Package ‘Luminescence’

September 20, 2019

Type Package

Title Comprehensive Luminescence Dating Data Analysis

Version 0.9.5

Date 2019-09-20

Author Sebastian Kreutzer [aut, trl, cre, dtc]
(https://orcid.org/0000-0002-0734-2199),
Christoph Burow [aut, trl, dtc]
(https://orcid.org/0000-0002-5023-4046),
Michael Dietze [aut] (https://orcid.org/0000-0001-6063-1726),
Margret C. Fuchs [aut],
Christoph Schmidt [aut] (https://orcid.org/0000-0002-2309-3209),
Manfred Fischer [aut, trl],
Johannes Friedrich [aut] (https://orcid.org/0000-0002-0805-9547),
Norbert Mercier [ctb] (https://orcid.org/0000-0002-6375-9108),
Rachel K. Smedley [ctb] (https://orcid.org/0000-0001-7773-5193),
Claire Christophe [ctb],
Antoine Zink [ctb] (https://orcid.org/0000-0001-7146-1101),
Julie Durcan [ctb] (https://orcid.org/0000-0001-8724-8022),
Georgina E. King [ctb, dtc] (https://orcid.org/0000-0003-1059-8192),
Anne Philippe [ctb],
Guillaume Guerin [ctb] (https://orcid.org/0000-0001-6298-5579),
Svenja Riedesel [ctb] (https://orcid.org/0000-0003-2936-8776),
Martin Autzen [ctb],
Pierre Guibert [ctb] (https://orcid.org/0000-0001-8969-8684),
Markus Fuchs [ths]

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

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.

Contact Package Developers <developers@r-luminescence.org>

License GPL-3

BugReports https://github.com/R-Lum/Luminescence/issues
Depends R (>= 3.4.0), utils, magrittr (>= 1.5)

LinkingTo Rcpp (>= 1.0.0), RcppArmadillo (>= 0.9.300.0.0)

Imports bbmle (>= 1.0.20), data.table (>= 1.12.0), DEoptim (>= 2.2-4),
httr (>= 1.4.0), matrixStats (>= 0.54.0), methods, minpack.lm
(>= 1.2), plotrix (>= 3.7), raster (>= 2.8-0), readxl (>= 1.3.0),
shape (>= 1.4.3), parallel, XML (>= 3.98-1.9), zoo (>= 1.8)

Suggests RLumShiny (>= 0.2.2), plotly (>= 4.9.0), rmarkdown (>= 1.12),
 rstudioapi (>= 0.7), rjags (>= 4-8), coda (>= 0.19-1), pander
(>= 0.6.1), testthat (>= 2.0.0), devtools (>= 2.0.0), R.rsp (>= 0.43.0)

VignetteBuilder R.rsp

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

Encoding UTF-8

Collate 'Analyse_SAR.OSLdata.R' 'CW2pPHMi.R' 'CW2pPLM.R' 'CW2pLMI.R'
'CW2pPMi.R' 'Luminescence-package.R' 'PSL2Risoe.BINfileData.R'
'RcppExports.R' 'replicate_RLum.R' 'RLum-class.R'
'smooth_RLum.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.Spectrum-class.R' 'RLum.Results-class.R'
'set_Risoe.BINfileData.R' 'get_Risoe.BINfileData.R'
'Risoe.BINfileData-class.R' 'Risoe.BINfileData2RLum.Analysis.R'
'addins_RLum.R' 'analyse_A12O3C_CrossTalk.R'
'analyse_A12O3C_ITC.R' 'analyse_A12O3C_Measurement.R'
'analyse_FadingMeasurement.R' 'analyse_IRSAR.RF.R'
'analyse_SAR.CWOSL.R' 'analyse_SAR.TL.R' 'analyse_baSAR.R'
'analyse_pIRIRSequence.R' 'analyse_portableOSL.R'
'app_RLum.R' 'apply_CosmicRayRemoval.R' 'apply_EfficiencyCorrection.R'
'calc_AliquotSize.R' 'calc_AverageDose.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_Huntley2006.R' 'calc_IEU.R' 'calc_Kars2008.R'
'calc_Lamothe2003.R' 'calc_MaxDose.R' 'calc_MinDose.R'
'calc_OSSLxTxRatio.R' 'calc_SourceDoseRate.R'
'calc_Statistics.R' 'calc_TLLxTxRatio.R'
'calc_ThermalLifetime.R' 'calc_WodaFuchs2008.R' 'calc_gSGC.R'
'convert_Activity2Concentration.R' 'convert_BIN2CSV.R'
'convert_Daybreak2CSV.R' 'convert_PSL2CSV.R'
'convert_RLum2Risoe.BINfileData.R'
'convert_Wavelength2Energy.R' 'convert_XSYG2CSV.R'
'extract_IrradiationTimes.R' 'fit_CWCurve.R' 'fit_LMCurve.R'
'fit_OSSLifeTimes.R' 'fit_SurfaceExposure.R'
R topics documented:

'fit_ThermalQuenching.R' 'get_Layout.R' 'get_Quote.R'
'get_rightAnswer.R' 'github.R' 'install_DevelopmentVersion.R'
'internal_as.latex.table.R' 'internals_RLum.R'
'merge_RLum.R' 'merge_RLum.Results.R'
'merge_Risoe.BINfileData.R' 'methods_DRAC.R' 'methods_RLum.R'
'plot_AbanicoPlot.R' 'plot_DRCSummary.R' 'plot_DRResults.R'
'plot_DetPlot.R' 'plot_FilterCombinations.R'
'plot_GrowthCurve.R' 'plot_Histogram.R' 'plot_KDE.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_PSL2R.R' 'read_SPE2R.R'
'read_XSYG2R.R' 'report_RLum.R' 'scale_GammaDose.R'
'template_DRAC.R' 'tune_Data.R' 'use_DRAC.R' 'utils_DRAC.R'
'verify_SingleGrainData.R' 'write_R2BIN.R' 'write_RLum2CSV.R'
'zzz.R'

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-09-20 11:20:02 UTC

R topics documented:

Luminescence-package .................................................. 6
analyse_A12O3C_CrossTalk ............................................. 8
analyse_A12O3C_ITC ...................................................... 10
analyse_A12O3C_Measurement ............................................ 12
analyse_baSAR ........................................................... 15
analyse_FadingMeasurement ........................................... 23
analyse_IRSAR.RF ...................................................... 26
analyse_pIRIRSequence ............................................... 33
analyse_portableOSL .................................................. 36
analyse_SAR.CWOSL ................................................... 38
Analyse_SAR.OSLdata .................................................. 42
analyse_SAR.TL ......................................................... 45
apply_CosmicRayRemoval ............................................. 47
apply_EfficiencyCorrection ........................................... 50
app_RLum ............................................................... 51
as ................................................................. 52
BaseDataSet.ConversionFactors ...................................... 53
BaseDataSet.CosmicDoseRate ......................................... 54
BaseDataSet.FractionalGammaDose .................................... 56
bin_RLum.Data .......................................................... 57
calc_AliquotSize ...................................................... 58
calc_AverageDose ........................................... 61
calc_CentralDose ........................................... 64
calc_CommonDose ........................................... 66
calc_CosmicDoseRate ...................................... 68
calc_FadingCorr ........................................... 73
calc_FastRatio ............................................. 76
calc_FiniteMixture ......................................... 79
calc_FuchsLang2001 ......................................... 82
calc_gSGC ..................................................... 84
calc_HomogeneityTest ...................................... 86
calc_Huntley2006 ........................................... 88
calc_IEU ....................................................... 93
calc_Kars2008 ............................................. 95
calc_Lamothe2003 .......................................... 97
calc_MaxDose ................................................ 99
calc_MinDose ............................................... 102
calc(OSLLxTxRatio .......................................... 108
calc_SourceDoseRate ....................................... 112
calc_Statistics ............................................ 114
calc_ThermalLifetime ...................................... 116
calc_TLLxTxRatio ........................................... 119
calc_WodaFuchs2008 ....................................... 121
convert_Activity2Concentration ....................... 123
convert_BIN2CSV ........................................... 125
convert_Daybreak2CSV .................................... 126
convert_PSL2CSV ........................................... 127
convert_RLum2Risoe.BINfileData ......................... 128
convert_Wavelength2Energy ............................... 130
convert_XSYG2CSV ......................................... 132
CW2pHMi ..................................................... 134
CW2pLM ...................................................... 137
CW2pLMi .................................................... 139
CW2pPMi ..................................................... 142
ExampleData.AI2O3C ...................................... 145
ExampleData.BINfileData .................................. 146
ExampleData.CW_OSL_Curve ............................. 148
ExampleData.DeValues ................................... 149
ExampleData.Fading ....................................... 150
ExampleData.FittingLM .................................... 152
ExampleData.LxTxData .................................... 152
ExampleData.LxTxOSLData ................................ 153
ExampleData.portableOSL ................................. 154
ExampleData.RLum.Analysis .............................. 154
ExampleData.RLum.Data.Image ......................... 155
ExampleData.ScaleGammaDose ......................... 156
ExampleData.SurfaceExposure ....................... 157
ExampleData.TR_OSL ...................................... 160
ExampleData.XSYG ......................................... 161
# R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>extdata</td>
<td>163</td>
</tr>
<tr>
<td>extract_IrradiationTimes</td>
<td>164</td>
</tr>
<tr>
<td>fit_CWCurve</td>
<td>167</td>
</tr>
<tr>
<td>fit_LMCurve</td>
<td>170</td>
</tr>
<tr>
<td>fit_OSSLifeTimes</td>
<td>174</td>
</tr>
<tr>
<td>fit_SurfaceExposure</td>
<td>178</td>
</tr>
<tr>
<td>fit_TermalQuenching</td>
<td>182</td>
</tr>
<tr>
<td>get_Layout</td>
<td>185</td>
</tr>
<tr>
<td>get_Quote</td>
<td>186</td>
</tr>
<tr>
<td>get_rightAnswer</td>
<td>187</td>
</tr>
<tr>
<td>get_Risoe.BINfileData</td>
<td>188</td>
</tr>
<tr>
<td>get_RLum</td>
<td>189</td>
</tr>
<tr>
<td>GitHub-API</td>
<td>190</td>
</tr>
<tr>
<td>install_DevelopmentVersion</td>
<td>192</td>
</tr>
<tr>
<td>length_RLum</td>
<td>193</td>
</tr>
<tr>
<td>merge_Risoe.BINfileData</td>
<td>194</td>
</tr>
<tr>
<td>merge_RLum</td>
<td>196</td>
</tr>
<tr>
<td>names_RLum</td>
<td>199</td>
</tr>
<tr>
<td>plot_AbanicoPlot</td>
<td>200</td>
</tr>
<tr>
<td>plot_DetPlot</td>
<td>207</td>
</tr>
<tr>
<td>plot_DRCSummary</td>
<td>210</td>
</tr>
<tr>
<td>plot_DRTResults</td>
<td>212</td>
</tr>
<tr>
<td>plot_FilterCombinations</td>
<td>216</td>
</tr>
<tr>
<td>plot_GrowthCurve</td>
<td>219</td>
</tr>
<tr>
<td>plot_Histogram</td>
<td>224</td>
</tr>
<tr>
<td>plot_KDE</td>
<td>227</td>
</tr>
<tr>
<td>plot_NRt</td>
<td>230</td>
</tr>
<tr>
<td>plot_RadialPlot</td>
<td>233</td>
</tr>
<tr>
<td>plot_Risoe.BINfileData</td>
<td>238</td>
</tr>
<tr>
<td>plot_RLum</td>
<td>240</td>
</tr>
<tr>
<td>plot_RLum.Analysis</td>
<td>242</td>
</tr>
<tr>
<td>plot_RLum.Data.Curve</td>
<td>244</td>
</tr>
<tr>
<td>plot_RLum.Data.Image</td>
<td>246</td>
</tr>
<tr>
<td>plot_RLum.Data.Spectrum</td>
<td>248</td>
</tr>
<tr>
<td>plot_RLum.Results</td>
<td>252</td>
</tr>
<tr>
<td>plot_ViolinPlot</td>
<td>253</td>
</tr>
<tr>
<td>PSL2Risoe.BINfileData</td>
<td>255</td>
</tr>
<tr>
<td>read_BIN2R</td>
<td>257</td>
</tr>
<tr>
<td>read_Daybreak2R</td>
<td>259</td>
</tr>
<tr>
<td>read_PSL2R</td>
<td>261</td>
</tr>
<tr>
<td>read_SPE2R</td>
<td>262</td>
</tr>
<tr>
<td>read_XSYG2R</td>
<td>265</td>
</tr>
<tr>
<td>replicate_RLum</td>
<td>268</td>
</tr>
<tr>
<td>report_RLum</td>
<td>269</td>
</tr>
<tr>
<td>Risoe.BINfileData2RLum.Analysis</td>
<td>272</td>
</tr>
<tr>
<td>RLM-class</td>
<td>274</td>
</tr>
<tr>
<td>scale_GammaDose</td>
<td>276</td>
</tr>
<tr>
<td>Second2Gray</td>
<td>281</td>
</tr>
</tbody>
</table>
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.

