The function returns the same data type as the input data type with the transformed curve values.

`list(list("RLum.Data.Curve"))`
package `RLum` `object` with two additional info elements:

$CW2pHMi.x.t	: transformed time values
$CW2pHMi.method	: used method for the production of the new data points

`list(list("data.frame"))`
with four columns:

$x	: time
$y.t	: transformed count values
$x.t	: transformed time values
$method	: used method for the production of the new data points

Function version

0.2.2 (2015-11-29 17:27:48)

Note

According to Bos & Wallinga (2012), the number of extrapolated points should be limited to avoid artificial intensity data. If `delta` is provided manually and more than two points are extrapolated, a warning message is returned.

The function `approx` may produce some `Inf` and `NaN` data. The function tries to manually interpolate these values by calculating the `mean` using the adjacent channels. If two invalid values are succeeding, the values are removed and no further interpolation is attempted.
In every case a warning message is shown.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

Based on comments and suggestions from:
Adrie J.J. Bos, Delft University of Technology, The Netherlands

R Luminescence Package Team

References

Bos, A.J.J. & Wallinga, J., 2012. How to visualize quartz OSL signal components. Radiation Measurements, 47, 752-758.

Further Reading

Bulur, E., 1996. An Alternative Technique For Optically Stimulated Luminescence (OSL) Experiment. Radiation Measurements, 26, 701-709.

Bulur, E., 2000. A simple transformation for converting CW-OSL curves to LM-OSL curves. Radiation Measurements, 32, 141-145.

See Also

`CW2pLM`, `CW2pLMi`, `CW2pPMi`, `fit_LMCurve`, `lm`, `RLum.Data.Curve`

Examples

```
##(1) - simple transformation

##load CW-OSL curve data
data(ExampleData.CW_OSL_Curve, envir = environment())

##transform values
values.transformed<-CW2pHMi(ExampleData.CW_OSL_Curve)

##plot
plot(values.transformed$x, values.transformed$y.t, log = "x")

##(2) - load CW-OSL curve from BIN-file and plot transformed values

##load BINfile
#BINfileData<-readBIN2R("[path to BIN-file]")
```

```
data(ExampleData.BINfileData, envir = environment())

##grep first CW-OSL curve from ALQ 1
curve.ID<-CWOSL.SAR.Data@METADATA[CWOSL.SAR.Data@METADATA[,"LTYPE"]=="OSL" &
                                    CWOSL.SAR.Data@METADATA[,"POSITION"]==1
                                  ,"ID"]

curve.HIGH<-CWOSL.SAR.Data@METADATA[CWOSL.SAR.Data@METADATA[,"ID"]==curve.ID[1]
                                    ,"HIGH"]

curve.NPOINTS<-CWOSL.SAR.Data@METADATA[CWOSL.SAR.Data@METADATA[,"ID"]==curve.ID[1]
                                       ,"NPOINTS"]

##combine curve to data set

curve<-data.frame(x = seq(curve.HIGH/curve.NPOINTS,curve.HIGH,
                          by = curve.HIGH/curve.NPOINTS),
                  y=unlist(CWOSL.SAR.Data@DATA[curve.ID[1]]))


##transform values

curve.transformed <- CW2pHMi(curve)

##plot curve
plot(curve.transformed$x, curve.transformed$y.t, log = "x")


##(3) - produce Fig. 4 from Bos & Wallinga (2012)

##load data
data(ExampleData.CW_OSL_Curve, envir = environment())
values <- CW_Curve.BosWallinga2012

##open plot area
plot(NA, NA,
     xlim=c(0.001,10),
     ylim=c(0,8000),
     ylab="pseudo OSL (cts/0.01 s)",
     xlab="t [s]",
     log="x",
     main="Fig. 4 - Bos & Wallinga (2012)")

values.t<-CW2pLMi(values, P=1/20)
lines(values[1:length(values.t[,1]),1],CW2pLMi(values, P=1/20)[,2],
      col="red" ,lwd=1.3)
text(0.03,4500,"LM", col="red" ,cex=.8)

values.t<-CW2pHMi(values, delta=40)
lines(values[1:length(values.t[,1]),1],CW2pHMi(values, delta=40)[,2],
      col="black", lwd=1.3)
text(0.005,3000,"HM", cex=.8)

values.t<-CW2pPMi(values, P=1/10)
lines(values[1:length(values.t[,1]),1],CW2pPMi(values, P=1/10)[,2],
      col="blue", lwd=1.3)
text(0.5,6500,"PM", col="blue" ,cex=.8)
```

CW2pLM *Transform a CW-OSL curve into a pLM-OSL curve*

Description

Transforms a conventionally measured continuous-wave (CW) curve into a pseudo linearly modulated (pLM) curve using the equations given in Bulur (2000).

Usage

```
CW2pLM(values)
```

Arguments

values	RLum.Data.Curve or data.frame (**required**): RLum.Data.Curve data object. Alternatively, a data.frame of the measured curve data of type stimulation time (t) (values[,1]) and measured counts (cts) (values[,2]) can be provided.

Details

According to Bulur (2000) the curve data are transformed by introducing two new parameters P (stimulation period) and u (transformed time):

$$P = 2 * max(t)$$

$$u = \sqrt{(2 * t * P)}$$

The new count values are then calculated by

$$ctsNEW = cts(u/P)$$

and the returned data.frame is produced by: data.frame(u,ctsNEW)

Value

The function returns the same data type as the input data type with the transformed curve values.

list(list("data.frame"))
: generic R data structure

list(list("RLum.Data.Curve"))
: package RLum object

Function version

0.4.1 (2015-11-29 17:27:48)

Note

The transformation is recommended for curves recorded with a channel resolution of at least 0.05 s/channel.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Bulur, E., 2000. A simple transformation for converting CW-OSL curves to LM-OSL curves. Radiation Measurements, 32, 141-145.

Further Reading

Bulur, E., 1996. An Alternative Technique For Optically Stimulated Luminescence (OSL) Experiment. Radiation Measurements, 26, 701-709.

See Also

CW2pHMi, CW2pLMi, CW2pPMi, fit_LMCurve, lm, RLum.Data.Curve

The output of the function can be further used for LM-OSL fitting: CW2pLMi, CW2pHMi, CW2pPMi, fit_LMCurve, RLum.Data.Curve, plot_RLum

Examples

```
##read curve from CWOSL.SAR.Data transform curve and plot values
data(ExampleData.BINfileData, envir = environment())

##read id for the 1st OSL curve
id.OSL <- CWOSL.SAR.Data@METADATA[CWOSL.SAR.Data@METADATA[,"LTYPE"] == "OSL","ID"]

##produce x and y (time and count data for the data set)
x<-seq(CWOSL.SAR.Data@METADATA[id.OSL[1],"HIGH"]/CWOSL.SAR.Data@METADATA[id.OSL[1],"NPOINTS"],
       CWOSL.SAR.Data@METADATA[id.OSL[1],"HIGH"],
     by = CWOSL.SAR.Data@METADATA[id.OSL[1],"HIGH"]/CWOSL.SAR.Data@METADATA[id.OSL[1],"NPOINTS"])
y <- unlist(CWOSL.SAR.Data@DATA[id.OSL[1]])
values <- data.frame(x,y)

##transform values
values.transformed <- CW2pLM(values)

##plot
plot(values.transformed)
```

CW2pLMi — *Transform a CW-OSL curve into a pLM-OSL curve via interpolation under linear modulation conditions*

Description

Transforms a conventionally measured continuous-wave (CW) OSL-curve into a pseudo linearly modulated (pLM) curve under linear modulation conditions using the interpolation procedure described by Bos & Wallinga (2012).

Usage

```
CW2pLMi(values, P)
```

Arguments

values	RLum.Data.Curve or data.frame (**required**): RLum.Data.Curve or data.frame with measured curve data of type stimulation time (t) (values[,1]) and measured counts (cts) (values[,2])
P	vector (optional): stimulation time in seconds. If no value is given the optimal value is estimated automatically (see details). Greater values of P produce more points in the rising tail of the curve.

Details

The complete procedure of the transformation is given in Bos & Wallinga (2012). The input data.frame consists of two columns: time (t) and count values (CW(t))

Nomenclature

P = stimulation time (s)
1/P = stimulation rate (1/s)

Internal transformation steps

(1) log(CW-OSL) values
(2) Calculate t' which is the transformed time:

$$t' = 1/2 * 1/P * t^2$$

(3) Interpolate CW(t'), i.e. use the log(CW(t)) to obtain the count values for the transformed time (t'). Values beyond min(t) and max(t) produce NA values.

(4) Select all values for t' < min(t), i.e. values beyond the time resolution of t. Select the first two values of the transformed data set which contain no NA values and use these values for a linear fit using lm.

(5) Extrapolate values for t' < min(t) based on the previously obtained fit parameters.

(6) Transform values using

$$pLM(t) = t/P * CW(t')$$

(7) Combine values and truncate all values for t' > max(t)

The number of values for t' < min(t) depends on the stimulation period (P) and therefore on the stimulation rate 1/P. To avoid the production of too many artificial data at the raising tail of the determined pLM curves it is recommended to use the automatic estimation routine for P*, i.e. provide no own value for* P.

Value

The function returns the same data type as the input data type with the transformed curve values.

`list(list("RLum.Data.Curve"))`
package RLum object with two additional info elements:

$CW2pLMi.x.t	: transformed time values
$CW2pLMi.method	: used method for the production of the new data points

Function version

0.3.1 (2015-11-29 17:27:48)

Note

According to Bos & Wallinga (2012) the number of extrapolated points should be limited to avoid artificial intensity data. If `P` is provided manually and more than two points are extrapolated, a warning message is returned.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne

Based on comments and suggestions from:
Adrie J.J. Bos, Delft University of Technology, The Netherlands

R Luminescence Package Team

References

Bos, A.J.J. & Wallinga, J., 2012. How to visualize quartz OSL signal components. Radiation Measurements, 47, 752-758.

Further Reading

Bulur, E., 1996. An Alternative Technique For Optically Stimulated Luminescence (OSL) Experiment. Radiation Measurements, 26, 701-709.

Bulur, E., 2000. A simple transformation for converting CW-OSL curves to LM-OSL curves. Radiation Measurements, 32, 141-145.

See Also

CW2pLM, CW2pHMi, CW2pPMi, fit_LMCurve, RLum.Data.Curve

Examples

```
##(1)
##load CW-OSL curve data
data(ExampleData.CW_OSL_Curve, envir = environment())

##transform values
```

```
values.transformed <- CW2pLMi(ExampleData.CW_OSL_Curve)

##plot
plot(values.transformed$x, values.transformed$y.t, log = "x")

##(2) - produce Fig. 4 from Bos & Wallinga (2012)
##load data
data(ExampleData.CW_OSL_Curve, envir = environment())
values <- CW_Curve.BosWallinga2012

##open plot area
plot(NA, NA,
     xlim = c(0.001,10),
     ylim = c(0,8000),
     ylab = "pseudo OSL (cts/0.01 s)",
     xlab = "t [s]",
     log = "x",
     main = "Fig. 4 - Bos & Wallinga (2012)")


values.t <- CW2pLMi(values, P = 1/20)
lines(values[1:length(values.t[,1]),1],CW2pLMi(values, P = 1/20)[,2],
      col = "red", lwd = 1.3)
text(0.03,4500,"LM", col = "red", cex = .8)

values.t <- CW2pHMi(values, delta = 40)
lines(values[1:length(values.t[,1]),1],CW2pHMi(values, delta = 40)[,2],
      col = "black", lwd = 1.3)
text(0.005,3000,"HM", cex =.8)

values.t <- CW2pPMi(values, P = 1/10)
lines(values[1:length(values.t[,1]),1], CW2pPMi(values, P = 1/10)[,2],
      col = "blue", lwd = 1.3)
text(0.5,6500,"PM", col = "blue", cex = .8)
```

CW2pPMi *Transform a CW-OSL curve into a pPM-OSL curve via interpolation under parabolic modulation conditions*

Description

Transforms a conventionally measured continuous-wave (CW) OSL-curve into a pseudo parabolic modulated (pPM) curve under parabolic modulation conditions using the interpolation procedure described by Bos & Wallinga (2012).

Usage

```
CW2pPMi(values, P)
```

Arguments

values	RLum.Data.Curve or data.frame (**required**): RLum.Data.Curve or data.frame with measured curve data of type stimulation time (t) (values[,1]) and measured counts (cts) (values[,2])
P	vector (optional): stimulation period in seconds. If no value is given, the optimal value is estimated automatically (see details). Greater values of P produce more points in the rising tail of the curve.

Details

The complete procedure of the transformation is given in Bos & Wallinga (2012). The input data.frame consists of two columns: time (t) and count values (CW(t))

Nomenclature

P = stimulation time (s)
1/P = stimulation rate (1/s)

Internal transformation steps

(1) log(CW-OSL) values

(2) Calculate t' which is the transformed time:

$$t' = (1/3) * (1/P^2)t^3$$

(3) Interpolate CW(t'), i.e. use the log(CW(t)) to obtain the count values for the transformed time (t'). Values beyond min(t) and max(t) produce NA values.

(4) Select all values for t' < min(t), i.e. values beyond the time resolution of t. Select the first two values of the transformed data set which contain no NA values and use these values for a linear fit using lm.

(5) Extrapolate values for t' < min(t) based on the previously obtained fit parameters. The extrapolation is limited to two values. Other values at the beginning of the transformed curve are set to 0.

(6) Transform values using

$$pLM(t) = t^2/P^2 * CW(t')$$

(7) Combine all values and truncate all values for t' > max(t)

The number of values for t' < min(t) depends on the stimulation period P. To avoid the production of too many artificial data at the raising tail of the determined pPM curve, it is recommended to use the automatic estimation routine for P, i.e. provide no value for P.

Value

The function returns the same data type as the input data type with the transformed curve values.

`list(list("RLum.Data.Curve"))`
package `RLum` `object` with two additional info elements:

$CW2pPMi.x.t	: transformed time values
$CW2pPMi.method	: used method for the production of the new data points

`list(list("data.frame"))`
with four columns:

$x	: time
$y.t	: transformed count values
$x.t	: transformed time values
$method	: used method for the production of the new data points

Function version

0.2.1 (2015-11-29 17:27:48)

Note

According to Bos & Wallinga (2012), the number of extrapolated points should be limited to avoid artificial intensity data. If `P` is provided manually, not more than two points are extrapolated.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

Based on comments and suggestions from:
Adrie J.J. Bos, Delft University of Technology, The Netherlands

R Luminescence Package Team

References

Bos, A.J.J. & Wallinga, J., 2012. How to visualize quartz OSL signal components. Radiation Measurements, 47, 752-758.

Further Reading

Bulur, E., 1996. An Alternative Technique For Optically Stimulated Luminescence (OSL) Experiment. Radiation Measurements, 26, 701-709.

Bulur, E., 2000. A simple transformation for converting CW-OSL curves to LM-OSL curves. Radiation Measurements, 32, 141-145.

See Also

`CW2pLM`, `CW2pLMi`, `CW2pHMi`, `fit_LMCurve`, `RLum.Data.Curve`

Examples

```
##(1)
##load CW-OSL curve data
```

```
data(ExampleData.CW_OSL_Curve, envir = environment())

##transform values
values.transformed <- CW2pPMi(ExampleData.CW_OSL_Curve)

##plot
plot(values.transformed$x,values.transformed$y.t, log = "x")

##(2) - produce Fig. 4 from Bos & Wallinga (2012)

##load data
data(ExampleData.CW_OSL_Curve, envir = environment())
values <- CW_Curve.BosWallinga2012

##open plot area
plot(NA, NA,
     xlim = c(0.001,10),
     ylim = c(0,8000),
     ylab = "pseudo OSL (cts/0.01 s)",
     xlab = "t [s]",
     log = "x",
     main = "Fig. 4 - Bos & Wallinga (2012)")

values.t <- CW2pLMi(values, P = 1/20)
lines(values[1:length(values.t[,1]),1],CW2pLMi(values, P = 1/20)[,2],
      col = "red",lwd = 1.3)
text(0.03,4500,"LM", col = "red", cex = .8)

values.t <- CW2pHMi(values, delta = 40)
lines(values[1:length(values.t[,1]),1], CW2pHMi(values, delta = 40)[,2],
      col = "black", lwd = 1.3)
text(0.005,3000,"HM", cex = .8)

values.t <- CW2pPMi(values, P = 1/10)
lines(values[1:length(values.t[,1]),1], CW2pPMi(values, P = 1/10)[,2],
      col = "blue", lwd = 1.3)
text(0.5,6500,"PM", col = "blue", cex = .8)
```

`ExampleData.BINfileData` *Example data from a SAR OSL and SAR TL measurement for the package Luminescence*

Description

Example data from a SAR OSL and TL measurement for package Luminescence directly extracted from a Risoe BIN-file and provided in an object of type Risoe.BINfileData-class

Format

`CWOSL.SAR.Data`: SAR OSL measurement data

`TL.SAR.Data`: SAR TL measurement data

Each class object contains two slots: (a) METADATA is a data.frame with all metadata stored in the BIN file of the measurements and (b) DATA contains a list of vectors of the measured data (usually count values).

Version

0.1

Note

Please note that this example data cannot be exported to a BIN-file using the function writeR2BIN as it was generated and implemented in the package long time ago. In the meantime the BIN-file format changed.

Source

CWOSL.SAR.Data

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT607
Location:	Saxony/Germany
Material:	Middle grain quartz measured on aluminum cups on a Risoe TL/OSL DA-15 reader
Reference:	unpublished

TL.SAR.Data

Lab:	Luminescence Laboratory of Cologne
Lab-Code:	LP1_5
Location:	Spain
Material:	Flint
Setup:	Risoe TL/OSL DA-20 reader (Filter: Semrock Brightline, HC475/50, N2, unpolished steel discs)
Reference:	unpublished
Remarks:	dataset limited to one position

References

CWOSL.SAR.Data: unpublished data

TL.SAR.Data: unpublished data

Examples

```
##show first 5 elements of the METADATA and DATA elements in the terminal
data(ExampleData.BINfileData, envir = environment())
CWOSL.SAR.Data@METADATA[1:5,]
CWOSL.SAR.Data@DATA[1:5]
```

`ExampleData.CW_OSL_Curve` *Example CW-OSL curve data for the package Luminescence*

Description

`data.frame` containing CW-OSL curve data (time, counts)

Format

Data frame with 1000 observations on the following 2 variables:

list("x") a numeric vector, time

list("y") a numeric vector, counts

Source

ExampleData.CW_OSL_Curve

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT607
Location:	Saxony/Germany
Material:	Middle grain quartz measured on aluminum cups on a Risoe TL/OSL DA-15 reader.
Reference:	unpublished data

CW_Curve.BosWallinga2012

Lab:	Netherlands Centre for Luminescence Dating (NCL)
Lab-Code:	NCL-2108077
Location:	Guadalentin Basin, Spain
Material:	Coarse grain quartz
Reference:	Bos & Wallinga (2012) and Baartman et al. (2011)

References

Baartman, J.E.M., Veldkamp, A., Schoorl, J.M., Wallinga, J., Cammeraat, L.H., 2011. Unravelling Late Pleistocene and Holocene landscape dynamics: The Upper Guadalentin Basin, SE Spain. Geomorphology, 125, 172-185.

Bos, A.J.J. & Wallinga, J., 2012. How to visualize quartz OSL signal components. Radiation Measurements, 47, 752-758.

Examples

```
data(ExampleData.CW_OSL_Curve, envir = environment())
plot(ExampleData.CW_OSL_Curve)
```

`ExampleData.DeValues` *Example De data sets for the package Luminescence*

Description

Equivalent dose (De) values measured for a fine grain quartz sample from a loess section in Rottewitz (Saxony/Germany) and for a coarse grain quartz sample from a fluvial deposit in the rock shelter of Cueva Anton (Murcia/Spain).

Format

A `list` with two elements, each containing a two column `data.frame`:

`$BT998`: De and De error values for a fine grain quartz sample from a loess section in Rottewitz.

`$CA1`: Single grain De and De error values for a coarse grain quartz sample from a fluvial deposit in the rock shelter of Cueva Anton

References

BT998

Unpublished data

CA1

Burow, C., Kehl, M., Hilgers, A., Weniger, G.-C., Angelucci, D., Villaverde, V., Zapata, J. and Zilhao, J. (2015). Luminescence dating of fluvial deposits in the rock shelter of Cueva Anton, Spain. Geochronometria 52, 107-125.

BT998

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT998
Location:	Rottewitz (Saxony/Germany)
Material:	Fine grain quartz measured on aluminum discs on a Risoe TL/OSL DA-15 reader
Units:	Values are given in seconds
Dose Rate:	Dose rate of the beta-source at measurement ca. 0.0438 Gy/s +/- 0.0019 Gy/s
Measurement Date:	2012-01-27

CA1

Lab:	Cologne Luminescence Laboratory (CLL)
Lab-Code:	C-L2941
Location:	Cueva Anton (Murcia/Spain)
Material:	Coarse grain quartz (200-250 microns) measured on single grain discs on a Risoe TL/OSL DA-20 r
Units:	Values are given in Gray
Measurement Date:	2012

Examples

```
##(1) plot values as histogram
data(ExampleData.DeValues, envir = environment())
plot_Histogram(ExampleData.DeValues$BT998, xlab = "De [s]")

##(2) plot values as histogram (with second to gray conversion)
data(ExampleData.DeValues, envir = environment())

De.values <- Second2Gray(ExampleData.DeValues$BT998,
                         dose.rate = c(0.0438, 0.0019))


plot_Histogram(De.values, xlab = "De [Gy]")
```

ExampleData.FittingLM *Example data for fit_LMCurve() in the package Luminescence*

Description

Lineraly modulated (LM) measurement data from a quartz sample from Norway including background measurement. Measurements carried out in the luminescence laboratory at the University of Bayreuth.

Format

Two objects (data.frames) with two columns (time and counts).

Source

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT900
Location:	Norway
Material:	Beach deposit, coarse grain quartz measured on aluminum discs on a Risoe TL/OSL DA-15 reader

References

Fuchs, M., Kreutzer, S., Fischer, M., Sauer, D., Soerensen, R., 2012. OSL and IRSL dating of raised beach sand deposits along the southeastern coast of Norway. Quaternary Geochronology, 10, 195-200.

Examples

```
##show LM data
data(ExampleData.FittingLM, envir = environment())
plot(values.curve,log="x")
```

ExampleData.LxTxData *Example Lx/Tx data from CW-OSL SAR measurement*

Description

LxTx data from a SAR measurement for the package Luminescence.

Format

A `data.frame` with 4 columns (Dose, LxTx, LxTx.Error, TnTx).

Source

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT607
Location:	Ostrau (Saxony-Anhalt/Germany)
Material:	Middle grain (38-63 μm) quartz measured on a Risoe TL/OSL DA-15 reader.

References

unpublished data

Examples

```
##plot Lx/Tx data vs dose [s]
data(ExampleData.LxTxData, envir = environment())
plot(LxTxData$Dose,LxTxData$LxTx)
```

ExampleData.LxTxOSLData *Example Lx and Tx curve data from an artificial OSL measurement*

Description

Lx and Tx data of continous wave (CW-) OSL signal curves.

Format

Two `data.frames` containing time and count values.

Source

Arbitrary OSL measurement.

References

unpublished data

Examples

```
##load data
data(ExampleData.LxTxOSLData, envir = environment())

##plot data
plot(Lx.data)
plot(Tx.data)
```

`ExampleData.RLum.Analysis`

Example data as `RLum.Analysis` *objects*

Description

Collection of different `RLum.Analysis` objects for protocol analysis.

Format

`IRSAR.RF.Data`: IRSAR.RF.Data on coarse grain feldspar

Each object contains data needed for the given protocol analysis.

Version

0.1

Source

IRSAR.RF.Data

These data were kindly provided by Tobias Lauer and Matthias Krbetschek.

Lab:	Luminescence Laboratory TU Bergakademie Freiberg
Lab-Code:	ZEU/SA1
Location:	Zeuchfeld (Zeuchfeld Sandur; Saxony-Anhalt/Germany)
Material:	K-feldspar (130-200 μm)
Reference:	Kreutzer et al. (2014)

References

IRSAR.RF.Data

Kreutzer, S., Lauer, T., Meszner, S., Krbetschek, M.R., Faust, D., Fuchs, M., 2014. Chronology of the Quaternary profile Zeuchfeld in Saxony-Anhalt / Germany - a preliminary luminescence dating study. Zeitschrift fuer Geomorphologie 58, 5-26. doi: 10.1127/0372-8854/2012/S-00112

Examples

```
##load data
data(ExampleData.RLum.Analysis, envir = environment())

##plot data
plot_RLum(IRSAR.RF.Data)
```

ExampleData.RLum.Data.Image

Example data as RLum.Data.Image *objects*

Description

Measurement of Princton Instruments camera imported with the function read_SPE2R to R to produce an RLum.Data.Image object.

Format

Object of class RLum.Data.Image

Version

0.1

Source

ExampleData.RLum.Data.Image

These data were kindly provided by Regina DeWitt.

Lab.:	Department of Physics, East-Carolina University, NC, USA
Lab-Code:	-
Location:	-
Material:	-
Reference:	-

Image data is a measurement of fluorescent ceiling lights with a cooled Princeton Instruments (TM) camera fitted on Risoe DA-20 TL/OSL reader.

Examples

```
##load data
data(ExampleData.RLum.Data.Image, envir = environment())

##plot data
plot_RLum(ExampleData.RLum.Data.Image)
```

`ExampleData.XSYG` *Example data for a SAR OSL measurement and a TL spectrum using a lexsyg reader*

Description

Example data from a SAR OSL measurement and a TL spectrum for package Luminescence imported from a Freiberg Instruments XSYG file using the function `read_XSYG2R`.

Format

`OSL.SARMeasurement`: SAR OSL measurement data

The data contain two elements: (a) `$Sequence.Header` is a data.frame with metadata from the measurement,(b) `Sequence.Object` contains an `RLum.Analysis` object for further analysis.

`TL.Spectrum`: TL spectrum data

`RLum.Data.Spectrum` object for further analysis. The spectrum was cleaned from cosmic-rays using the function `apply_CosmicRayRemoval`. Note that no quantum efficiency calibration was performed.

Version

0.1

Source

OSL.SARMeasurement

Lab:	Luminescence Laboratory Giessen
Lab-Code:	no code
Location:	not specified
Material:	Coarse grain quartz
	on steel cups on lexsyg research reader
Reference:	unpublished

TL.Spectrum

Lab:	Luminescence Laboratory Giessen
Lab-Code:	BT753
Location:	Dolni Vestonice/Czech Republic
Material:	Fine grain polymineral
	on steel cups on lexsyg rearch reader
Reference:	Fuchs et al., 2013
Spectrum:	Integration time 19 s, channel time 20 s
Heating:	1 K/s, up to 500 deg. C

References

Unpublished data measured to serve as example data for that package. Location origin of sample BT753 is given here:

Fuchs, M., Kreutzer, S., Rousseau, D.D., Antoine, P., Hatte, C., Lagroix, F., Moine, O., Gauthier, C., Svoboda, J., Lisa, L., 2013. The loess sequence of Dolni Vestonice, Czech Republic: A new OSL-based chronology of the Last Climatic Cycle. Boreas, 42, 664–677.

See Also

read_XSYG2R, RLum.Analysis, RLum.Data.Spectrum, plot_RLum, plot_RLum.Analysis, plot_RLum.Data.Spectrum

Examples

```
##show data
data(ExampleData.XSYG, envir = environment())

## ========================================
##(1) OSL.SARMeasurement
OSL.SARMeasurement

##show $Sequence.Object
OSL.SARMeasurement$Sequence.Object

##grep OSL curves and plot the first curve
OSLcurve <- get_RLum(OSL.SARMeasurement$Sequence.Object,
recordType="OSL")[[1]]
plot_RLum(OSLcurve)

## ========================================
##(2) TL.Spectrum
TL.Spectrum

##plot simple spectrum (2D)
plot_RLum.Data.Spectrum(TL.Spectrum,
                        plot.type="contour",
                        xlim = c(310,750),
                        ylim = c(0,300),
                        bin.rows=10,
                        bin.cols = 1)

##plot 3d spectrum (uncomment for usage)
# plot_RLum.Data.Spectrum(TL.Spectrum, plot.type="persp",
# xlim = c(310,750), ylim = c(0,300), bin.rows=10,
# bin.cols = 1)
```

---

extract_IrradiationTimes

Extract irradiation times from an XSYG file

Description

Extracts irradiation times, dose and times since last irradiation, from a Freiberg Instruments XSYG-file. These information can be further used to update an existing BINX-file

Usage

```
extract_IrradiationTimes(object, file.BINX, recordType = c("irradiation (NA)",
  "IRSL (UVVIS)", "OSL (UVVIS)", "TL (UVVIS)"), compatibility.mode = TRUE,
  txtProgressBar = TRUE)
```

Arguments

object	character (**required**) or RLum.Analysis object or list: path and file name of the XSYG file or an RLum.Analysis produced by the function read_XSYG2R; alternatively a list of RLum.Analysis can be provided. **Note**: If an RLum.Analysis is used, any input for the arguments file.BINX and recordType will be ignored!
file.BINX	character (optional): path and file name of an existing BINX-file. If a file name is provided the file will be updated with the information from the XSYG file in the same folder as the original BINX-file. Note: The XSYG and the BINX-file have to be originate from the same measurement!
recordType	character (with default): select relevant curves types from the XSYG file or RLum.Analysis object. As the XSYG-file format comprises much more information than usually needed for routine data analysis and allowed in the BINX-file format, only the relevant curves are selected by using the function get_RLum. The argument recordType works as described for this function. Note: A wrong selection will causes a function error. Please change this argument only if you have reasons to do so.
compatibility.mode	logical (with default): this option is parsed only if a BIN/BINX file is produced and it will reset all position values to a max. value of 48, cf.write_R2BIN
txtProgressBar	logical (with default): enables TRUE or disables FALSE the progression bars during import and export

Details

The function was written to compensate missing information in the BINX-file output of Freiberg Instruments lexsyg readers. As all information are available within the XSYG-file anyway, these information can be extracted and used for further analysis or/and to stored in a new BINX-file, which can be further used by other software, e.g. Analyst (Geoff Duller).