**Full list of authors and contributors** (alphabetic order)

- Martin Autzen  
  DTU NUTECH Center for Nuclear Technologies
- Christoph Burow  
  University of Cologne, Germany*
- Claire Christophe  
  IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne, France
- Michael Dietze  
  GFZ Helmholtz Centre Potsdam, Germany
- Julie Durcan  
  University of Oxford, United Kingdom
- Pierre Guibert  
  IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne, France
- Manfred Fischer  
  University of Bayreuth, Germany
- Margret C. Fuchs  
  Helmholtz-Zentrum Dresden-Rossendorf, Helmholtz-Institute Freiberg for Resource Technology, Freiberg
- Johannes Friedrich  
  Chair of Geomorphology, University of Bayreuth, Germany
- Guillaume Guérin  
  IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne, France
- Georgina E. King  
  University of Lausanne, Switzerland
- Sebastian Kreutzer  
  IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne, France
- Norbert Mercier  
  IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne, France
- Svenja Riedesel  
  Aberystwyth University, United Kingdom
- Christoph Schmidt  
  Chair of Geomorphology, University of Bayreuth, Germany
- Rachel K. Smedley  
  Liverpool University, United Kingdom
- Anne Philippe  
  Universite de Nantes and ANJA INRIA, Rennes, France
- Antoine Zink  
  C2RMF, Palais du Louvre, Paris, France
Supervisor of the initial version in 2012
Markus Fuchs, Justus-Liebig-University Giessen, Germany

Support contact
<developers@r-luminescence.org>

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

Bug reporting
* <developers@r-luminescence.org> or
* https://github.com/R-Lum/Luminescence/issues

Project website
* http://www.r-luminescence.org

Project source code repository
* https://github.com/R-Lum/Luminescence

Related package projects
* https://cran.r-project.org/package=RLumShiny
* https://cran.r-project.org/package=RLumModel

Package maintainer
Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS- Université Bordeaux Montaigne, France,
<sebastian.kreutzer@u-bordeaux-montaigne.fr>

Funding
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: "RLum.Network: Ein Wissenschaftsnetzwerk zur Analyse von Lumineszenzdaten mit R" (2014-2018)

Between 2014–2019, the work of Sebastian Kreutzer as maintainer of the package was supported by LabEx LaScArBxSK (ANR - n. ANR-10-LABX-52).

References


---

**analyse_Al2O3C_CrossTalk**

**Al2O3:C Reader Cross Talk Analysis**

**Description**

The function provides the analysis of cross-talk measurements on a FI lexsyg SMART reader using Al2O3:C chips.

**Usage**

```r
analyse_Al2O3C_CrossTalk(object, signal_integral = NULL, dose_points = c(0, 4), recordType = c("OSL (UVVIS)"), irradiation_time_correction = NULL, method_control = NULL, plot = TRUE, ...)
```

**Arguments**

- `object`: RLum.Analysis (required): measurement input
- `signal_integral`: numeric (optional): signal integral, used for the signal and the background. If nothing is provided the full range is used
- `dose_points`: numeric (with default): vector with dose points, if dose points are repeated, only the general pattern needs to be provided. Default values follow the suggestions made by Kreutzer et al., 2018
- `recordType`: character (with default): input curve selection, which is passed to function get_RLum. To deactivate the automatic selection set the argument to NULL
- `irradiation_time_correction`: numeric or RLum.Results (optional): information on the used irradiation time correction obtained by another experiments.
- `method_control`: list (optional): optional parameters to control the calculation. See details for further explanations
- `plot`: logical (with default): enable/disable plot output
- `...`: further arguments that can be passed to the plot output
analyse_Al2O3C_CrossTalk

Value

Function returns results numerically and graphically:

[ NUMERICAL OUTPUT ]

RLum.Results-object

slot: @data

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$data</td>
<td>data.frame</td>
<td>summed apparent dose table</td>
</tr>
<tr>
<td>$data_full</td>
<td>data.frame</td>
<td>full apparent dose table</td>
</tr>
<tr>
<td>$fit</td>
<td>lm</td>
<td>the linear model obtained from fitting</td>
</tr>
<tr>
<td>$col.seq</td>
<td>numeric</td>
<td>the used colour vector</td>
</tr>
</tbody>
</table>

slot: @info

The original function call

[ PLOT OUTPUT ]

- An overview of the obtained apparent dose values

Function version

0.1.2

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

References

analyse_Al2O3C_ITC

See Also

analyse_Al2O3C_ITC

Examples

```r
# load data
data(ExampleData.Al2O3C, envir = environment())

# run analysis
analyse_Al2O3C_CrossTalk(data_CrossTalk)
```

---

**analyse_Al2O3C_ITC**

*Al2O3 Irradiation Time Correction Analysis*

**Description**

The function provides a very particular analysis to correct the irradiation time while irradiating Al2O3:C chips in a luminescence reader.

**Usage**

```r
analyse_Al2O3C_ITC(object, signal_integral = NULL, dose_points = c(2, 4, 8, 12, 16), recordType = c("OSL (UVVIS)"), method_control = NULL, verbose = TRUE, plot = TRUE, ...)
```

**Arguments**

- **object** `RLum.Analysis` or `list` *(required)*: results obtained from the measurement. Alternatively a list of `RLum.Analysis` objects can be provided to allow an automatic analysis.
- **signal_integral** `numeric` *(optional)*: signal integral, used for the signal and the background. If nothing is provided the full range is used. Argument can be provided as list.
- **dose_points** `numeric` *(with default)*: vector with dose points, if dose points are repeated, only the general pattern needs to be provided. Default values follow the suggestions made by Kreutzer et al., 2018. Argument can be provided as list.
- **recordType** `character` *(with default)*: input curve selection, which is passed to function `get_RLum`. To deactivate the automatic selection set the argument to NULL
- **method_control** `list` *(optional)*: optional parameters to control the calculation. See details for further explanations
- **verbose** `logical` *(with default)*: enable/disable verbose mode
- **plot** `logical` *(with default)*: enable/disable plot output
- **...** further arguments that can be passed to the plot output
Details

Background: Due to their high dose sensitivity Al2O3:C chips are usually irradiated for only a very short duration or under the closed beta-source within a luminescence reader. However, due to its high dose sensitivity, during the movement towards the beta-source, the pellet already receives and non-negligible dose. Based on measurements following a protocol suggested by Kreutzer et al., 2018, a dose response curve is constructed and the intersection (absolute value) with the time axis is taken as real irradiation time.

method_control

To keep the generic argument list as clear as possible, arguments to allow a deeper control of the method are all preset with meaningful default parameters and can be handled using the argument method_control only, e.g., method_control = list(fit.method = "LIN"). Supported arguments are:

ARGUMENT FUNCTION DESCRIPTION
mode plot_GrowthCurve as in plot_GrowthCurve; sets the mode used for fitting
fit.method plot_GrowthCurve as in plot_GrowthCurve; sets the function applied for fitting

Value

Function returns results numerically and graphically:

[ NUMERICAL OUTPUT ]

RLum.Results-object

slot: @data

Element Type Description
$data data.frame correction value and error
$table data.frame table used for plotting
$table_mean data.frame table used for fitting
$fit lm or nls the fitting as returned by the function plot_GrowthCurve

slot: @info

The original function call

[ PLOT OUTPUT ]

• A dose response curve with the marked correction values

Function version

0.1.1
## How to cite


## Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

## References


## See Also

`plot_GrowthCurve`

## Examples

```r
# load data
data(ExampleData.Al2O3C, envir = environment())

# run analysis
analyse_Al2O3C_ITC(data_ITC)
```

## Description

The function provides the analysis routines for measurements on a FI lexsyg SMART reader using Al2O3:C chips according to Kreutzer et al., 2018

## Usage

```r
analyse_Al2O3C_Measurement(object, signal_integral = NULL, 
dose_points = c(0, 4), recordType = c("OSL (UVVIS)", "TL (UVVIS)")
, calculate_TL_dose = FALSE, irradiation_time_correction = NULL,
cross_talk_correction = NULL, travel_dosimeter = NULL,
test_parameters = NULL, verbose = TRUE, plot = TRUE, ...)
```
Arguments

- **object**: `RLum.Analysis` *(required)*: measurement input
- **signal_integral**: numeric *(optional)*: signal integral, used for the signal and the background. Example: `c(1:10)` for the first 10 channels. If nothing is provided the full range is used
- **dose_points**: numeric *(with default)*: vector with dose points, if dose points are repeated, only the general pattern needs to be provided. Default values follow the suggestions made by Kreutzer et al., 2018
- **recordType**: character *(with default)*: input curve selection, which is passed to function `get_RLum`. To deactivate the automatic selection set the argument to `NULL`
- **calculate_TL_dose**: logical *(with default)*: Enables/disables experimental dose estimation based on the TL curves. Taken is the ratio of the peak sums of each curves +/- 5 channels.
- **irradiation_time_correction**: numeric or `RLum.Results` *(optional)*: information on the used irradiation time correction obtained by another experiments. If a numeric is provided it has to be of length two: mean, standard error
- **cross_talk_correction**: numeric or `RLum.Results` *(optional)*: information on the used irradiation time correction obtained by another experiments. If a numeric vector is provided it has to be of length three: mean, 2.5 % quantile, 97.5 % quantile.
- **travel_dosimeter**: numeric *(optional)*: specify the position of the travel dosimeter (so far measured at the same time). The dose of travel dosimeter will be subtracted from all other values.
- **test_parameters**: list *(with default)*: set test parameters. Supported parameters are: `TL_peak_shift` All input: numeric values, NA and NULL (s. Details)
- **verbose**: logical *(with default)*: enable/disable verbose mode
- **plot**: logical *(with default)*: enable/disable plot output, if object is of type list, a numeric vector can be provided to limit the plot output to certain aliquots

Details

**Working with a travel dosimeter**

The function allows to define particular aliquots as travel dosimeters. For example: `travel_dosimeter = c(1,3,5)` sets aliquots 1, 3 and 5 as travel dosimeters. These dose values of this dosimeters are combined and automatically subtracted from the obtained dose values of the other dosimeters.

**Calculate TL dose**

The argument `calculate_TL_dose` provides the possibility to experimentally calculate a TL-dose, i.e. an apparent dose value derived from the TL curve ratio. However, it should be noted that this value is only a fallback in case something went wrong during the measurement of the optical
stimulation. The TL derived dose value is corrected for cross-talk and for the irradiation time, but not considered if a travel dosimeter is defined.

**Test parameters**

- **TL_peak_shift numeric** (default: 15):
  Checks whether the TL peak shift is bigger > 15 K, indicating a problem with the thermal contact of the chip.

- **stimulation_power numeric** (default: 0.05):
  So far available, information on the delievered optical stimulation are compared. Compared are the information from the first curves with all others. If the ratio differs more from unity than the defined by the threshold, a warning is returned.

**Value**

Function returns results numerically and graphically:

---

**[ NUMERICAL OUTPUT ]**

---

RLum.Results-object

**slot: @data**

**Element** | **Type**        | **Description**                             
-------------|-----------------|---------------------------------------------
$data$       | data.frame      | the estimated equivalent dose              
$data_table$ | data.frame      | full dose and signal table                 
$test_parameters$ | data.frame       | results with test parameters               
$data_TDcorrected$ | data.frame       | travel dosimeter corrected results (only if TD was provided)

*Note: If correction the irradiation time and the cross-talk correction method is used, the De values in the table data table are already corrected, i.e. if you want to get an uncorrected value, you can use the column CT_CORRECTION remove the correction*

**slot: @info**

The original function call

---

**[ PLOT OUTPUT ]**

---

- OSL and TL curves, combined on two plots.

**Function version**

0.2.4
Bayesian models (baSAR) applied on luminescence data

**How to cite**


**Author(s)**

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

**References**


**See Also**

analyse_Al2O3C_ITC

**Examples**

```r
# load data
data(ExampleData.Al2O3C, envir = environment())

# run analysis
analyse_Al2O3C_Measurement(data_CrossTalk)
```

**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

analyze_baSAR(object, XLS_file = NULL, aliquot_range = NULL, source_doserate = NULL, signal_integral, signal_integral.Tx = NULL, background_integral, background_integral.Tx = NULL, irradiation_times = 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, distribution_plot = "kde", plot = TRUE, plot_reduced = TRUE, plot.single = FALSE, verbose = TRUE, ...)

Arguments

object  
Risoe.BINfileData, RLum.Results, list of RLum.Analysis, 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/BINX-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). Alternatively a data.frame of similar structure can be provided.

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 measurement and its uncertainty in Gy/s, e.g., source_doserate = c(0.12, 0.04). Parameter 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, see 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.

**irradiation_times**

_numeric (optional): if set this vector replaces all irradiation times for one aliquot and one cycle (Lx and Tx curves) and recycles it for all others cycles and aliquots. Please note that if this argument is used, for every(!) single curve in the dataset an irradiation time needs to be set.

**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 error, 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): equation used for the fitting of the dose-response curve using the function plot_GrowthCurve and then for the Bayesian modelling. 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 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

**distribution_plot**

_character (with default): sets the final distribution plot that shows equivalent doses obtained using the frequentist approach and sets in the central dose as comparison obtained using baSAR. Allowed input is 'abanico' or 'kde'. If set to NULL nothing is plotted.

**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), readxl::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 hierarchical 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:

\[
\text{systematic.error} = \frac{1}{n} \sum SE(\text{source.doserate})
\]

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

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:

1. Select all valid aliquots using the function verify_SingleGrainData
2. Calculate Lx/Tx values using the function calc_OSLLxTxRatio
3. 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:

1. Calculate Lx/Tx values using the function `calc_OSLLxTxRatio`
2. 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower_centralD</td>
<td>numeric</td>
<td>sets the lower bound for the expected De range. Change it only if you know what you are doing!</td>
</tr>
<tr>
<td>upper_centralD</td>
<td>numeric</td>
<td>sets the upper bound for the expected De range. Change it only if you know what you are doing!</td>
</tr>
<tr>
<td>n.chains</td>
<td>integer</td>
<td>sets number of parallel chains for the model (default = 3) (cf. rjags::jags.model)</td>
</tr>
<tr>
<td>inits</td>
<td>list</td>
<td>option to set initialisation values (cf. rjags::jags.model)</td>
</tr>
<tr>
<td>thin</td>
<td>numeric</td>
<td>thinning interval for monitoring the Bayesian process (cf. rjags::jags.model)</td>
</tr>
<tr>
<td>variable.names</td>
<td>character</td>
<td>set the variables to be monitored during the MCMC run, default: 'central_D', 'sigma_D', 'D', 'Q', 'a', 'b', 'c', 'g'. Note: only variables present in the model can be monitored.</td>
</tr>
</tbody>
</table>

**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 function via the argument `baSAR_model` as character. The model has to be provided in the JAGS dialect of the BUGS language (cf. rjags::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):

```r
method_control = list(
  inits = list(
    list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 1),
    list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 2),
    list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 3)
  ))
```
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))

Q: The lines in the baSAR-model appear to be in a wrong logical order?

A: This is correct and allowed (cf. JAGS manual)

**Additional arguments support via the ... argument**

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

<table>
<thead>
<tr>
<th>Supported argument</th>
<th>Corresponding function</th>
<th>Default</th>
<th><strong>Short description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>verify_SingleGrainData</td>
<td>30</td>
<td>change rejection threshold for curve selection</td>
</tr>
<tr>
<td>sheet</td>
<td>readxl::read_excel</td>
<td>1</td>
<td>select XLS-sheet for import</td>
</tr>
<tr>
<td>col_names</td>
<td>readxl::read_excel</td>
<td>TRUE</td>
<td>first row in XLS-file is header</td>
</tr>
<tr>
<td>col_types</td>
<td>readxl::read_excel</td>
<td>NULL</td>
<td>limit import to specific columns</td>
</tr>
<tr>
<td>skip</td>
<td>readxl::read_excel</td>
<td>0</td>
<td>number of rows to be skipped during import</td>
</tr>
<tr>
<td>n.records</td>
<td>read_BIN2R</td>
<td>NULL</td>
<td>limit records during BIN-file import</td>
</tr>
<tr>
<td>duplicated.rm</td>
<td>read_BIN2R</td>
<td>TRUE</td>
<td>remove duplicated records in the BIN-file import</td>
</tr>
<tr>
<td>pattern</td>
<td>read_BIN2R</td>
<td>TRUE</td>
<td>select BIN-file by name pattern</td>
</tr>
<tr>
<td>position</td>
<td>read_BIN2R</td>
<td>NULL</td>
<td>limit import to a specific position</td>
</tr>
<tr>
<td>background.count.distribution</td>
<td>calc_OSLLxTxtRatio</td>
<td>&quot;non-poisson&quot;</td>
<td>set assumed count distribution</td>
</tr>
<tr>
<td>fit.weights</td>
<td>plot_GrowthCurve</td>
<td>TRUE</td>
<td>enables / disables fit weights</td>
</tr>
<tr>
<td>fit.bounds</td>
<td>plot_GrowthCurve</td>
<td>TRUE</td>
<td>enables / disables fit bounds</td>
</tr>
<tr>
<td>NumberIterations.MC</td>
<td>plot_GrowthCurve</td>
<td>100</td>
<td>number of MC runs for error calculation</td>
</tr>
<tr>
<td>output.plot</td>
<td>plot_GrowthCurve</td>
<td>TRUE</td>
<td>enables / disables dose response curve plot</td>
</tr>
<tr>
<td>output.plotExtended</td>
<td>plot_GrowthCurve</td>
<td>TRUE</td>
<td>enables / disables extended dose response curve plot</td>
</tr>
</tbody>
</table>

**Value**

Function returns results numerically and graphically:

<table>
<thead>
<tr>
<th>[ NUMERICAL OUTPUT ]</th>
</tr>
</thead>
</table>

**RLum.Results-object**

**slot:** @data

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$summary</td>
<td>data.frame</td>
<td>statistical summary, including the central dose</td>
</tr>
<tr>
<td>$mcmc</td>
<td>mcmc</td>
<td>coda::mcmc.list object including raw output</td>
</tr>
<tr>
<td>$models</td>
<td>character</td>
<td>implemented models used in the baSAR-model core</td>
</tr>
</tbody>
</table>
$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 removed NULL

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 (frequentist) approach and the central dose with the HPDs marked within. This figure is only provided for a comparison, no further statistical conclusion should be drawn from it.

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

Function version
0.1.33

How to cite

Note
If you provide more than one BIN-file, it is strongly recommended 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. , RLum Developer Team

References

Further reading

See Also
read_BIN2R, calc_OSLLxTxRatio, plot_GrowthCurve, readxl::read_excel, verify_SingleGrainData, rjags::jags.model, rjags::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",
)
**analyse_FadingMeasurement**

```r
plot = FALSE,
n.MCMC = 200
```

```r
print(results)
```

```r
###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_FadingMeasurement**

*Analyse fading measurements and returns the fading rate per decade (g-value)*

---

**Description**

The function analyses fading measurements and returns a fading rate including an error estimation. The function is not limited to standard fading measurements, as can be seen, e.g., Huntley and Lamothe 2001. Additionally, the density of recombination centres (rho') is estimated after Kars et al. 2008.

**Usage**

```r
analyse_FadingMeasurement(object, structure = c("Lx", "Tx"),
  signal.integral, background.integral, t_star = "half", n.MC = 100,
  verbose = TRUE, plot = TRUE, plot.single = FALSE, ...)
```

**Arguments**

- **object**: `RLum.Analysis` (*required*): input object with the measurement data. Alternatively, a list containing `RLum.Analysis` objects or a `data.frame` with three
columns (x = LxTx, y = LxTx error, z = time since irradiation) can be provided. Can also be a wide table, i.e., a data.frame with a number of columns divisible by 3 and where each triplet has the before mentioned column structure. **Please note:** The input object should solely consists of the curve needed for the data analysis, i.e., only IRSL curves representing Lx (and Tx).

If data from multiple aliquots are provided please see the details below with regard to Lx/Tx normalisation. The function assumes that all your measurements are related to one (comparable) sample. If you to treat independent samples, you have use this function in a loop.

**structure** character *(with default)*: sets the structure of the measurement data. Allowed are 'Lx' or c('Lx', 'Tx'). Other input is ignored.

**signal.integral** vector *(required)*: vector with the limits for the signal integral. Not required if a data.frame with LxTx values are provided.

**background.integral** vector *(required)*: vector with the bounds for the background integral. Not required if a data.frame with LxTx values are provided.

**t_star** character *(with default)*: method for calculating the time elapsed since irradiation. Options are: ‘half’, which is \( t_{star} := t_{1} + (t_{2} - t_{1})/2 \) (Auclair et al., 2003) and ‘end’, which takes the time between irradiation and the measurement step. Default is ‘half’.

**n.MC** integer *(with default)*: number for Monte Carlo runs for the error estimation.

**verbose** logical *(with default)*: enables/disables verbose mode.

**plot** logical *(with default)*: enables/disables plot output.

**plot.single** logical *(with default)*: enables/disables single plot mode, i.e., one plot window per plot. Alternatively a vector specifying the plot to be drawn, e.g., plot.single = c(3,4) draws only the last two plots.

*(optional)* further arguments that can be passed to internally used functions (see details).

**Details**

All provided output corresponds to the \( tc \) value obtained by this analysis. Additionally in the output object the g-value normalised to 2-days is provided. The output of this function can be passed to the function `calc_FadingCorr`.

**Fitting and error estimation**

For the fitting the function `stats::lm` is used without applying weights. For the error estimation all input values, except \( tc \), as the precision can be considered as sufficiently high enough with regard to the underlying problem, are sampled assuming a normal distribution for each value with the value as the mean and the provided uncertainty as standard deviation.

**Density of recombination centres**

The density of recombination centres, expressed by the dimensionless variable \( \rho' \), is estimated by fitting equation 5 in Kars et al. 2008 to the data. For the fitting the function `stats::nls` is used without applying weights. For the error estimation the same procedure as for the g-value is applied (see above).
Multiple aliquots & Lx/Tx normalisation

Be aware that this function will always normalise all Lx/Tx values by the Lx/Tx value of the prompt measurement of the first aliquot. This implicitly assumes that there are no systematic inter-aliquot variations in Lx/Tx values. If deemed necessary to normalise the Lx/Tx values of each aliquot by its individual prompt measurement please do so **before** running `analyse_FadingMeasurement` and provide the already normalised values for object instead.

Value

An `RLum.Results` object is returned:

Slot: `@data`

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>fading_results</td>
<td>data.frame</td>
<td>results of the fading measurement in a table</td>
</tr>
<tr>
<td>fit</td>
<td>lm</td>
<td>object returned by the used linear fitting function <code>stats::lm</code></td>
</tr>
<tr>
<td>rho_prime</td>
<td>data.frame</td>
<td>results of rho' estimation after Kars et al. 2008</td>
</tr>
<tr>
<td>LxTx_table</td>
<td>data.frame</td>
<td>Lx/Tx table, if curve data had been provided</td>
</tr>
<tr>
<td>irr.times</td>
<td>integer</td>
<td>vector with the irradiation times in seconds</td>
</tr>
</tbody>
</table>

Slot: `@info`

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>call</td>
<td>the original function call</td>
</tr>
</tbody>
</table>

Function version

0.1.11

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France)
Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


See Also

calc_OSLLxTxRatio, read_BIN2R, read_XSYG2R, extract_IrradiationTimes

Examples

```r
## load example data (sample UNIL/NB123, see ?ExampleData.Fading)
data("ExampleData.Fading", envir = environment())

## (1) get fading measurement data (here a three column data.frame)
fading_data <- ExampleData.Fading$fading.data$IR50

## (2) run analysis
g_value <- analyse_FadingMeasurement(
fading_data,
   plot = TRUE,
   verbose = TRUE,
   n.MC = 10)

## (3) this can be further used in the function
## to correct the age according to Huntley & Lamothe, 2001
results <- calc_FadingCorr(
   age.faded = c(100,2),
   g_value = g_value,
   n.MC = 10)
```

---

**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**

```r
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 chosen 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 (see Details)

n.MC numeric (with default): set number of Monte Carlo runs for start parameter estimation (method = "FIT") or error estimation (method = "SLIDE"). This value can be set to NULL to skip the MC runs. 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

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 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)):

\[
\phi(D) = \phi_0 - \Delta \phi(1 - e^{(-\lambda \times D)})^\beta
\]

with \(\phi(D)\) the dose dependent IR-RF flux, \(\phi_0\) the initial IR-RF flux, \(\Delta \phi\) the dose dependent change of the IR-RF flux, \(\lambda\) the exponential parameter, \(D\) the dose and \(\beta\) the dispersive factor.