Typical application example: g-value estimation from fading measurements using the Analyst or any other self written script.

Beside the some simple data transformation steps the function applies the functions read_XSYG2R, read_BIN2R, write_R2BIN for data import and export.

Value

An RLum.Results object is returned with the following structure:
.. $irr.times (data.frame)

If a BINX-file path and name is set, the output will be additionally transferred into a new BINX-file with the function name as suffix. For the output the path of the input BINX-file itself is used. Note that this will not work if the input object is a file path to an XSYG-file. In this case the argument input is ignored.

In the self call mode (input is a `list` of `RLum.Analysis` objects a list of `RLum.Results` is returned.

Function version

0.3.0 (2016-05-03 11:10:26)

Note

The produced output object contains still the irradiation steps to keep the output transparent. However, for the BINX-file export this steps are removed as the BINX-file format description does not allow irradiations as separat sequences steps.

Know issue: The 'fading correction' menu in the Analyst will not work appear with the produced BIN/BINX-file due to hidden bits, which are not reproduced by the function `write_R2BIN()` or if it appears it stops with a floating point error.

Negative values for `TIMESINCELAS.STEP`? Yes, this is possible and no bug, as in the XSYG file multiple curves are stored for one step. Example: A TL step may comprise three curves: (a) counts vs. time, (b) measured temperature vs. time and (c) predefined temperature vs. time. Three curves, but they are all belonging to one TL measurement step, but with regard to the time stamps this could produce negative values as the important function (`read_XSYG2R`) do not change the order of entries for one step towards a correct time order.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Duller, G., 2007. Analyst.

See Also

`RLum.Analysis`, `RLum.Results`, `Risoe.BINfileData`, `read_XSYG2R`, `read_BIN2R`, `write_R2BIN`

Examples

```
## (1) - example for your own data
##
## set files and run function
#
#   file.XSYG <- file.choose()
#   file.BINX <- file.choose()
#
#     output <- extract_IrradiationTimes(file.XSYG = file.XSYG, file.BINX = file.BINX)
#     get_RLum(output)
```

```
#
## export results additionally to a CSV.file in the same directory as the XSYG-file
#       write.table(x = get_RLum(output),
#                   file = paste0(file.BINX,"_extract_IrradiationTimes.csv"),
#                   sep = ";",
#                   row.names = FALSE)
```

`fit_CWCurve` *Nonlinear Least Squares Fit for CW-OSL curves [beta version]*

Description

The function determines the weighted least-squares estimates of the component parameters of a CW-OSL signal for a given maximum number of components and returns various component parameters. The fitting procedure uses the `nls` function with the `port` algorithm.

Usage

```
fit_CWCurve(values, n.components.max, fit.failure_threshold = 5,
  fit.method = "port", fit.trace = FALSE, fit.calcError = FALSE,
  LED.power = 36, LED.wavelength = 470, cex.global = 0.6,
  sample_code = "Default", output.path, output.terminal = TRUE,
  output.terminalAdvanced = TRUE, plot = TRUE, ...)
```

Arguments

`values`	`RLum.Data.Curve` or data.frame (**required**): x, y data of measured values (time and counts). See examples.
`n.components.max`	vector (optional): maximum number of components that are to be used for fitting. The upper limit is 7.
`fit.failure_threshold`	vector (with default): limits the failed fitting attempts.
`fit.method`	character (with default): select fit method, allowed values: `'port'` and `'LM'`. `'port'` uses the 'port' routine usint the funtion `nls` `'LM'` utilises the function `nlsLM` from the package `minpack.lm` and with that the Levenberg-Marquardt algorithm.
`fit.trace`	logical (with default): traces the fitting process on the terminal.
`fit.calcError`	logical (with default): calculate 1-sigma error range of components using `confint`
`LED.power`	numeric (with default): LED power (max.) used for intensity ramping in mW/cm^2. **Note:** The value is used for the calculation of the absolute photoionisation cross section.
`LED.wavelength`	numeric (with default): LED wavelength used for stimulation in nm. **Note:** The value is used for the calculation of the absolute photoionisation cross section.
`cex.global`	numeric (with default): global scaling factor.
`sample_code`	character (optional): sample code used for the plot and the optional output table (mtext).

`output.path` character (optional): output path for table output containing the results of the fit. The file name is set automatically. If the file already exists in the directory, the values are appended.

`output.terminal` logical (with default): terminal ouput with fitting results.

`output.terminalAdvanced` logical (with default): enhanced terminal output. Requires `output.terminal = TRUE`. If `output.terminal = FALSE` no advanced output is possible.

`plot` logical (with default): returns a plot of the fitted curves.

`...` further arguments and graphical parameters passed to `plot`.

Details

Fitting function

The function for the CW-OSL fitting has the general form:

$$y = I0_1 * \lambda_1 * exp(-\lambda_1 * x) + , \ldots , + I0_i * \lambda_i * exp(-\lambda_i * x)$$

where $0 < i < 8$

and λ is the decay constant and $N0$ the intial number of trapped electrons.
(for the used equation cf. Boetter-Jensen et al., 2003)

Start values

Start values are estimated automatically by fitting a linear function to the logarithmized input data set. Currently, there is no option to manually provide start parameters.

Goodness of fit

The goodness of the fit is given as pseudoR^2 value (pseudo coefficient of determination). According to Lave (1970), the value is calculated as:

$$pseudoR^2 = 1 - RSS/TSS$$

where $RSS = Residual\ Sum\ of\ Squares$
and $TSS = Total\ Sum\ of\ Squares$

Error of fitted component parameters

The 1-sigma error for the components is calculated using the function `confint`. Due to considerable calculation time, this option is deactived by default. In addition, the error for the components can be estimated by using internal R functions like `summary`. See the `nls` help page for more information.

For details on the nonlinear regression in R, see Ritz & Streibig (2008).

Value

`plot` (optional) the fitted CW-OSL curves are returned as plot.

`table` (optional) an output table (*.csv) with parameters of the fitted components is provided if the `output.path` is set.

`list(list("RLum.Results"))`
beside the plot and table output options, an `RLum.Results` object is returned.

`fit`: an `nls` object (`$fit`) for which generic R functions are provided, e.g. summary, confint, profile. For more details, see nls.

`output.table`: a data.frame containing the summarised parameters including the error
`component.contribution.matrix`: matrix containing the values for the component to sum contribution plot (`$component.contribution.matrix`).

Matrix structure:
Column 1 and 2: time and `rev(time)` values
Additional columns are used for the components, two for each component, containing I0 and n0. The last columns `cont.` provide information on the relative component contribution for each time interval including the row sum for this values.

`object` beside the plot and table output options, an `RLum.Results` object is returned.

`fit`: an `nls` object (`$fit`) for which generic R functions are provided, e.g. summary, confint, profile. For more details, see nls.

`output.table`: a data.frame containing the summarised parameters including the error
`component.contribution.matrix`: matrix containing the values for the component to sum contribution plot (`$component.contribution.matrix`).

Matrix structure:
Column 1 and 2: time and `rev(time)` values
Additional columns are used for the components, two for each component, containing I0 and n0. The last columns `cont.` provide information on the relative component contribution for each time interval including the row sum for this values.

Function version

0.5.1 (2015-11-29 17:27:48)

Note

Beta version - This function has not been properly tested yet and should therefore not be used for publication purposes!

The pseudo-R^2 may not be the best parameter to describe the goodness of the fit. The trade off between the `n.components` and the pseudo-R^2 value is currently not considered.

The function **does not** ensure that the fitting procedure has reached a global minimum rather than a local minimum!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Boetter-Jensen, L., McKeever, S.W.S., Wintle, A.G., 2003. Optically Stimulated Luminescence Dosimetry. Elsevier Science B.V.

Lave, C.A.T., 1970. The Demand for Urban Mass Transportation. The Review of Economics and Statistics, 52 (3), 320-323.

Ritz, C. & Streibig, J.C., 2008. Nonlinear Regression with R. In: R. Gentleman, K. Hornik, G. Parmigiani, eds., Springer, p. 150.

See Also

fit_LMCurve, plot,nls, RLum.Data.Curve, RLum.Results, get_RLum, nlsLM

Examples

```
##load data
data(ExampleData.CW_OSL_Curve, envir = environment())

##fit data
fit <- fit_CWCurve(values = ExampleData.CW_OSL_Curve,
                   main = "CW Curve Fit",
                   n.components.max = 4,
                   log = "x")
```

---

fit_LMCurve *Nonlinear Least Squares Fit for LM-OSL curves*

Description

The function determines weighted nonlinear least-squares estimates of the component parameters of an LM-OSL curve (Bulur 1996) for a given number of components and returns various component parameters. The fitting procedure uses the function nls with the port algorithm.

Usage

```
fit_LMCurve(values, values.bg, n.components = 3, start_values,
  input.dataType = "LM", fit.method = "port", sample_code = "",
  sample_ID = "", LED.power = 36, LED.wavelength = 470,
  fit.trace = FALSE, fit.advanced = FALSE, fit.calcError = FALSE,
  bg.subtraction = "polynomial", verbose = TRUE, plot = TRUE,
  plot.BG = FALSE, ...)
```

Arguments

values	RLum.Data.Curve or data.frame (**required**): x,y data of measured values (time and counts). See examples.
values.bg	RLum.Data.Curve or data.frame (optional): x,y data of measured values (time and counts) for background subtraction.
n.components	integer (with default): fixed number of components that are to be recognised during fitting (min = 1, max = 7).
start_values	data.frame (optional): start parameters for lm and xm data for the fit. If no start values are given, an automatic start value estimation is attempted (see details).
input.dataType	character (with default): alter the plot output depending on the input data: "LM" or "pLM" (pseudo-LM). See: CW2pLM
fit.method	character (with default): select fit method, allowed values: 'port' and 'LM'. 'port' uses the 'port' routine usint the funtion nls 'LM' utilises the function nlsLM from the package minpack.lm and with that the Levenberg-Marquardt algorithm.
sample_code	character (optional): sample code used for the plot and the optional output table (mtext).
sample_ID	character (optional): additional identifier used as column header for the table output.
LED.power	numeric (with default): LED power (max.) used for intensity ramping in mW/cm^2. **Note:** This value is used for the calculation of the absolute photoionisation cross section.
LED.wavelength	numeric (with default): LED wavelength in nm used for stimulation. **Note:** This value is used for the calculation of the absolute photoionisation cross section.
fit.trace	logical (with default): traces the fitting process on the terminal.
fit.advanced	logical (with default): enables advanced fitting attempt for automatic start parameter recognition. Works only if no start parameters are provided. **Note:** It may take a while and it is not compatible with fit.method = "LM".
fit.calcError	logical (with default): calculate 1-sigma error range of components using confint.
bg.subtraction	character (with default): specifies method for background subtraction (polynomial, linear, channel, see Details). **Note:** requires input for values.bg.
verbose	logical (with default): terminal output with fitting results.
plot	logical (with default): returns a plot of the fitted curves.
plot.BG	logical (with default): returns a plot of the background values with the fit used for the background subtraction.
...	Further arguments that may be passed to the plot output, e.g. xlab, xlab, main, log.

Details

Fitting function

The function for the fitting has the general form:

$$y = (exp(0.5) * Im_1 * x / xm_1) * exp(-x^2/(2 * xm_1^2)) + , \ldots , + exp(0.5) * Im_i * x / xm_i) * exp(-x^2/(2 * xm_i^2))$$

where $1 < i < 8$
This function and the equations for the conversion to b (detrapping probability) and n0 (proportional to initially trapped charge) have been taken from Kitis et al. (2008):

$$xm_i = \sqrt{max(t)/b_i}$$

$$Im_i = exp(-0.5)n0/xm_i$$

Background subtraction

Three methods for background subtraction are provided for a given background signal (`values.bg`). `polynomial`: default method. A polynomial function is fitted using glm and the resulting function is used for background subtraction:

$$y = a * x^4 + b * x^3 + c * x^2 + d * x + e$$

`linear`: a linear function is fitted using glm and the resulting function is used for background subtraction:

$$y = a * x + b$$

`channel`: the measured background signal is subtracted channelwise from the measured signal.

Start values

The choice of the initial parameters for the `nls`-fitting is a crucial point and the fitting procedure may mainly fail due to ill chosen start parameters. Here, three options are provided:

(a) If no start values (`start_values`) are provided by the user, a cheap guess is made by using the detrapping values found by Jain et al. (2003) for quartz for a maximum of 7 components. Based on these values, the pseudo start parameters xm and Im are recalculated for the given data set. In all cases, the fitting starts with the ultra-fast component and (depending on `n.components`) steps through the following values. If no fit could be achieved, an error plot (for `plot = TRUE`) with the pseudo curve (based on the pseudo start parameters) is provided. This may give the opportunity to identify appropriate start parameters visually.

(b) If start values are provided, the function works like a simple `nls` fitting approach.

(c) If no start parameters are provided and the option `fit.advanced = TRUE` is chosen, an advanced start paramter estimation is applied using a stochastical attempt. Therefore, the recalculated start parameters **(a)** are used to construct a normal distribution. The start parameters are then sampled randomly from this distribution. A maximum of 100 attempts will be made. **Note:** This process may be time consuming.

Goodness of fit

The goodness of the fit is given by a pseudoR^2 value (pseudo coefficient of determination). According to Lave (1970), the value is calculated as:

$$pseudoR^2 = 1 - RSS/TSS$$

where $RSS = Residual\ Sum\ of\ Squares$
and $TSS = Total\ Sum\ of\ Squares$

Error of fitted component parameters

The 1-sigma error for the components is calculated using the function confint. Due to considerable calculation time, this option is deactived by default. In addition, the error for the components can be estimated by using internal R functions like summary. See the nls help page for more information.
For more details on the nonlinear regression in R, see Ritz & Streibig (2008).

Value

Various types of plots are returned. For details see above.
Furthermore an `RLum.Results` object is returned with the following structure:

data:
.. $fit : `nls` (nls object)
.. $output.table : `data.frame` with fitting results
.. $component.contribution.matrix : `list` component distribution matrix
.. $call : `call` the original function call

Matrix structure for the distribution matrix:

Column 1 and 2: time and `rev(time)` values
Additional columns are used for the components, two for each component, containing I0 and n0. The last columns `cont.` provide information on the relative component contribution for each time interval including the row sum for this values.

Function version

0.3.1 (2016-05-02 09:36:06)

Note

The pseudo-R^2 may not be the best parameter to describe the goodness of the fit. The trade off between the `n.components` and the pseudo-R^2 value currently remains unconsidered.

The function **does not** ensure that the fitting procedure has reached a global minimum rather than a local minimum! In any case of doubt, the use of manual start values is highly recommended.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Bulur, E., 1996. An Alternative Technique For Optically Stimulated Luminescence (OSL) Experiment. Radiation Measurements, 26, 5, 701-709.

Jain, M., Murray, A.S., Boetter-Jensen, L., 2003. Characterisation of blue-light stimulated luminescence components in different quartz samples: implications for dose measurement. Radiation Measurements, 37 (4-5), 441-449.

Kitis, G. & Pagonis, V., 2008. Computerized curve deconvolution analysis for LM-OSL. Radiation Measurements, 43, 737-741.

Lave, C.A.T., 1970. The Demand for Urban Mass Transportation. The Review of Economics and Statistics, 52 (3), 320-323.

Ritz, C. & Streibig, J.C., 2008. Nonlinear Regression with R. R. Gentleman, K. Hornik, & G. Parmigiani, eds., Springer, p. 150.

See Also

fit_CWCurve, plot, nls, nlsLM, get_RLum

Examples

```
##(1) fit LM data without background subtraction
data(ExampleData.FittingLM, envir = environment())
fit_LMCurve(values = values.curve, n.components = 3, log = "x")

##(2) fit LM data with background subtraction and export as JPEG
## -alter file path for your preferred system
##jpeg(file = "~/Desktop/Fit_Output\%03d.jpg", quality = 100,
## height = 3000, width = 3000, res = 300)
data(ExampleData.FittingLM, envir = environment())
fit_LMCurve(values = values.curve, values.bg = values.curveBG,
            n.components = 2, log = "x", plot.BG = TRUE)
##dev.off()

##(3) fit LM data with manual start parameters
data(ExampleData.FittingLM, envir = environment())
fit_LMCurve(values = values.curve,
            values.bg = values.curveBG,
            n.components = 3,
            log = "x",
            start_values = data.frame(Im = c(170,25,400), xm = c(56,200,1500)))
```

---

get_Layout *Collection of layout definitions*

Description

This helper function returns a list with layout definitions for homogeneous plotting.

Usage

```
get_Layout(layout)
```

Arguments

layout	character or list object (required): name of the layout definition to be returned. If name is provided the respective definition is returned. One of the following supported layout definitions is possible: "default", "journal.1", "small", "empty". User-specific layout definitions must be provided as a list object of predefined structure, see details.

Details

The easiest way to create a user-specific layout definition is perhaps to create either an empty or a default layout object and fill/modify the definitions (`user.layout <- get_Layout(data = "empty")`).

Value

A list object with layout definitions for plot functions.

Function version

0.1 (2016-05-17 22:39:50)

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

Examples

```
## read example data set
data(ExampleData.DeValues, envir = environment())

## show structure of the default layout definition
layout.default <- get_Layout(layout = "default")
str(layout.default)

## show colour definitions for Abanico plot, only
layout.default$abanico$colour

## set Abanico plot title colour to orange
layout.default$abanico$colour$main <- "orange"

## create Abanico plot with modofied layout definition
plot_AbanicoPlot(data = ExampleData.DeValues,
                 layout = layout.default)

## create Abanico plot with predefined layout "journal"
plot_AbanicoPlot(data = ExampleData.DeValues,
                 layout = "journal")
```

---

get_Quote *Function to return essential quotes*

Description

This function returns one of the collected essential quotes in the growing library. If called without any parameters, a random quote is returned.

Usage

```
get_Quote(ID, author, separated = FALSE)
```

Arguments

ID	character, qoute ID to be returned.
author	character, all quotes by specified author.
separated	logical, return result in separated form.

Value

Returns a character with quote and respective (false) author.

Function version

0.1.1 (2016-09-09 10:32:17)

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

Examples

```
## ask for an arbitrary qoute
get_Quote()
```

---

get_rightAnswer *Function to get the right answer*

Description

This function returns just the right answer

Usage

```
get_rightAnswer(...)
```

Arguments

...	you can pass an infinite number of further arguments

Value

Returns the right answer

Function version

0.1.0 (2015-11-29 17:27:48)

Author(s)

inspired by R.G.
R Luminescence Package Team

Examples

```
## you really want to know?
get_rightAnswer()
```

`get_Risoe.BINfileData` *General accessor function for RLum S4 class objects*

Description

Function calls object-specific get functions for RisoeBINfileData S4 class objects.

Usage

```
get_Risoe.BINfileData(object, ...)
```

Arguments

`object`	`Risoe.BINfileData` (**required**): S4 object of class `RLum`
`...`	further arguments that one might want to pass to the specific get function

Details

The function provides a generalised access point for specific `Risoe.BINfileData` objects. Depending on the input object, the corresponding get function will be selected. Allowed arguments can be found in the documentations of the corresponding `Risoe.BINfileData` class.

Value

Return is the same as input objects as provided in the list.

Function version

0.1.0 (2015-11-29 17:27:48)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

`Risoe.BINfileData`

get_RLum *General accessor function for RLum S4 class objects*

Description

Function calls object-specific get functions for RLum S4 class objects.

Usage

```
get_RLum(object, ...)

## S4 method for signature 'list'
get_RLum(object, null.rm = FALSE, ...)
```

Arguments

object	RLum (**required**): S4 object of class RLum or an object of type list containing only objects of type RLum
null.rm	logical (with default): option to get rid of empty and NULL objects
...	further arguments that will be passed to the object specific methods. For furter details on the supported arguments please see the class documentation: RLum.Data.Curve, RLum.Data.Spectrum, RLum.Data.Image, RLum.Analysis and RLum.Results

Details

The function provides a generalised access point for specific RLum objects.
Depending on the input object, the corresponding get function will be selected. Allowed arguments can be found in the documentations of the corresponding RLum class.

Value

Return is the same as input objects as provided in the list.

Methods (by class)

- list: Returns a list of RLum objects that had been passed to get_RLum

Function version

0.3.0 (2016-05-02 09:40:57)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##Example based using data and from the calc_CentralDose() function

##load example data
data(ExampleData.DeValues, envir = environment())

##apply the central dose model 1st time
temp1 <- calc_CentralDose(ExampleData.DeValues$CA1)

##get results and store them in a new object
temp.get <- get_RLum(object = temp1)
```

`length_RLum` *General accessor function for RLum S4 class objects*

Description

Function calls object-specific get functions for RLum S4 class objects.

Usage

```
length_RLum(object)
```

Arguments

`object`	`RLum`	(**required**): S4 object of class `RLum`

Details

The function provides a generalised access point for specific `RLum` objects.
Depending on the input object, the corresponding get function will be selected. Allowed arguments can be found in the documentations of the corresponding `RLum` class.

Value

Return is the same as input objects as provided in the list.

Function version

0.1.0 (2016-05-02 09:36:06)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

`RLum.Data.Curve`, `RLum.Data.Image`, `RLum.Data.Spectrum`, `RLum.Analysis`, `RLum.Results`

merge_Risoe.BINfileData
Merge Risoe.BINfileData objects or Risoe BIN-files

Description

Function allows merging Risoe BIN/BINX files or Risoe.BINfileData objects.

Usage

```
merge_Risoe.BINfileData(input.objects, output.file,
  keep.position.number = FALSE, position.number.append.gap = 0)
```

Arguments

input.objects	character or Risoe.BINfileData (**required**): Character vector with path and files names (e.g. input.objects = c("path/file1.bin","path/file2.bin") or Risoe.BINfileData objects (e.g. input.objects = c(object1, object2))
output.file	character (optional): File output path and name. If no value is given, a Risoe.BINfileData is returned instead of a file.
keep.position.number	logical (with default): Allows keeping the original position numbers of the input objects. Otherwise the position numbers are recalculated.
position.number.append.gap	integer (with default): Set the position number gap between merged BIN-file sets, if the option keep.position.number = FALSE is used. See details for further information.

Details

The function allows merging different measurements to one file or one object.
The record IDs are recalculated for the new object. Other values are kept for each object. The number of input objects is not limited.

position.number.append.gap option

If the option keep.position.number = FALSE is used, the position numbers of the new data set are recalculated by adding the highest position number of the previous data set to the each position number of the next data set. For example: The highest position number is 48, then this number will be added to all other position numbers of the next data set (e.g. 1 + 48 = 49)

However, there might be cases where an additional addend (summand) is needed before the next position starts. Example:

Position number set (A): 1,3,5,7
Position number set (B): 1,3,5,7

With no additional summand the new position numbers would be: 1,3,5,7,8,9,10,11. That might be unwanted. Using the argument position.number.append.gap = 1 it will become: 1,3,5,7,9,11,13,15,17.

Value

Returns a file or a Risoe.BINfileData object.

Function version

0.2.5 (2016-09-09 10:32:17)

Note

The validity of the output objects is not further checked.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Duller, G., 2007. Analyst.

See Also

Risoe.BINfileData, read_BIN2R, write_R2BIN

Examples

```
##merge two objects
data(ExampleData.BINfileData, envir = environment())

object1 <- CWOSL.SAR.Data
object2 <- CWOSL.SAR.Data

object.new <- merge_Risoe.BINfileData(c(object1, object2))
```

---

merge_RLum *General merge function for RLum S4 class objects*

Description

Function calls object-specific merge functions for RLum S4 class objects.

Usage

```
merge_RLum(objects, ...)
```

Arguments

objects	list of RLum (**required**): list of S4 object of class RLum
...	further arguments that one might want to pass to the specific merge function

Details

The function provides a generalised access point for merge specific RLum objects.
Depending on the input object, the corresponding merge function will be selected. Allowed arguments can be found in the documentations of each merge function. Empty list elements (NULL) are automatically removed from the input list.

object		corresponding merge function
RLum.Data.Curve	:	merge_RLum.Data.Curve
RLum.Analysis	:	merge_RLum.Analysis
RLum.Results	:	merge_RLum.Results

Value

Return is the same as input objects as provided in the list.

Function version

0.1.2 (2016-05-02 09:36:06)

Note

So far not for every RLum object a merging function exists.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

#

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##Example based using data and from the calc_CentralDose() function

##load example data
data(ExampleData.DeValues, envir = environment())

##apply the central dose model 1st time
temp1 <- calc_CentralDose(ExampleData.DeValues$CA1)

##apply the central dose model 2nd time
temp2 <- calc_CentralDose(ExampleData.DeValues$CA1)

##merge the results and store them in a new object
temp.merged <- get_RLum(merge_RLum(objects = list(temp1, temp2)))
```

`merge_RLum.Analysis` *Merge function for RLum.Analysis S4 class objects*

Description

Function allows merging of RLum.Analysis objects and adding of allowed objects to an RLum.Analysis.

Usage

```
merge_RLum.Analysis(objects)
```

Arguments

`objects`	`list` of `RLum.Analysis` (**required**): list of S4 objects of class `RLum.Analysis`. Furthermore other objects of class `RLum` can be added, see details.

Details

This function simply allowing to merge `RLum.Analysis` objects. Additionally other `RLum` objects can be added to an existing `RLum.Analysis` object. Supported objects to be added are: `RLum.Data.Curve`, `RLum.Data.Spectrum` and `RLum.Data.Image`.

The order in the new `RLum.Analysis` object is the object order provided with the input list.

Value

Return an `RLum.Analysis` object.

Function version

0.2.0 (2016-05-02 09:36:06)

Note

The information for the slot 'protocol' is taken from the first `RLum.Analysis` object in the input list. Therefore at least one object of type `RLum.Analysis` has to be provided.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

-

See Also

`merge_RLum`, `RLum.Analysis`, `RLum.Data.Curve`, `RLum.Data.Spectrum`, `RLum.Data.Image`, `RLum`

Examples

```
##merge different RLum objects from the example data
data(ExampleData.RLum.Analysis, envir = environment())
data(ExampleData.BINfileData, envir = environment())

object <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)
curve <- get_RLum(object)[[2]]

temp.merged <- merge_RLum.Analysis(list(curve, IRSAR.RF.Data, IRSAR.RF.Data))
```

---

merge_RLum.Data.Curve *Merge function for RLum.Data.Curve S4 class objects*

Description

Function allows merging of RLum.Data.Curve objects in different ways

Usage

```
merge_RLum.Data.Curve(object, merge.method = "mean", method.info)
```

Arguments

object	list of RLum.Data.Curve (**required**): list of S4 objects of class RLum.Curve.
merge.method	character (**required**): method for combining of the objects, e.g. 'mean', 'sum', see details for further information and allowed methods. Note: Elements in slot info will be taken from the first curve in the list.
method.info	numeric (optional): allows to specify how info elements of the input objects are combined, e.g. 1 means that just the elements from the first object are kept, 2 keeps only the info elements from the 2 object etc. If nothing is provided all elements are combined.

Details

This function simply allowing to merge RLum.Data.Curve objects without touching the objects itself. Merging is always applied on the 2nd colum of the data matrix of the object.

Supported merge operations are RLum.Data.Curve

"sum"

All count values will be summed up using the function rowSums.

"mean"

The mean over the count values is calculated using the function rowMeans.

`"median"`

The median over the count values is calculated using the function `rowMedians`.

`"sd"`

The standard deviation over the count values is calculated using the function `rowSds`.

`"var"`

The variance over the count values is calculated using the function `rowVars`.

`"min"`

The min values from the count values is chosen using the function `rowMins`.

`"max"`

The max values from the count values is chosen using the function `rowMins`.

`"-"`

The row sums of the last objects are subtracted from the first object.

`"*"`

The row sums of the last objects are mutliplied with the first object.

`"/"`

Values of the first object are divided by row sums of the last objects.

Value

Returns an `RLum.Data.Curve` object.