To obtain the palaeodose \(D_e\) the function is changed to:

\[
D_e = \ln\left(\frac{-(\phi(D) - \phi_0)}{(-\lambda \times \phi)^{1/\beta} + 1}\right) - \lambda
\]

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

**method = "SLIDE"**

For this method, the natural curve is slide along the x-axis until congruence with the regenerated curve is reached. Instead of fitting this allows working with the original data without the need for 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. For the mathematical details of the implementation see Frouin et al., 2017

**method.control**

To keep the generic argument list as clear as possible, arguments to control the methods for \(D_e\) 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:

<table>
<thead>
<tr>
<th>ARGUMENT</th>
<th>METHOD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>FIT, SLIDE</td>
<td>as in nls; shows sum of squared residuals</td>
</tr>
<tr>
<td>trace_vslide</td>
<td>SLIDE</td>
<td>logical argument to enable or disable the tracing of the vertical sliding</td>
</tr>
<tr>
<td>maxiter</td>
<td>FIT</td>
<td>as in nls</td>
</tr>
<tr>
<td>warnOnly</td>
<td>FIT</td>
<td>as in nls</td>
</tr>
<tr>
<td>minFactor</td>
<td>FIT</td>
<td>as in nls</td>
</tr>
<tr>
<td>correct_onset</td>
<td>SLIDE</td>
<td>The logical argument shifts the curves along the x-axis by the first channel, as light is expected</td>
</tr>
<tr>
<td>show_density</td>
<td>SLIDE</td>
<td>logical (with default) enables or disables KDE plots for MC run results. If the distribution is</td>
</tr>
<tr>
<td>show_fit</td>
<td>SLIDE</td>
<td>logical (with default) enables or disables the plot of the fitted curve routinely obtained during</td>
</tr>
<tr>
<td>n.MC</td>
<td>SLIDE</td>
<td>integer (with default): This controls the number of MC runs within the sliding (assessing the</td>
</tr>
<tr>
<td>vslide_range</td>
<td>SLIDE</td>
<td>logical or numeric or character (with default): This argument sets the boundaries for a vertical</td>
</tr>
<tr>
<td>cores</td>
<td>SLIDE</td>
<td>number or character (with default): set number of cores to be allocated for a parallel process</td>
</tr>
</tbody>
</table>
**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 of $RF_{nat}$ 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 minpack.lm::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

The function returns numerical output and an (optional) plot.

[ NUMERICAL OUTPUT ]

RLum.Results-object

slot: @data

[.:. $data : data.frame]

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>numeric</td>
<td>the obtained equivalent dose</td>
</tr>
<tr>
<td>DE.ERROR</td>
<td>numeric</td>
<td>(only method = &quot;SLIDE&quot;) standard deviation obtained from MC runs</td>
</tr>
<tr>
<td>DE.LOWER</td>
<td>numeric</td>
<td>2.5% quantile for De values obtained by MC runs</td>
</tr>
<tr>
<td>DE.UPPER</td>
<td>numeric</td>
<td>97.5% quantile for De values obtained by MC runs</td>
</tr>
<tr>
<td>DE.STATUS</td>
<td>character</td>
<td>test parameter status</td>
</tr>
<tr>
<td>RF_NAT.LIM</td>
<td>character</td>
<td>used RF_nat curve limits</td>
</tr>
<tr>
<td>RF_REG.LIM</td>
<td>character</td>
<td>used RF_reg curve limits</td>
</tr>
<tr>
<td>POSITION</td>
<td>integer</td>
<td>(optional) position of the curves</td>
</tr>
<tr>
<td>DATE</td>
<td>character</td>
<td>(optional) measurement date</td>
</tr>
<tr>
<td>SEQUENCE_NAME</td>
<td>character</td>
<td>(optional) sequence name</td>
</tr>
<tr>
<td>UID</td>
<td>character</td>
<td>unique data set ID</td>
</tr>
</tbody>
</table>

[.:. $De.MC : numeric]
A numeric vector with all the De values obtained by the MC runs.

[.:. $test_parameters : data.frame]

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>numeric</td>
<td>aliquot position</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>character</td>
<td>test parameter name</td>
</tr>
<tr>
<td>THRESHOLD</td>
<td>numeric</td>
<td>set test parameter threshold value</td>
</tr>
<tr>
<td>VALUE</td>
<td>numeric</td>
<td>the calculated test parameter value (to be compared with the threshold)</td>
</tr>
<tr>
<td>STATUS</td>
<td>character</td>
<td>test parameter status either &quot;OK&quot; or &quot;FAILED&quot;</td>
</tr>
<tr>
<td>SEQUENCE_NAME</td>
<td>character</td>
<td>name of the sequence, so far available</td>
</tr>
<tr>
<td>UID</td>
<td>character</td>
<td>unique data set ID</td>
</tr>
</tbody>
</table>

[.:. $fit : data.frame]
An nls object produced by the fitting.

[.:. $slide : list]
A list with data produced during the sliding. Some elements are previously reported with the summary object data. List elements are:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>De</td>
<td>numeric</td>
<td>the final De obtained with the sliding approach</td>
</tr>
<tr>
<td>De.MC</td>
<td>numeric</td>
<td>all De values obtained by the MC runs</td>
</tr>
<tr>
<td>residuals</td>
<td>numeric</td>
<td>the obtained residuals for each channel of the curve</td>
</tr>
<tr>
<td>trend.fit</td>
<td>lm</td>
<td>fitting results produced by the fitting of the residuals</td>
</tr>
<tr>
<td>RF_nat.slided</td>
<td>matrix</td>
<td>the slided RF_nat curve</td>
</tr>
<tr>
<td>t_n.id</td>
<td>numeric</td>
<td>the index of the t_n offset</td>
</tr>
<tr>
<td>I_n</td>
<td>numeric</td>
<td>the vertical intensity offset if a vertical slide was applied</td>
</tr>
<tr>
<td>algorithm_error</td>
<td>numeric</td>
<td>the vertical sliding suffers from a systematic effect induced by the used algorithm. The returned value is the standard error of the residuals while expanding the vertical sliding range. I can be added as systematic error to the final De error; so far wanted.</td>
</tr>
<tr>
<td>vslide_range</td>
<td>numeric</td>
<td>the range used for the vertical sliding</td>
</tr>
<tr>
<td>squared_residuals</td>
<td>numeric</td>
<td>the squared residuals (horizontal sliding)</td>
</tr>
</tbody>
</table>

**slot:** @info

The original function call (methods::language-class-object)

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

---

**[ PLOT OUTPUT ]**

The slided IR-RF curves with the finally obtained De

**Function version**

0.7.5

**How to cite**


**Note**

This function assumes that there is no sensitivity change during the measurements (natural vs. regenerated signal), which is in contrast to the findings by Buylaert et al. (2012).

**Author(s)**

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team
References


See Also

RLum.Analysis, RLum.Results, get_RLum, nls, minpack.lm::nlsLM, parallel::mclapply

Examples

```r
#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 parameter results
get_RLum(results, data.object = "data")
get_RLum(results, data.object = "test_parameters")
```
analyse_pIRIRSequence

##(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 measurement 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 or list of RLum.Analysis objects (required): input object containing data for analysis. If a list is provided the functions tries to iterate 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 none 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. Furthermore, the arguments main (headers), log (IRSL curves), cex (control the size) and mtext.outer (additional text on the plot area) can be passed to influence the plotting. If the input is list, main can be passed as vector or list.

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.

**Defining the sequence structure**

The argument sequence.structure expects a shortened pattern of your sequence structure and was mainly introduced to ease the use of the function. For example: If your measurement data contains the following curves: TL, IRSL, IRSL, TL, IRSL, TL, the sequence pattern in sequence.structure becomes c(TL', 'IRSL', 'IRSL'). The second part of your sequence for one cycle should be similar and can be discarded. If this is not the case (e.g., additional hotbleach) such curves have to be removed before using the function.

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:

<table>
<thead>
<tr>
<th>DATA.OBJECT</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>..$data:</td>
<td>data.frame</td>
<td>Table with De values</td>
</tr>
<tr>
<td>..$LnLxTnTx.table:</td>
<td>data.frame</td>
<td>with the LnLxTnTx values</td>
</tr>
<tr>
<td>..$rejection.criteria:</td>
<td>data.frame</td>
<td>rejection criteria</td>
</tr>
<tr>
<td>..$Formula:</td>
<td>list</td>
<td>Function used for fitting of the dose response curve</td>
</tr>
<tr>
<td>..$call:</td>
<td>call</td>
<td>the original function call</td>
</tr>
</tbody>
</table>
The output should be accessed using the function `get_RLum`.

Function version

0.2.4

How to cite


Note

Best graphical output can be achieved by using the function `pdf` with the following options:

```r
df(file = "<YOUR FILENAME>", height = 15, width = 15)
```

Author(s)

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

References


See Also

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

Examples

```r
### NOTE: For this example existing example data are used. These data are non pIRIR data.
###
## 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:
tempfile <- tempfile(fileext = ".pdf")
pdf(file = tempfile, 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)
Description
The function analyses CW-OSL curve data produced by a SUERC portable OSL reader and pro-
duces a combined plot of OSL/IRSL signal intensities, OSL/IRSL depletion ratios and the IRSL/OSL
ratio.

Usage
analyse_portableOSL(object, signal.integral, invert = FALSE,
    normalise = FALSE, plot = TRUE, ...)

Arguments
object RLum.Analysis (required): RLum.Analysis object produced by read_PSL2R.
signal.integral vector (required): A vector of two values specifying the lower and upper chan-
nel used to calculate the OSL/IRSL signal. Can be provided in form of c(1,5) or 1:5.
invert logical (with default): TRUE to calculate and plot the data in reverse order.
normalise logical (with default): TRUE to normalise the OSL/IRSL signals by the mean of
all corresponding data curves.
plot logical (with default): enable/disable plot output
... currently not used.

Details
This function only works with RLum.Analysis objects produced by read_PSL2R. It further as-
sumes (or rather requires) an equal amount of OSL and IRSL curves that are pairwise combined for
calculating the IRSL/OSL ratio. For calculating the depletion ratios the cumulative signal of the last
n channels (same number of channels as specified by signal.integral) is divided by cumulative
signal of the first n channels (signal.integral).

Value
Returns an S4 RLum.Results object.

Function version
0.0.3

How to cite
Burow, C., 2019. analyse_portableOSL(): Analyse portable CW-OSL measurements. Function
version 0.0.3. In: Kreutzer, S., Burow, C., Dietze, M., Fuchs, M.C., Schmidt, C., Fischer, M.,
Friedrich, J., 2019. Luminescence: Comprehensive Luminescence Dating Data AnalysisR package
version 0.9.5. https://CRAN.R-project.org/package=Luminescence

Author(s)
Christoph Burow, University of Cologne (Germany), RLum Developer Team
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, onlyLxTxTable = 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.
analyse_SAR.CWOSL

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 = list(recycling.ratio = 10). 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 containing 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 numeric 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.

onlyLxTxTable
logical (**with default**): If TRUE the dose response curve fitting and plotting is skipped. This allows to get hands on the Lx/Tx table for large datasets without the need for a curve fitting.

... 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 useful 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 length. Function output will be just one single RLum.Results object.

Please be careful when using this option. It may allow a fast and 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 function 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:

- **data** 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 be 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**

0.8.8
How to cite


Note

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

The function currently does support only 'OSL', 'IRSL' and 'POSL' data!

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS-Université Bordeaux Montaigne (France), RLum Developer Team

References


See Also

calc.OSLLxTxRatio, plot_GrowthCurve, RLum.Analysis, RLum.Results, get_RLum

Examples

```r
# 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,
)```
Analyse_SAR.OSLdata

**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**

```r
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 (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.
Analyse_SAR.OSLdata

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 for allowed data types.
keep.SEL logical (default): option allowing to use the SEL element of the Risoe.BINfileData 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

[recycling 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

How to cite

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), RLum Developer Team

References

See Also
calc_OSLLxTxRatio, Risoe.BINfileData, read_BIN2R, plot_GrowthCurve

Examples
```r
## 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

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, integral_input = "channel", 
sequence.structure = c("PREHEAT", "SIGNAL", "BACKGROUND"), 
rejection.criteria = list(recycling.ratio = 10, recuperation.rate = 10), dose.points, log = "", ...)

Arguments

object RLum.Analysis or a list of such objects (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)
inintegral_input character (with default): defines the input for the the arguments signal.integral.min and signal.integral.max. These limits can be either provided 'channel' number (the default) or 'temperature'. If 'temperature' is chosen the best matching channel is selected.
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

[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)

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
LnLxTxNnTx.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

0.3.0

How to cite


Note

THIS IS A BETA VERSION
None TL curves will be removed from the input object without further warning.
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 RLum.Data.Spectrum objects and such objects embedded in list or RLum.Analysis objects.

Usage

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

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CRNS-Universite Bordeaux Montaigne (France), RLum Developer Team

References


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

Examples

```r
# 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 = object,
  signal.integral.min = 210,
  signal.integral.max = 220,
  fit.method = "EXP OR LIN",
  sequence.structure = c("SIGNAL", "BACKGROUND"))
```
Arguments

object: \texttt{RLum.Data.Spectrum} or \texttt{RLum.Analysis} \textbf{(required)}: input object to be treated. This can be also provided as \texttt{list}. If an \texttt{RLum.Analysis} object is provided, only the \texttt{RLum.Data.Spectrum} objects are treated. Please note: this mixing of objects do not work for a list of \texttt{RLum.Data} objects.

method: \texttt{character} \textbf{(with default)}: Defines method that is applied for cosmic ray removal. Allowed methods are \texttt{smooth}, the default, (\texttt{smooth}), \texttt{smooth.spline} (\texttt{smooth.spline}) and \texttt{Pych}. See details for further information.

method.Pych.smoothing: \texttt{integer} \textbf{(with default)}: Smoothing parameter for cosmic ray removal according to \texttt{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: \texttt{numeric} \textbf{(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: \texttt{integer} \textbf{(with default)}: on which part the function cosmic ray removal should be applied on:

\begin{itemize}
  \item 1 = along the time axis (line by line),
  \item 2 = along the wavelength axis (column by column).
\end{itemize}

Note: This argument currently only affects the methods \texttt{smooth} and \texttt{smooth.spline}.

verbose: \texttt{logical} \textbf{(with default)}: Option to suppress terminal output.

plot: \texttt{logical} \textbf{(with default)}: If \texttt{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 \texttt{method = "Pych"} a graphical output is provided.

... further arguments and graphical parameters that will be passed to the \texttt{smooth} function.

Details

method = "Pych"

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

\begin{itemize}
  \item For interpolation between neighbouring values the median and not the mean is used.
  \item The number of breaks to construct the histogram is set to: \texttt{length(number.of.input.values)/2}
\end{itemize}

For further details see references below.

method = "smooth"

Method uses the function \texttt{smooth} to remove cosmic rays.

Arguments that can be passed are: \texttt{kind}, \texttt{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.

**Function version**
0.3.0

**How to cite**

**Author(s)**
Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France) , RLum Developer Team

**References**

**See Also**

**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**

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

**Arguments**

- `object` \texttt{RLum.Data.Spectrum} or \texttt{RLum.Analysis} (required): S4 object of class \texttt{RLum.Data.Spectrum}, \texttt{RLum.Analysis} or a list of such objects. Other objects in the list are skipped.
- `spectral.efficiency` \texttt{data.frame} (required): Data set containing wavelengths (x-column) and relative spectral response values (y-column) (values between 0 and 1). The provided data will be used to correct all spectra if object is a list.

**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 \texttt{stats::approx}.

If the energy calibration differs for both data sets, \texttt{NA} values are produced that will be removed from the matrix.

**Value**

Returns same object as provided as input.

**Function version**

0.2.0

**How to cite**

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, UMR 5060, CNRS-Université Bordeaux Montaigne (France)
Johannes Friedrich, University of Bayreuth (Germany), RLum Developer Team

See Also

RLum.Data.Spectrum, RLum.Analysis

Examples

```r
##(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 RLumShiny::app_RLum from the package RLumShiny::RLumShiny-package. For further details and examples please see the manual of this package.

**Usage**

`app_RLum(app = NULL, ...)`

**Arguments**

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

**Function version**

0.1.1
How to cite

Author(s)
Christoph Burow, University of Cologne (Germany), RLum Developer Team

as

as() - RLum-object coercion

Description
for [RLum.Analysis-class]
for [RLum.Data.Curve-class]
for [RLum.Data.Image-class]
for [RLum.Data.Spectrum-class]
for [RLum.Results-class]

Arguments
from RLum, 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 consists of RLum.Analysis objects.

RLum.Data.Curve

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

RLum.Data.Image

from to
**BaseDataSet.ConversionFactors**  

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**  
methods::as

---

**BaseDataSet.ConversionFactors**  

*Base data set of dose-rate conversion factors*

**Description**  
Collection of published dose-rate conversion factors to convert concentrations of radioactive isotopes to dose rate values.

**Format**  
A list with three elements with dose-rate conversion factors sorted by article and radiation type (alpha, beta, gamma):

- AdamiecAitken1998: Conversion factors from Tables 5 and 6
- Guerinetal2011: Conversion factors from Tables 1, 2 and 3
- Liritzisetal2013: Conversion factors from Tables 1, 2 and 3
Version
0.1

Source
All gamma conversion factors were carefully read from the tables given in the references above.

References

Examples
```r
## Load data
data("BaseDataSet.ConversionFactors")
```

---

**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 the "AGE" program by Rainer Gruen (cf. Gruen 2009). These data essentially reproduce the graph shown in Fig. 1 of Prescott & Hutton (1988).
- **values.factor.Alitude**: data frame containing altitude factors for adjusting geomagnetic field-change factors. Values were read from Fig. 1 in Prescott & Hutton (1994).
- **values.par.FJH**: data frame containing values for parameters F, J and H (read from Fig. 2 in Prescott & Hutton 1994)

\[
D_c = D_0 \ast (F + J \ast \exp((\text{altitude}/1000)/H))
\]

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:

\[ y = 2 \times 10^{-6} \times x^2 - 0.0008 \times x + 0.2535 \]

(For depths <40 g cm^-2)

\[ y = -6 \times 10^{-8} \times x^3 + 2 \times 10^{-5} \times x^2 - 0.0025 \times x + 0.2969 \]

values.factor.Altitude

Page: 499
Figure: 1
Fit: 2-degree polynomial in the form of

\[ y = -0.026 \times x^2 + 0.6628 \times x + 1.0435 \]

values.par.FJH

Page: 500
Figure: 2
Fits: 3-degree polynomials and linear fits

F (non-linear part, \( \lambda < 36.5 \) deg.):

\[ y = -7 \times 10^{-7} \times x^3 - 8 \times 10^{-5} \times x^2 - 0.0009 \times x + 0.3988 \]

F (linear part, \( \lambda > 36.5 \) deg.):

\[ y = -0.0001 \times x + 0.2347 \]

J (non-linear part, \( \lambda < 34 \) deg.):

\[ y = 5 \times 10^{-6} \times x^3 - 5 \times 10^{-5} \times x^2 + 0.0026 \times x + 0.5177 \]
J (linear part, $\lambda > 34$ deg.):

$$y = 0.0005 * x + 0.7388$$

H (non-linear part, $\lambda < 36$ deg.):

$$y = -3 \times 10^{-6} * x^3 - 5 \times 10^{-5} * x^2 - 0.0031 * x + 4.398$$

H (linear part, $\lambda > 36$ deg.):

$$y = 0.0002 * x + 4.0914$$

References


Examples

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

**BaseDataSet.FractionalGammaDose**

*Base data set of fractional gamma-dose values*

**Description**

Collection of (un-)published fractional gamma dose-rate values to scale the gamma-dose rate considering layer-to-layer variations in soil radioactivity.

**Format**

A list with fractional gamma dose-rate values sorted by article:

- **Aitken1985**: Fractional gamma-dose values from table H.1

**Version**

0.1
Fractional gamma dose values were carefully read from the tables given in the references above.

References


Examples

```r
## Load data
data("BaseDataSet.FractionalGammaDose")
```

---

**bin_RLum.Data**

**Channel binning - method dispatcher**

**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 specific 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**

0.2.0
How to cite

Note
Currently only RLum.Data objects of class RLum.Data.Curve and RLum.Data.Spectrum are supported!

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5050, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

See Also

Examples

```r
##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

---

**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 (required): empirical value for mean packing density. If packing.density = "Inf" a hexagonal structure on an infinite plane with a packing density of 0.906... 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:

\[ n = \frac{\pi * x^2}{\pi * y^2} * d \]

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 \(\pi/\sqrt{12} = 0.9068...\) 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 \(\mu_x, \mu_y, \mu_d\) 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 \( \mu_y = \text{mean.grain.size} \) and \( \sigma_y = (\text{max.grain.size} - \text{min.grain.size})/4 \) so that 95% of all grains are within the provided 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 \( \mu = \text{sample.diameter} \) 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 aliquots 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 \( \text{MC.iter} \) (see ...). The results are visualised in a bar- and boxplot together with a statistical summary.

**Value**

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

- \$.summary \textbf{data.frame} summary of all relevant calculation results.
- \$.args \textbf{list} used arguments
- \$.call \textbf{call} the function call
- \$.MC \textbf{list} results of the Monte Carlo simulation

The output should be accessed using the function \textbf{get_RLum}.

**Function version**

0.31

**How to cite**


**Author(s)**

Christoph Burow, University of Cologne (Germany), RLum Developer Team
calc_AverageDose

References


Further reading


Examples

```r
## 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_AverageDose

Calculate the Average Dose and the dose rate dispersion

Description

This functions calculates the Average Dose and their extrinsic dispersion and estimates the standard errors by bootstrapping based on the Average Dose Model by Guerin et al., 2017

Usage

```r
calc_AverageDose(data, sigma_m = NULL, Nb_BE = 500, na.rm = TRUE,
plot = TRUE, verbose = TRUE, ...)
```

Arguments

data

RLum.Results or data.frame (required): for data.frame: two columns with De (data[,1]) and De error (values[,2])

sigma_m

numeric (required): the overdispersion resulting from a dose recovery experiment, i.e. when all grains have received the same dose. Indeed in such a case, any overdispersion (i.e. dispersion on top of analytical uncertainties) is, by definition, an unrecognised measurement uncertainty.
calc_AverageDose

###Nb_BE
integer (with default): sample size used for the bootstrapping

###na.rm
logical (with default): exclude NA values from the data set prior to any further operation.

###plot
logical (with default): enables/disables plot output

###verbose
logical (with default): enables/disables terminal output

... further arguments that can be passed to `graphics::hist`. As three plots are returned all arguments need to be provided as list, e.g., `main = list("Plot 1","Plot 2","Plot 3")`. Note: not all arguments of `hist` are supported, but the output of `hist` is returned and can be used of own plots.

Further supported arguments: `mtext` (character), `rug` (TRUE/FALSE).

###Details

####sigma_m

The program requires the input of a known value of sigma_m, which corresponds to the intrinsic overdispersion, as determined by a dose recovery experiment. Then the dispersion in doses (sigma_d) will be that over and above sigma_m (and individual uncertainties sigma_wi).

###Value

The function returns numerical output and an (optional) plot.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVERAGE_DOSE</td>
<td>numeric</td>
<td>the obtained average dose</td>
</tr>
<tr>
<td>AVERAGE_DOSE.SE</td>
<td>numeric</td>
<td>the average dose error</td>
</tr>
<tr>
<td>SIGMA_D</td>
<td>numeric</td>
<td>sigma</td>
</tr>
<tr>
<td>SIGMA_D.SE</td>
<td>numeric</td>
<td>standard error of the sigma</td>
</tr>
<tr>
<td>IC_AVERAGE_DOSE.LEVEL</td>
<td>character</td>
<td>confidence level average dose</td>
</tr>
<tr>
<td>IC_AVERAGE_DOSELOWER</td>
<td>character</td>
<td>lower quantile of average dose</td>
</tr>
<tr>
<td>IC_AVERAGE_DOSE.UPPER</td>
<td>character</td>
<td>upper quantile of average dose</td>
</tr>
<tr>
<td>IC_SIGMA_D.LEVEL</td>
<td>integer</td>
<td>confidence level sigma</td>
</tr>
<tr>
<td>IC_SIGMA_D.LOWER</td>
<td>character</td>
<td>lower sigma quantile</td>
</tr>
<tr>
<td>IC_SIGMA_D.UPPER</td>
<td>character</td>
<td>upper sigma quantile</td>
</tr>
<tr>
<td>L_MAX</td>
<td>character</td>
<td>maximum likelihood value</td>
</tr>
</tbody>
</table>

---

**RLum.Results-object**

**slot:** @data

[.. $summary : data.frame]
calc_AverageDose

[ .. $dstar : matrix ]
Matrix with bootstrap values

[ .. $hist : list ]
Object as produced by the function histogram

[ PLOT OUTPUT ]

The function returns two different plot panels.
(1) An abanico plot with the dose values
(2) A histogram panel comprising 3 histograms with the equivalent dose and the bootstrapped average dose and the sigma values.

Function version
0.1.4

How to cite

Note
This function has beta status!

Author(s)
Claire Christophe, IRAMAT-CRP2A, Universite de Nantes (France), Anne Philippe, Universite de Nantes, (France), Guillaume Guerin, IRAMAT-CRP2A, Universite Bordeaux Montaigne, (France), Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne, (France), RLum Developer Team

References

Further reading
See Also

read.table, graphics::hist

Examples

```r
##Example 01 using package example data
##load example data
data(ExampleData.DeValues, envir = environment())

##calculate Average dose
##(use only the first 56 values here)
AD <- calc_AverageDose(ExampleData.DeValues$CA1[1:56,], sigma_m = 0.1)

##plot De and set Average dose as central value
plot_AbanicoPlot(
  data = ExampleData.DeValues$CA1[1:56,],
  z.0 = AD$summary$AVERAGE_DOSE)
```

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, na.rm = FALSE, plot = TRUE, ...)

Arguments

data `RLum.Results` or `data.frame` *(required)*: for `data.frame`: two columns with De (data[,1]) and De error (data[,2])
sigmab `numeric` *(default)*: additional spread in De values. This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100). **NOTE**: For the logged model (`log = TRUE`) this value must be a fraction, e.g. 0.2 (= 20 %). If the un-logged model is used (`log = FALSE`), sigmab must be provided in the same absolute units of the De values (seconds or Gray).

log `logical` *(default)*: fit the (un-)logged central age model to De data

na.rm `logical` *(default)*: strip NA values before the computation proceeds

plot `logical` *(default)*: plot output

... further arguments (`trace`, `verbose`).
calc_CentralDose

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 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
- \$.profile: data.frame the log likelihood profile for \( \sigma \)

The output should be accessed using the function get_RLum

Function version
1.4.0

How to cite

Author(s)
Christoph Burow, University of Cologne (Germany)
Based on a rewritten S script of Rex Galbraith, 2010, RLum Developer Team

References
Further reading


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

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

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

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 (data[,2])
calc_CommonDose

sigmab numeric (with default): additional spread in De values. This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100). **NOTE**: For the logged model (log = TRUE) this value must be a fraction, e.g. 0.2 (= 20 %). If the un-logged model is used (log = FALSE), sigmab must be provided in the same absolute units of the De values (seconds or Gray).

log logical (with default): fit the (un-)logged central 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

0.1.1

How to cite


Author(s)

Christoph Burow, University of Cologne (Germany), RLum Developer Team
References


Further reading


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