S3-generic support

This function is fully operational via S3-generics: `` `+` ``, `` `-` ``, `` `/` ``, `` `*` ``, `merge`

Function version

0.2.0 (2016-09-09 10:32:17)

Note

The information from the slot 'recordType' is taken from the first `RLum.Data.Curve` object in the input list. The slot 'curveType' is filled with the name `merged`.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

-

See Also

merge_RLum, RLum.Data.Curve

Examples

```
##load example data
data(ExampleData.XSYG, envir = environment())

##grep first and 3d TL curves
TL.curves  <- get_RLum(OSL.SARMeasurement$Sequence.Object, recordType = "TL (UVVIS)")
TL.curve.1 <- TL.curves[[1]]
TL.curve.3 <- TL.curves[[3]]

##plot single curves
plot_RLum(TL.curve.1)
plot_RLum(TL.curve.3)

##subtract the 1st curve from the 2nd and plot
TL.curve.merged <- merge_RLum.Data.Curve(list(TL.curve.3, TL.curve.1), merge.method = "/")
plot_RLum(TL.curve.merged)
```

---

merge_RLum.Results *Merge function for RLum.Results S4-class objects*

Description

Function merges objects of class RLum.Results. The slots in the objects are combined depending on the object type, e.g., for data.frame and matrix rows are appended.

Usage

```
merge_RLum.Results(objects)
```

Arguments

objects	list	(required): a list of RLum.Results objects

Function version

0.2.0 (2016-05-02 09:36:06)

Note

The originator is taken from the first element and not reset to merge_RLum

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

methods_RLum *methods_RLum*

Description

Methods for S3-generics implemented for the package 'Luminescence'. This document summarises all implemented S3-generics. The name of the function is given before the first dot, after the dot the name of the object that is supported by this method is given, e.g. `plot.RLum.Data.Curve` can be called by `plot(object, ...)`, where `object` is the `RLum.Data.Curve` object.

Usage

```
## S3 method for class 'list'
plot(x, y, ...)

## S3 method for class 'RLum.Results'
plot(x, y, ...)

## S3 method for class 'RLum.Analysis'
plot(x, y, ...)

## S3 method for class 'RLum.Data.Curve'
plot(x, y, ...)

## S3 method for class 'RLum.Data.Spectrum'
plot(x, y, ...)

## S3 method for class 'RLum.Data.Image'
plot(x, y, ...)

## S3 method for class 'Risoe.BINfileData'
plot(x, y, ...)

## S3 method for class 'RLum.Results'
hist(x, ...)

## S3 method for class 'RLum.Data.Image'
hist(x, ...)

## S3 method for class 'RLum.Data.Curve'
hist(x, ...)

## S3 method for class 'RLum.Analysis'
hist(x, ...)

## S3 method for class 'RLum.Results'
summary(object, ...)

## S3 method for class 'RLum.Analysis'
summary(object, ...)
```

```
## S3 method for class 'RLum.Data.Image'
summary(object, ...)

## S3 method for class 'RLum.Data.Curve'
summary(object, ...)

## S3 method for class 'Risoe.BINfileData'
subset(x, subset, records.rm = TRUE, ...)

bin.RLum.Data.Curve(x, ...)

## S3 method for class 'RLum.Results'
length(x, ...)

## S3 method for class 'RLum.Analysis'
length(x, ...)

## S3 method for class 'RLum.Data.Curve'
length(x, ...)

## S3 method for class 'Risoe.BINfileData'
length(x, ...)

## S3 method for class 'RLum.Data.Curve'
dim(x)

## S3 method for class 'RLum.Data.Spectrum'
dim(x)

## S3 method for class 'RLum'
rep(x, ...)

## S3 method for class 'RLum.Data.Curve'
names(x, ...)

## S3 method for class 'RLum.Data.Spectrum'
names(x, ...)

## S3 method for class 'RLum.Data.Image'
names(x, ...)

## S3 method for class 'RLum.Analysis'
names(x, ...)

## S3 method for class 'RLum.Results'
names(x, ...)

## S3 method for class 'Risoe.BINfileData'
names(x)

## S3 method for class 'RLum.Data.Spectrum'
row.names(x, ...)
```

```
## S3 method for class 'RLum.Data.Curve'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)

## S3 method for class 'RLum.Data.Spectrum'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)

## S3 method for class 'RLum.Results'
as.list(x, ...)

## S3 method for class 'RLum.Data.Curve'
as.list(x, ...)

## S3 method for class 'RLum.Analysis'
as.list(x, ...)

## S3 method for class 'RLum.Data.Curve'
as.matrix(x, ...)

## S3 method for class 'RLum.Data.Spectrum'
as.matrix(x, ...)

is.RLum(x, ...)

is.RLum.Data(x, ...)

is.RLum.Data.Curve(x, ...)

is.RLum.Data.Spectrum(x, ...)

is.RLum.Data.Image(x, ...)

is.RLum.Analysis(x, ...)

is.RLum.Results(x, ...)

## S3 method for class 'RLum'
merge(x, y, ...)

## S3 method for class 'RLum.Analysis'
unlist(x, recursive = TRUE, ...)

## S3 method for class 'RLum.Data.Curve'
x + y

## S3 method for class 'RLum.Data.Curve'
x - y

## S3 method for class 'RLum.Data.Curve'
x * y
```

```
## S3 method for class 'RLum.Data.Curve'
x / y

## S3 method for class 'RLum.Data.Curve'
x[y, z, drop = TRUE]

## S3 method for class 'RLum.Data.Spectrum'
x[y, z, drop = TRUE]

## S3 method for class 'RLum.Data.Image'
x[y, z, drop = TRUE]

## S3 method for class 'RLum.Analysis'
x[i, drop = FALSE]

## S3 method for class 'RLum.Results'
x[i, drop = TRUE]

## S3 method for class 'RLum.Analysis'
x[[i]]

## S3 method for class 'RLum.Results'
x[[i]]

## S3 method for class 'RLum.Data.Curve'
x$i

## S3 method for class 'RLum.Analysis'
x$i

## S3 method for class 'RLum.Results'
x$i
```

Arguments

x	RLum or Risoe.BINfileData (**required**): input opject
y	integer (optional): the row index of the matrix, data.frame
...	further arguments that can be passed to the method
object	RLum (**required**): input opject
subset	[subset] expression (**required**): logical expression indicating elements or rows to keep, this function works in Risoe.BINfileData objects like subset.data.frame, but takes care of the object structure
records.rm	[subset] logical (with default): remove records from data set, can be disabled, to just set the column SET to TRUE or FALSE
row.names	logical (with default): enables or disables row names (as.data.frame)
optional	logical (with default): logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional (see as.data.frame)
recursive	logical (with default): enables or disables further subsetting (unlist)
z	integer (optional): the column index of the matrix, data.frame

drop	logical (with default): keep object structure or drop it
i	character (optional): name of the wanted record type or data object

Details

The term S3-generics sounds complicated, however, it just means that something has been implemented in the package to increase the usability for users new in R and who are not familiar with the underlying RLum-object structure of the package. The practical outcome is that operations and functions presented in standard books on R can be used without knowing the specifica of the R package 'Luminescence'. For examples see the example section.

Note

methods_RLum are not really new functions, everything given here are mostly just surrogates for existing functions in the package.

Examples

```
##load example data
data(ExampleData.RLum.Analysis, envir = environment())


##combine curve is various ways
curve1 <- IRSAR.RF.Data[[1]]
curve2 <-  IRSAR.RF.Data[[1]]
curve1 + curve2
curve1 - curve2
curve1 / curve2
curve1 * curve2


##`$` access curves
IRSAR.RF.Data$RF
```

---

model_LuminescenceSignals

Model Luminescence Signals (wrapper)

Description

Wrapper for the function model_LuminescenceSignals from the package RLumModel-package. For the further details and examples please see the manual of this package.

Usage

```
model_LuminescenceSignals(model, sequence, lab.dose_rate = 1,
  simulate_sample_history = FALSE, plot = TRUE, verbose = TRUE,
  show.structure = FALSE, ...)
```

Arguments

model	character (**required**): set model to be used. Available models are: "Bailey2001", "Bailey2002", "Bailey2004", "Pagonis2007", "Pagonis2008"
sequence	list (**required**): set sequence to model as list or as *.seq file from the Riso sequence editor. To simulate SAR measurements there is an extra option to set the sequence list (cf. details).
lab.dose_rate	numeric (with default): laboratory dose rate in XXX Gy/s for calculating seconds into Gray in the *.seq file.
simulate_sample_history	logical (with default): FALSE (with default): simulation begins at laboratory conditions, TRUE: simulations begins at crystallization (all levels 0) process
plot	logical (with default): Enables or disables plot output
verbose	logical (with default): Verbose mode on/off
show.structure	logical (with default): Shows the structure of the result. Recommended to show record.id to analyse concentrations.
...	further arguments and graphical parameters passed to plot.default. See details for further information.

Function version

0.1.0 (2016-05-02 09:36:06)

Author(s)

Johannes Friedrich, University of Bayreuth (Germany),
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaige (France),

R Luminescence Package Team

---

names_RLum *S4-names function for RLum S4 class objects*

Description

Function calls object-specific names functions for RLum S4 class objects.

Usage

```
names_RLum(object)
```

Arguments

object	RLum (**required**): S4 object of class RLum

Details

The function provides a generalised access point for specific RLum objects.
Depending on the input object, the corresponding 'names' function will be selected. Allowed arguments can be found in the documentations of the corresponding RLum class.

Value

Returns a character

Function version

0.1.0 (2015-11-29 17:27:48)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

---

plot_AbanicoPlot *Function to create an Abanico Plot.*

Description

A plot is produced which allows comprehensive presentation of data precision and its dispersion around a central value as well as illustration of a kernel density estimate, histogram and/or dot plot of the dose values.

Usage

```
plot_AbanicoPlot(data, na.rm = TRUE, log.z = TRUE, z.0 = "mean.weighted",
  dispersion = "qr", plot.ratio = 0.75, rotate = FALSE, mtext, summary,
  summary.pos, summary.method = "MCM", legend, legend.pos, stats,
  rug = FALSE, kde = TRUE, hist = FALSE, dots = FALSE,
  boxplot = FALSE, y.axis = TRUE, error.bars = FALSE, bar, bar.col,
  polygon.col, line, line.col, line.lty, line.label, grid.col, frame = 1,
  bw = "SJ", output = FALSE, interactive = FALSE, ...)
```

Arguments

data	data.frame or RLum.Results object (required): for data.frame two columns: De (data[,1]) and De error (data[,2]). To plot several data sets in one plot the data sets must be provided as list, e.g. list(data.1, data.2).
na.rm	logical (with default): exclude NA values from the data set prior to any further operations.
log.z	logical (with default): Option to display the z-axis in logarithmic scale. Default is TRUE.
z.0	character or numeric: User-defined central value, used for centering of data. One out of "mean", "mean.weighted" and "median" or a numeric value (not its logarithm). Default is "mean.weighted".

dispersion	character (with default): measure of dispersion, used for drawing the scatter polygon. One out of "qr" (quartile range), "pnn" (symmetric percentile range with nn the lower percentile, e.g. "p05" depicting the range between 5 and 95 "sd" (standard deviation) and "2sd" (2 standard deviations), default is "qr". Note that "sd" and "2sd" are only meaningful in combination with "z.0 = 'mean'" because the unweighted mean is used to center the polygon.
plot.ratio	numeric: Relative space, given to the radial versus the cartesian plot part, deault is 0.75.
rotate	logical: Option to turn the plot by 90 degrees.
mtext	character: additional text below the plot title.
summary	character (optional): add statistic measures of centrality and dispersion to the plot. Can be one or more of several keywords. See details for available keywords. Results differ depending on the log-option for the z-scale (see details).
summary.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the statistical summary. Alternatively, the keyword "sub" may be specified to place the summary below the plot header. However, this latter option in only possible if mtext is not used.
summary.method	character (with default): keyword indicating the method used to calculate the statistic summary. One out of "unweighted", "weighted" and "MCM". See calc_Statistics for details.
legend	character vector (optional): legend content to be added to the plot.
legend.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the legend to be plotted.
stats	character: additional labels of statistically important values in the plot. One or more out of the following: "min", "max", "median".
rug	logical: Option to add a rug to the KDE part, to indicate the location of individual values.
kde	logical: Option to add a KDE plot to the dispersion part, default is TRUE.
hist	logical: Option to add a histogram to the dispersion part. Only meaningful when not more than one data set is plotted.
dots	logical: Option to add a dot plot to the dispersion part. If number of dots exceeds space in the dispersion part, a square indicates this.
boxplot	logical: Option to add a boxplot to the dispersion part, default is FALSE.
y.axis	logical: Option to hide y-axis labels. Useful for data with small scatter.
error.bars	logical: Option to show De-errors as error bars on De-points. Useful in combination with y.axis = FALSE, bar.col = "none".
bar	numeric (with default): option to add one or more dispersion bars (i.e., bar showing the 2-sigma range) centered at the defined values. By default a bar is drawn according to "z.0". To omit the bar set "bar = FALSE".
bar.col	character or numeric (with default): colour of the dispersion bar. Default is "grey60".
polygon.col	character or numeric (with default): colour of the polygon showing the data scatter. Sometimes this polygon may be omitted for clarity. To disable it use FALSE or polygon = FALSE. Default is "grey80".
line	numeric: numeric values of the additional lines to be added.
line.col	character or numeric: colour of the additional lines.

line.lty	integer: line type of additional lines
line.label	character: labels for the additional lines.
grid.col	character or numeric (with default): colour of the grid lines (originating at [0,0] and strechting to the z-scale). To disable grid lines use FALSE. Default is "grey".
frame	numeric (with default): option to modify the plot frame type. Can be one out of 0 (no frame), 1 (frame originates at 0,0 and runs along min/max isochrons), 2 (frame embraces the 2-sigma bar), 3 (frame embraces the entire plot as a rectangle).Default is 1.
bw	character (with default): bin-width for KDE, choose a numeric value for manual setting.
output	logical: Optional output of numerical plot parameters. These can be useful to reproduce similar plots. Default is FALSE.
interactive	logical (with default): create an interactive abanico plot (requires the 'plotly' package)
...	Further plot arguments to pass. xlab must be a vector of length 2, specifying the upper and lower x-axes labels.

Details

The Abanico Plot is a combination of the classic Radial Plot (plot_RadialPlot) and a kernel density estimate plot (e.g plot_KDE). It allows straightforward visualisation of data precision, error scatter around a user-defined central value and the combined distribution of the values, on the actual scale of the measured data (e.g. seconds, equivalent dose, years). The principle of the plot is shown in Galbraith & Green (1990). The function authors are thankful for the thoughtprovocing figure in this article.
The semi circle (z-axis) of the classic Radial Plot is bent to a straight line here, which actually is the basis for combining this polar (radial) part of the plot with any other cartesian visualisation method (KDE, histogram, PDF and so on). Note that the plot allows dispaying two measures of distribution. One is the 2-sigma bar, which illustrates the spread in value errors, and the other is the polygon, which stretches over both parts of the Abanico Plot (polar and cartesian) and illustrates the actual spread in the values themselfes.
Since the 2-sigma-bar is a polygon, it can be (and is) filled with shaded lines. To change density (lines per inch, default is 15) and angle (default is 45 degrees) of the shading lines, specify these parameters. See ?polygon() for further help.
The Abanico Plot supports other than the weighted mean as measure of centrality. When it is obvious that the data is not (log-)normally distributed, the mean (weighted or not) cannot be a valid measure of centrality and hence central dose. Accordingly, the median and the weighted median can be chosen as well to represent a proper measure of centrality (e.g. centrality = "median.weighted"). Also user-defined numeric values (e.g. from the central age model) can be used if this appears appropriate.
The proportion of the polar part and the cartesian part of the Abanico Plot can be modfied for display reasons (plot.ratio = 0.75). By default, the polar part spreads over 75 % and leaves 25 % for the part that shows the KDE graph.

A statistic summary, i.e. a collection of statistic measures of centrality and dispersion (and further measures) can be added by specifying one or more of the following keywords:

- "n" (number of samples)
- "mean" (mean De value)

- "median" (median of the De values)
- "sd.rel" (relative standard deviation in percent)
- "sd.abs" (absolute standard deviation)
- "se.rel" (relative standard error)
- "se.abs" (absolute standard error)
- "in.2s" (percent of samples in 2-sigma range)
- "kurtosis" (kurtosis)
- "skewness" (skewness)

Note that the input data for the statistic summary is sent to the function calc_Statistics() depending on the log-option for the z-scale. If "log.z = TRUE", the summary is based on the logarithms of the input data. If "log.z = FALSE" the linearly scaled data is used.
Note as well, that "calc_Statistics()" calculates these statistic measures in three different ways: unweighted, weighted and MCM-based (i.e., based on Monte Carlo Methods). By default, the MCM-based version is used. If you wish to use another method, indicate this with the appropriate keyword using the argument summary.method.

The optional parameter layout allows to modify the entire plot more sophisticated. Each element of the plot can be addressed and its properties can be defined. This includes font type, size and decoration, colours and sizes of all plot items. To infer the definition of a specific layout style cf. get_Layout() or type eg. for the layout type "journal" get_Layout("journal"). A layout type can be modified by the user by assigning new values to the list object.

It is possible for the z-scale to specify where ticks are to be drawn by using the parameter at, e.g. at = seq(80, 200, 20), cf. function documentation of axis. Specifying tick positions manually overrides a zlim-definition.

Value

returns a plot object and, optionally, a list with plot calculus data.

Function version

0.1.10 (2016-09-09 10:32:17)

Author(s)

Michael Dietze, GFZ Potsdam (Germany),
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
Inspired by a plot introduced by Galbraith & Green (1990)
R Luminescence Package Team

References

Galbraith, R. & Green, P., 1990. Estimating the component ages in a finite mixture. International Journal of Radiation Applications and Instrumentation. Part D. Nuclear Tracks and Radiation Measurements, 17 (3), 197-206.

Dietze, M., Kreutzer, S., Burow, C., Fuchs, M.C., Fischer, M., Schmidt, C., 2015. The abanico plot: visualising chronometric data with individual standard errors. Quaternary Geochronology. doi:10.1016/j.quageo.2015.09.003

See Also

plot_RadialPlot, plot_KDE, plot_Histogram

Examples

```
## load example data and recalculate to Gray
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <- ExampleData.DeValues$CA1

## plot the example data straightforward
plot_AbanicoPlot(data = ExampleData.DeValues)

## now with linear z-scale
plot_AbanicoPlot(data = ExampleData.DeValues,
                 log.z = FALSE)

## now with output of the plot parameters
plot1 <- plot_AbanicoPlot(data = ExampleData.DeValues,
                          output = TRUE)
str(plot1)
plot1$zlim

## now with adjusted z-scale limits
plot_AbanicoPlot(data = ExampleData.DeValues,
                 zlim = c(10, 200))

## now with adjusted x-scale limits
plot_AbanicoPlot(data = ExampleData.DeValues,
                 xlim = c(0, 20))

## now with rug to indicate individual values in KDE part
plot_AbanicoPlot(data = ExampleData.DeValues,
                 rug = TRUE)

## now with a smaller bandwidth for the KDE plot
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bw = 0.04)

## now with a histogram instead of the KDE plot
plot_AbanicoPlot(data = ExampleData.DeValues,
                 hist = TRUE,
                 kde = FALSE)

## now with a KDE plot and histogram with manual number of bins
plot_AbanicoPlot(data = ExampleData.DeValues,
                 hist = TRUE,
                 breaks = 20)

## now with a KDE plot and a dot plot
plot_AbanicoPlot(data = ExampleData.DeValues,
                 dots = TRUE)

## now with user-defined plot ratio
plot_AbanicoPlot(data = ExampleData.DeValues,
                 plot.ratio = 0.5)
```

```
## now with user-defined central value
plot_AbanicoPlot(data = ExampleData.DeValues,
                 z.0 = 70)

## now with median as central value
plot_AbanicoPlot(data = ExampleData.DeValues,
                 z.0 = "median")

## now with the 17-83 percentile range as definition of scatter
plot_AbanicoPlot(data = ExampleData.DeValues,
                 z.0 = "median",
                 dispersion = "p17")

## now with user-defined green line for minimum age model
CAM <- calc_CentralDose(ExampleData.DeValues,
                        plot = FALSE)

plot_AbanicoPlot(data = ExampleData.DeValues,
                 line = CAM,
                 line.col = "darkgreen",
                 line.label = "CAM")

## now create plot with legend, colour, different points and smaller scale
plot_AbanicoPlot(data = ExampleData.DeValues,
                 legend = "Sample 1",
                 col = "tomato4",
                 bar.col = "peachpuff",
                 pch = "R",
                 cex = 0.8)

## now without 2-sigma bar, polygon, grid lines and central value line
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bar.col = FALSE,
                 polygon.col = FALSE,
                 grid.col = FALSE,
                 y.axis = FALSE,
                 lwd = 0)

## now with direct display of De errors, without 2-sigma bar
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bar.col = FALSE,
                 ylab = "",
                 y.axis = FALSE,
                 error.bars = TRUE)

## now with user-defined axes labels
plot_AbanicoPlot(data = ExampleData.DeValues,
                 xlab = c("Data error (%)",
                          "Data precision"),
                 ylab = "Scatter",
                 zlab = "Equivalent dose [Gy]")

## now with minimum, maximum and median value indicated
plot_AbanicoPlot(data = ExampleData.DeValues,
                 stats = c("min", "max", "median"))

## now with a brief statistical summary as subheader
```

```
plot_AbanicoPlot(data = ExampleData.DeValues,
                 summary = c("n", "in.2s"))

## now with another statistical summary
plot_AbanicoPlot(data = ExampleData.DeValues,
                 summary = c("mean.weighted", "median"),
                 summary.pos = "topleft")

## now a plot with two 2-sigma bars for one data set
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bar = c(30, 100))

## now the data set is split into sub-groups, one is manipulated
data.1 <- ExampleData.DeValues[1:30,]
data.2 <- ExampleData.DeValues[31:62,] * 1.3

## now a common dataset is created from the two subgroups
data.3 <- list(data.1, data.2)

## now the two data sets are plotted in one plot
plot_AbanicoPlot(data = data.3)

## now with some graphical modification
plot_AbanicoPlot(data = data.3,
                 z.0 = "median",
                 col = c("steelblue4", "orange4"),
                 bar.col = c("steelblue3", "orange3"),
                 polygon.col = c("steelblue1", "orange1"),
                 pch = c(2, 6),
                 angle = c(30, 50),
                 summary = c("n", "in.2s", "median"))

## create Abanico plot with predefined layout definition
plot_AbanicoPlot(data = ExampleData.DeValues,
                 layout = "journal")

## now with predefined layout definition and further modifications
plot_AbanicoPlot(data = data.3,
                 z.0 = "median",
                 layout = "journal",
                 col = c("steelblue4", "orange4"),
                 bar.col = adjustcolor(c("steelblue3", "orange3"),
                                       alpha.f = 0.5),
                 polygon.col = c("steelblue3", "orange3"))

## for further information on layout definitions see documentation
## of function get_Layout()

## now with manually added plot content
## create empty plot with numeric output
AP <- plot_AbanicoPlot(data = ExampleData.DeValues,
                       pch = NA,
                       output = TRUE)

## identify data in 2 sigma range
in_2sigma <- AP$data[[1]]$data.in.2s
```

```
## restore function-internal plot parameters
par(AP$par)

## add points inside 2-sigma range
points(x = AP$data[[1]]$precision[in_2sigma],
       y = AP$data[[1]]$std.estimate.plot[in_2sigma],
       pch = 16)

## add points outside 2-sigma range
points(x = AP$data[[1]]$precision[!in_2sigma],
       y = AP$data[[1]]$std.estimate.plot[!in_2sigma],
       pch = 1)
```

---

plot_DetPlot *Create De(t) plot*

Description

Plots the equivalent dose (De) in dependency of the chosen signal integral (cf. Bailey et al., 2003). The function is simply passing several arguments to the function plot and the used analysis functions and runs it in a loop. Example: legend.pos for legend position, legend for legend text.

Usage

```
plot_DetPlot(object, signal.integral.min, signal.integral.max,
  background.integral.min, background.integral.max, method = "shift",
  signal_integral.seq = NULL, analyse_function = "analyse_SAR.CWOSL",
  analyse_function.control = list(), n.channels = NULL,
  show_ShineDownCurve = TRUE, respect_RC.Status = FALSE, verbose = TRUE,
  ...)
```

Arguments

object	RLum.Analysis (**required**): input object containing data for analysis
signal.integral.min	integer (**required**): lower bound of the signal integral.
signal.integral.max	integer (**required**): upper bound of the signal integral.
background.integral.min	integer (**required**): lower bound of the background integral.
background.integral.max	integer (**required**): upper bound of the background integral.
method	character (with default): method applied for constructing the De(t) plot. shift (the default): the chosen signal integral is shifted the shine down curve, expansion: the chosen signal integral is expanded each time by its length
signal_integral.seq	numeric (optional): argument to provide an own signal integral sequence for constructing the De(t) plot

analyse_function
: character (with default): name of the analyse function to be called. Supported functions are: 'analyse_SAR.CWOSL', 'analyse_pIRIRSequence'

analyse_function.control
: list (optional): arguments to be passed to the supported analyse functions ('analyse_SAR.CWOSL', 'analyse_pIRIRSequence')

n.channels
: integer (optional): number of channels used for the De(t) plot. If nothing is provided all De-values are calculated and plotted until the start of the background integral.

show_ShineDownCurve
: logical (with default): enables or disables shine down curve in the plot output

respect_RC.Status
: logical (with default): remove De-values with 'FAILED' RC.Status from the plot (cf. analyse_SAR.CWOSL and analyse_pIRIRSequence)

verbose
: logical (with default): enables or disables terminal feedback

...
: further arguments and graphical parameters passed to plot.default, analyse_SAR.CWOSL and analyse_pIRIRSequence. See details for further information.

Details

method

The original method presented by Baiely et al., 2003 shifted the signal integrals and slightly extended them accounting for changes in the counting statistics. Example: c(1:3, 3:5, 5:7). However, here also another method is provided allowing to expand the signal integral by consectutively expaning the integral by its chosen length. Example: c(1:3, 1:5, 1:7)

Note that in both cases the integral limits are overlap. The finally applied limits are part of the function output.

Value

A plot and an RLum.Results object with the produced De values

@data:

Object	Type	Description
De.values	data.frame	table with De values
signal_integral.seq	numeric	integral sequence used for the calculation

@info:

Object	Type	Description
call	call	the original function call

Function version

0.1.0 (2016-05-19 23:48:19)

Note

The entire analysis is based on the used analysis functions, namely analyse_SAR.CWOSL and analyse_pIRIRSequence. However, the integrity checks of this function are not that thoughtful as in these functions itself. It means, that every sequence should be checked carefully before running long calculations using serveral hundreds of channels.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Bailey, R.M., Singarayer, J.S., Ward, S., Stokes, S., 2003. Identification of partial resetting using De as a function of illumination time. Radiation Measurements 37, 511-518. doi:10.1016/S1350-4487(03)00063-5

See Also

plot, analyse_SAR.CWOSL, analyse_pIRIRSequence

Examples

```
## Not run:
##load data
##ExampleData.BINfileData contains two BINfileData objects
##CWOSL.SAR.Data and TL.SAR.Data
data(ExampleData.BINfileData, envir = environment())

##transform the values from the first position in a RLum.Analysis object
object <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)

plot_DetPlot(object,
             signal.integral.min = 1,
             signal.integral.max = 3,
             background.integral.min = 900,
             background.integral.max = 1000,
             n.channels = 5,
)

## End(Not run)
```

---

plot_DRTResults *Visualise dose recovery test results*

Description

The function provides a standardised plot output for dose recovery test measurements.

Usage

```
plot_DRTResults(values, given.dose = NULL, error.range = 10, preheat,
  boxplot = FALSE, mtext, summary, summary.pos, legend, legend.pos,
  par.local = TRUE, na.rm = FALSE, ...)
```

Arguments

values	RLum.Results or data.frame, (**required**): input values containing at least De and De error. To plot more than one data set in one figure, a list of the individual data sets must be provided (e.g. list(dataset.1, dataset.2)).
given.dose	numeric (optional): given dose used for the dose recovery test to normalise data. If only one given dose is provided this given dose is valid for all input data sets (i.e., values is a list). Otherwise a given dose for each input data set has to be provided (e.g., given.dose = c(100,200)). If no given.dose values are plotted without normalisation (might be useful for preheat plateau tests). Note: Unit has to be the same as from the input values (e.g., Seconds or Gray).
error.range	numeric: symmetric error range in percent will be shown as dashed lines in the plot. Set error.range to 0 to void plotting of error ranges.
preheat	numeric: optional vector of preheat temperatures to be used for grouping the De values. If specified, the temperatures are assigned to the x-axis.
boxplot	logical: optionally plot values, that are grouped by preheat temperature as boxplots. Only possible when preheat vector is specified.
mtext	character: additional text below the plot title.
summary	character (optional): adds numerical output to the plot. Can be one or more out of: "n" (number of samples), "mean" (mean De value), "mean.weighted" (error-weighted mean), "median" (median of the De values), "sdrel" (relative standard deviation in percent), "sdabs" (absolute standard deviation), "serel" (relative standard error) and "seabs" (absolute standard error).
summary.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the statistical summary. Alternatively, the keyword "sub" may be specified to place the summary below the plot header. However, this latter option in only possible if mtext is not used.
legend	character vector (optional): legend content to be added to the plot.
legend.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the legend to be plotted.
par.local	logical (with default): use local graphical parameters for plotting, e.g. the plot is shown in one column and one row. If par.local = FALSE, global parameters are inherited, i.e. parameters provided via par() work
na.rm	logical: indicating wether NA values are removed before plotting from the input data set
...	further arguments and graphical parameters passed to plot.

Details

Procedure to test the accuracy of a measurement protocol to reliably determine the dose of a specific sample. Here, the natural signal is erased and a known laboratory dose administered which is treated as unknown. Then the De measurement is carried out and the degree of congruence between administered and recovered dose is a measure of the protocol's accuracy for this sample.
In the plot the normalised De is shown on the y-axis, i.e. obtained De/Given Dose.

Value

A plot is returned.

Function version

0.1.10 (2016-09-09 10:32:17)

Note

Further data and plot arguments can be added by using the appropiate R commands.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

References

Wintle, A.G., Murray, A.S., 2006. A review of quartz optically stimulated luminescence characteristics and their relevance in single-aliquot regeneration dating protocols. Radiation Measurements, 41, 369-391.

See Also

plot

Examples

```
## read example data set and misapply them for this plot type
data(ExampleData.DeValues, envir = environment())

## plot values
plot_DRTResults(values = ExampleData.DeValues$BT998[7:11,],
given.dose = 2800, mtext = "Example data")

## plot values with legend
plot_DRTResults(values = ExampleData.DeValues$BT998[7:11,],
                given.dose = 2800,
                legend = "Test data set")

## create and plot two subsets with randomised values
x.1 <- ExampleData.DeValues$BT998[7:11,]
x.2 <- ExampleData.DeValues$BT998[7:11,] * c(runif(5, 0.9, 1.1), 1)

plot_DRTResults(values = list(x.1, x.2),
                given.dose = 2800)

## some more user-defined plot parameters
plot_DRTResults(values = list(x.1, x.2),
                given.dose = 2800,
                pch = c(2, 5),
                col = c("orange", "blue"),
```

```
                xlim = c(0, 8),
                ylim = c(0.85, 1.15),
                xlab = "Sample aliquot")

## plot the data with user-defined statistical measures as legend
plot_DRTResults(values = list(x.1, x.2),
                given.dose = 2800,
                summary = c("n", "mean.weighted", "sd"))

## plot the data with user-defined statistical measures as sub-header
plot_DRTResults(values = list(x.1, x.2),
                given.dose = 2800,
                summary = c("n", "mean.weighted", "sd"),
                summary.pos = "sub")

## plot the data grouped by preheat temperatures
plot_DRTResults(values = ExampleData.DeValues$BT998[7:11,],
                given.dose = 2800,
                preheat = c(200, 200, 200, 240, 240))
## read example data set and misapply them for this plot type
data(ExampleData.DeValues, envir = environment())

## plot values
plot_DRTResults(values = ExampleData.DeValues$BT998[7:11,],
                given.dose = 2800, mtext = "Example data")
## plot two data sets grouped by preheat temperatures
plot_DRTResults(values = list(x.1, x.2),
                given.dose = 2800,
                preheat = c(200, 200, 200, 240, 240))

## plot the data grouped by preheat temperatures as boxplots
plot_DRTResults(values = ExampleData.DeValues$BT998[7:11,],
                given.dose = 2800,
                preheat = c(200, 200, 200, 240, 240),
                boxplot = TRUE)
```

`plot_FilterCombinations` *Plot filter combinations along with the (optional) net transmission window*

Description

The function allows to plot transmission windows for different filters. Missing data for specific wavelenghts are automatically interpolated for the given filter data using the function `approx`. With that a standardised output is reached and a net transmission window can be shown.

Usage

```
plot_FilterCombinations(filters, wavelength_range = 200:1000,
  show_net_transmission = TRUE, plot = TRUE, ...)
```

Arguments

`filters`	`list` (**required**): a named list of filter data for each filter to be shown. The filter data itself should be either provided as `data.frame` or `matrix`. (for more options s. Details)
`wavelength_range`	`numeric` (with default): wavelength range used for the interpolation
`show_net_transmission`	`logical` (with default): show net transmission window as polygon.
`plot`	`logical` (with default): enables or disables the plot output
`...`	further arguments that can be passed to control the plot output. Suppored are `main`, `xlab`, `ylab`, `xlim`, `ylim`, `type`, `lty`, `lwd`. For non common plotting parameters see the details section.

Details

How to provide input data?

CASE 1

The function expects that all filter values are either of type `matrix` or `data.frame` with two columns. The first columens contains the wavelength, the second the relative transmission (but not in percentage, i.e. the maximum transmission can be only become 1).

In this case only the transmission window is show as provided. Changes in filter thickness and relection factor are not considered.

CASE 2

The filter data itself are provided as list element containing a `matrix` or `data.frame` and additional information on the thickness of the filter, e.g., `list(filter1 = list(filter_matrix, d = 2))`. The given filter data are always considered as standard input and the filter thickness value is taken into account by

$$Transmission = Transmission^(d)$$

with d given in the same dimension as the original filter data.

CASE 3

Same as CASE 2 but additionally a reflection factor P is provided, e.g., `list(filter1 = list(filter_matrix, d = 2,` The final transmission becomes:

$$Transmission = Transmission^(d) * P$$

Advanced plotting parameters

The following further non-common plotting parameters can be passed to the function:

Argument	Datatype	Description
legend	logical	enable/disable legend
legend.pos	character	change legend position (legend)
legend.text	character	same as the argument legend in (legend)
net_transmission.col	col	colour of net transmission window polygon
grid	list	full list of arguments that can be passd to the function grid

For further modifications standard additional R plot functions are recommend, e.g., the legend can be fully customised by disabling the standard legend and use the function legend instead.

Value

Returns an S4 object of type RLum.Results.

@data

Object	Type	Description
net_transmission_window	matrix	the resulting net transmission window
filter_matrix	matrix	the filter matrix used for plotting

@info

Object	Type	Description
call	call	the original function call

Function version

0.1.0 (2016-08-26 10:45:14)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montagine (France)

R Luminescence Package Team

See Also

RLum.Results, approx

Examples

```
## (For legal reasons no real filter data are provided)

## Create filter sets
filter1 <- density(rnorm(100, mean = 450, sd = 20))
filter1 <- matrix(c(filter1$x, filter1$y/max(filter1$y)), ncol = 2)
filter2 <- matrix(c(200:799,rep(c(0,0.8,0),each = 200)), ncol = 2)

## Example 1 (standard)
plot_FilterCombinations(filters = list(filter1, filter2))

## Example 2 (with d and P value and name for filter 2)
results <- plot_FilterCombinations(
```

```
filters = list(filter_1 = filter1, Rectangle = list(filter2, d = 2, P = 0.6)))
results
```

`plot_GrowthCurve` *Fit and plot a growth curve for luminescence data (Lx/Tx against dose)*

Description

A dose response curve is produced for luminescence measurements using a regenerative protocol.

Usage

```
plot_GrowthCurve(sample, na.rm = TRUE, fit.method = "EXP",
  fit.force_through_origin = FALSE, fit.weights = TRUE,
  fit.includingRepeatedRegPoints = TRUE, fit.NumberRegPoints = NULL,
  fit.NumberRegPointsReal = NULL, fit.bounds = TRUE,
  NumberIterations.MC = 100, output.plot = TRUE,
  output.plotExtended = TRUE, output.plotExtended.single = FALSE,
  cex.global = 1, txtProgressBar = TRUE, verbose = TRUE, ...)
```

Arguments

`sample`	`data.frame` (**required**): data frame with three columns for x=Dose,y=LxTx,z=LxTx.Error, y1=TnTx. The column for the test dose response is optional, but requires 'TnTx' as column name if used. For exponential fits at least three dose points (including the natural) should be provided.
`na.rm`	`logical` (with default): excludes `NA` values from the data set prior to any further operations.
`fit.method`	`character` (with default): function used for fitting. Possible options are: `LIN`, `QDR`, `EXP`, `EXP OR LIN`, `EXP+LIN` or `EXP+EXP`. See details.
`fit.force_through_origin`	`logical` (with default) allow to force the fitted function through the origin. For `method = "EXP+EXP"` the function will go to the origin in either case, so this option will have no effect.
`fit.weights`	`logical` (with default): option whether the fitting is done with or without weights. See details.
`fit.includingRepeatedRegPoints`	`logical` (with default): includes repeated points for fitting (`TRUE`/`FALSE`).
`fit.NumberRegPoints`	`integer` (optional): set number of regeneration points manually. By default the number of all (!) regeneration points is used automatically.
`fit.NumberRegPointsReal`	`integer` (optional): if the number of regeneration points is provided manually, the value of the real, regeneration points = all points (repeated points) including reg 0, has to be inserted.
`fit.bounds`	`logical` (with default): set lower fit bounds for all fitting parameters to 0. Limited for the use with the fit methods `EXP`, `EXP+LIN` and `EXP OR LIN`. Argument to be inserted for experimental application only!

NumberIterations.MC
: integer (with default): number of Monte Carlo simulations for error estimation. See details.

output.plot
: logical (with default): plot output (TRUE/FALSE).

output.plotExtended
: logical (with default): If TRUE, 3 plots on one plot area are provided: (1) growth curve, (2) histogram from Monte Carlo error simulation and (3) a test dose response plot. If FALSE, just the growth curve will be plotted. **Requires:** output.plot = TRUE.

output.plotExtended.single
: logical (with default): single plot output (TRUE/FALSE) to allow for plotting the results in single plot windows. Requires output.plot = TRUE and output.plotExtended = TRUE.

cex.global
: numeric (with default): global scaling factor.

txtProgressBar
: logical (with default): enables or disables txtProgressBar. If verbose = FALSE also no txtProgressBar is shown.

verbose
: logical (with default): enables or disables terminal feedback.

...
: Further arguments and graphical parameters to be passed. Note: Standard arguments will only be passed to the growth curve plot. Supported: xlim, ylim, main, xlab, ylab

Details

Fitting methods

For all options (except for the LIN, QDR and the EXP OR LIN), the nlsLM function with the LM (Levenberg-Marquardt algorithm) algorithm is used. Note: For historical reasons for the Monte Carlo simulations partly the function nls using the port algorithm.

The solution is found by transforming the function or using uniroot.

LIN: fits a linear function to the data using lm:

$$y = m * x + n$$

QDR: fits a linear function to the data using lm:

$$y = a + b * x + c * x^2$$

EXP: try to fit a function of the form

$$y = a * (1 - exp(-(x + c)/b))$$

Parameters b and c are approximated by a linear fit using lm. Note: b = D0

EXP OR LIN: works for some cases where an EXP fit fails. If the EXP fit fails, a LIN fit is done instead.

EXP+LIN: tries to fit an exponential plus linear function of the form:

$$y = a * (1 - exp(-(x + c)/b) + (g * x))$$

The De is calculated by iteration.
Note: In the context of luminescence dating, this function has no physical meaning. Therefore, no

D0 value is returned.

EXP+EXP: tries to fit a double exponential function of the form

$$y = (a1 * (1 - exp(-(x)/b1))) + (a2 * (1 - exp(-(x)/b2)))$$

This fitting procedure is not robust against wrong start parameters and should be further improved.

Fit weighting

If the option `fit.weights = TRUE` is chosen, weights are calculated using provided signal errors (Lx/Tx error):

$$fit.weights = 1/error/(sum(1/error))$$

Error estimation using Monte Carlo simulation

Error estimation is done using a Monte Carlo (MC) simulation approach. A set of Lx/Tx values is constructed by randomly drawing curve data from samled from normal distributions. The normal distribution is defined by the input values (mean = value, sd = value.error). Then, a growth curve fit is attempted for each dataset resulting in a new distribution of single De values. The sd of this distribution is becomes then the error of the De. With increasing iterations, the error value becomes more stable. **Note:** It may take some calculation time with increasing MC runs, especially for the composed functions (EXP+LIN and EXP+EXP).
Each error estimation is done with the function of the chosen fitting method.

Subtitle information

To avoid plotting the subtitle information, provide an empty user mtext `mtext = ""`. To plot any other subtitle text, use `mtext`.

Value

Along with a plot (so far wanted) an `RLum.Results` object is returned containing, the slot `data` contains the following elements:

DATA.OBJECT	TYPE	DESCRIPTION
..$De :	data.frame	Table with De values
..$De.MC :	numeric	Table with De values from MC runs
..$Fit :	nls or lm	object from the fitting for EXP, EXP+LIN and EXP+EXP. In case of a resulting linear fit w
..$Formula :	expression	Fitting formula as R expression
..$call :	call	The original function call

Function version

1.8.16 (2016-09-09 10:32:17)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France),
Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

See Also

nls, RLum.Results, get_RLum, nlsLM, lm, uniroot

Examples

```
##(1) plot growth curve for a dummy data.set and show De value
data(ExampleData.LxTxData, envir = environment())
temp <- plot_GrowthCurve(LxTxData)
get_RLum(temp)

##(1a) to access the fitting value try
get_RLum(temp, data.object = "Fit")

##(2) plot the growth curve only - uncomment to use
##pdf(file = "~/Desktop/Growth_Curve_Dummy.pdf", paper = "special")
plot_GrowthCurve(LxTxData)
##dev.off()

##(3) plot growth curve with pdf output - uncomment to use, single output
##pdf(file = "~/Desktop/Growth_Curve_Dummy.pdf", paper = "special")
plot_GrowthCurve(LxTxData, output.plotExtended.single = TRUE)
##dev.off()

##(4) plot resulting function for given intervall x
x <- seq(1,10000, by = 100)
plot(
 x = x,
 y = eval(temp$Formula),
 type = "l"
)
```

---

plot_Histogram *Plot a histogram with separate error plot*

Description

Function plots a predefined histogram with an accompanying error plot as suggested by Rex Galbraith at the UK LED in Oxford 2010.

Usage

```
plot_Histogram(data, na.rm = TRUE, mtext, cex.global, se, rug, normal_curve,
  summary, summary.pos, colour, interactive = FALSE, ...)
```

Arguments

data	data.frame or RLum.Results object (required): for data.frame: two columns: De (data[,1]) and De error (data[,2])
na.rm	logical (with default): excludes NA values from the data set prior to any further operations.
mtext	character (optional): further sample information (mtext).
cex.global	numeric (with default): global scaling factor.
se	logical (optional): plots standard error points over the histogram, default is FALSE.
rug	logical (optional): adds rugs to the histogram, default is TRUE.
normal_curve	logical (with default): adds a normal curve to the histogram. Mean and sd are calculated from the input data. More see details section.
summary	character (optional): add statistic measures of centrality and dispersion to the plot. Can be one or more of several keywords. See details for available keywords.
summary.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the statistical summary. Alternatively, the keyword "sub" may be specified to place the summary below the plot header. However, this latter option in only possible if mtext is not used. In case of coordinate specification, y-coordinate refers to the right y-axis.
colour	numeric or character (with default): optional vector of length 4 which specifies the colours of the following plot items in exactly this order: histogram bars, rug lines, normal distribution curve and standard error points (e.g., c("grey","black", "red", "grey")).
interactive	logical (with default): create an interactive histogram plot (requires the 'plotly' package)
...	further arguments and graphical parameters passed to plot or hist. If y-axis labels are provided, these must be specified as a vector of length 2 since the plot features two axes (e.g. ylab = c("axis label 1", "axis label 2")). Y-axes limits (ylim) must be provided as vector of length four, with the first two elements specifying the left axes limits and the latter two elements giving the right axis limits.

Details

If the normal curve is added, the y-axis in the histogram will show the probability density.

A statistic summary, i.e. a collection of statistic measures of centrality and dispersion (and further measures) can be added by specifying one or more of the following keywords: "n" (number of samples), "mean" (mean De value), "mean.weighted" (error-weighted mean), "median" (median of the De values), "sdrel" (relative standard deviation in percent), "sdrel.weighted" (error-weighted relative standard deviation in percent), "sdabs" (absolute standard deviation), "sdabs.weighted" (error-weighted absolute standard deviation), "serel" (relative standard error), "serel.weighted" (error-weighted relative standard error), "seabs" (absolute standard error), "seabs.weighted" (error-weighted absolute standard error), "kurtosis" (kurtosis) and "skewness" (skewness).

Function version

0.4.4 (2016-07-16 11:28:11)

Note

The input data is not restricted to a special type.

Author(s)

Michael Dietze, GFZ Potsdam (Germany),
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

hist, plot

Examples

```
## load data
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <-
  Second2Gray(ExampleData.DeValues$BT998, dose.rate = c(0.0438,0.0019))

## plot histogram the easiest way
plot_Histogram(ExampleData.DeValues)

## plot histogram with some more modifications
plot_Histogram(ExampleData.DeValues,
               rug = TRUE,
               normal_curve = TRUE,
               cex.global = 0.9,
               pch = 2,
               colour = c("grey", "black", "blue", "green"),
               summary = c("n", "mean", "sdrel"),
               summary.pos = "topleft",
               main = "Histogram of De-values",
               mtext = "Example data set",
               ylab = c(expression(paste(D[e], " distribution")),
                        "Standard error"),
               xlim = c(100, 250),
               ylim = c(0, 0.1, 5, 20))
```

---

plot_KDE *Plot kernel density estimate with statistics*

Description

Plot a kernel density estimate of measurement values in combination with the actual values and associated error bars in ascending order. If enabled, the boxplot will show the usual distribution parameters (median as bold line, box delimited by the first and third quartile, whiskers defined by the extremes and outliers shown as points) and also the mean and standard deviation as pale bold line and pale polygon, respectively.

Usage

```
plot_KDE(data, na.rm = TRUE, values.cumulative = TRUE, order = TRUE,
  boxplot = TRUE, rug = TRUE, summary, summary.pos,
  summary.method = "MCM", bw = "nrd0", output = FALSE, ...)
```

Arguments

data	data.frame or RLum.Results object (required): for data.frame: two columns: De (values[,1]) and De error (values[,2]). For plotting multiple data sets, these must be provided as list (e.g. list(dataset1,dataset2)).
na.rm	logical (with default): exclude NA values from the data set prior to any further operations.
values.cumulative	logical (with default): show cumulative individual data.
order	logical: Order data in ascending order.
boxplot	logical (with default): optionally show a boxplot (depicting median as thick central line, first and third quartile as box limits, whiskers denoting +/- 1.5 interquartile ranges and dots further outliers).
rug	logical (with default): optionally add rug.
summary	character (optional): add statistic measures of centrality and dispersion to the plot. Can be one or more of several keywords. See details for available keywords.
summary.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the statistical summary. Alternatively, the keyword "sub" may be specified to place the summary below the plot header. However, this latter option in only possible if mtext is not used. In case of coordinate specification, y-coordinate refers to the right y-axis.
summary.method	character (with default): keyword indicating the method used to calculate the statistic summary. One out of "unweighted", "weighted" and "MCM". See calc_Statistics for details.
bw	character (with default): bin-width, chose a numeric value for manual setting.
output	logical: Optional output of numerical plot parameters. These can be useful to reproduce similar plots. Default is FALSE.
...	further arguments and graphical parameters passed to plot.

Details

The function allows passing several plot arguments, such as main, xlab, cex. However, as the figure is an overlay of two separate plots, ylim must be specified in the order: c(ymin_axis1, ymax_axis1, ymin_axis2, ymax_axis2) when using the cumulative values plot option. See examples for some further explanations. For details on the calculation of the bin-width (parameter bw) see density.

A statistic summary, i.e. a collection of statistic measures of centrality and dispersion (and further measures) can be added by specifying one or more of the following keywords:

- "n" (number of samples)
- "mean" (mean De value)
- "median" (median of the De values)
- "sd.rel" (relative standard deviation in percent)

- "sd.abs" (absolute standard deviation)
- "se.rel" (relative standard error)
- "se.abs" (absolute standard error)
- "in.2s" (percent of samples in 2-sigma range)
- "kurtosis" (kurtosis)
- "skewness" (skewness)

Note that the input data for the statistic summary is sent to the function calc_Statistics() depending on the log-option for the z-scale. If "log.z = TRUE", the summary is based on the logarithms of the input data. If "log.z = FALSE" the linearly scaled data is used.
Note as well, that "calc_Statistics()" calculates these statistic measures in three different ways: unweighted, weighted and MCM-based (i.e., based on Monte Carlo Methods). By default, the MCM-based version is used. If you wish to use another method, indicate this with the appropriate keyword using the argument summary.method.

Function version

3.5.3 (2016-09-09 10:32:17)

Note

The plot output is no 'probability density' plot (cf. the discussion of Berger and Galbraith in Ancient TL; see references)!

Author(s)

Michael Dietze, GFZ Potsdam (Germany),
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne
R Luminescence Package Team

See Also

density, plot

Examples

```
## read example data set
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <-
  Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))

## create plot straightforward
plot_KDE(data = ExampleData.DeValues)

## create plot with logarithmic x-axis
plot_KDE(data = ExampleData.DeValues,
         log = "x")

## create plot with user-defined labels and axes limits
plot_KDE(data = ExampleData.DeValues,
         main = "Dose distribution",
```

```
         xlab = "Dose (s)",
         ylab = c("KDE estimate", "Cumulative dose value"),
         xlim = c(100, 250),
         ylim = c(0, 0.08, 0, 30))

## create plot with boxplot option
plot_KDE(data = ExampleData.DeValues,
         boxplot = TRUE)

## create plot with statistical summary below header
plot_KDE(data = ExampleData.DeValues,
         summary = c("n", "median", "skewness", "in.2s"))

## create plot with statistical summary as legend
plot_KDE(data = ExampleData.DeValues,
         summary = c("n", "mean", "sd.rel", "se.abs"),
         summary.pos = "topleft")

## split data set into sub-groups, one is manipulated, and merge again
data.1 <- ExampleData.DeValues[1:15,]
data.2 <- ExampleData.DeValues[16:25,] * 1.3
data.3 <- list(data.1, data.2)

## create plot with two subsets straightforward
plot_KDE(data = data.3)

## create plot with two subsets and summary legend at user coordinates
plot_KDE(data = data.3,
         summary = c("n", "median", "skewness"),
         summary.pos = c(110, 0.07),
         col = c("blue", "orange"))

## example of how to use the numerical output of the function
## return plot output to draw a thicker KDE line
KDE_out <- plot_KDE(data = ExampleData.DeValues,
output = TRUE)
```

`plot_NRt` *Visualise natural/regenerated signal ratios*

Description

This function creates a Natural/Regenerated signal vs. time (NR(t)) plot as shown in Steffen et al. 2009

Usage

```
plot_NRt(data, log = FALSE, smooth = c("none", "spline", "rmean"), k = 3,
  legend = TRUE, legend.pos = "topright", ...)
```

Arguments

data	a `list`, `data.frame`, `matrix` or `RLum.Analysis` object (**required**). X,Y data of measured values (time and counts). See details on individual data structure.

log	character (optional): logarithmic axes (c("x", "y", "xy")).
smooth	character (optional): apply data smoothing. Use "rmean" to calculate the rolling where k determines the width of the rolling window (see rollmean). "spline" applies a smoothing spline to each curve (see smooth.spline)
k	integer (with default): integer width of the rolling window.
legend	logical (with default): show or hide the plot legend.
legend.pos	character (with default): keyword specifying the position of the legend (see legend).
...	further parameters passed to plot (also see par).

Details

This function accepts the individual curve data in many different formats. If data is a list, each element of the list must contain a two column data.frame or matrix containing the XY data of the curves (time and counts). Alternatively, the elements can be objects of class RLum.Data.Curve. Input values can also be provided as a data.frame or matrix where the first column contains the time values and each following column contains the counts of each curve.

Value

Returns a plot and RLum.Analysis object.

Author(s)

Christoph Burow, University of Cologne (Germany)

References

Steffen, D., Preusser, F., Schlunegger, F., 2009. OSL quartz underestimation due to unstable signal components. Quaternary Geochronology, 4, 353-362.

See Also

plot

Examples

```
## load example data
data("ExampleData.BINfileData", envir = environment())

## EXAMPLE 1

## convert Risoe.BINfileData object to RLum.Analysis object
data <- Risoe.BINfileData2RLum.Analysis(object = CWOSL.SAR.Data, pos = 8, ltype = "OSL")

## extract all OSL curves
allCurves <- get_RLum(data)

## keep only the natural and regenerated signal curves
pos <- seq(1, 9, 2)
curves <- allCurves[pos]

## plot a standard NR(t) plot
```

```
plot_NRt(curves)

## re-plot with rolling mean data smoothing
plot_NRt(curves, smooth = "rmean", k = 10)

## re-plot with a logarithmic x-axis
plot_NRt(curves, log = "x", smooth = "rmean", k = 5)

## re-plot with custom axes ranges
plot_NRt(curves, smooth = "rmean", k = 5,
         xlim = c(0.1, 5), ylim = c(0.4, 1.6),
         legend.pos = "bottomleft")

## re-plot with smoothing spline on log scale
plot_NRt(curves, smooth = "spline", log = "x",
         legend.pos = "top")

## EXAMPLE 2

# you may also use this function to check whether all
# TD curves follow the same shape (making it a TnTx(t) plot).
posTD <- seq(2, 14, 2)
curves <- allCurves[posTD]

plot_NRt(curves, main = "TnTx(t) Plot",
         smooth = "rmean", k = 20,
         ylab = "TD natural / TD regenerated",
         xlim = c(0, 20), legend = FALSE)

## EXAMPLE 3

# extract data from all positions
data <- lapply(1:24, FUN = function(pos) {
   Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos = pos, ltype = "OSL")
})

# get individual curve data from each aliquot
aliquot <- lapply(data, get_RLum)

# set graphical parameters
par(mfrow = c(2, 2))

# create NR(t) plots for all aliquots
for (i in 1:length(aliquot)) {
   plot_NRt(aliquot[[i]][pos],
            main = paste0("Aliquot #", i),
            smooth = "rmean", k = 20,
            xlim = c(0, 10),
            cex = 0.6, legend.pos = "bottomleft")
}

# reset graphical parameters
par(mfrow = c(1, 1))
```

plot_RadialPlot *Function to create a Radial Plot*

Description

A Galbraith's radial plot is produced on a logarithmic or a linear scale.

Usage

```
plot_RadialPlot(data, na.rm = TRUE, negatives = "remove", log.z = TRUE,
  central.value, centrality = "mean.weighted", mtext, summary, summary.pos,
  legend, legend.pos, stats, rug = FALSE, plot.ratio, bar.col,
  y.ticks = TRUE, grid.col, line, line.col, line.label, output = FALSE, ...)
```

Arguments

data	data.frame or RLum.Results object (required): for data.frame two columns: De (data[,1]) and De error (data[,2]). To plot several data sets in one plot, the data sets must be provided as list, e.g. list(data.1, data.2).
na.rm	logical (with default): excludes NA values from the data set prior to any further operations.
negatives	character (with default): rule for negative values. Default is "remove" (i.e. negative values are removed from the data set).
log.z	logical (with default): Option to display the z-axis in logarithmic scale. Default is TRUE.
central.value	numeric: User-defined central value, primarily used for horizontal centering of the z-axis.
centrality	character or numeric (with default): measure of centrality, used for automatically centering the plot and drawing the central line. Can either be one out of "mean", "median", "mean.weighted" and "median.weighted" or a numeric value used for the standardisation.
mtext	character: additional text below the plot title.
summary	character (optional): add statistic measures of centrality and dispersion to the plot. Can be one or more of several keywords. See details for available keywords.
summary.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the statistical summary. Alternatively, the keyword "sub" may be specified to place the summary below the plot header. However, this latter option is only possible if mtext is not used.
legend	character vector (optional): legend content to be added to the plot.
legend.pos	numeric or character (with default): optional position coordinates or keyword (e.g. "topright") for the legend to be plotted.
stats	character: additional labels of statistically important values in the plot. One or more out of the following: "min", "max", "median".
rug	logical: Option to add a rug to the z-scale, to indicate the location of individual values

plot.ratio	numeric: User-defined plot area ratio (i.e. curvature of the z-axis). If omitted, the default value (4.5/5.5) is used and modified automatically to optimise the z-axis curvature. The parameter should be decreased when data points are plotted outside the z-axis or when the z-axis gets too elliptic.
bar.col	character or numeric (with default): colour of the bar showing the 2-sigma range around the central value. To disable the bar, use "none". Default is "grey".
y.ticks	logical: Option to hide y-axis labels. Useful for data with small scatter.
grid.col	character or numeric (with default): colour of the grid lines (originating at [0,0] and stretching to the z-scale). To disable grid lines, use "none". Default is "grey".
line	numeric: numeric values of the additional lines to be added.
line.col	character or numeric: colour of the additional lines.
line.label	character: labels for the additional lines.
output	logical: Optional output of numerical plot parameters. These can be useful to reproduce similar plots. Default is FALSE.
...	Further plot arguments to pass. xlab must be a vector of length 2, specifying the upper and lower x-axes labels.

Details

Details and the theoretical background of the radial plot are given in the cited literature. This function is based on an S script of Rex Galbraith. To reduce the manual adjustments, the function has been rewritten. Thanks to Rex Galbraith for useful comments on this function.
Plotting can be disabled by adding the argument plot = "FALSE", e.g. to return only numeric plot output.

Earlier versions of the Radial Plot in this package had the 2-sigma-bar drawn onto the z-axis. However, this might have caused misunderstanding in that the 2-sigma range may also refer to the z-scale, which it does not! Rather it applies only to the x-y-coordinate system (standardised error vs. precision). A spread in doses or ages must be drawn as lines originating at zero precision (x0) and zero standardised estimate (y0). Such a range may be drawn by adding lines to the radial plot (line, line.col, line.label, cf. examples).