```r
## 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.
**calc_CosmicDoseRate**

**Usage**

```r
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
- `...` further arguments (verbose to disable/enable console output).

**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)

   \[ \text{absorber} = \text{depth}_1 \times \text{density}_1 + \text{depth}_2 \times \text{density}_2 + ... + \text{depth}_n \times \text{density}_n \]

2. If `half.depth = TRUE`

   \[ \text{absorber} = \text{absorber}/2 \]

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)
\[ D_0 = C/((absorber + d)^a + a) \times (absorber + H) \times \exp(-B \times absorber) \]

b) If absorber is < 167 g/cm² (soft- and hard-component): derive \( D_0 \) from Fig. 1 in Prescott & Hutton (1988).


\[ \lambda = \arcsin(0.203 \times \cos(latitude) \times \cos(longitude - 291) + 0.979 \times \sin(latitude)) \]

(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.

\[ D_c = D_0 \times (F + J \times \exp((altitude/1000)/H)) \]

(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.

\[ D_c' = D_c \times correctionFactor \]

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) and 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/cm³.

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

How to cite


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 $10^4$ kg/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 $10-10^4$ kg/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

$$D = 0.21 \times \exp(-0.070 \times \text{absorber} + 0.0005 \times \text{absorber}^2)$$

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" program (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), RLum Developer Team

References


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,
calc_FadingCorr

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, interval = c(0.01, 500), 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
- interval numeric (with default): a vector containing the end-points (age interval) of the interval to be searched for the root in 'ka'. This argument is passed to the function stats::uniroot used for solving the equation.
- txtProgressBar logical (with default): enables or disables txtProgressBar
- verbose logical (with default): enables or disables terminal output
Details

As the g-value slightly 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 recalculated to tc of the measurement used for estimating the age applying the following equation:

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

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 \texttt{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 resources and it may take a while, (b) the length (size) of the output vector \texttt{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.

\texttt{seed}

This option allows to recreate previously calculated results by setting the seed for the R random number generator (see \texttt{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
calc_FadingCorr

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

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>age.corr</td>
<td>data.frame</td>
<td>Corrected age</td>
</tr>
<tr>
<td>age.corr.MC</td>
<td>numeric</td>
<td>MC simulation results with all possible ages from that simulation</td>
</tr>
</tbody>
</table>

Slot: @info

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>character</td>
<td>the original function call</td>
</tr>
</tbody>
</table>

Function version

0.4.2

How to cite


Note

Special thanks to Sebastien Huot for his support and clarification via e-mail.

Author(s)

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

References


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 = 1728000,
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 = 1728000,
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 = 1728000,
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, Ch_L2 = NULL,
Ch_L3 = NULL, x = 1, x2 = 0.1, dead.channels = c(0, 0),
fitCW.sigma = FALSE, fitCW.curve = FALSE, plot = TRUE, ...)

calc_FastRatio

Arguments

object: RLuc.Analysis, RLuc.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.

Ch_L2: numeric (optional): An integer specifying the channel for L2.

Ch_L3: numeric (optional): A vector of length 2 with integer values specifying the start and end channels for L3 (e.g., c(40, 50)).

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 RLuc.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: RLuc.Results object if either fitCW.sigma or fitCW.curve is TRUE

args: list of used arguments

call: [call] the function call
Function version

0.1.1

How to cite


Author(s)

Georgina E. King, University of Bern (Switzerland)
Julie A. Durcan, University of Oxford (United Kingdom)
Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


Further reading


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**

```r
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 component 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**

0.4.1

**How to cite**


**Author(s)**

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

**References**


**Further reading**

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

```r
## 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

```r
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
calc_FuchsLang2001

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

How to cite


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), RLum Developer Team
References


See Also

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

Examples

```r
## 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, Y0, Y0.error, range). 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.
calc_gSGC

\[ n \text{.MC} \]\textbf{integer (with default):} number of Monte Carlo simulation runs for error estimation, see details.

\[ \text{verbose} \]\textbf{logical:} enable or disable terminal output

\[ \text{plot} \]\textbf{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 \texttt{uniroot}. Large values for \texttt{n.MC} will significantly increase the computation time.

Value

Returns an S4 object of type \texttt{RLum.Results}.

@data

$\text{De.value (data.frame)}$

.. $\text{De}$

.. $\text{De.error}$

.. $\text{Eta}$

$\text{De.MC (list)}$ contains the matrices from the error estimation.

$\text{uniroot (list)}$ contains the \texttt{uniroot} outputs of the De estimations

@info

‘$\text{call" (call)}$ the original function call

Function version

0.1.1

How to cite

Kreutzer, S., 2019. calc_gSGC(): Calculate De value based on the gSGC by Li et al., 2015. Function version 0.1.1. In: Kreutzer, S., Burow, C., Dietze, M., Fuchs, M.C., Schmidt, C., Fischer, M., Friedrich, J., 2019. Luminescence: Comprehensive Luminescence Dating Data Analysis R package version 0.9.5. https://CRAN.R-project.org/package=Luminescence

Author(s)

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

References

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): perform the homogeneity 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 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

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

Function version

0.3.0

How to cite


Author(s)

Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

References


See Also

pchisq

Examples

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

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

## using the data presented by Galbraith (2003)
df <-
data.frame(
  x = c(30.1, 53.8, 54.3, 29.0, 47.6, 44.2, 43.1),
  y = c(4.8, 7.1, 6.8, 4.3, 5.2, 5.9, 3.0))
calc_HomogeneityTest(df)
Apply the Huntley (2006) model

Description

A function to calculate the expected sample specific fraction of saturation based on the model of Huntley (2006) using the approach as implemented in Kars et al. (2008) or Guralnik et al. (2015).

Usage

calc_Huntley2006(data, LnTn = NULL, rhop, ddot, readerDdot, normalise = TRUE, fit.method = c("EXP", "GOK")[1], lower.bounds = c(-Inf, -Inf, -Inf), summary = TRUE, plot = TRUE, ...)

Arguments

data data.frame (required): A data.frame with one of the following structures:
- A three column data frame with numeric values on a) dose (s), b) LxTx and c) LxTx error.
- If a two column data frame is provided it is automatically assumed that errors on LxTx are missing. A third column will be attached with an arbitrary 5 % error on the provided LxTx values.
- Can also be a wide table, i.e. a data.frame with a number of columns divisible by 3 and where each triplet has the aforementioned column structure.

<table>
<thead>
<tr>
<th>dose (s)</th>
<th>LxTx</th>
<th>LxTx error</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ,1]</td>
<td>[ ,2]</td>
<td>[ ,3]</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------------</td>
</tr>
</tbody>
</table>

NOTE: The function assumes the first row of the function to be the Ln/Tn-value. If you want to provide more than one Ln/Tn-value consider using the argument LnTn.

LnTn data.frame (optional): This argument should only be used to provide more than one Ln/Tn-value. It assumes a two column data frame with the following structure:

<table>
<thead>
<tr>
<th>LnTn</th>
<th>LnTn error</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ,1]</td>
<td>[ ,2]</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
</tbody>
</table>

[1, ]| LnTn_1 | LnTn_1 error |
The function will calculate a **mean** \( \text{Ln}/Tn \)-value and uses either the standard deviation or the highest individual error, whichever is larger. If another mean value (e.g. a weighted mean or median) or error is preferred, this value must be calculated beforehand and used in the first row in the data frame for argument data.

**NOTE:** If you provide \( \text{Ln}/Tn \)-values with this argument the data frame for the data-argument must **not** contain any \( \text{Ln}/Tn \)-values!

**rhop**

**numeric (required):** The density of recombination centres (\( \rho' \)) and its error (see Huntley 2006), given as numeric vector of length two. Note that \( \rho' \) must **not** be provided as the common logarithm. Example: \( \text{rhop} = c(2.92e-06, 4.93e-07) \).

**ddot**

**numeric (required):** Environmental dose rate and its error, given as a numeric vector of length two. Expected unit: Gy/ka. Example: \( \text{ddot} = c(3.7, 0.4) \).

**readerDdot**

**numeric (required):** Dose rate of the irradiation source of the OSL reader and its error, given as a numeric vector of length two. Expected unit: Gy/s. Example: \( \text{readerDdot} = c(0.08, 0.01) \).

**normalise**

**logical (with default):** If TRUE (the default) all measured and computed \( Lx/Tx \) values are normalised by the pre-exponential factor \( A \) (see details).

**fit.method**

**character (with default):** Fit function of the dose response curve. Can either be \( \text{EXP} \) (the default) or \( \text{GOK} \). Note that \( \text{EXP} \) (single saturating exponential) is the original function the model after Huntley (2006) and Kars et al. (2008) was designed to use. The use of a general-order kinetics function (\( \text{GOK} \)) is an experimental adaption of the model and should be used with great care.

**lower.bounds**

**numeric (with default):** Only applicable for \( \text{fit.method} = \text{GOK} \). A vector of length 3 that contains the lower bound values for fitting the general-order kinetics function using \texttt{minpack.lm::nlsLM}. In most cases, the default values (\( c(-\text{Inf}, -\text{Inf}, -\text{Inf}) \)) are appropriate for finding a best fit, but sometimes it may be useful to restrict the lower bounds to e.g. \( c(0, 0, 0) \). The values of the vector are for parameters \( a, D0 \) and \( c \) in that particular order (see details in \texttt{Luminescence::plot\_GrowthCurve}).

**summary**

**logical (with default):** If TRUE (the default) various parameters provided by the user and calculated by the model are added as text on the right-hand side of the plot.

**plot**

**logical (with default):** enables/disables plot output.

... Further parameters:

- **verbose** logical: Show or hide console output
- **n.MC** numeric: Number of Monte Carlo iterations (default = 100000). **Note** that it is generally advised to have a large number of Monte Carlo iterations for the results to converge. Decreasing the number of iterations will often result in unstable estimates.

All other arguments are passed to \texttt{plot} and \texttt{plot\_GrowthCurve}. 
Details

This function applies the approach described in Kars et al. (2008) or Guralnik et al. (2015), which are both developed from the model of Huntley (2006) to calculate the expected sample specific fraction of saturation of a feldspar and also to calculate fading corrected age using this model. \( \rho' \) (\( \rho_{\text{hop}} \)), the density of recombination centres, is a crucial parameter of this model and must be determined separately from a fading measurement. The function `analyse_FadingMeasurement` can be used to calculate the sample specific \( \rho' \) value.

**Kars et al. (2008) - Single saturating exponential**

To apply the approach after Kars et al. (2008) use `fit.method = "EXP"`.

Firstly, the unfaded \( D_0 \) value is determined through applying equation 5 of Kars et al. (2008) to the measured \( LxTx \) data as a function of irradiation time, and fitting the data with a single saturating exponential of the form:

\[
LxTx(t^*) = Ax\phi(t^*)(1 - \exp((-t^*/D_0)))
\]

where

\[
\phi(t^*) = \exp(-\rho'xln(1.8xs_{\text{tilde}}t^*))^3
\]

after King et al. (2016) where \( A \) is a pre-exponential factor, \( t^* \) (s) is the irradiation time, starting at the mid-point of irradiation (Auclair et al. 2003) and \( s_{\text{tilde}} \) (3x10^15 s^-1) is the athermal frequency factor after Huntley (2006).

Using fit parameters \( A \) and \( D_0 \), the function then computes a natural dose response curve using the environmental dose rate, \( D_\text{dot} \) (Gy/s) and equations [1] and [2]. Computed \( LxTx \) values are then fitted using the `plot_GrowthCurve` function and the laboratory measured \( LnTn \) can then be interpolated onto this curve to determine the fading corrected \( De \) value, from which the fading corrected age is calculated.

**Guralnik et al. (2015) - General-order kinetics**

To apply the approach after Guralnik et al. (2015) use `fit.method = "GOK"`.

The approach of Guralnik et al. (2015) is very similar to that of Kars et al. (2008), but instead of using a single saturating exponential the model fits a general-order kinetics function of the form:

\[
LxTx(t^*) = Ax\phi(t^*)(1 - (1 + (1/D_0)x(t^*x)c)^{-1/c})
\]

where \( A \), \( \phi \), \( t^* \) and \( D_0 \) are the same as above and \( c \) is a dimensionless kinetic order modifier (cf. equation 10 in Guralnik et al., 2015).

**Level of saturation**

The `calc_Huntley2006` function also calculates the level of saturation (\( n/N \)) and the field saturation (i.e. athermal steady state, \( (n/N)_SS \)) value for the sample under investigation using the sample specific \( \rho' \), unfaded \( D_0 \) and \( D_\text{dot} \) values, following the approach of Kars et al. (2008).

**Uncertainties**
Uncertainties are reported at 1 sigma and are assumed to be normally distributed and are estimated using monte-carlo resamples \((n.\text{MC} = 1000)\) of \(\rho'\) and LxTx during dose response curve fitting, and of \(\rho'\) in the derivation of \((n/N)\) and \((n/N)_\text{SS}\).

**Age calculated from 2*D0 of the simulated natural DRC**

In addition to the age calculated from the equivalent dose derived from \(Ln/Tn\) projected on the simulated natural dose response curve (DRC), this function also calculates an age from twice the characteristic saturation dose \((D0)\) of the simulated natural DRC. This can be a useful information for (over)saturated samples (i.e. no intersect of \(Ln/Tn\) on the natural DRC) to obtain at least a "minimum age" estimate of the sample. In the console output this value is denoted by "Age @2D0 (ka):".

**Value**

An `RLum.Results` object is returned:

Slot: @data

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>results</td>
<td>data.frame</td>
<td>results of the of Kars et al. 2008 model</td>
</tr>
<tr>
<td>data</td>
<td>data.frame</td>
<td>original input data</td>
</tr>
<tr>
<td>Ln</td>
<td>numeric</td>
<td>Ln and its error</td>
</tr>
<tr>
<td>LxTx_tables</td>
<td>list</td>
<td>A list of data.frames containing data on dose, LxTx and LxTx error for each of the dose response curves</td>
</tr>
<tr>
<td>fits</td>
<td>list</td>
<td>A list of nls objects produced by <code>minpack.lm::nlsLM</code> when fitting the dose response curves</td>
</tr>
</tbody>
</table>

Slot: @info

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>call</td>
<td>the original function call</td>
</tr>
<tr>
<td>args</td>
<td>list</td>
<td>arguments of the original function call</td>
</tr>
</tbody>
</table>

**Function version**

0.4.1

**How to cite**


**Note**

This function has BETA status and should not be used for publication work!
Author(s)

Georgina E. King, University of Bern (Switzerland)
Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


Further reading


Examples