A statistic summary, i.e. a collection of statistic measures of centrality and dispersion (and further measures) can be added by specifying one or more of the following keywords: "n" (number of samples), "mean" (mean De value), "mean.weighted" (error-weighted mean), "median" (median of the De values), "sdrel" (relative standard deviation in percent), "sdrel.weighted" (error-weighted relative standard deviation in percent), "sdabs" (absolute standard deviation), "sdabs.weighted" (error-weighted absolute standard deviation), "serel" (relative standard error), "serel.weighted" (error-weighted relative standard error), "seabs" (absolute standard error), "seabs.weighted" (error-weighted absolute standard error), "in.2s" (percent of samples in 2-sigma range), "kurtosis" (kurtosis) and "skewness" (skewness).

Value

Returns a plot object.

Function version

0.5.3 (2016-05-19 23:47:38)

Author(s)

Michael Dietze, GFZ Potsdam (Germany),
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
Based on a rewritten S script of Rex Galbraith, 2010
R Luminescence Package Team

References

Galbraith, R.F., 1988. Graphical Display of Estimates Having Differing Standard Errors. Technometrics, 30 (3), 271-281.

Galbraith, R.F., 1990. The radial plot: Graphical assessment of spread in ages. International Journal of Radiation Applications and Instrumentation. Part D. Nuclear Tracks and Radiation Measurements, 17 (3), 207-214.

Galbraith, R. & Green, P., 1990. Estimating the component ages in a finite mixture. International Journal of Radiation Applications and Instrumentation. Part D. Nuclear Tracks and Radiation Measurements, 17 (3) 197-206.

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks And Radiation Measurements, 21 (4), 459-470.

Galbraith, R.F., 1994. Some Applications of Radial Plots. Journal of the American Statistical Association, 89 (428), 1232-1242.

Galbraith, R.F., 2010. On plotting OSL equivalent doses. Ancient TL, 28 (1), 1-10.

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

See Also

plot, plot_KDE, plot_Histogram

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <- Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))

## plot the example data straightforward
plot_RadialPlot(data = ExampleData.DeValues)

## now with linear z-scale
plot_RadialPlot(data = ExampleData.DeValues,
                log.z = FALSE)

## now with output of the plot parameters
plot1 <- plot_RadialPlot(data = ExampleData.DeValues,
                         log.z = FALSE,
                         output = TRUE)
plot1
```

```
plot1$zlim

## now with adjusted z-scale limits
plot_RadialPlot(data = ExampleData.DeValues,
               log.z = FALSE,
               zlim = c(100, 200))

## now the two plots with serious but seasonally changing fun
#plot_RadialPlot(data = data.3, fun = TRUE)

## now with user-defined central value, in log-scale again
plot_RadialPlot(data = ExampleData.DeValues,
                central.value = 150)

## now with a rug, indicating individual De values at the z-scale
plot_RadialPlot(data = ExampleData.DeValues,
                rug = TRUE)

## now with legend, colour, different points and smaller scale
plot_RadialPlot(data = ExampleData.DeValues,
                legend.text = "Sample 1",
                col = "tomato4",
                bar.col = "peachpuff",
                pch = "R",
                cex = 0.8)

## now without 2-sigma bar, y-axis, grid lines and central value line
plot_RadialPlot(data = ExampleData.DeValues,
                bar.col = "none",
                grid.col = "none",
                y.ticks = FALSE,
                lwd = 0)

## now with user-defined axes labels
plot_RadialPlot(data = ExampleData.DeValues,
                xlab = c("Data error (%)",
                         "Data precision"),
                ylab = "Scatter",
                zlab = "Equivalent dose [Gy]")

## now with minimum, maximum and median value indicated
plot_RadialPlot(data = ExampleData.DeValues,
                central.value = 150,
                stats = c("min", "max", "median"))

## now with a brief statistical summary
plot_RadialPlot(data = ExampleData.DeValues,
                summary = c("n", "in.2s"))

## now with another statistical summary as subheader
plot_RadialPlot(data = ExampleData.DeValues,
                summary = c("mean.weighted", "median"),
                summary.pos = "sub")

## now the data set is split into sub-groups, one is manipulated
data.1 <- ExampleData.DeValues[1:15,]
data.2 <- ExampleData.DeValues[16:25,] * 1.3
```

```
## now a common dataset is created from the two subgroups
data.3 <- list(data.1, data.2)

## now the two data sets are plotted in one plot
plot_RadialPlot(data = data.3)

## now with some graphical modification
plot_RadialPlot(data = data.3,
                col = c("darkblue", "darkgreen"),
                bar.col = c("lightblue", "lightgreen"),
                pch = c(2, 6),
                summary = c("n", "in.2s"),
                summary.pos = "sub",
                legend = c("Sample 1", "Sample 2"))
```

`plot_Risoe.BINfileData` *Plot single luminescence curves from a BIN file object*

Description

Plots single luminescence curves from an object returned by the read_BIN2R function.

Usage

```
plot_Risoe.BINfileData(BINfileData, position, run, set, sorter = "POSITION",
  ltype = c("IRSL", "OSL", "TL", "RIR", "RBR", "RL"), curve.transformation,
  dose_rate, temp.lab, cex.global = 1, ...)
```

Arguments

`BINfileData`	Risoe.BINfileData-class (**required**): requires an S4 object returned by the read_BIN2R function.
`position`	vector (optional): option to limit the plotted curves by position (e.g. `position = 1`, `position = c(1,3,5)`).
`run`	vector (optional): option to limit the plotted curves by run (e.g., `run = 1`, `run = c(1,3,5)`).
`set`	vector (optional): option to limit the plotted curves by set (e.g., `set = 1`, `set = c(1,3,5)`).
`sorter`	character (with default): the plot output can be ordered by "POSITION","SET" or "RUN". POSITION, SET and RUN are options defined in the Risoe Sequence Editor.
`ltype`	character (with default): option to limit the plotted curves by the type of luminescence stimulation. Allowed values: `"IRSL"`, `"OSL"`,`"TL"`, `"RIR"`, `"RBR"` (corresponds to LM-OSL), `"RL"`. All type of curves are plotted by default.
`curve.transformation`	character (optional): allows transforming CW-OSL and CW-IRSL curves to pseudo-LM curves via transformation functions. Allowed values are: `CW2pLM`, `CW2pLMi`, `CW2pHMi` and `CW2pPMi`. See details.

dose_rate	numeric (optional): dose rate of the irradition source at the measurement date. If set, the given irradiation dose will be shown in Gy. See details.
temp.lab	character (optional): option to allow for different temperature units. If no value is set deg. C is chosen.
cex.global	numeric (with default): global scaling factor.
...	further undocumented plot arguments.

Details

Nomenclature

See Risoe.BINfileData-class

curve.transformation

This argument allows transforming continuous wave (CW) curves to pseudo (linear) modulated curves. For the transformation, the functions of the package are used. Currently, it is not possible to pass further arguments to the transformation functions. The argument works only for ltype OSL and IRSL.

Irradiation time

Plotting the irradiation time (s) or the given dose (Gy) requires that the variable IRR_TIME has been set within the BIN-file. This is normally done by using the 'Run Info' option within the Sequence Editor or by editing in R.

Value

Returns a plot.

Function version

0.4.1 (2015-11-29 17:27:48)

Note

The function has been successfully tested for the Sequence Editor file output version 3 and 4.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France),
Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

References

Duller, G., 2007. Analyst. pp. 1-45.

See Also

Risoe.BINfileData-class,read_BIN2R, CW2pLM, CW2pLMi, CW2pPMi, CW2pHMi

Examples

```
##load data
data(ExampleData.BINfileData, envir = environment())

##plot all curves from the first position to the desktop
#pdf(file = "~/Desktop/CurveOutput.pdf", paper = "a4", height = 11, onefile = TRUE)

##example - load from *.bin file
#BINfile<- file.choose()
#BINfileData<-read_BIN2R(BINfile)

#par(mfrow = c(4,3), oma = c(0.5,1,0.5,1))
#plot_Risoe.BINfileData(CWOSL.SAR.Data,position = 1)
#mtext(side = 4, BINfile, outer = TRUE, col = "blue", cex = .7)
#dev.off()
```

---

plot_RLum *General plot function for RLum S4 class objects*

Description

Function calls object specific plot functions for RLum S4 class objects.

Usage

```
plot_RLum(object, ...)
```

Arguments

object	RLum (**required**): S4 object of class RLum. Optional a list containing objects of class RLum can be provided. In this case the function tries to plot every object in this list according to its RLum class.
...	further arguments and graphical parameters that will be passed to the specific plot functions. The only argument that is supported directly is main (setting the plot title). In contrast to the normal behaviour main can be here provided as list and the arguments in the list will dispatched to the plots if the object is of type list as well.

Details

The function provides a generalised access point for plotting specific RLum objects.
Depending on the input object, the corresponding plot function will be selected. Allowed arguments can be found in the documentations of each plot function.

object		corresponding plot function
RLum.Data.Curve	:	plot_RLum.Data.Curve
RLum.Data.Spectrum	:	plot_RLum.Data.Spectrum
RLum.Data.Image	:	plot_RLum.Data.Image
RLum.Analysis	:	plot_RLum.Analysis
RLum.Results	:	plot_RLum.Results

Value

Returns a plot.

Function version

0.4.2 (2016-09-09 10:32:17)

Note

The provided plot output depends on the input object.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

#

See Also

plot_RLum.Data.Curve, RLum.Data.Curve, plot_RLum.Data.Spectrum, RLum.Data.Spectrum, plot_RLum.Data.Image, RLum.Data.Image, plot_RLum.Analysis, RLum.Analysis, plot_RLum.Results, RLum.Results

Examples

```
#load Example data
data(ExampleData.CW_OSL_Curve, envir = environment())

#transform data.frame to RLum.Data.Curve object
temp <- as(ExampleData.CW_OSL_Curve, "RLum.Data.Curve")

#plot RLum object
plot_RLum(temp)
```

---

plot_RLum.Analysis *Plot function for an RLum.Analysis S4 class object*

Description

The function provides a standardised plot output for curve data of an RLum.Analysis S4 class object

Usage

```
plot_RLum.Analysis(object, subset, nrows, ncols, abline = NULL,
  combine = FALSE, curve.transformation, plot.single = FALSE, ...)
```

Arguments

object	RLum.Analysis (**required**): S4 object of class RLum.Analysis
subset	named list (optional): subsets elements for plotting. The arguments in the named list will be directly passed to the function get_RLum (e.g., subset = list(curveType = "me
nrows	integer (optional): sets number of rows for plot output, if nothing is set the function tries to find a value.
ncols	integer (optional): sets number of columns for plot output, if nothing is set the function tries to find a value.
abline	list (optional): allows to add ablines to the plot. Argument are provided in a list and will be forwared to the function abline, e.g., list(v = c(10, 100)) adds two vertical lines add 10 and 100 to all plots. In contrast list(v = c(10), v = c(100) adds a vertical at 10 to the first and a vertical line at 100 to the 2nd plot.
combine	logical (with default): allows to combine all RLum.Data.Curve objects in one single plot.
curve.transformation	character (optional): allows transforming CW-OSL and CW-IRSL curves to pseudo-LM curves via transformation functions. Allowed values are: CW2pLM, CW2pLMi, CW2pHMi and CW2pPMi. See details.
plot.single	logical (with default): global par settings are considered, normally this should end in one plot per page
...	further arguments and graphical parameters will be passed to the plot function. Supported arguments: main, mtext, log, lwd, lty type, pch, col, norm, xlim,ylim, xlab, ylab... and for combine = TRUE also: sub, legend, legend.text, legend.pos (typical plus 'outside'), legend.col, smooth. All arguments can be provided as vector or list to gain in full control of all plot settings.

Details

The function produces a multiple plot output. A file output is recommended (e.g., pdf).

curve.transformation

This argument allows transforming continuous wave (CW) curves to pseudo (linear) modulated curves. For the transformation, the functions of the package are used. Currently, it is not possible to pass further arguments to the transformation functions. The argument works only for ltype OSL and IRSL.

Please note: The curve transformation within this functions works roughly, i.e. every IRSL or OSL curve is transformed, without considerung whether it is measured with the PMT or not! However, for a fast look it might be helpful.

Value

Returns multiple plots.

Function version

0.3.6 (2016-09-09 10:32:17)

Note

Not all arguments available for plot will be passed! Only plotting of RLum.Data.Curve and RLum.Data.Spectrum objects are currently supported.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

#

See Also

plot, plot_RLum, plot_RLum.Data.Curve

Examples

```
##load data
data(ExampleData.BINfileData, envir = environment())

##convert values for position 1
temp <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)

##plot (combine) TL curves in one plot
plot_RLum.Analysis(
temp,
subset = list(recordType = "TL"),
combine = TRUE,
norm = TRUE,
abline = list(v = c(110))
)
```

---

plot_RLum.Data.Curve *Plot function for an RLum.Data.Curve S4 class object*

Description

The function provides a standardised plot output for curve data of an RLum.Data.Curve S4 class object

Usage

```
plot_RLum.Data.Curve(object, par.local = TRUE, norm = FALSE,
  smooth = FALSE, ...)
```

Arguments

object	RLum.Data.Curve (**required**): S4 object of class RLum.Data.Curve
par.local	logical (with default): use local graphical parameters for plotting, e.g. the plot is shown in one column and one row. If par.local = FALSE, global parameters are inherited.
norm	logical (with default): allows curve normalisation to the highest count value
smooth	logical (with default): provides an automatic curve smoothing based on rollmean
...	further arguments and graphical parameters that will be passed to the plot function

Details

Only single curve data can be plotted with this function. Arguments according to plot.

Value

Returns a plot.

Function version

0.2.0 (2016-05-02 09:36:06)

Note

Not all arguments of plot will be passed!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

#

See Also

plot, plot_RLum

Examples

```
##plot curve data

#load Example data
data(ExampleData.CW_OSL_Curve, envir = environment())

#transform data.frame to RLum.Data.Curve object
temp <- as(ExampleData.CW_OSL_Curve, "RLum.Data.Curve")

#plot RLum.Data.Curve object
plot_RLum.Data.Curve(temp)
```

plot_RLum.Data.Image *Plot function for an* RLum.Data.Image *S4 class object*

Description

The function provides a standardised plot output for image data of an RLum.Data.ImageS4 class object, mainly using the plot functions provided by the raster package.

Usage

```
plot_RLum.Data.Image(object, par.local = TRUE, plot.type = "plot.raster",
  ...)
```

Arguments

object	RLum.Data.Image (**required**): S4 object of class RLum.Data.Image
par.local	logical (with default): use local graphical parameters for plotting, e.g. the plot is shown in one column and one row. If par.local = FALSE global parameters are inherited.
plot.type	character (with default): plot types. Supported types are plot.raster, plotRGB or contour
...	further arguments and graphical parameters that will be passed to the specific plot functions.

Details

Details on the plot functions

Image is visualised as 2D plot usinng generic plot types provided by other packages.

Supported plot types:

plot.type = "plot.raster"

Uses the standard plot function for raster data from the package raster: plot. For each raster layer in a raster brick one plot is produced.

Arguments that are passed through the function call:

main,axes, xlab, ylab, xlim, ylim, col

plot.type = "plotRGB"

Uses the function plotRGB from the raster package. Only one image plot is produced as all layers in a brick a combined. This plot type is useful to see whether any signal is recorded by the camera. Arguments that are passed through the function call:

main,axes, xlab, ylab, ext, interpolate, maxpixels, alpha, colNA, stretch

`plot.type = "contour"`

Uses the function contour plot function from the `raster` function (`contour`). For each raster layer one contour plot is produced. Arguments that are passed through the function call:

`main`,`axes`, `xlab`, `ylab`, `xlim`, `ylim`, `col`

Value

Returns a plot.

Function version

0.1 (2015-11-29 17:27:48)

Note

This function has been created to faciliate the plotting of image data imported by the function `read_SPE2R`. However, so far the function is not optimized to handle image data > ca. 200 MByte and thus plotting of such data is extremely slow.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

-

See Also

`RLum.Data.Image`, `plot`, `plot_RLum`, `raster`,

Examples

```
##load data
data(ExampleData.RLum.Data.Image, envir = environment())

##plot data
plot_RLum.Data.Image(ExampleData.RLum.Data.Image)
```

plot_RLum.Data.Spectrum

Plot function for an RLum.Data.Spectrum S4 class object

Description

The function provides a standardised plot output for spectrum data of an RLum.Data.Spectrum S4 class object

Usage

```
plot_RLum.Data.Spectrum(object, par.local = TRUE, plot.type = "contour",
  optical.wavelength.colours = TRUE, bg.channels, bin.rows = 1,
  bin.cols = 1, rug = TRUE, limit_counts = NULL, xaxis.energy = FALSE,
  legend.text, ...)
```

Arguments

object	RLum.Data.Spectrum or matrix (**required**): S4 object of class RLum.Data.Spectrum or a matrix containing count values of the spectrum. Please note that in case of a matrix rownames and colnames are set automatically if not provided.
par.local	logical (with default): use local graphical parameters for plotting, e.g. the plot is shown in one column and one row. If par.local = FALSE global parameters are inherited.
plot.type	character (with default): plot type, for 3D-plot use persp, or interactive, for a 2D-plot contour, single or multiple.lines (along the time or temperature axis) or transect (along the wavelength axis)
optical.wavelength.colours	logical (with default): use optical wavelength colour palette. Note: For this, the spectrum range is limited: c(350,750). Own colours can be set with the argument col.
bg.channels	vector (optional): defines channel for background subtraction If a vector is provided the mean of the channels is used for subtraction. Note: Background subtraction is applied prior to channel binning
bin.rows	integer (with defaul): allow summing-up wavelength channels (horizontal binning), e.g. bin.rows = 2 two channels are summed up
bin.cols	integer (with default): allow summing-up channel counts (vertical binning) for plotting, e.g. bin.cols = 2 two channels are summed up
rug	logical (with default): enables or disables colour rug. Currently only implemented for plot type multiple.lines and single
limit_counts	numeric (optional): value to limit all count values to this value, i.e. all count values above this threshold will be replaced by this threshold. This is helpfull especially in case of TL-spectra.
xaxis.energy	logical (with default): enables or disables energy instead of wavelength axis. Note: This option means not only simnply redrawing the axis, insteadly the spectrum in terms of intensity is recalculated, s. details.

`legend.text`	`character` (with default): possiblity to provide own legend text. This argument is only considered for plot types providing a legend, e.g. `plot.type="transect"`
`...`	further arguments and graphical parameters that will be passed to the `plot` function.

Details

Matrix structure

(cf. `RLum.Data.Spectrum`)

- `rows` (x-values): wavelengths/channels (xlim, xlab)
- `columns` (y-values): time/temperature (ylim, ylab)
- `cells` (z-values): count values (zlim, zlab)

Note: This nomenclature is valid for all plot types of this function!

Nomenclature for value limiting

`xlim`: Limits values along the wavelength axis
`ylim`: Limits values along the time/temperature axis
`zlim`: Limits values along the count value axis

Energy axis re-calculation

If the argument `xaxis.energy = TRUE` is chosen, instead intensity vs. wavelength the spectrum is plotted as intensiyt vs. energy. Therefore the entire spectrum is re-recaluated (e.g., Appendix 4 in Blasse and Grabmeier, 1994):

The intensity of the spectrum (z-values) is re-calcualted using the following equation:

$$\phi_E = \phi_\lambda * \lambda^2/(hc)$$

with ϕ_E the intensity per interval of energy E (eV), ϕ_λ the intensity per interval of wavelength λ (nm) and h (eV/s) the Planck constant and c (m/s) the velocity of light.

For transforming the wavelength axis (x-values) the equation

$$E = hc/\lambda$$

is used. For further details please see the cited the literature.

Details on the plot functions

Spectrum is visualised as 3D or 2D plot. Both plot types are based on internal R plot functions.

`plot.type = "persp"`

Arguments that will be passed to `persp`:

- `shade`: default is `0.4`
- `phi`: default is `15`
- `theta`: default is `-30`
- `expand`: default is `1`

- `ticktype`: default is `detailed`, `r`: default is `10`

Note: Further parameters can be adjusted via `par`. *For example to set the background transparent and reduce the thickness of the lines use:* `par(bg = NA, lwd = 0.7)` *previous the function call.*

`plot.type = "single"`

Per frame a single curve is returned. Frames are time or temperature steps.

`plot.type = "multiple.lines"`

All frames plotted in one frame.

`plot.type = "transect"`

Depending on the selected wavelength/channel range a transect over the time/temperature (y-axis) will be plotted along the wavelength/channels (x-axis). If the range contains more than one channel, values (z-values) are summed up. To select a transect use the `xlim` argument, e.g. `xlim = c(300,310)` plot along the summed up count values of channel 300 to 310.

Further arguments that will be passed (depending on the plot type)

`xlab`, `ylab`, `zlab`, `xlim`, `ylim`, `zlim`, `main`, `mtext`, `pch`, `type`, `col`, `border`, `box lwd`, `bty`

Value

Returns a plot.

Function version

0.5.0 (2016-09-09 10:32:17)

Note

Not all additional arguments (`...`) will be passed similarly!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Blasse, G., Grabmaier, B.C., 1994. Luminescent Materials. Springer.

See Also

`RLum.Data.Spectrum`, `plot`, `plot_RLum`, `persp`, `plot_ly`, `contour`

Examples

```
##load example data
data(ExampleData.XSYG, envir = environment())

##(1)plot simple spectrum (2D) - contour
plot_RLum.Data.Spectrum(TL.Spectrum,
                        plot.type="contour",
                        xlim = c(310,750),
                        ylim = c(0,300),
                        bin.rows=10,
                        bin.cols = 1)

##(2) plot spectrum (3D)
plot_RLum.Data.Spectrum(TL.Spectrum,
                        plot.type="persp",
                        xlim = c(310,750),
                        ylim = c(0,100),
                        bin.rows=10,
                        bin.cols = 1)

##(3) plot multiple lines (2D) - multiple.lines (with ylim)
plot_RLum.Data.Spectrum(TL.Spectrum,
                        plot.type="multiple.lines",
                        xlim = c(310,750),
                        ylim = c(0,100),
                        bin.rows=10,
                        bin.cols = 1)

## Not run:
 ##(4) interactive plot using the package plotly
 plot_RLum.Data.Spectrum(TL.Spectrum, plot.type="interactive",
 xlim = c(310,750), ylim = c(0,300), bin.rows=10,
 bin.cols = 1)

 ##(5) alternative using the package fields
 fields::image.plot(get_RLum(TL.Spectrum))
 contour(get_RLum(TL.Spectrum), add = TRUE)


## End(Not run)
```

---

plot_RLum.Results *Plot function for an RLum.Results S4 class object*

Description

The function provides a standardised plot output for data of an RLum.Results S4 class object

Usage

```
plot_RLum.Results(object, single = TRUE, ...)
```

Arguments

object	RLum.Results (**required**): S4 object of class RLum.Results
single	logical (with default): single plot output (TRUE/FALSE) to allow for plotting the results in as few plot windows as possible.
...	further arguments and graphical parameters will be passed to the plot function.

Details

The function produces a multiple plot output. A file output is recommended (e.g., pdf).

Value

Returns multiple plots.

Function version

0.2.1 (2016-05-16 22:24:15)

Note

Not all arguments available for plot will be passed! Only plotting of RLum.Results objects are supported.

Author(s)

Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

#

See Also

plot, plot_RLum,

Examples

```
###load data
data(ExampleData.DeValues, envir = environment())

# apply the un-logged minimum age model
mam <- calc_MinDose(data = ExampleData.DeValues$CA1, sigmab = 0.2, log = TRUE, plot = FALSE)

##plot
plot_RLum.Results(mam)

# estimate the number of grains on an aliquot
grains<- calc_AliquotSize(grain.size = c(100,150), sample.diameter = 1, plot = FALSE, MC.iter = 100)

##plot
```

```
plot_RLum.Results(grains)
```

`plot_ViolinPlot` *Create a violin plot*

Description

Draws a kernal densiy plot in combination with a boxplot in its middle. The shape of the violin is constructed using a mirrored density curve. This plot is especially designed for cases where the individual errors are zero or to small to be visualised. The idea for this plot is based on the the 'volcano plot' in the ggplot2 package by Hadely Wickham and Winston Chang. The general idea for the Violin Plot seems to be introduced by Hintze and Nelson (1998).

Usage

```
plot_ViolinPlot(data, boxplot = TRUE, rug = TRUE, summary = NULL,
  summary.pos = "sub", na.rm = TRUE, ...)
```

Arguments

`data`	`numeric` or `RLum.Results` object (required): input data for plotting. Alternatively a `data.frame` or a `matrix` can be provided, but only the first column will be considered by the function
`boxplot`	`logical` (with default): enable or disable boxplot
`rug`	`logical` (with default): enable or disable rug
`summary`	`character` (optional): add statistic measures of centrality and dispersion to the plot. Can be one or more of several keywords. See details for available keywords.
`summary.pos`	`numeric` or `character` (with default): optional position keywords (cf., `legend`) for the statistical summary. Alternatively, the keyword `"sub"` may be specified to place the summary below the plot header. However, this latter option in only possible if `mtext` is not used.
`na.rm`	`logical` (with default): exclude NA values from the data set prior to any further operations.
`...`	further arguments and graphical parameters passed to `plot.default`, `density` and `boxplot`. See details for further information

Details

The function is passing several arguments to the function `plot`, `density`, `boxplot`: Supported arguments are: `xlim`, `main`, `xlab`, `ylab`, `col.violin`, `col.boxplot`, `mtext`, `cex`, `mtext`

`Valid summary keywords`

'n', 'mean', 'median', 'sd.abs', 'sd.rel', 'se.abs', 'se.rel', 'skewness', 'kurtosis'

Function version

0.1.2 (2016-05-17 13:27:04)

Note

Although the code for this function was developed independently and just the idea for the plot was based on the 'ggplot2' package plot type 'volcano', it should be mentioned that, beyond this, two other R packages exist providing a possibility to produces this kind of plot, namely: 'vioplot' and 'violinmplot' (see References for details).

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Daniel Adler (2005). vioplot: A violin plot is a combination of a box plot and a kernel density plot. R package version 0.2 http://CRAN.R-project.org/package=violplot

Hintze, J.L., Nelson, R.D., 1998. A Box Plot-Density Trace Synergism. The American Statistician 52, 181-184.

Raphael W. Majeed (2012). violinmplot: Combination of violin plot with mean and standard deviation. R package version 0.2.1. http://CRAN.R-project.org/package=violinmplot

Wickham. H (2009). ggplot2: elegant graphics for data analysis. Springer New York.

See Also

density, plot, boxplot, rug, calc_Statistics

Examples

```
## read example data set
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <- Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))

## create plot straightforward
plot_ViolinPlot(data = ExampleData.DeValues)
```

---

read_BIN2R *Import Risoe BIN-file into R*

Description

Import a *.bin or a *.binx file produced by a Risoe DA15 and DA20 TL/OSL reader into R.

Usage

```
read_BIN2R(file, show.raw.values = FALSE, position = NULL,
  n.records = NULL, zero_data.rm = TRUE, duplicated.rm = FALSE,
  fastForward = FALSE, show.record.number = FALSE, txtProgressBar = TRUE,
  forced.VersionNumber = NULL, pattern = NULL, verbose = TRUE, ...)
```

Arguments

file	character or list (**required**): path and file name of the BIN/BINX file. If input is a list it should comprise only characters representing each valid path and BIN/BINX-file names. Alternatively the input character can be just a directory (path), in this case the the function tries to detect and import all BIN/BINX files found in the directory.
show.raw.values	logical (with default): shows raw values from BIN file for LTYPE, DTYPE and LIGHTSOURCE without translation in characters. Can be provided as list if file is a list.
position	numeric (optional): imports only the selected position. Note: the import performance will not benefit by any selection made here. Can be provided as list if file is a list.
n.records	raw (optional): limits the number of imported records. Can be used in combination with show.record.number for debugging purposes, e.g. corrupt BIN-files. Can be provided as list if file is a list.
zero_data.rm	logical (with default): remove erroneous data with no count values. As such data are usally not needed for the subsequent data analysis they will be removed by default. Can be provided as list if file is a list.
duplicated.rm	logical (with default): remove duplicated entries if TRUE. This may happen due to an erroneous produced BIN/BINX-file. This option compares only predeccessor and successor. Can be provided as list if file is a list.
fastForward	logical (with default): if TRUE for a more efficient data processing only a list of RLum.Analysis objects is returned instead of a Risoe.BINfileData-class object. Can be provided as list if file is a list.
show.record.number	logical (with default): shows record number of the imported record, for debugging usage only. Can be provided as list if file is a list.
txtProgressBar	logical (with default): enables or disables txtProgressBar.
forced.VersionNumber	integer (optional): allows to cheat the version number check in the function by own values for cases where the BIN-file version is not supported. Can be provided as list if file is a list. Note: The usage is at own risk, only supported BIN-file versions have been tested.
pattern	character (optional): argument that is used if only a path is provided. The argument will than be passed to the function list.files used internally to construct a list of wanted files
verbose	logical (with default): enables or disables verbose mode
...	further arguments that will be passed to the function Risoe.BINfileData2RLum.Analysis. Please note that any matching argument automatically sets fastForward = TRUE

Details

The binary data file is parsed byte by byte following the data structure published in the Appendices of the Analyst manual p. 42.

For the general BIN-file structure, the reader is referred to the Risoe website: http://www.nutech.dtu.dk/

Value

Returns an S4 Risoe.BINfileData-class object containing two slots:

METADATA	A data.frame containing all variables stored in the bin-file.
DATA	A list containing a numeric vector of the measured data. The ID corresponds to the record ID in METADATA.

If `fastForward = TRUE` a list of `RLum.Analysis` object is returned. The internal coercing is done using the function `Risoe.BINfileData2RLum.Analysis`

Function version

0.15.0 (2016-06-13 21:17:19)

Note

The function works for BIN/BINX-format versions 03, 04, 06, 07 and 08. The version number depends on the used Sequence Editor.

ROI data sets introduced with BIN-file version 8 are not supported and skipped durint import.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Margret C. Fuchs, HZDR Freiberg, (Germany)
R Luminescence Package Team

References

DTU Nutech, 2016. The Squence Editor, Users Manual, February, 2016. http://www.nutech.dtu.dk/english/Products-and-Services/Dosimetry/Radiation-Measurement-Instruments/TL_OSL_reader/Manuals

See Also

`write_R2BIN`, `Risoe.BINfileData`, `readBin`, `merge_Risoe.BINfileData`, `RLum.Analysis` `txtProgressBar`, `list.files`

Examples

```
##(1) import Risoe BIN-file to R (uncomment for usage)

#FILE <- file.choose()
#temp <- read_BIN2R(FILE)
#temp
```

read_Daybreak2R *Import Daybreak ASCII dato into R*

Description

Import a *.txt (ASCII) file produced by a Daybreak reader into R.

Usage

```
read_Daybreak2R(file, verbose = TRUE, txtProgressBar = TRUE)
```

Arguments

file	character or list (**required**): path and file name of the file to be imported. Alternatively a list of file names can be provided or just the path a folder containing measurement data. Please note that the specific, common, file extension (txt) is likely leading to function failures during import when just a path is provided.
verbose	logical (with default): enables or disables terminal feedback
txtProgressBar	logical (with default): enables or disables txtProgressBar.

Value

A list of RLum.Analysis objects (each per position) is provided.

Function version

0.2.1 (2016-05-02 09:36:06)

Note

[BETA VERSION] This function version still needs to be properly tested.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
Based on a suggestion by Willian Amidon and Andrew Louis Gorin.
R Luminescence Package Team

References

-

See Also

RLum.Analysis, RLum.Data.Curve

Examples

```
## This function has no example yet.
```

read_SPE2R *Import Princeton Intruments (TM) SPE-file into R*

Description

Function imports Princeton Instruments (TM) SPE-files into R environment and provides RLum objects as output.

Usage

```
read_SPE2R(file, output.object = "RLum.Data.Image", frame.range,
  txtProgressBar = TRUE)
```

Arguments

file	character (**required**): spe-file name (including path), e.g. [WIN]: read_SPE2R("C:/Desktop/test.spe"), [MAC/LINUX]: readSPER("/User/test/Desktop/test.spe")
output.object	character (with default): set RLum output object. Allowed types are "RLum.Data.Spectrum", "RLum.Data.Image" or "matrix"
frame.range	vector (optional): limit frame range, e.g. select first 100 frames by frame.range = c(1,100)
txtProgressBar	logical (with default): enables or disables txtProgressBar.

Details

Function provides an import routine for the Princton Instruments SPE format. Import functionality is based on the file format description provided by Princton Instruments and a MatLab script written by Carl Hall (s. references).

Value

Depending on the chosen option the functions returns three different type of objects:

output.object.

RLum.Data.Spectrum

An object of type RLum.Data.Spectrum is returned. Row sums are used to integrate all counts over one channel.

RLum.Data.Image

An object of type RLum.Data.Image is returned. Due to performace reasons the import is aborted for files containing more than 100 frames. This limitation can be overwritten manually by using the argument frame.frange.

matrix

Returns a matrix of the form: Rows = Channels, columns = Frames. For the transformation the function get_RLum is used, meaning that the same results can be obtained by using the function get_RLum on an RLum.Data.Spectrum or RLum.Data.Image object.

Function version

0.1.0 (2016-05-02 09:42:32)

Note

The function does not test whether the input data are spectra or pictures for spatial resolved analysis!

The function has been successfully tested for SPE format versions 2.x.

Currently not all information provided by the SPE format are supported.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Princeton Instruments, 2014. Princeton Instruments SPE 3.0 File Format Specification, Version 1.A, ftp://ftp.princetoninstruments.com/Public/Manuals/Princeton%20Instruments/SPE%203.0%20File%20Format%20Specification.pdf

Hall, C., 2012: readSPE.m. http://www.mathworks.com/matlabcentral/fileexchange/35940-readspe/content/readSPE.m

See Also

readBin, RLum.Data.Spectrum, raster

Examples

```
## to run examples uncomment lines and run the code

##(1) Import data as RLum.Data.Spectrum object
#file <- file.choose()
#temp <- read_SPE2R(file)
#temp

##(2) Import data as RLum.Data.Image object
#file <- file.choose()
#temp <- read_SPE2R(file, output.object = "RLum.Data.Image")
#temp

##(3) Import data as matrix object
#file <- file.choose()
#temp <- read_SPE2R(file, output.object = "matrix")
#temp

##(4) Export raw data to csv, if temp is a RLum.Data.Spectrum object
# write.table(x = get_RLum(temp),
#             file = "[your path and filename]",
#             sep = ";", row.names = FALSE)
```

read_XSYG2R *Import XSYG files to R*

Description

Imports XSYG files produced by a Freiberg Instrument lexsyg reader into R.

Usage

```
read_XSYG2R(file, recalculate.TL.curves = TRUE, fastForward = FALSE,
  import = TRUE, pattern = ".xsyg", txtProgressBar = TRUE)
```

Arguments

file	character or list (**required**): path and file name of the XSYG file. If input is a list it should comprise only characters representing each valid path and xsyg-file names. Alternatively the input character can be just a directory (path), in this case the the function tries to detect and import all xsyg files found in the directory.
recalculate.TL.curves	logical (with default): if set to TRUE, TL curves are returned as temperature against count values (see details for more information) Note: The option overwrites the time vs. count TL curve. Select FALSE to import the raw data delivered by the lexsyg. Works for TL curves and spectra.
fastForward	logical (with default): if TRUE for a more efficient data processing only a list of RLum.Analysis objects is returned.
import	logical (with default): if set to FALSE, only the XSYG file structure is shown.
pattern	regex (with default): optional regular expression if file is a link to a folder, to select just specific XSYG-files
txtProgressBar	logical (with default): enables TRUE or disables FALSE the progression bar during import

Details

How does the import function work?

The function uses the xml package to parse the file structure. Each sequence is subsequently translated into an RLum.Analysis object.

General structure XSYG format

```
<?xml?
<Sample>
<Sequence>
<Record>
<Curve name="first curve" />
<Curve name="curve with data">
```

```
x0 , y0 ; x1 , y1 ; x2 , y2 ; x3 , y3
</Curve>
</Record>
</Sequence>
</Sample>
```

So far, each XSYG file can only contain one `<Sample></Sample>`, but multiple sequences.

Each record may comprise several curves.

TL curve recalculation

On the FI lexsyg device TL curves are recorded as time against count values. Temperature values are monitored on the heating plate and stored in a separate curve (time vs. temperature). If the option `recalculate.TL.curves = TRUE` is chosen, the time values for each TL curve are replaced by temperature values.

Practically, this means combining two matrices (Time vs. Counts and Time vs. Temperature) with different row numbers by their time values. Three cases are considered:

HE: Heating element
PMT: Photomultiplier tube
Interpolation is done using the function `approx`

CASE (1): `nrow(matrix(PMT)) > nrow(matrix(HE))`

Missing temperature values from the heating element are calculated using time values from the PMT measurement.

CASE (2): `nrow(matrix(PMT)) < nrow(matrix(HE))`

Missing count values from the PMT are calculated using time values from the heating element measurement.

CASE (3): `nrow(matrix(PMT)) == nrow(matrix(HE))`

A new matrix is produced using temperature values from the heating element and count values from the PMT.

Note: Please note that due to the recalculation of the temperature values based on values delivered by the heating element, it may happen that mutiple count values exists for each temperature value and temperature values may also decrease during heating, not only increase.

Advanced file import

To allow for a more efficient usage of the function, instead of single path to a file just a directory can be passed as input. In this particular case the function tries to extract all XSYG-files found in the directory and import them all. Using this option internally the function constructs as list of the

XSYG-files found in the directory. Please note no recursive detection is supported as this may lead to endless loops.

Value

Using the option `import = FALSE`

A list consisting of two elements is shown:

`Sample`	data.frame with information on file.
`Sequences`	data.frame with information on the sequences stored in the XSYG file

.

Using the option `import = TRUE` **(default)**

A list is provided, the list elements contain:

`Sequence.Header`
: data.frame with information on the sequence.

`Sequence.Object`
: `RLum.Analysis` containing the curves.

Function version

0.5.7 (2016-09-05 20:21:40)

Note

This function is a beta version as the XSYG file format is not yet fully specified. Thus, further file operations (merge, export, write) should be done using the functions provided with the package `xml`.

So far, no image data import is provided!
Corresponding values in the XSXG file are skipped.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Grehl, S., Kreutzer, S., Hoehne, M., 2013. Documentation of the XSYG file format. Unpublished Technical Note. Freiberg, Germany

Further reading

XML: `http://en.wikipedia.org/wiki/XML`

See Also

`xml`, `RLum.Analysis`, `RLum.Data.Curve`, `approx`

Examples

```
##(1) import XSYG file to R (uncomment for usage)

#FILE <- file.choose()
#temp <- read_XSYG2R(FILE)

##(2) additional examples for pure XML import using the package XML
##    (uncomment for usage)

  ##import entire XML file
  #FILE <- file.choose()
  #temp <- XML::xmlRoot(XML::xmlTreeParse(FILE))

  ##search for specific subnodes with curves containing 'OSL'
  #getNodeSet(temp, "//Sample/Sequence/Record[@recordType = 'OSL']/Curve")

##(2) How to extract single curves ... after import
data(ExampleData.XSYG, envir = environment())

##grep one OSL curves and plot the first curve
OSLcurve <- get_RLum(OSL.SARMeasurement$Sequence.Object, recordType="OSL")[[1]]

##(3) How to see the structure of an object?
structure_RLum(OSL.SARMeasurement$Sequence.Object)
```

`replicate_RLum` *General replication function for RLum S4 class objects*

Description

Function replicates RLum S4 class objects and returns a list for this objects

Usage

```
replicate_RLum(object, times = NULL)
```

Arguments

`object`	an object of class `RLum` (**required**)
`times`	`integer` (optional): number for times each element is repeated element

Value

Returns a `list` of the object to be repeated

Function version

0.1.0 (2015-11-29 17:27:48)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

RLum,

`report_RLum` *Create a HTML report for (RLum) objects*

Description

This function creates a HTML report for a given object, listing its complete structure and content. The object itself is saved as a serialised .Rds file. The report file serves both as a convenient way of browsing through objects with complex data structures as well as a mean of properly documenting and saving objects.

Usage

```
report_RLum(object, file = tempfile(), title = "RLum.Report",
  compact = TRUE, timestamp = TRUE, launch.browser = FALSE,
  css.file = NULL, quiet = TRUE, clean = TRUE, ...)
```

Arguments

`object`	(**required**): The object to be reported on, preferably of any `RLum`-class.
`file`	`character` (with default): A character string naming the output file. If no filename is provided a temporary file is created.
`title`	`character` (with default): A character string specifying the title of the document.
`compact`	`logical` (with default): When `TRUE` the following report components are hidden: `@.pid`, `@.uid`, `'Object structure'`, `'Session Info'` and only the first and last 5 rows of long matrices and data frames are shown. See details.
`timestamp`	`logical` (with default): `TRUE` to add a timestamp to the filename (suffix).
`launch.browser`	`logical` (with default): `TRUE` to open the HTML file in the system's default web browser after it has been rendered.
`css.file`	`character` (optional): Path to a CSS file to change the default styling of the HTML document.
`quiet`	`logical` (with default): `TRUE` to supress printing of the pandoc command line.
`clean`	`logical` (with default): `TRUE` to clean intermediate files created during rendering.
`...`	further arguments passed to or from other methods and to control the document's structure (see details).

Details

The HTML report is created with `render` and has the following structure:

Section	Description
Header	A summary of general characteristics of the object
Object content	A comprehensive list of the complete structure and content of the provided object.
Object structure	Summary of the objects structure given as a table
File	Information on the saved RDS file
Session Info	Captured output from sessionInfo()
Plots	(optional) For RLum-class objects a variable number of plots

The structure of the report can be controlled individually by providing one or more of the following arguments (all logical):

Argument	Description
header	Hide or show general information on the object
main	Hide or show the object's content
structure	Hide or show object's structure
rds	Hide or show information on the saved RDS file
session	Hide or show the session info
plot	Hide or show the plots (depending on object)

Note that these arguments have higher precedence than compact.

Further options that can be provided via the ... argument:

Argument	Description
short_table	If TRUE only show the first and last 5 rows of lang tables.
theme	Specifies the Bootstrap theme to use for the report. Valid themes include "default", "cerulean", "journal",
highlight	Specifies the syntax highlighting style. Supported styles include "default", "tango", "pygments", "kate", "
css	TRUE or FALSE to enable/disable custom CSS styling

The following arguments can be used to customise the report via CSS (Cascading Style Sheets):

Argument	Description
font_family	Define the font family of the HTML document (default: arial)
headings_size	Size of the <h1> to <h6> tags used to define HTML headings (default: 166%).
content_color	Color of the object's content (default: #a72925).

Note that these arguments must all be of class character and follow standard CSS syntax. For exhaustive CSS styling you can provide a custom CSS file for argument css.file. CSS styling can be turned of using css = FALSE.

Value

Writes a HTML and .Rds file.

Function version

0.1.0 (2016-09-09 10:32:17)

Note

This function requires the R packages 'rmarkdown', 'pander' and 'rstudioapi'.

Author(s)

Christoph Burow, University of Cologne (Germany)

R Luminescence Package Team

See Also

render, pander_return, openFileInOS, viewer, browseURL

Examples

```
## Not run:
## Example: RLum.Results ----

# load example data
data("ExampleData.DeValues")

# apply the MAM-3 age model and save results
mam <- calc_MinDose(ExampleData.DeValues$CA1, sigmab = 0.2)

# create the HTML report
report_RLum(object = mam, file = "~/CA1_MAM.Rmd",
            timestamp = FALSE,
            title = "MAM-3 for sample CA1")

# when creating a report the input file is automatically saved to a
# .Rds file (see saveRDS()).
mam_report <- readRDS("~/CA1_MAM.Rds")
all.equal(mam, mam_report)


## Example: Temporary file & Viewer/Browser ----

# (a)
# Specifying a filename is not necessarily required. If no filename is provided,
# the report is rendered in a temporary file. If you use the RStudio IDE, the
# temporary report is shown in the interactive Viewer pane.
report_RLum(object = mam)

# (b)
# Additionally, you can view the HTML report in your system's default web browser.
report_RLum(object = mam, launch.browser = TRUE)


## Example: RLum.Analysis ----

data("ExampleData.RLum.Analysis")

# create the HTML report (note that specifying a file
# extension is not necessary)
report_RLum(object = IRSAR.RF.Data, file = "~/IRSAR_RF")
```

```
## Example: RLum.Data.Curve ----

data.curve <- get_RLum(IRSAR.RF.Data)[[1]]

# create the HTML report
report_RLum(object = data.curve, file = "~/Data_Curve")

## Example: Any other object ----
x <- list(x = 1:10,
          y = runif(10, -5, 5),
          z = data.frame(a = LETTERS[1:20], b = dnorm(0:9)),
          NA)

report_RLum(object = x, file = "~/arbitray_list")

## End(Not run)
```

Risoe.BINfileData-class
Class "Risoe.BINfileData"

Description

S4 class object for luminescence data in R. The object is produced as output of the function read_BIN2R.

Usage

```
## S4 method for signature 'Risoe.BINfileData'
show(object)

## S4 method for signature 'data.frame,list'
set_Risoe.BINfileData(METADATA, DATA, .RESERVED)

## S4 method for signature 'Risoe.BINfileData'
get_Risoe.BINfileData(object, ...)
```

Arguments

object	an object of class Risoe.BINfileData
METADATA	Object of class "data.frame" containing the meta information for each curve.
DATA	Object of class "list" containing numeric vector with count data.
.RESERVED	Object of class "list" containing list of undocumented raw values for internal use only.
...	other arguments that might be passed

Methods (by generic)

- show: Show structure of RLum and Risoe.BINfile class objects
- set_Risoe.BINfileData: The Risoe.BINfileData is normally produced as output of the function read_BIN2R. This construction method is intended for internal usage only.
- get_Risoe.BINfileData: Formal get-method for Risoe.BINfileData object. It does not allow accessing the object directly, it is just showing a terminal message.

Slots

METADATA Object of class "data.frame" containing the meta information for each curve.

DATA Object of class "list" containing numeric vector with count data.

.RESERVED Object of class "list" containing list of undocumented raw values for internal use only.

Objects from the Class

Objects can be created by calls of the form new("Risoe.BINfileData", ...).

Function version

0.3.0

Note

Internal METADATA - object structure

#	Name	Data Type	V	Description
[,1]	ID	numeric	RLum	Unique record ID (same ID as in slot DATA)
[,2]	SEL	logic	RLum	Record selection, not part official BIN-format, triggered by TAG
[,3]	VERSION	raw	03-08	BIN-file version number
[,4]	LENGTH	integer	03-08	Length of this record
[,5]	PREVIOUS	integer	03-08	Length of previous record
[,6]	NPOINTS	integer	03-08	Number of data points in the record
[,7]	RECTYPE	integer	08	Record type
[,8]	RUN	integer	03-08	Run number
[,9]	SET	integer	03-08	Set number
[,10]	POSITION	integer	03-08	Position number
[,11]	GRAIN	integer	03-04	Grain number
[,12]	GRAINNUMBER	integer	06-08	Grain number
[,13]	CURVENO	integer	06-08	Curve number
[,14]	XCOORD	integer	03-08	X position of a single grain
[,15]	YCOORD	integer	03-08	Y position of a single grain
[,16]	SAMPLE	factor	03-08	Sample name
[,17]	COMMENT	factor	03-08	Comment name
[,18]	SYSTEMID	integer	03-08	Risoe system id
[,19]	FNAME	factor	06-08	File name (*.bin/*.binx)
[,20]	USER	facotr	03-08	User name
[,21]	TIME	character	03-08	Data collection time (hh-mm-ss)
[,22]	DATE	factor	03-08	Data collection date (ddmmyy)
[,23]	DTYPE	character	03-08	Data type
[,24]	BL_TIME	numeric	03-08	Bleaching time
[,25]	BL_UNIT	integer	03-08	Bleaching unit (mJ, J, secs, mins, hrs)
[,26]	NORM1	numeric	03-08	Normalisation factor (1)

[,27]	NORM2	numeric	03-08	Normalisation factor (2)
[,28]	NORM3	numeric	03-08	Normalisation factor (3)
[,29]	BG	numeric	03-08	Background level
[,30]	SHIFT	integer	03-08	Number of channels to shift data
[,31]	TAG	integer	03-08	Tag, triggers SEL
[,32]	LTYPE	character	03-08	Luminescence type
[,33]	LIGHTSOURCE	character	03-08	Light source
[,34]	LPOWER	numeric	03-08	Optical stimulation power
[,35]	LIGHTPOWER	numeric	06-08	Optical stimulation power
[,36]	LOW	numeric	03-08	Low (temperature, time, wavelength)
[,37]	HIGH	numeric	03-08	High (temperature, time, wavelength)
[,38]	RATE	numeric	03-08	Rate (heating rate, scan rate)
[,39]	TEMPERATURE	integer	03-08	Sample temperature
[,40]	MEASTEMP	integer	06-08	Measured temperature
[,41]	AN_TEMP	numeric	03-08	Annealing temperature
[,42]	AN_TIME	numeric	03-08	Annealing time
[,43]	TOLDELAY	integer	03-08	TOL 'delay' channels
[,44]	TOLON	integer	03-08	TOL 'on' channels
[,45]	TOLOFF	integer	03-08	TOL 'off' channels
[,46]	IRR_TIME	numeric	03-08	Irradiation time
[,47]	IRR_TYPE	integer	03-08	Irradiation type (alpha, beta or gamma)
[,48]	IRR_UNIT	integer	03-04	Irradiation unit (Gy, Rads, secs, mins, hrs)
[,49]	IRR_DOSERATE	numeric	06-08	Irradiation dose rate (Gy/s)
[,50]	IRR_DOSERATEERR	numeric	06-08	Irradiation dose rate error (Gy/s)
[,51]	TIMESINCEIRR	integer	06-08	Time since irradiation (s)
[,52]	TIMETICK	numeric	06-08	Time tick for pulsing (s)
[,53]	ONTIME	integer	06-08	On-time for pulsing (in time ticks)
[,54]	STIMPERIOD	integer	06-08	Stimulation period (on+off in time ticks)
[,55]	GATE_ENABLED	raw	06-08	PMT signal gating enabled
[,56]	ENABLE_FLAGS	raw	06-08	PMT signal gating enabled
[,57]	GATE_START	integer	06-08	Start gating (in time ticks)
[,58]	GATE_STOP	ingeter	06-08	Stop gating (in time ticks), 'Gateend' for version 04, here only G
[,59]	PTENABLED	raw	06-08	Photon time enabled
[,60]	DTENABLED	raw	06-08	PMT dead time correction enabled
[,61]	DEADTIME	numeric	06-08	PMT dead time (s)
[,62]	MAXLPOWER	numeric	06-08	Stimulation power to 100 percent (mW/cm^2)
[,63]	XRF_ACQTIME	numeric	06-08	XRF acquisition time (s)
[,64]	XRF_HV	numeric	06-08	XRF X-ray high voltage (V)
[,65]	XRF_CURR	integer	06-08	XRF X-ray current (uA)
[,66]	XRF_DEADTIMEF	numeric	06-08	XRF dead time fraction
[,67]	SEQUENCE	character	03-04	Sequence name
[,68]	DETECTOR_ID	raw	07-08	Detector ID
[,69]	LOWERFILTER_ID	integer	07-08	Lower filter ID in reader
[,70]	UPPERFILTER_ID	integer	07-08	Uper filter ID in reader
[,71]	ENOISEFACTOR	numeric	07-08	Excess noise filter, usage unknown
[,72]	MARKPOS_X1	numeric	08	Coordinates marker position 1
[,73]	MARKPOS_Y1	numeric	08	Coordinates marker position 1
[,74]	MARKPOS_X2	numeric	08	Coordinates marker position 2
[,75]	MARKPOS_Y2	numeric	08	Coordinates marker position 2
[,76]	MARKPOS_X3	numeric	08	Coordinates marker position 3
[,77]	MARKPOS_Y3	numeric	08	Coordinates marker position 3
[,78]	MARKPOS_X4	numeric	08	Coordinates marker position 4

[,79]	MARKPOS_Y4	`numeric`	08	Coordinates marker position 4
[,80]	EXTR_START	`numeric`	08	usage unknown
[,81]	EXTR_END	`numeric`	08	usage unknown

V = BIN-file version (RLum means that it does not depend on a specific BIN version)

Note that the `Risoe.BINfileData` object combines all values from different versions from the BIN-file, reserved bits are skipped, however, the function `write_R2BIN` reset arbitrary reserved bits. Invalid values for a specific version are set to `NA`. Furthermore, the internal R data types do not necessarily match the required data types for the BIN-file data import! Data types are converted during data import.

LTYPE values

[,0]	TL	: Thermoluminescence
[,1]	OSL	: Optically stimulated luminescence
[,2]	IRSL	: Infrared stimulated luminescence
[,3]	M-IR	: Infrared monochromator scan
[,4]	M-VIS	: Visible monochromator scan
[,5]	TOL	: Thermo-optical luminescence
[,6]	TRPOSL	: Time Resolved Pulsed OSL
[,7]	RIR	: Ramped IRSL
[,8]	RBR	: Ramped (Blue) LEDs
[,9]	USER	: User defined
[,10]	POSL	: Pulsed OSL
[,11]	SGOSL	: Single Grain OSL
[,12]	RL	: Radio Luminescence
[,13]	XRF	: X-ray Fluorescence

DTYPE values

[,0]	0	Natural
[,1]	1	N+dose
[,2]	2	Bleach
[,3]	3	Bleach+dose
[,4]	4	Natural (Bleach)
[,5]	5	N+dose (Bleach)
[,6]	6	Dose
[,7]	7	Background

LIGHTSOURCE values

[,0]	0	Non
[,1]	1	Lamp
[,2]	2	IR diodes/IR Laser
[,3]	3	Calibration LED
[,4]	4	Blue Diodes
[,5]	5	White lite
[,6]	6	Green laser (single grain)
[,7]	7	IR laser (single grain)

(information on the BIN/BINX file format are kindly provided by Risoe, DTU Nutech)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

Risoe DTU, 2013. The Sequence Editor User Manual - Feb 2013 and Risoe DTU, 2016. The Sequence Editor User Manual - Feburar 2016

http://www.nutech.dtu.dk/

See Also

plot_Risoe.BINfileData, read_BIN2R, write_R2BIN, merge_Risoe.BINfileData, Risoe.BINfileData2RLum.Anal

Examples

```
showClass("Risoe.BINfileData")
```

Risoe.BINfileData2RLum.Analysis

Convert Risoe.BINfileData object to an RLum.Analysis object

Description

Converts values from one specific position of a Risoe.BINfileData S4-class object to an RLum.Analysis object.

Usage

```
Risoe.BINfileData2RLum.Analysis(object, pos = NULL, grain = NULL,
  run = NULL, set = NULL, ltype = NULL, dtype = NULL,
  protocol = "unknown", txtProgressBar = FALSE)
```

Arguments

object	Risoe.BINfileData (**required**): Risoe.BINfileData object
pos	numeric (optional): position number of the Risoe.BINfileData object for which the curves are stored in the RLum.Analysis object. If length(position)>1 a list of RLum.Analysis objects is returned. If nothing is provided every position will be converted. If the position is not valid NA is returned.
grain	vector, numeric (optional): grain number from the measurement to limit the converted data set (e.g., grain = c(1:48)). Please be aware that this option may lead to unwanted effects, as the output is strictly limited to the choosen grain number for all position numbers
run	vector, numeric (optional): run number from the measurement to limit the converted data set (e.g., run = c(1:48)).

`set`	`vector`, `numeric` (optional): set number from the measurement to limit the converted data set (e.g., `set = c(1:48)`).
`ltype`	`vector`, `character` (optional): curve type to limit the converted data. Commonly allowed values are: `IRSL`, `OSL`, `TL`, `RIR`, `RBR` and `USER` (see also `Risoe.BINfileData`)
`dtype`	`vector`, `character` (optional): data type to limit the converted data. Commonly allowed values are listed in `Risoe.BINfileData`
`protocol`	`character` (optional): sets protocol type for analysis object. Value may be used by subsequent analysis functions.
`txtProgressBar`	`logical` (with default): enables or disables `txtProgressBar`.

Details

The `RLum.Analysis` object requires a set of curves for specific further protocol analyses. However, the `Risoe.BINfileData` usually contains a set of curves for different aliquots and different protocol types that may be mixed up. Therefore, a conversion is needed.

Value

Returns an `RLum.Analysis` object.

Function version

0.4.1 (2016-09-09 10:32:17)

Note

The `protocol` argument of the `RLum.Analysis` object is set to 'unknown' if not stated otherwise.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

#

See Also

`Risoe.BINfileData`, `RLum.Analysis`, `read_BIN2R`

Examples

```
##load data
data(ExampleData.BINfileData, envir = environment())

##convert values for position 1
Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos = 1)
```

RLum-class *Class* "RLum"

Description

Abstract class for data in the package Luminescence

Usage

```
## S4 method for signature 'RLum'
replicate_RLum(object, times = NULL)
```

Arguments

object	an object of class RLum (**required**)
times	integer (optional): number for times each element is repeated element

Methods (by generic)

- replicate_RLum: Replication method RLum-objects

Slots

originator Object of class character containing the name of the producing function for the object. Set automatically by using the function set_RLum.

info Object of class list for additional information on the object itself

.uid Object of class character for a unique object identifier. This id is usually calculated using the internal function .create_UID() if the funtion set_RLum is called.

.pid Object of class character for a parent id. This allows nesting RLum-objects at will. The parent id can be the uid of another object.

Objects from the Class

A virtual Class: No objects can be created from it.

Class version

0.4.0

Note

RLum is a virtual class.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

RLum.Data, RLum.Analysis

Examples

```
showClass("RLum")
```

RLum.Analysis-class *Class* "RLum.Analysis"

Description

Object class to represent analysis data for protocol analysis, i.e. all curves, spectra etc. from one measurements. Objects from this class are produced, by e.g. read_XSYG2R, read_Daybreak2R

Usage

```
## S4 method for signature 'RLum.Analysis'
show(object)

## S4 method for signature 'RLum.Analysis'
set_RLum(class, originator, .uid, .pid,
  protocol = NA_character_, records = list(), info = list())

## S4 method for signature 'RLum.Analysis'
get_RLum(object, record.id = NULL,
  recordType = NULL, curveType = NULL, RLum.type = NULL,
  protocol = "UNKNOWN", get.index = NULL, drop = TRUE, recursive = TRUE,
  info.object = NULL)