```r
## Load example data (sample UNIL/NB123, see ?ExampleData.Fading)
data("ExampleData.Fading", envir = environment())

## (1) Set all relevant parameters
# a. fading measurement data (IR50)
fading_data <- ExampleData.Fading$fading.data$IR50

# b. Dose response curve data
data <- ExampleData.Fading$equivalentDose.data$IR50

## (2) Define required function parameters
ddot <- c(7.00, 0.004)
readerDdot <- c(0.134, 0.0067)

# Analyse fading measurement and get an estimate of rho'.
# Note that the RLum.Results object can be directly used for further processing.
# The number of MC runs is reduced for this example
rhop <- analyse_FadingMeasurement(fading_data, plot = TRUE, verbose = FALSE, n.MC = 10)

## (3) Apply the Kars et al. (2008) model to the data
kars <- calc_Huntley2006(data = data,
rhop = rhop,
ddot = ddot,
readerDdot = readerDdot,
n.MC = 25)
```
## Not run:
# You can also provide LnTn values separately via the 'LnTn' argument.
# Note, however, that the data frame for 'data' must then NOT contain
# a LnTn value. See argument descriptions!
LnTn <- data.frame(LnTn = c(1.84833, 2.24833),
                  LnTn.error = c(0.17, 0.22))
LxTx <- data[2:nrow(data),]
kars <- calc_Huntley2006(data = LxTx,
                        LnTn = LnTn,
                        rhop = rhop,
                        ddot = ddot,
                        readerDdot = readerDdot,
                        n.MC = 25)
## End(Not run)

---

**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, a, b, interval, decimal.point, plot, ...

data  RLuc.Results or data.frame (required): for data.frame: two columns with De (data[,1]) and De error (values[,2])

a  numeric (required): slope

b  numeric (required): intercept

interval  numeric (required): 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 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
- `.tables` list a list of data frames containing all calculation tables

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

Function version

0.1.1

How to cite


Author(s)

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

RLum Developer Team

References


See Also

plot, calc_CommonDose, calc_CentralDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

```r
## 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)
```
Description

A function to calculate the expected sample specific fraction of saturation following Kars et al. (2008) and Huntley (2006). This function is deprecated and will eventually be removed. Please use calc_Huntley2006() instead.

Usage

calc_Kars2008(fit.method = "EXP", ...)

Arguments

- **fit.method**: character (with default): Fit function of the dose response curve. Can either be EXP (the default) or GOK. Note that EXP (single saturating exponential) is the original function the model after Huntley (2006) and Kars et al. (2008) was designed to use. The use of a general-order kinetics function (GOK) is an experimental adaption of the model and should only be used with great care.

- ... Parameters passed to calc_Huntley2006.

Details

This function applies the approach described in Kars et al. (2008), developed from the model of Huntley (2006) to calculate the expected sample specific fraction of saturation of a feldspar and also to calculate fading corrected age using this model. \( \rho' \) (\( \rho \)hop), the density of recombination centres, is a crucial parameter of this model and must be determined separately from a fading measurement. The function analyse_FadingMeasurement can be used to calculate the sample specific \( \rho' \) value.

Value

An RLum.Results object is returned:

Function version

0.4.0

How to cite

Note

This function is deprecated and will eventually be removed from the package. Please use the function `calc_Huntley2006()` instead (use `fit.method = "EXP"` to apply the model after Kars et al., 2008).

Author(s)

Georgina E. King, University of Bern (Switzerland)
Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


Further reading


Examples

```R
## Load example data (sample UNIL/NB123, see ?ExampleData.Fading)
data("ExampleData.Fading", envir = environment())

## (1) Set all relevant parameters
# a. fading measurement data (IR50)
fading_data <- ExampleData.Fading$fading.data$IR50

# b. Dose response curve data
data <- ExampleData.Fading$equivalentDose.data$IR50

## (2) Define required function parameters
ddot <- c(7.00, 0.004)
readerDdot <- c(0.134, 0.0067)

# Analyse fading measurement and get an estimate of rho'.
# Note that the RLum.Results object can be directly used for further processing.
# The number of MC runs is reduced for this example
rhop <- analyse_FadingMeasurement(fading_data, plot = TRUE, verbose = FALSE, n.MC = 10)

## (3) Apply the Kars et al. (2008) model to the data
kars <- suppressWarnings(calc_Kars2008(data = data, rhop = rhop)
```

calc_Lamothe2003

Apply fading correction after Lamothe et al., 2003

Description

This function applies the fading correction for the prediction of long-term fading as suggested by Lamothe et al., 2003. The function basically adjusts the Ln/Tn values and fit a new dose-response curve using the function plot_GrowthCurve.

Usage

calc_Lamothe2003(object, dose_rate.envir, dose_rate.source, g_value, tc = NULL, tc.g_value = tc, verbose = TRUE, plot = TRUE, ...)

Arguments

- **object**: RLum.Results data.frame (required): Input data for applying the fading correction. Allowed are (1) data.frame with three columns (dose, LxTx, LxTx error), (2) RLum.Results object created by the function analyse_SAR.CWOSL or analyse_pIRIRSequence
- **dose_rate.envir**: numeric vector of length 2 (required): Environmental dose rate in mGy/a
- **dose_rate.source**: numeric vector of length 2 (required): Irradiation source dose rate in Gy/s
- **g_value**: numeric vector of length 2 (required): g_value in %/decade recalculated at the moment the equivalent dose was calculated, i.e. tc is either similar for the g-value measurement and the De measurement or needs be to recalculated (cf. calc_FadingCorr). Inserting a normalised g-value, e.g., normalised to 2-days, will lead to wrong results
- **tc**: numeric (optional): time in seconds between irradiation and the prompt measurement used in the De estimation (cf. Huntley & Lamothe 2001). If set to NULL it is assumed that tc is similar for the equivalent dose estimation and the g-value estimation
- **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 along with the time used for the De estimation. 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. Note: If this value is not NULL the functions expects a numeric value for tc.
verbose logical (with default): Enables/disables terminal verbose mode
plot logical (with default): Enables/disables plot output
... further arguments passed to the function plot_GrowthCurve

Value
The function returns are graphical output produced by the function plot_GrowthCurve and an R Lum Results.

[NUMERICAL OUTPUT]

RLum.Results-object
slot: @data

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$data</td>
<td>data.frame</td>
<td>the fading corrected values</td>
</tr>
<tr>
<td>$fit</td>
<td>nls</td>
<td>the object returned by the dose response curve fitting</td>
</tr>
</tbody>
</table>

'slot: @info
The original function call

Function version
0.1.0

How to cite

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), Norbert Mercier, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

References

See Also

plot_GrowthCurve, calc_FadingCorr, analyse_SAR.CWOSL, analyse_pIRIRSequence

Examples

```r
# 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,
  verbose = FALSE,
  plot = FALSE,
  onlyLxTxTable = TRUE
)

# run fading correction
results_corr <- calc_Lamothe2003(
  object = results,
  dose_rate.envir = c(1.676, 0.180),
  dose_rate.source = c(0.184, 0.003),
  g_value = c(2.36, 0.6),
  plot = TRUE,
  fit.method = "EXP"
)
```

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 similar approach as described in Olley et al. (2006).

Usage

```r
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 (data[,2]).
sigmab numeric (required): additional spread in De values. This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Wallinga 2012, p. 100). NOTE: For the logged model (log = TRUE) this value must be a fraction, e.g. 0.2 (= 20 %). If the un-logged model is used (log = FALSE), sigmab must be provided in the same absolute units of the De values (seconds or Gray). See details (calc_MinDose).
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-parameter 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

0.3.1

How to cite

Burow, C., 2019. calc_MaxDose(): Apply the maximum age model to a given De distribution. Function version 0.3.1. In: Kreutzer, S., Burow, C., Dietze, M., Fuchs, M.C., Schmidt, C., Fischer, M., Friedrich, J., 2019. Luminescence: Comprehensive Luminescence Dating Data Analysis R package version 0.9.5. https://CRAN.R-project.org/package=Luminescence

Author(s)

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

References


Further reading


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

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, log.output = FALSE, plot = TRUE,
multicore = FALSE, ...)

Arguments

data  RLum.Results or data.frame (required): for data.frame: two columns with De (data[,1]) and De error (data[,2]).
sigmab  numeric (required): additional spread in De values. This value represents the expected overdispersion in the data should the sample be well-bleached (Cunningham & Walling 2012, p. 100). NOTE: For the logged model (log = TRUE) this value must be a fraction, e.g. 0.2 (= 20 %). If the un-logged model is used (log = FALSE), sigmab must be provided in the same absolute units of the De values (seconds or Gray). See details.
log  logical (with default): fit the (un-)logged minimum dose model to De data.
par  numeric (with default): apply the 3- or 4-parameter 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).
calc_MinDose

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. NOTE that the initial values must always be given in the absolute units. The the logged model is applied (log = TRUE), the provided init.values are automatically log transformed.

level logical (with default): the confidence level required (defaults to 0.95).

log.output logical (with default): If TRUE the console output will also show the logged values of the final parameter estimates and confidence intervals (only applicable if log = TRUE).

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 may drastically reduce the computation time. Note that this option is highly experimental and may not work on 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-parameter minimum age model is applied, where gamma=mu. For par=4 the 4-parameter model is applied instead.

(Un-)logged model

In the original version of the minimum dose model, the basic data are the natural logarithms of the De estimates and relative standard errors of the De estimates. The value for sigmab must be provided as a ratio (e.g. 0.2 for 20 %). 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. Note that the un-logged model requires sigmab to be in the same absolute unit as the provided De values (seconds or Gray).

While the original (logged) version of the minimum 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

\[ h = \frac{2 \times \sigma_{DE}}{\sqrt{n}} \]

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 bootstrap 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 available, otherwise there will be a massive perfomance hit.

Likelihood profiles

The likelihood profiles are generated and plotted by the bbmle package. The profile likelihood plots look different to ordinary profile likelihood as

"[...] the plot method for likelihood profiles displays the square root of the the deviance difference (twice the difference in negative log-likelihood from the best fit), so it will be V-shaped for cases where the quadratic approximation works well [...]." (Bolker 2016).

For more details on the profile likelihood calculations and plots please see the vignettes of the bbmle package (also available here: https://CRAN.R-project.org/package=bbmle).

Value

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

$.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

0.4.4

How to cite

Burow, C., 2019. calc_MinDose(): Apply the (un-)logged minimum age model (MAM) after Galbraith et al. (1999) to a given De distribution. Function version 0.4.4. In: Kreutzer, S., Burow, C., Dietze, M., Fuchs, M.C., Schmidt, C., Fischer, M., Friedrich, J., 2019. Luminescence: Comprehensive Luminescence Dating Data Analysis R package version 0.9.5. https://CRAN.R-project.org/package=Luminescence

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.