## S4 method for signature 'RLum.Analysis'
structure_RLum(object, fullExtent = FALSE)

## S4 method for signature 'RLum.Analysis'
length_RLum(object)

## S4 method for signature 'RLum.Analysis'
names_RLum(object)
```

Arguments

object	[show_RLum][get_RLum][names_RLum][length_RLum] [structure_RLum]] an object of class RLum.Analysis (**required**)
class	[set_RLum] character (**required**): name of the RLum class to be created
originator	[set_RLum] character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	[set_RLum] character (automatic): sets an unique ID for this object using the internal C++ function .create_UID.
.pid	[set_RLum] character (with default): option to provide a parent id for nesting at will.
protocol	[set_RLum] character (optional): sets protocol type for analysis object. Value may be used by subsequent analysis functions.

records	[set_RLum] list (**required**): list of RLum.Analysis objects
info	[set_RLum] list (optional): a list containing additional info data for the object

set_RLum:

Returns an RLum.Analysis object.

record.id	[get_RLum] numeric or logical (optional): IDs of specific records. If of type logical the entire id range is assuemd and TRUE and FALSE indicates the selection.
recordType	[get_RLum] character (optional): record type (e.g., "OSL"). Can be also a vector, for multiple matching, e.g., recordType = c("OSL", "IRSL")
curveType	[get_RLum] character (optional): curve type (e.g. "predefined" or "measured")
RLum.type	[get_RLum] character (optional): RLum object type. Defaults to "RLum.Data.Curve" and "RLum.Data.Spectrum".
get.index	[get_RLum] logical (optional): return a numeric vector with the index of each element in the RLum.Analysis object.
drop	[get_RLum] logical (with default): coerce to the next possible layer (which are RLum.Data-objects), drop = FALSE keeps the original RLum.Analysis
recursive	[get_RLum] logical (with default): if TRUE (the default) and the result of the 'get_RLum' request is a single object this object will be unlisted, means only the object itself and no list containing exactly one object is returned. Mostly this makes things easier, however, if this method is used within a loop this might undesired.
info.object	[get_RLum] character (optional): name of the wanted info element
fullExtent	[structure_RLum] logical (with default): extents the returned data.frame to its full extent, i.e. all info elements are part of the return as well. The default valule is FALSE as the data frame might become rather big.

Value

get_RLum:

Returns:
(1) list of RLum.Data objects or
(2) Single RLum.Data object, if only one object is contained and recursive = FALSE or
(3) RLum.Analysis ojects for drop = FALSE

structure_RLum:

Returns data.frame showing the structure.

length_RLum

Returns the number records in this object.

names_RLum

Returns the names of the record types (recordType) in this object.

Methods (by generic)

- show: Show structure of RLum.Analysis object
- set_RLum: Construction method for RLum.Analysis objects.
- get_RLum: Accessor method for RLum.Analysis object.
  The slots record.id, recordType, curveType and RLum.type are optional to allow for records limited by their id (list index number), their record type (e.g. recordType = "OSL") or object type.
  Example: curve type (e.g. curveType = "predefined" or curveType ="measured")
  The selection of a specific RLum.type object superimposes the default selection. Currently supported objects are: RLum.Data.Curve and RLum.Data.Spectrum
- structure_RLum: Method to show the structure of an RLum.Analysis object.
- length_RLum: Returns the length of the object, i.e., number of stored records.
- names_RLum: Returns the names of the RLum.Data objects objects (same as shown with the show method)

Slots

protocol Object of class character describing the applied measurement protocol

records Object of class list containing objects of class RLum.Data

Objects from the Class

Objects can be created by calls of the form set_RLum("RLum.Analysis", ...).

Class version

0.4.6

Note

The method structure_RLum is currently just avaiblable for objects containing RLum.Data.Curve.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

Risoe.BINfileData2RLum.Analysis, Risoe.BINfileData, RLum

Examples

```
showClass("RLum.Analysis")

##set empty object
set_RLum(class = "RLum.Analysis")

###use example data
##load data
data(ExampleData.RLum.Analysis, envir = environment())
```

```
##show curves in object
get_RLum(IRSAR.RF.Data)

##show only the first object, but by keeping the object
get_RLum(IRSAR.RF.Data, record.id = 1, drop = FALSE)
```

RLum.Data-class *Class* "RLum.Data"

Description

Generalized virtual data class for luminescence data.

Objects from the Class

A virtual Class: No objects can be created from it.

Class version

0.2.1

Note

Just a virtual class.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

RLum, RLum.Data.Curve, RLum.Data.Spectrum

Examples

```
showClass("RLum.Data")
```

RLum.Data.Curve-class *Class* "RLum.Data.Curve"

Description

Class for representing luminescence curve data.

Usage

```
## S4 method for signature 'RLum.Data.Curve'
show(object)

## S4 method for signature 'RLum.Data.Curve'
set_RLum(class, originator, .uid, .pid,
  recordType = NA_character_, curveType = NA_character_, data = matrix(0,
  ncol = 2), info = list())

## S4 method for signature 'RLum.Data.Curve'
get_RLum(object, info.object = NULL)

## S4 method for signature 'RLum.Data.Curve'
length_RLum(object)

## S4 method for signature 'RLum.Data.Curve'
names_RLum(object)

## S4 method for signature 'RLum.Data.Curve'
bin_RLum.Data(object, bin_size = 2)
```

Arguments

object	[show_RLum][get_RLum][length_RLum][names_RLum] an object of class RLum.Data.Curve (**required**)
class	[set_RLum] character (**required**): name of the RLum class to create
originator	[set_RLum] character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	[set_RLum] character (automatic): sets an unique ID for this object using the internal C++ function .create_UID.
.pid	[set_RLum] character (with default): option to provide a parent id for nesting at will.
recordType	[set_RLum] character (optional): record type (e.g., "OSL")
curveType	[set_RLum] character (optional): curve type (e.g., "predefined" or "measured")
data	[set_RLum] matrix (**required**): raw curve data. If data itself is a RLum.Data.Curve-object this can be used to re-construct the object (s. Details)
info	[set_RLum] list (optional): info elements
info.object	[get_RLum] character (optional): name of the wanted info element
bin_size	[bin_RLum] integer (with default): set number of channels used for each bin, e.g. bin_size = 2 means that two channels are binned.

Value

`set_RLum`

Returns an `RLum.Data.Curve` object.

`get_RLum`

(1) A `matrix` with the curve values or
(2) only the info object if `info.object` was set.

`length_RLum`

Number of channels in the curve (row number of the matrix)

`names_RLum`

Names of the info elements (slot `info`)

`bin_RLum.Data`

Same object as input, after applying the binning.

Methods (by generic)

- `show`: Show structure of `RLum.Data.Curve` object
- `set_RLum`: Construction method for RLum.Data.Curve object. The slot info is optional and predefined as empty list by default.
- `get_RLum`: Accessor method for RLum.Data.Curve object. The argument info.object is optional to directly access the info elements. If no info element name is provided, the raw curve data (matrix) will be returned.
- `length_RLum`: Returns the length of the curve object, which is the maximum of the value time/temperature of the curve (corresponding to the stimulation length)
- `names_RLum`: Returns the names info elements coming along with this curve object
- `bin_RLum.Data`: Allows binning of specific objects

Slots

`recordType` Object of class "character" containing the type of the curve (e.g. "TL" or "OSL")

`curveType` Object of class "character" containing curve type, allowed values are measured or predefined

`data` Object of class `matrix` containing curve x and y data. 'data' can also be of type `RLum.Data.Curve` to change object values without deconstructing the object. For example: `set_RLum(class = 'RLum.Data.Curve',` would just change the recordType. Missing arguements the value is taken from the input object in 'data' (which is already an RLum.Data.Curve object in this example)

Create objects from this Class

Objects can be created by calls of the form `set_RLum(class = "RLum.Data.Curve", ...)`.

Class version

0.4.1

Note

The class should only contain data for a single curve. For additional elements the slot `info` can be used (e.g. providing additional heating ramp curve). Objects from the class `RLum.Data.Curve` are produced by other functions (partyl within `RLum.Analysis` objects), namely: `Risoe.BINfileData2RLum.Analysis`, `read_XSYG2R`

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

`RLum`, `RLum.Data`, `plot_RLum`, `merge_RLum`

Examples

```
showClass("RLum.Data.Curve")

##set empty curve object
set_RLum(class = "RLum.Data.Curve")
```

RLum.Data.Image-class *Class* "RLum.Data.Image"

Description

Class for representing luminescence image data (TL/OSL/RF). Such data are for example produced by the function `read_SPE2R`

Usage

```
## S4 method for signature 'RLum.Data.Image'
show(object)

## S4 method for signature 'RLum.Data.Image'
set_RLum(class, originator, .uid, .pid,
  recordType = "Image", curveType = NA_character_,
  data = raster::brick(raster::raster(matrix())), info = list())

## S4 method for signature 'RLum.Data.Image'
get_RLum(object, info.object)

## S4 method for signature 'RLum.Data.Image'
names_RLum(object)
```

Arguments

object	[show_RLum][get_RLum][names_RLum] an object of class RLum.Data.Image
class	[set_RLum]character: name of the RLum class to create
originator	[set_RLum] character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	[set_RLum] character (automatic): sets an unique ID for this object using the internal C++ function .create_UID.
.pid	[set_RLum] character (with default): option to provide a parent id for nesting at will.
recordType	[set_RLum] character: record type (e.g. "OSL")
curveType	[set_RLum] character: curve type (e.g. "predefined" or "measured")
data	[set_RLum] matrix: raw curve data. If data is of type RLum.Data.Image this can be used to re-construct the object.
info	[set_RLum] list: info elements
info.object	[get_RLum] character name of the info object to returned

Value

set_RLum

Returns an object from class RLum.Data.Image

get_RLum

(1) Returns the data object (brick)
(2) only the info object if info.object was set.

names_RLum

Returns the names of the info elements

Methods (by generic)

- show: Show structure of RLum.Data.Image object
- set_RLum: Construction method for RLum.Data.Image object. The slot info is optional and predefined as empty list by default..
- get_RLum: Accessor method for RLum.Data.Image object. The argument info.object is optional to directly access the info elements. If no info element name is provided, the raw image data (RasterBrick) will be returned.
- names_RLum: Returns the names info elements coming along with this curve object

Slots

recordType Object of class character containing the type of the curve (e.g. "OSL image", "TL image")

curveType Object of class character containing curve type, allowed values are measured or predefined

data Object of class brick containing images (raster data).

info Object of class list containing further meta information objects

Objects from the Class

Objects can be created by calls of the form `set_RLum("RLum.Data.Image", ...)`.

Class version

0.4.0

Note

The class should only contain data for a set of images. For additional elements the slot `info` can be used.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

`RLum`, `RLum.Data`, `plot_RLum`, `read_SPE2R`

Examples

```
showClass("RLum.Data.Image")

##create empty RLum.Data.Image object
set_RLum(class = "RLum.Data.Image")
```

RLum.Data.Spectrum-class

Class "RLum.Data.Spectrum"

Description

Class for representing luminescence spectra data (TL/OSL/RF).

Usage

```
## S4 method for signature 'RLum.Data.Spectrum'
show(object)

## S4 method for signature 'RLum.Data.Spectrum'
set_RLum(class, originator, .uid, .pid,
  recordType = "Spectrum", curveType = NA_character_, data = matrix(),
  info = list())

## S4 method for signature 'RLum.Data.Spectrum'
get_RLum(object, info.object)

## S4 method for signature 'RLum.Data.Spectrum'
names_RLum(object)
```

Arguments

object	[show_RLum][get_RLum][names_RLum] an object of class RLum.Data.Spectrum
class	[set_RLum] character (automatic): name of the RLum class to create.
originator	character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	[set_RLum] character (automatic): sets an unique ID for this object using the internal C++ function .create_UID.
.pid	[set_RLum] character (with default): option to provide a parent id for nesting at will.
recordType	[set_RLum] character: record type (e.g. "OSL")
curveType	[set_RLum] character: curve type (e.g. "predefined" or "measured")
data	[set_RLum] matrix: raw curve data. If data is of type RLum.Data.Spectrum, this can be used to re-construct the object.
info	[set_RLum] list: info elements
info.object	[get_RLum] character (optional): the name of the info object to be called

Value

[set_RLum]

An object from the class RLum.Data.Spectrum

get_RLum

(1) A matrix with the spectrum values or
(2) only the info object if info.object was set.

names_RLum

The names of the info objects

Methods (by generic)

- show: Show structure of RLum.Data.Spectrum object
- set_RLum: Construction method for RLum.Data.Spectrum object. The slot info is optional and predefined as empty list by default
- get_RLum: Accessor method for RLum.Data.Spectrum object. The argument info.object is optional to directly access the info elements. If no info element name is provided, the raw curve data (matrix) will be returned
- names_RLum: Returns the names info elements coming along with this curve object

Slots

recordType Object of class character containing the type of the curve (e.g. "TL" or "OSL")

curveType Object of class character containing curve type, allowed values are measured or predefined

data Object of class matrix containing spectrum (count) values. Row labels indicate wavelength/pixel values, column labels are temperature or time values.

info Object of class list containing further meta information objects

Objects from the Class

Objects can be created by calls of the form `set_RLum("RLum.Data.Spectrum", ...)`.

Class version

0.4.0

Note

The class should only contain data for a single spectra data set. For additional elements the slot `info` can be used. Objects from this class are automatically created by, e.g., `read_XSYG2R`

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

`RLum`, `RLum.Data`, `plot_RLum`

Examples

```
showClass("RLum.Data.Spectrum")

##show example data
data(ExampleData.XSYG, envir = environment())
TL.Spectrum

##show data matrix
get_RLum(TL.Spectrum)

##plot spectrum
## Not run:
plot_RLum(TL.Spectrum)

## End(Not run)
```

RLum.Results-class *Class* "RLum.Results"

Description

Object class contains results data from functions (e.g., `analyse_SAR.CWOSL`).

Usage

```
## S4 method for signature 'RLum.Results'
show(object)

## S4 method for signature 'RLum.Results'
set_RLum(class, originator, .uid, .pid,
  data = list(), info = list())
```

```
## S4 method for signature 'RLum.Results'
get_RLum(object, data.object, info.object = NULL,
  drop = TRUE)

## S4 method for signature 'RLum.Results'
length_RLum(object)

## S4 method for signature 'RLum.Results'
names_RLum(object)
```

Arguments

object	[get_RLum] RLum.Results (required): an object of class RLum.Results to be evaluated
class	[set_RLum] character **(required)**: name of the RLum class to create
originator	[set_RLum] character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	[set_RLum] character (automatic): sets an unique ID for this object using the internal C++ function .create_UID.
.pid	[set_RLum] character (with default): option to provide a parent id for nesting at will.
data	[set_RLum] list (optional): a list containing the data to be stored in the object
info	[set_RLum] list (optional): a list containing additional info data for the object
data.object	[get_RLum] character or numeric: name or index of the data slot to be returned
info.object	[get_RLum] character (optional): name of the wanted info element
drop	[get_RLum] logical (with default): coerce to the next possible layer (which are data objects, drop = FALSE keeps the original RLum.Results

Value

set_RLum:

Returns an object from the class RLum.Results

get_RLum:

Returns:
(1) Data object from the specified slot
(2) list of data objects from the slots if 'data.object' is vector or
(3) an RLum.Results for drop = FALSE.

length_RLum

Returns the number of data elements in the RLum.Results object.

names_RLum

Returns the names of the data elements in the object.

Methods (by generic)

- show: Show structure of RLum.Results object
- set_RLum: Construction method for an RLum.Results object.
- get_RLum: Accessor method for RLum.Results object. The argument data.object allows directly accessing objects delivered within the slot data. The default return object depends on the object originator (e.g., fit_LMCurve). If nothing is specified always the first data.object will be returned.

  Note: Detailed specification should be made in combination with the originator slot in the receiving function if results are pipped.
- length_RLum: Returns the length of the object, i.e., number of stored data.objects
- names_RLum: Returns the names data.objects

Slots

data Object of class "list" containing output data

Objects from the Class

Objects can be created by calls of the form new("RLum.Results", ...).

Class version

0.5.1

Note

The class is intended to store results from functions to be used by other functions. The data in the object should always be accessed by the method get_RLum.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)

See Also

RLum, plot_RLum, merge_RLum

Examples

```
showClass("RLum.Results")

##create an empty object from this class
set_RLum(class = "RLum.Results")

##use another function to show how it works

##Basic calculation of the dose rate for a specific date
 dose.rate <-  calc_SourceDoseRate(
   measurement.date = "2012-01-27",
   calib.date = "2014-12-19",
   calib.dose.rate = 0.0438,
   calib.error = 0.0019)
```

```
##show object
dose.rate

##get results
get_RLum(dose.rate)

##get parameters used for the calcualtion from the same object
get_RLum(dose.rate, data.object = "parameters")

##alternatively objects can be accessed using S3 generics, such as
dose.rate$parameters
```

Second2Gray *Converting equivalent dose values from seconds (s) to gray (Gy)*

Description

Conversion of absorbed radiation dose in seconds (s) to the SI unit gray (Gy) including error propagation. Normally used for equivalent dose data.

Usage

```
Second2Gray(data, dose.rate, error.propagation = "omit")
```

Arguments

`data`	`data.frame` (**required**): input values, structure: data (`values[,1]`) and data error (`values [,2]`) are required
`dose.rate`	`RLum.Results` or `data.frame` or `numeric` (**required**): `RLum.Results` needs to be orginated from the function `calc_SourceDoseRate`, for `vector` dose rate in Gy/s and dose rate error in Gy/s
`error.propagation`	`character` (with default): error propagation method used for error calculation (`omit`, `gaussian` or `absolute`), see details for further information

Details

Calculation of De values from seconds (s) to gray (Gy)

$$De[Gy] = De[s] * DoseRate[Gy/s])$$

Provided calculation error propagation methods for error calculation (with 'se' as the standard error and 'DR' of the dose rate of the beta-source):

(1) `omit` (default)

$$se(De)[Gy] = se(De)[s] * DR[Gy/s]$$

In this case the standard error of the dose rate of the beta-source is treated as systematic (i.e. non-random), it error propagation is omitted. However, the error must be considered during calculation of the final age. (cf. Aitken, 1985, pp. 242). This approach can be seen as method (2) (gaussian) for the case the (random) standard error of the beta-source calibration is 0. Which particular method is requested depends on the situation and cannot be prescriptive.

(2) `gaussian` error propagation

$$se(De)[Gy] = \sqrt{((DR[Gy/s] * se(De)[s])^2 + (De[s] * se(DR)[Gy/s])^2)}$$

Applicable under the assumption that errors of De and se are uncorrelated.

(3) `absolute` error propagation

$$se(De)[Gy] = abs(DR[Gy/s] * se(De)[s]) + abs(De[s] * se(DR)[Gy/s])$$

Applicable under the assumption that errors of De and se are not uncorrelated.

Value

Returns a data.frame with converted values.

Function version

0.6.0 (2015-11-29 17:27:48)

Note

If no or a wrong error propagation method is given, the execution of the function is stopped. Furthermore, if a `data.frame` is provided for the dose rate values is has to be of the same length as the data frame provided with the argument `data`

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France),
Michael Dietze, GFZ Potsdam (Germany),
Margret C. Fuchs, HZDR, Helmholtz-Institute Freiberg for Resource Technology (Germany)
R Luminescence Package Team

References

Aitken, M.J., 1985. Thermoluminescence dating. Academic Press.

See Also

`calc_SourceDoseRate`

Examples

```
##(A) for known source dose rate at date of measurement
## - load De data from the example data help file
data(ExampleData.DeValues, envir = environment())
## - convert De(s) to De(Gy)
Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))




##(B) for source dose rate calibration data
## - calculate source dose rate first
dose.rate <-  calc_SourceDoseRate(measurement.date = "2012-01-27",
                                  calib.date = "2014-12-19",
                                  calib.dose.rate = 0.0438,
                                  calib.error = 0.0019)
# read example data
data(ExampleData.DeValues, envir = environment())

# apply dose.rate to convert De(s) to De(Gy)
Second2Gray(ExampleData.DeValues$BT998, dose.rate)
```

---

set_Risoe.BINfileData *General accessor function for RLum S4 class objects*

Description

Function calls object-specific get functions for RisoeBINfileData S4 class objects.

Usage

```
set_Risoe.BINfileData(METADATA, DATA, .RESERVED)
```

Arguments

METADATA	x
DATA	x
.RESERVED	x

Details

The function provides a generalised access point for specific Risoe.BINfileData objects. Depending on the input object, the corresponding get function will be selected. Allowed arguments can be found in the documentations of the corresponding Risoe.BINfileData class.

Value

Return is the same as input objects as provided in the list.

Function version

0.1 (2015-11-29 17:27:48)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

Risoe.BINfileData

---

set_RLum *General set function for RLum S4 class objects*

Description

Function calls object-specific set functions for RLum S4 class objects.

Usage

```
set_RLum(class, originator, .uid = .create_UID(), .pid = NA_character_, ...)
```

Arguments

class	RLum (**required**): name of the S4 class to create
originator	character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	character (automatic): sets an unique ID for this object using the internal C++ function .create_UID.
.pid	character (with default): option to provide a parent id for nesting at will.
...	further arguments that one might want to pass to the specific set method

Details

The function provides a generalised access point for specific RLum objects.
Depending on the given class, the corresponding method to create an object from this class will be selected. Allowed additional arguments can be found in the documentations of the corresponding RLum class: RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis and RLum.Results

Value

Returns an object of the specified class.

Function version

0.3.0 (2016-05-02 09:43:47)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##produce empty objects from each class
set_RLum(class = "RLum.Data.Curve")
set_RLum(class = "RLum.Data.Spectrum")
set_RLum(class = "RLum.Data.Spectrum")
set_RLum(class = "RLum.Analysis")
set_RLum(class = "RLum.Results")

##produce a curve object with arbitrary curve values
object <- set_RLum(
class = "RLum.Data.Curve",
curveType = "arbitrary",
recordType = "OSL",
data = matrix(c(1:100,exp(-c(1:100))),ncol = 2))

##plot this curve object
plot_RLum(object)
```

---

sTeve *sTeve - sophisticated tool for efficient data validation and evaluation*

Description

This function provides a sophisticated routine for comprehensive luminescence dating data analysis.

Usage

```
sTeve(n_frames = 10, t_animation = 2, n.tree = 7, type)
```

Arguments

n_frames	integer (with default): n frames
t_animation	integer (with default): t animation
n.tree	integer (with default): How many trees do you want to cut?
type	integer (optional): Make a decision: 1, 2 or 3

Details

This amazing sophisticated function validates your data seriously.

Value

Validates your data.

Note

This function should not be taken too seriously.

Author(s)

R Luminescence Team, 2012-2013

References

#

See Also

plot_KDE

Examples

```
##no example available
```

---

structure_RLum *General structure function for RLum S4 class objects*

Description

Function calls object-specific get functions for RLum S4 class objects.

Usage

```
structure_RLum(object, ...)
```

Arguments

object	RLum (**required**): S4 object of class RLum
...	further arguments that one might want to pass to the specific structure method

Details

The function provides a generalised access point for specific RLum objects.
Depending on the input object, the corresponding structure function will be selected. Allowed arguments can be found in the documentations of the corresponding RLum class.

Value

Returns a data.frame with structure of the object.

Function version

0.2.0 (2016-05-02 09:36:06)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##load example data
data(ExampleData.XSYG, envir = environment())

##show structure
structure_RLum(OSL.SARMeasurement$Sequence.Object)
```

---

template_DRAC *Create a DRAC input data template (v1.1)*

Description

This function returns a DRAC input template (v1.1) to be used in conjunction with the use_DRAC() function

Usage

```
template_DRAC(nrow = 1, notification = TRUE)
```

Arguments

nrow	integer (with default): specifies the number of rows of the template (i.e., the number of data sets you want to submit)
notification	logical (with default): show or hide the notification

Value

A list.

Author(s)

Christoph Burow, University of Cologne (Germany)

References

Durcan, J.A., King, G.E., Duller, G.A.T., 2015. DRAC: Dose Rate and Age Calculator for trapped charge dating. Quaternary Geochronology 28, 54-61. doi:10.1016/j.quageo.2015.03.012

See Also

as.data.frame list

Examples

```
# create a new DRAC input input
input <- template_DRAC()

# show content of the input
print(input)
print(input$`Project ID`)
print(input[[4]])


## Example: DRAC Quartz example
# note that you only have to assign new values where they
# are different to the default values
input$`Project ID` <- "DRAC-Example"
input$`Sample ID` <- "Quartz"
input$`Conversion factors` <- "AdamiecAitken1998"
input$`ExternalU (ppm)` <- 3.4
input$`errExternal U (ppm)` <- 0.51
input$`External Th (ppm)` <- 14.47
input$`errExternal Th (ppm)` <- 1.69
input$`External K (%)` <- 1.2
input$`errExternal K (%)` <- 0.14
input$`Calculate external Rb from K conc?` <- "N"
input$`Calculate internal Rb from K conc?` <- "N"
input$`Scale gammadoserate at shallow depths?` <- "N"
input$`Grain size min (microns)` <- 90
input$`Grain size max (microns)` <- 125
input$`Water content ((wet weight - dry weight)/dry weight) %` <- 5
input$`errWater content %` <- 2
input$`Depth (m)` <- 2.2
input$`errDepth (m)` <- 0.22
input$`Overburden density (g cm-3)` <- 1.8
input$`errOverburden density (g cm-3)` <- 0.1
input$`Latitude (decimal degrees)` <- 30.0000
input$`Longitude (decimal degrees)` <- 70.0000
input$`Altitude (m)` <- 150
input$`De (Gy)` <- 20
input$`errDe (Gy)` <- 0.2

# use DRAC
## Not run:
output <- use_DRAC(input)

## End(Not run)
```

---

tune_Data *Tune data for experimental purpose*

Description

The error can be reduced and sample size increased for specific purpose.

Usage

```
tune_Data(data, decrease.error = 0, increase.data = 0)
```

Arguments

data	data.frame (**required**): input values, structure: data (values[,1]) and data error (values [,2]) are required
decrease.error	numeric: factor by which the error is decreased, ranges between 0 and 1.
increase.data	numeric: factor by which the error is decreased, ranges between 0 and inf.

Value

Returns a data.frame with tuned values.

Function version

0.5.0 (2015-11-29 17:27:48)

Note

You should not use this function to improve your poor data set!

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
R Luminescence Package Team

References

#

See Also

#

Examples

```
## load example data set
data(ExampleData.DeValues, envir = environment())
x <- ExampleData.DeValues$CA1

## plot original data
plot_AbanicoPlot(data = x,
                 summary = c("n", "mean"))

## decrease error by 10 %
plot_AbanicoPlot(data = tune_Data(x, decrease.error = 0.1),
                 summary = c("n", "mean"))

## increase sample size by 200 %
```

```
#plot_AbanicoPlot(data = tune_Data(x, increase.data = 2) ,
#                 summary = c("n", "mean"))
```

---

use_DRAC *Use DRAC to calculate dose rate data*

Description

The function provides an interface from R to DRAC. An R-object or a pre-formatted XLS/XLSX file is passed to the DRAC website and the results are re-imported into R.

Usage

```
use_DRAC(file, name, ...)
```

Arguments

file	character: spreadsheet to be passed to the DRAC website for calculation. Can also be a DRAC template object obtained from template_DRAC().
name	character: Optional user name submitted to DRAC. If omitted, a random name will be generated
...	Further arguments.

Value

Returns an RLum.Results object containing the following elements:

DRAC — list: a named list containing the following elements in slot @data:

$highlights	data.frame	summary of 25 most important input/output fields
$header	character	HTTP header from the DRAC server response
$labels	data.frame	descriptive headers of all input/output fields
$content	data.frame	complete DRAC input/output table
$input	data.frame	DRAC input table
$output	data.frame	DRAC output table

data	character or list path to the input spreadsheet or a DRAC template
call	call the function call
args	list used arguments

The output should be accessed using the function get_RLum.

Function version

0.1.0 (2015-12-05 15:52:49)

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Michael Dietze, GFZ Potsdam (Germany), Christoph Burow, University of Cologne (Germany)

R Luminescence Package Team

References

Durcan, J.A., King, G.E., Duller, G.A.T., 2015. DRAC: Dose Rate and Age Calculator for trapped charge dating. Quaternary Geochronology 28, 54-61. doi:10.1016/j.quageo.2015.03.012

Examples

```
## (1) Method using the DRAC spreadsheet

file <-  "/PATH/TO/DRAC_Input_and_Output_Template.xlsx"

# send the actual IO template spreadsheet to DRAC
## Not run:
use_DRAC(file = file)

## End(Not run)



## (2) Method using an R template object

# Create a template
input <- template_DRAC()

# Fill the template with values
input$`Project ID` <- "DRAC-Example"
input$`Sample ID` <- "Quartz"
input$`Conversion factors` <- "AdamiecAitken1998"
input$`ExternalU (ppm)` <- 3.4
input$`errExternal U (ppm)` <- 0.51
input$`External Th (ppm)` <- 14.47
input$`errExternal Th (ppm)` <- 1.69
input$`External K (%)` <- 1.2
input$`errExternal K (%)` <- 0.14
input$`Calculate external Rb from K conc?` <- "N"
input$`Calculate internal Rb from K conc?` <- "N"
input$`Scale gammadoserate at shallow depths?` <- "N"
input$`Grain size min (microns)` <- 90
input$`Grain size max (microns)` <- 125
input$`Water content ((wet weight - dry weight)/dry weight) %` <- 5
input$`errWater content %` <- 2
input$`Depth (m)` <- 2.2
input$`errDepth (m)` <- 0.22
input$`Overburden density (g cm-3)` <- 1.8
input$`errOverburden density (g cm-3)` <- 0.1
input$`Latitude (decimal degrees)` <- 30.0000
input$`Longitude (decimal degrees)` <- 70.0000
input$`Altitude (m)` <- 150
input$`De (Gy)` <- 20
```

```
input$`errDe (Gy)` <- 0.2

# use DRAC
## Not run:
output <- use_DRAC(input)

## End(Not run)
```

`verify_SingleGrainData` *Verify single grain data sets and check for invalid grains, i.e. zero light level grains*

Description

This function tries to identify automatically zero light level curves (grains) from single grain data measurements.

Usage

```
verify_SingleGrainData(object, threshold = 10, cleanup = FALSE,
  cleanup_level = "aliquot", verbose = TRUE, plot = FALSE)
```

Arguments

`object`	`Risoe.BINfileData` or `RLum.Analysis` (**required**): input object. The function also accepts a list with objects of allowed type.
`threshold`	`numeric` (with default): numeric threshold value for the allowed difference between the `mean` and the `var` of the count values (see details)
`cleanup`	`logical` (with default): if set to `TRUE` curves indentified as zero light level curves are automatically removed. Ouput is an object as same type as the input, i.e. either `Risoe.BINfileData` or `RLum.Analysis`
`cleanup_level`	`character` (with default): selects the level for the cleanup of the input data sets. Two options are allowed: `"curve"` or `"aliquot"`. If `"curve"` is selected every single curve marked as `invalid` is removed. If `"aliquot"` is selected, curves of one aliquot (grain or disc) can be marked as invalid, but will not be removed. An aliquot will be only removed if all curves of this aliquot are marked as invalid.
`verbose`	`logical` (with default): enables or disables terminal feedback
`plot`	`logical` (with default): enables or disables graphical feedback

Details

How the method works?

The function compares the expected values ($E(X)$) and the variance ($Var(X)$) of the count values for each curve. Assuming that the background roughly follows a poisson distribution the absolute difference of both values should be zero or at least around zero as

$$E(x) = Var(x) = \lambda$$

Thus the function checks for:

$$abs(E(x) - Var(x)) >= \Theta$$

With Θ an arbitray, user defined, threshold. Values above indicating curves comprising a signal.

Note: the absolute difference of $E(X)$ and $Var(x)$ instead of the ratio was chosen as both can become 0 which would result in `Inf` values.

Value

The function returns

[NUMERICAL OUTPUT]

`RLum.Reuslts`-object

slot: `@data`

Element	Type	Description
`$unique_pairs`	`data.frame`	the unique position and grain pairs
`$selection_id`	`numeric`	the selection as record ID
`$selection_full`	`data.frame`	implemented models used in the baSAR-model core

slot: `@info`

The original function call

Output variation

For `cleanup = TRUE` the same object as the input, but with cleaned up (invalid curves removed). This means: Either an `Risoe.BINfileData` or an `RLum.Analysis` object is returned in such cases. An `Risoe.BINfileData` object can be exported to a BIN-file by using the function `write_R2BIN`.

Function version

0.2.0 (2016-06-20 19:34:56)

Note

This function can work with `Risoe.BINfileData` objects or `RLum.Analysis` objects (or a list of it). However, the function is highly optimised for `Risoe.BINfileData` objects as it make sense to remove identify invalid grains before the conversion to an `RLum.Analysis` object.

The function checking for invalid curves works rather robust and it is likely that Reg0 curves within a SAR cycle are removed as well. Therefore it is strongly recommended to use the argument `cleanup = TRUE` carefully.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

-

See Also

Risoe.BINfileData, RLum.Analysis, write_R2BIN, read_BIN2R

Examples

```
##01 - basic example I
##just show how to apply the function
data(ExampleData.XSYG, envir = environment())

##verify and get data.frame out of it
verify_SingleGrainData(OSL.SARMeasurement$Sequence.Object)$selection_full

##02 - basic example II
data(ExampleData.BINfileData, envir = environment())
id <- verify_SingleGrainData(object = CWOSL.SAR.Data,
cleanup_level = "aliquot")$selection_id

## Not run:
##03 - advanced example I
##importing and exporting a BIN-file

##select and import file
file <- file.choose()
object <- read_BIN2R(file)

##remove invalid aliquots(!)
object <- verify_SingleGrainData(object, cleanup = TRUE)

##export to new BIN-file
write_R2BIN(object, paste0(dirname(file),"/", basename(file), "_CLEANED.BIN"))

## End(Not run)
```

`write_R2BIN` *Export Risoe.BINfileData into Risoe BIN-file*

Description

Exports a Risoe.BINfileData object in a *.bin or *.binx file that can be opened by the Analyst software or other Risoe software.

Usage

```
write_R2BIN(object, file, version, compatibility.mode = FALSE,
  txtProgressBar = TRUE)
```

Arguments

object	Risoe.BINfileData (**required**): input object to be stored in a bin file.
file	character (**required**): file name and path of the output file [WIN]: write_R2BIN(object, "C:/Desktop/test.bin"), [MAC/LINUX]: write_R2BIN("/User/test/Desktop/test.bin")
version	character (optional): version number for the output file. If no value is provided the highest version number from the Risoe.BINfileData is taken automatically. Note: This argument can be used to convert BIN-file versions.
compatibility.mode	logical (with default): this option recalculates the position values if necessary and set the max. value to 48. The old position number is appended as comment (e.g., 'OP: 70). This option accounts for potential compatibility problems with the Analyst software.
txtProgressBar	logical (with default): enables or disables txtProgressBar.

Details

The structure of the exported binary data follows the data structure published in the Appendices of the Analyst manual p. 42.

If LTYPE, DTYPE and LIGHTSOURCE are not of type character, no transformation into numeric values is done.

Value

Write a binary file.

Function version

0.4.0 (2016-06-13 21:17:19)

Note

The function just roughly checks the data structures. The validity of the output data depends on the user.

The validity of the file path is not further checked.
BIN-file conversions using the argument version may be a lossy conversion, depending on the chosen input and output data (e.g., conversion from version 08 to 07 to 06 to 04 or 03).

Warning

Although the coding was done carefully it seems that the BIN/BINX-files produced by Risoe DA 15/20 TL/OSL readers slightly differ on the byte level. No obvious differences are observed in the METADATA, however, the BIN/BINX-file may not fully compatible, at least not similar to the once directly produced by the Risoe readers!

ROI definitions (introduced in BIN-file version 8) are not supported! There are furthermore ignored by the function read_BIN2R.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
R Luminescence Package Team

References

DTU Nutech, 2016. The Squence Editor, Users Manual, February, 2016. http://www.nutech.dtu.dk/english/Products-and-Services/Dosimetry/Radiation-Measurement-Instruments/TL_OSL_reader/Manuals

See Also

read_BIN2R, Risoe.BINfileData, writeBin

Examples

```
##uncomment for usage

##data(ExampleData.BINfileData, envir = environment())
##write_R2BIN(CWOSL.SAR.Data, file="[your path]/output.bin")
```

Index

∗Topic **IO**
extract_IrradiationTimes, 103
merge_Risoe.BINfileData, 119
read_BIN2R, 177
read_Daybreak2R, 180
read_SPE2R, 181
read_XSYG2R, 183
write_R2BIN, 223
∗Topic **aplot**
plot_FilterCombinations, 144
plot_RLum.Analysis, 165
plot_RLum.Data.Curve, 167
plot_RLum.Data.Image, 169
plot_RLum.Data.Spectrum, 171
plot_RLum.Results, 174
∗Topic **classes**
Risoe.BINfileData-class, 190
RLum-class, 196
RLum.Analysis-class, 197
RLum.Data-class, 200
RLum.Data.Curve-class, 201
RLum.Data.Image-class, 203
RLum.Data.Spectrum-class, 205
RLum.Results-class, 207
∗Topic **datagen**
analyse_baSAR, 6
analyse_IRSAR.RF, 13
analyse_pIRIRSequence, 19
analyse_SAR.CWOSL, 22
Analyse_SAR.OSLdata, 25
analyse_SAR.TL, 28
calc_FadingCorr, 49
calc_gSGC, 59
calc_OSLLxTxRatio, 71
calc_Statistics, 77
calc_ThermalLifetime, 79
calc_TLLxTxRatio, 81
plot_FilterCombinations, 144
verify_SingleGrainData, 221
∗Topic **datasets**
BaseDataSet.CosmicDoseRate, 35
ExampleData.BINfileData, 94
ExampleData.CW_OSL_Curve, 96
ExampleData.DeValues, 97
ExampleData.RLum.Analysis, 100
ExampleData.RLum.Data.Image, 101
ExampleData.XSYG, 102
∗Topic **dplot**
Analyse_SAR.OSLdata, 25
calc_FuchsLang2001, 57
fit_CWCurve, 106
fit_LMCurve, 109
plot_DRTResults, 141
plot_Risoe.BINfileData, 162
plot_RLum, 164
∗Topic **manip**
apply_CosmicRayRemoval, 30
apply_EfficiencyCorrection, 32
calc_SourceDoseRate, 75
CW2pHMi, 83
CW2pLM, 87
CW2pLMi, 88
CW2pPMi, 91
extract_IrradiationTimes, 103
merge_Risoe.BINfileData, 119
Risoe.BINfileData2RLum.Analysis, 194
Second2Gray, 210
sTeve, 214
tune_Data, 217
verify_SingleGrainData, 221
∗Topic **methods**
RLum.Analysis-class, 197
RLum.Results-class, 207
∗Topic **models**
fit_CWCurve, 106
fit_LMCurve, 109
∗Topic **package**
Luminescence-package, 4
∗Topic **plot**
analyse_pIRIRSequence, 19
analyse_SAR.CWOSL, 22
analyse_SAR.TL, 28
∗Topic **utilities**
bin_RLum.Data, 37
get_Risoe.BINfileData, 116

get_RLum, 117
length_RLum, 118
merge_RLum, 120
merge_RLum.Analysis, 122
merge_RLum.Data.Curve, 123
names_RLum, 131
replicate_RLum, 186
set_Risoe.BINfileData, 212
set_RLum, 213
structure_RLum, 215
*.RLum.Data.Curve *(methods_RLum)*, 126
+.RLum.Data.Curve *(methods_RLum)*, 126
-.RLum.Data.Curve *(methods_RLum)*, 126
/.RLum.Data.Curve *(methods_RLum)*, 126
[.RLum.Analysis *(methods_RLum)*, 126
[.RLum.Data.Curve *(methods_RLum)*, 126
[.RLum.Data.Image *(methods_RLum)*, 126
[.RLum.Data.Spectrum *(methods_RLum)*, 126
[.RLum.Results *(methods_RLum)*, 126
[[.RLum.Analysis *(methods_RLum)*, 126
[[.RLum.Results *(methods_RLum)*, 126
$.RLum.Analysis *(methods_RLum)*, 126
$.RLum.Data.Curve *(methods_RLum)*, 126
$.RLum.Results *(methods_RLum)*, 126

abline, *166*
analyse_baSAR, 6
analyse_IRSAR.RF, 13
analyse_pIRIRSequence, 19, *27*, *140, 141*
analyse_SAR.CWOSL, *19, 20*, 22, *27*, *74*, *140, 141*, *207*
Analyse_SAR.OSLdata, *24*, 25, *74*
analyse_SAR.TL, 28, *82*
app_RLum, *34*, 34
apply_CosmicRayRemoval, 30, *32*
apply_EfficiencyCorrection, 32
approx, *33*, *85*, *144*, *146*, *184, 185*
array, *80*
as, 34, *35*
as.data.frame, *129*, *217*
as.data.frame.RLum.Data.Curve *(methods_RLum)*, 126
as.data.frame.RLum.Data.Spectrum *(methods_RLum)*, 126
as.list.RLum.Analysis *(methods_RLum)*, 126
as.list.RLum.Data.Curve *(methods_RLum)*, 126
as.list.RLum.Results *(methods_RLum)*, 126
as.matrix.RLum.Data.Curve *(methods_RLum)*, 126
as.matrix.RLum.Data.Spectrum *(methods_RLum)*, 126
BaseDataSet.CosmicDoseRate, 35, *48*
bin.RLum.Data.Curve *(methods_RLum)*, 126
bin_RLum.Data, 37
bin_RLum.Data,RLum.Data.Curve-method *(RLum.Data.Curve-class)*, 201
boxplot, *176, 177*
boxplot.default, *12*
brick, *204*
browseURL, *189*

calc_AliquotSize, 39
calc_CentralDose, 41, *45*, *57*, *59*, *63*, *66*, *70*
calc_CommonDose, *43*, 43, *57*, *59*, *63*, *66*, *70*
calc_CosmicDoseRate, 45
calc_FadingCorr, 49
calc_FastRatio, 52
calc_FiniteMixture, *43*, *45*, 54, *59*, *63*, *66*, *70*
calc_FuchsLang2001, *43*, *45*, *57*, 57, *63*, *66*, *70*
calc_gSGC, 59
calc_HomogeneityTest, 61
calc_IEU, 62
calc_MaxDose, 64, *70*
calc_MinDose, *43*, *45*, *57*, *59*, *63*, *65, 66*, 66
calc_OSLLxTxRatio, *7–10*, *12*, *20*, *23*, *25–27*, 71
calc_SourceDoseRate, 75, *210, 211*
calc_Statistics, 77, *133*, *153*, *177*
calc_ThermalLifetime, 79
calc_TLLxTxRatio, *29, 30*, 81
call, *20*, *40*, *42*, *44*, *47*, *53*, *56*, *58*, *61*, *63*, *68*, *219*
character, *7–9*, *13*, *19*, *22*, *26*, *28–30*, *34*, *55*, *59*, *72*, *75*, *77*, *79*, *104*, *106, 107*, *110*, *113*, *115*, *119*, *123*, *130–134*, *139, 140*, *142*, *147*, *151*, *153*, *156*, *158, 159*, *162, 163*, *166*, *169*, *171, 172*, *176*, *178*, *180, 181*, *183*, *187, 188*, *195–199*, *201*, *204*, *206*, *208*, *210*, *213*, *219*, *221*, *224*
coda.samples, *12*
confint, *106–108*, *110*, *112*
contour, *170*, *173*
CW2pHMi, 83, *88*, *90*, *93*, *163*
CW2pLM, *85*, 87, *90*, *93*, *110*, *163*
CW2pLMi, *85*, *88*, 88, *93*, *163*
CW2pPMi, *85*, *88*, *90*, 91, *163*

data.frame, *17*, *20*, *24*, *27*, *29*, *32*, *34*, *40–44*, *47*, *52–54*, *56*, *58, 59*, *61–64*, *66*, *68*, *72*, *77*, *81–83*, *87*, *89*, *92*, *95*, *97*, *102*, *106*, *108*, *110*, *125*, *132*, *142*,

145, *147*, *151*, *153*, *155*, *158*, *176*, *179*, *185*, *198*, *210*, *211*, *218*, *219*
Date, *75*
density, *153*, *154*, *176*, *177*
dim.RLum.Data.Curve *(*methods_RLum*)*, 126
dim.RLum.Data.Spectrum *(*methods_RLum*)*, 126

ExampleData.BINfileData, 94
ExampleData.CW_OSL_Curve, 96
ExampleData.DeValues, 97
ExampleData.FittingLM, 98
ExampleData.LxTxData, 99
ExampleData.LxTxOSLData, 99
ExampleData.RLum.Analysis, 100
ExampleData.RLum.Data.Image, 101
ExampleData.XSYG, 102
expression, *129*, *149*
extract_IrradiationTimes, 103

fit_CWCurve, *53*, *54*, 106, *113*
fit_LMCurve, *85*, *88*, *90*, *93*, *109*, 109
formula, *24*

get_Layout, 113
get_Quote, 114
get_rightAnswer, 115
get_Risoe.BINfileData, 116
get_Risoe.BINfileData,Risoe.BINfileData-method *(*Risoe.BINfileData-class*)*, 190
get_RLum, *17*, *18*, *20*, *24*, *25*, *29*, *30*, *40*, *42*, *44*, *47*, *51*, *54*, *56*, *60*, *61*, *63*, *68*, *74*, *76*, *80*, *104*, *109*, *113*, *117*, 117, *150*, *166*, *181*, *219*
get_RLum,list-method *(*get_RLum*)*, 117
get_RLum,RLum.Analysis-method *(*RLum.Analysis-class*)*, 197
get_RLum,RLum.Data.Curve-method *(*RLum.Data.Curve-class*)*, 201
get_RLum,RLum.Data.Image-method *(*RLum.Data.Image-class*)*, 203
get_RLum,RLum.Data.Spectrum-method *(*RLum.Data.Spectrum-class*)*, 205
get_RLum,RLum.Results-method *(*RLum.Results-class*)*, 207
glm, *111*
grid, *146*

hist, *151*, *152*
hist.RLum.Analysis *(*methods_RLum*)*, 126
hist.RLum.Data.Curve *(*methods_RLum*)*, 126
hist.RLum.Data.Image *(*methods_RLum*)*, 126
hist.RLum.Results *(*methods_RLum*)*, 126

integer, *8*, *9*, *15*, *19*, *22*, *28*, *31*, *50*, *60*, *72*, *75*, *77*, *82*, *110*, *119*, *129*, *134*, *139*, *140*, *147*, *148*, *156*, *166*, *171*, *178*, *186*, *196*, *201*, *214*, *216*
is.RLum *(*methods_RLum*)*, 126

jags.model, *9*, *10*, *12*

legend, *14*, *146*, *156*, *176*
length.Risoe.BINfileData *(*methods_RLum*)*, 126
length.RLum.Analysis *(*methods_RLum*)*, 126
length.RLum.Data.Curve *(*methods_RLum*)*, 126
length.RLum.Results *(*methods_RLum*)*, 126
length_RLum, 118
length_RLum,RLum.Analysis-method *(*RLum.Analysis-class*)*, 197
length_RLum,RLum.Data.Curve-method *(*RLum.Data.Curve-class*)*, 201
length_RLum,RLum.Results-method *(*RLum.Results-class*)*, 207
list, *7–9*, *13*, *14*, *17*, *19*, *20*, *22*, *23*, *27*, *29*, *34*, *35*, *40*, *42*, *44*, *47*, *53*, *56*, *58*, *60*, *61*, *63*, *68*, *73*, *76*, *79*, *82*, *97*, *104*, *113*, *117*, *120*, *122*, *123*, *125*, *131*, *140*, *145*, *155*, *164*, *166*, *178–180*, *183*, *186*, *196*, *198*, *199*, *201*, *204*, *206*, *208*, *217*, *219*
list.files, *178*, *179*
lm, *84*, *85*, *88*, *89*, *92*, *148–150*
logical, *8*, *14*, *15*, *19*, *23*, *26*, *31*, *39*, *41*, *43*, *46*, *50*, *53*, *55*, *58*, *60–62*, *64*, *66*, *67*, *78*, *79*, *104*, *106*, *107*, *110*, *115*, *117*, *119*, *129–134*, *140*, *142*, *145*, *147*, *148*, *151*, *153*, *156*, *158*, *159*, *166*, *168*, *169*, *171*, *175*, *176*, *178*, *180*, *181*, *183*, *187*, *195*, *198*, *208*, *216*, *221*, *224*
Luminescence *(*Luminescence-package*)*, 4
Luminescence-package, 4

matplot, *80*
matrix, *34*, *56*, *80*, *108*, *125*, *145*, *155*, *171*, *176*, *201*, *202*, *204*, *206*
merge.RLum *(*methods_RLum*)*, 126
merge_Risoe.BINfileData, 119, *179*, *194*
merge_RLum, 120, *122*, *125*, *203*, *209*
merge_RLum.Analysis, 122
merge_RLum.Data.Curve, 123
merge_RLum.Results, 125
methods_RLum, 126
mle2, *68*

model_LuminescenceSignals, *130*, 130
mtext, *151*

names.Risoe.BINfileData *(*methods_RLum*)*, 126
names.RLum.Analysis *(*methods_RLum*)*, 126
names.RLum.Data.Curve *(*methods_RLum*)*, 126
names.RLum.Data.Image *(*methods_RLum*)*, 126
names.RLum.Data.Spectrum *(*methods_RLum*)*, 126
names.RLum.Results *(*methods_RLum*)*, 126
names_RLum, 131
names_RLum,RLum.Analysis-method *(*RLum.Analysis-class*)*, 197
names_RLum,RLum.Data.Curve-method *(*RLum.Data.Curve-class*)*, 201
names_RLum,RLum.Data.Image-method *(*RLum.Data.Image-class*)*, 203
names_RLum,RLum.Data.Spectrum-method *(*RLum.Data.Spectrum-class*)*, 205
names_RLum,RLum.Results-method *(*RLum.Results-class*)*, 207
nlminb, *67*
nls, *15*, *17*, *18*, *106–113*, *148–150*
nlsLM, *16*, *18*, *109*, *113*, *148*, *150*
numeric, *7*, *9*, *14*, *16*, *19*, *22*, *23*, *26*, *29*, *31*, *39*, *41*, *43*, *45*, *46*, *49*, *52–55*, *58*, *62*, *64*, *66–68*, *72*, *75*, *77*, *79*, *80*, *106*, *110*, *123*, *131–134*, *139*, *142*, *145*, *148*, *151*, *153*, *158*, *159*, *163*, *171*, *176*, *178*, *194*, *195*, *198*, *208*, *210*, *218*, *221*

openFileInOS, *189*

pander_return, *189*
par, *156*
pchisq, *62*
pdf, *166*, *175*
persp, *172*, *173*
plot, *43*, *58*, *59*, *63*, *107*, *109*, *113*, *139*, *141–143*, *151–154*, *156*, *160*, *167–170*, *173*, *175–177*
plot.default, *29*, *79*, *131*, *140*, *176*
plot.list *(*methods_RLum*)*, 126
plot.Risoe.BINfileData *(*methods_RLum*)*, 126
plot.RLum.Analysis *(*methods_RLum*)*, 126
plot.RLum.Data.Curve *(*methods_RLum*)*, 126
plot.RLum.Data.Image *(*methods_RLum*)*, 126
plot.RLum.Data.Spectrum *(*methods_RLum*)*, 126
plot.RLum.Results *(*methods_RLum*)*, 126
plot_AbanicoPlot, *78*, 132
plot_DetPlot, 139
plot_DRTResults, 141
plot_FilterCombinations, 144
plot_GrowthCurve, *8–10*, *12*, *19*, *20*, *23*, *25*, *27*, *29*, *30*, *74*, 147
plot_Histogram, *136*, 150, *160*
plot_KDE, *136*, 152, *160*, *215*
plot_ly, *173*
plot_NRt, 155
plot_RadialPlot, *136*, 158
plot_Risoe.BINfileData, 162, *194*
plot_RLum, *76*, *88*, *103*, 164, *167*, *168*, *170*, *173*, *175*, *203*, *205*, *207*, *209*
plot_RLum.Analysis, *103*, *164*, *165*, 165
plot_RLum.Data.Curve, *164*, *165*, *167*, 167
plot_RLum.Data.Image, *164*, *165*, 169
plot_RLum.Data.Spectrum, *103*, *164*, *165*, 171
plot_RLum.Results, *164*, *165*, 174
plot_ViolinPlot, 176
plotRGB, *169*
profile, *108*
profile.mle2, *68*

raster, *169*, *170*, *182*
raw, *178*
read_BIN2R, *8–10*, *12*, *26*, *27*, *104*, *105*, *120*, *162*, *163*, 177, *190*, *194*, *195*, *223*, *225*
read_Daybreak2R, 180, *197*
read_excel, *8*, *10*, *12*
read_SPE2R, *101*, *170*, 181, *203*, *205*
read_XSYG2R, *102–105*, 183, *197*, *203*, *207*
readBin, *179*, *182*
regex, *183*
render, *187*, *189*
rep.RLum *(*methods_RLum*)*, 126
replicate_RLum, 186
replicate_RLum,RLum-method *(*RLum-class*)*, 196
report_RLum, 187
Risoe.BINfileData, *7*, *9*, *105*, *116*, *119*, *120*, *129*, *179*, *190*, *194*, *195*, *199*, *212*, *213*, *221–225*
Risoe.BINfileData-class, *24*, *26*, *27*, *94*, *162*, *178*, *179*, 190
Risoe.BINfileData2RLum.Analysis, *178*, *179*, *194*, 194, *199*, *203*
rjags, *11*

RLum, *34*, *84*, *87*, *90*, *93*, *117, 118*, *120–122*, *129*, *131*, *164*, *186, 187*, *196*, *199, 200*, *203*, *205*, *207*, *209*, *213*, *215*
RLum-class, 196
RLum.Analysis, *13*, *18–20*, *22, 23*, *25*, *28*, *30*, *35*, *52*, *54*, *100*, *102–105*, *117, 118*, *121, 122*, *132*, *139*, *155, 156*, *164–166*, *179, 180*, *183*, *185*, *195–199*, *203*, *213, 214*, *216*, *221–223*
RLum.Analysis-class, 197
RLum.Data, *37, 38*, *196*, *198, 199*, *203*, *205*, *207*
RLum.Data-class, 200
RLum.Data.Curve, *38*, *52*, *54*, *72*, *74*, *81–83*, *85*, *87–90*, *92, 93*, *106*, *109, 110*, *117, 118*, *121–125*, *132*, *156*, *164–166*, *168*, *180*, *185*, *199–202*, *213, 214*, *216*
RLum.Data.Curve-class, 201
RLum.Data.Image, *101*, *117, 118*, *121, 122*, *132*, *164, 165*, *169, 170*, *181*, *204*, *213, 214*, *216*
RLum.Data.Image-class, 203
RLum.Data.Spectrum, *30–33*, *102, 103*, *117, 118*, *121, 122*, *132*, *164, 165*, *171–173*, *181, 182*, *200*, *206*, *213, 214*, *216*
RLum.Data.Spectrum-class, 205
RLum.Results, *7*, *9*, *17, 18*, *20*, *23–25*, *29, 30*, *35*, *40–44*, *47*, *49*, *51*, *53, 54*, *56*, *58–64*, *66*, *68*, *73*, *76, 77*, *80*, *82*, *104, 105*, *108, 109*, *117, 118*, *121*, *125*, *132*, *140*, *142*, *146*, *150, 151*, *153*, *158*, *164, 165*, *175, 176*, *208*, *210*, *213, 214*, *216*, *219*
RLum.Results-class, 207
RLumModel-package, *130*
RLumShiny-package, *34*
rnorm, *80*
rollmean, *156*, *168*
row.names.RLum.Data.Spectrum *(methods_RLum)*, 126
rowMeans, *123*
rowMedians, *124*
rowMins, *124*
rowSds, *124*
rowSums, *123*
rowVars, *124*
rug, *177*
runApp, *34*

sd, *149*
Second2Gray, *75, 76*, 210
set.seed, *50*
set_Risoe.BINfileData, 212
set_Risoe.BINfileData,data.frame,list-method *(Risoe.BINfileData-class)*, 190
set_RLum, *196*, 213
set_RLum,RLum.Analysis-method *(RLum.Analysis-class)*, 197
set_RLum,RLum.Data.Curve-method *(RLum.Data.Curve-class)*, 201
set_RLum,RLum.Data.Image-method *(RLum.Data.Image-class)*, 203
set_RLum,RLum.Data.Spectrum-method *(RLum.Data.Spectrum-class)*, 205
set_RLum,RLum.Results-method *(RLum.Results-class)*, 207
show,Risoe.BINfileData-method *(Risoe.BINfileData-class)*, 190
show,RLum.Analysis-method *(RLum.Analysis-class)*, 197
show,RLum.Data.Curve-method *(RLum.Data.Curve-class)*, 201
show,RLum.Data.Image-method *(RLum.Data.Image-class)*, 203
show,RLum.Data.Spectrum-method *(RLum.Data.Spectrum-class)*, 205
show,RLum.Results-method *(RLum.Results-class)*, 207
smooth, *30–32*
smooth.spline, *30–32*, *156*
sTeve, 214
structure_RLum, *199*, 215
structure_RLum,RLum.Analysis-method *(RLum.Analysis-class)*, 197
subset.data.frame, *129*
subset.Risoe.BINfileData *(methods_RLum)*, 126
summary, *107, 108*, *112*
summary.RLum.Analysis *(methods_RLum)*, 126
summary.RLum.Data.Curve *(methods_RLum)*, 126
summary.RLum.Data.Image *(methods_RLum)*, 126
summary.RLum.Results *(methods_RLum)*, 126

template_DRAC, 216
tune_Data, 217
txtProgressBar, *50*, *178–181*, *195*, *224*

uniroot, *50, 51*, *60*, *148*
unlist.RLum.Analysis *(methods_RLum)*, 126
use_DRAC, 219

vector, *7*, *13*, *19*, *26*, *28*, *49*, *72*, *83*, *89*, *92*, *106*, *162*, *171*, *179*, *181*, *194, 195*
verify_SingleGrainData, *9, 10*, *12*, 221
viewer, *189*

write_R2BIN, *104, 105*, *120*, *179*, *193, 194*, *222, 223*, 223
writeBin, *225*

xml, *183*, *185*