References


Further reading


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-parameter MAM.
calc_MinDose(data = ExampleData.DeValues$CA1, sigmab = 0.1)

# Not run:
# (2) Re-run the model, but save results to a variable and turn
# plotting of the log-likelihood profiles off.
calc_MinDose

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$de, 1), "\U000B1",
                                      round(res$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$de, 1)) ~ ", " ~
                                  rho == .(round(res$p0, 2)))

# (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,
```r
calc_OSLLxTxRatio

# 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**

```r
calc_OSLLxTxRatio(Lx.data, Tx.data = NULL, signal.integral,
    signal.integral.Tx = NULL, background.integral,
    background.integral.Tx = NULL,
    background.count.distribution = "non-poisson",
    use_previousBG = FALSE, 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)
calc_OSLLxTxRatio

Tx.data

**RLum.Data.Curve** or **data.frame** *(optional)*: requires a CW-OSL shine down curve \((x = \text{time}, y = \text{counts})\). 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

use_previousBG

**logical** *(with default)*: If set to **TRUE** the background of the Lx-signal is subtracted also from the Tx-signal. Please note that in this case separate signal integral limits for the Tx signal are not allowed and will be reset.

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. \( \text{sigmaB} = \text{c}(300, 300) \). **Note:** If only one value is provided this value is taken for both (LnTx and TnTx) signals.

sigma0

**numeric** *(with default)*: allow adding an extra component of error to the final Lx/Tx error value (e.g., instrumental error, 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.

sigma0

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:
\[ se(LxTx) = \sqrt{se(LxTx)^2 + (LxTx \ast sig0)^2} \]

**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 \( \sigma^2 \) (i.e. the overdispersion value). Therefore the relative standard error is calculated as:

- **poisson**

  \[ rse(\mu_S) \approx \sqrt{\left(Y_0 + Y_1/k\right)^2}/Y_0 - Y_1/k \]

- **non-poisson**

  \[ rse(\mu_S) \approx \sqrt{\left(Y_0 + Y_1/k^2 + \sigma^2(1 + 1/k)\right)/Y_0 - Y_1/k} \]

**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 RLuc.Results.

Slot data contains a list with the following structure:

@data

$\text{LxTx.table} \text{ (data.frame)}$

.. $\text{LnLx}$

.. $\text{LnLx.BG}$

.. $\text{TnTx}$

.. $\text{TnTx.BG}$

.. $\text{Net.LnLx}$

.. $\text{Net.LnLx.Error}$

.. $\text{Net.TnTx.Error}$

.. $\text{LxTx}$

.. $\text{LxTx.Error}$

$\text{calc.parameters} \text{ (list)}$

.. $\text{sigmab.LnLx}$

.. $\text{sigmab.TnLx}$

.. $k$

@info

$\text{call} \text{ (original function call)}$

**Function version**

0.7.0
### How to cite


### 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), RLum Developer Team

### References


### See Also

RLum.Data.Curve, Analyse_SAR.OSL.data, plot_GrowthCurve, analyse_SAR.CWOSL

### Examples

```r
# 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 = Sys.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 (with default): Date of measurement in "YYYY-MM-DD". 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 irradiation source (Sr-90, Co-60, Cs-137, 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 predict 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.

\[
dose.rate = D_0 \times \exp(-\log(2)/T.1/2 \times t)
\]

with: \(D_0\) <- 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
calc_SourceDoseRate

<table>
<thead>
<tr>
<th>#</th>
<th>Source type</th>
<th>T.1/2</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Sr-90</td>
<td>28.90 y</td>
<td>NNDC, Brookhaven National Laboratory</td>
</tr>
<tr>
<td>[3]</td>
<td>Co-60</td>
<td>5.274 y</td>
<td>NNDC, Brookhaven National Laboratory</td>
</tr>
<tr>
<td>[4]</td>
<td>Cs-137</td>
<td>30.08 y</td>
<td>NNDC, Brookhaven National Laboratory</td>
</tr>
</tbody>
</table>

Value

Returns an S4 object of type R Lum.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

0.3.2

How to cite

Fuchs, M.C., Kreutzer, S., 2019. calc_SourceDoseRate(): Calculation of the source dose rate via the date of measurement. Function version 0.3.2. In: Kreutzer, S., Burow, C., Dietze, M., Fuchs, M.C., Schmidt, C., Fischer, M., Friedrich, J., 2019. Luminescence: Comprehensive Luminescence Dating Data AnalysisR package version 0.9.5. https://CRAN.R-project.org/package=Luminescence

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) original 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, UMR 5060, CNRS - Université Bordeaux Montaigne (France)
RLum Developer Team
References


See Also

Second2Gray, get_RLum, plot_RLum

Examples

```r
##(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
calc_Statistics

Description

This function calculates a number of descriptive statistics for estimates with a given standard error (SE), most fundamentally using error-weighted approaches.

Usage

calc_Statistics(data, weight.calc = "square", digits = NULL, n.MCM = NULL, 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. NULL (the default) disables MC runs.

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) 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

0.1.7

How to cite

Author(s)

Michael Dietze, GFZ Potsdam (Germany), RLum Developer Team

Examples

```r
## 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

```r
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 calculated. A vector can be provided.
calc_TermalLifetime

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 configure 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, currently 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)

\[ \tau = \frac{1}{s \exp\left(\frac{E}{kT}\right)} \]

where: \( \tau \) 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 RLLum.Results object is returned along with a plot (for profiling = TRUE). The output object contain the following slots:

@data
calc_ThermalLifetime

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

How to cite

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 parameters.

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), RLum Developer Team

References

See Also
graphics::matplot, stats::rnorm, get_RLum

Examples

```r
## 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,
)```

calc_TLLxTxRatio

### Description

Calculate the $L/x$ ratio for a given set of TL curves [beta version].

### Usage

```r
calc_TLLxTxRatio(Lx.data.signal, Lx.data.background = NULL, Tx.data.signal, Tx.data.background = NULL, 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.
calc_TLLxTxRatio

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

Uncertainty estimation

The standard errors are calculated using the following generalised equation:

$$ SE_{signal} <- |Signal_{net} \times BG_f / BG_{signal} $$

where $BG_f$ is a term estimated by calculating the standard deviation of the sum of the $L_x$ background counts and the sum of the $T_x$ background counts. However, if both signals are similar the error becomes zero.

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

0.3.3

How to cite


Note

This function has still BETA status! Please further note that a similar background for both curves results in a zero error and is therefore set to NA.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, Université Bordeaux Montaigne (France)
Christoph Schmidt, University of Bayreuth (Germany) . RLum Developer Team
calc_WodaFuchs2008

See Also

RLum.Results, analyse_SAR.TL

Examples

```r
# 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)
```

**calc_WodaFuchs2008**  
*Obtain the equivalent dose using the approach by Woda and Fuchs 2008*

**Description**

The function generates a histogram-like reorganisation of the data, to assess counts per bin. The log-transformed counts per bin are used to calculate the second derivative of the data (i.e., the curvature of the curve) and to find the central value of the bin hosting the distribution maximum. A normal distribution model is fitted to the counts per bin data to estimate the dose distribution parameters. The uncertainty of the model is estimated based on all input equivalent doses smaller that of the modelled central value.

**Usage**

```r
calc_WodaFuchs2008(data, breaks = NULL, plot = TRUE, ...)
```

**Arguments**

data:  
A `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)`).
calc_WodaFuchs2008

breaks numeric: Either number or locations of breaks. See [hist] for details. If missing, the number of breaks will be estimated based on the bin width (as function of median error).

plot logical (with default): enable plot output.

Function version

0.2.0

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), Michael Dietze, GFZ Potsdam (Germany), RLum Developer Team

References


See Also
calc_FuchsLang2001, calc_CentralDose

Examples

```r
## read example data set
data(ExampleData.DeValues, envir = environment())

results <- calc_WodaFuchs2008(
data = ExampleData.DeValues$CA1,
xlab = expression(paste(D[e], " [Gy]"))
)
```
**Description**

The function performs the conversion of the specific activities into concentrations and vice versa for the nuclides U-238, Th-232 and K-40 to harmonise the measurement unit with the required data input unit of potential analytical tools for, e.g. dose rate calculation or related functions such as `use_DRAC`.

**Usage**

```r
convert_Activity2Concentration(data, input_unit = "Bq/kg", verbose = TRUE)
```

**Arguments**

- **data**: `data.frame` (required): provide dose rate data (activity or concentration) in three columns. The first column indicates the nuclides, the 2nd column measured value and in the 3rd column its error value. Allowed nuclide data are 'U-238', 'Th-232' and 'K-40'. See examples for an example.

- **input_unit**: `character` (with default): specify unit of input data given in the dose rate data frame, choose between 'Bq/kg' and 'ppm/%' the default is 'Bq/kg'.

- **verbose**: `logical` (with default): enable or disable verbose mode

**Details**

The conversion from nuclide activity of a sample to nuclide concentration is performed using conversion factors that are based on the mass-related specific activity of the respective nuclides. The factors can be calculated using the equation:

\[
A = \text{avogadronumber} \times \frac{N.\text{freq}}{N.\text{mol.mass}} \times \frac{\ln(2)}{N.\text{half.life}}
\]

\[
f = A / 10^6
\]

where:

- `A` - specific activity of the nuclide
- `N.freq` - natural frequency of the isotop
- `N.mol.mass` molare mass
- `n.half.life` half-life of the nuclide

example for U238:
convert_Activity2Concentration

- \textit{avogadronumber} = 6.02214199 \times 10^{23}
- \textit{uran半life} = 1.41 \times 10^{17} \text{ (in s)}
- \textit{uran.mol.mass} = 0.23802891 \text{ (in kg/mol)}
- \textit{uran.freq} = 0.992745 \text{ (in mol)}
- \textit{A.U} = \text{avogadronumber} \times \text{uran.freq} / \text{uran.mol.mass} \times \ln(2) / \text{uran.half.life} \text{ (specific activity in Bq/kg)}
- \textit{f.U} = A.kg/10^6

Function version

0.1.0

How to cite


Author(s)

Margret C. Fuchs, Helmholtz-Institut Freiberg for Resource Technology (Germany), RLum Developer Team

References


Examples

```r
##construct data.frame
data <- data.frame(
  NUCLIDES = c("U-238", "Th-232", "K-40"),
  VALUE = c(40, 80, 100),
  VALUE_ERROR = c(4, 8, 10),
  stringsAsFactors = FALSE)

##perform analysis
convert_Activity2Concentration(data)
```
convert_BIN2CSV

Export Risoe BIN-file(s) to CSV-files

Description
This function is a wrapper function around the functions read_BIN2R and write_RLum2CSV and it imports a Risoe BIN-file and directly exports its content to CSV-files. If nothing is set for the argument path (write_RLum2CSV) the input folder will become the output folder.

Usage
convert_BIN2CSV(file, ...)

Arguments
file character (required): name of the BIN-file to be converted to CSV-files
...
  further arguments that will be passed to the function read_BIN2R and write_RLum2CSV

Value
The function returns either a CSV-file (or many of them) or for the option export == FALSE a list comprising objects of type data.frame and matrix

Function version
0.1.0

How to cite

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France) , RLum Developer Team

See Also
RLum.Analysis, RLum.Data, RLum.Results, utils::write.table, write_RLum2CSV, read_BIN2R
Examples

```r
##transform Risoe.BINfileData values to a list
data(ExampleData.BINfileData, envir = environment())
convert_BIN2CSV(subset(CWOSL.SAR.Data, POSITION == 1), export = FALSE)

## Not run:
##select your BIN-file
file <- file.choose()

##convert
convert_BIN2CSV(file)

## End(Not run)
```

---

**convert.Daybreak2CSV**

*Export measurement data produced by a Daybreak luminescence reader to CSV-files*

**Description**

This function is a wrapper function around the functions `read.Daybreak2R` and `write.RLum2CSV` and it imports an Daybreak-file (TXT-file, DAT-file) and directly exports its content to CSV-files. If nothing is set for the argument `path` (`write.RLum2CSV`) the input folder will become the output folder.

**Usage**

```r
convert.Daybreak2CSV(file, ...)
```

**Arguments**

- `file` character (required): name of the Daybreak-file (TXT-file, DAT-file) to be converted to CSV-files
- `...` further arguments that will be passed to the function `read.Daybreak2R` and `write.RLum2CSV`

**Value**

The function returns either a CSV-file (or many of them) or for the option `export = FALSE` a list comprising objects of type `data.frame` and `matrix`

**Function version**

0.1.0
How to cite

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), RLum Developer Team

See Also
RLum.Analysis, RLum.Data, RLum.Results, utils::write.table, write_RLum2CSV, read_Daybreak2R

Examples
```r
## Not run:
## select your BIN-file
file <- file.choose()

## convert
convert_Daybreak2CSV(file)

## End(Not run)
```

---

**convert_PSL2CSV**

*Export PSL-file(s) to CSV-files*

**Description**

This function is a wrapper function around the functions read_PSL2R and write_RLum2CSV and it imports an PSL-file (SUERC portable OSL reader file format) and directly exports its content to CSV-files. If nothing is set for the argument `path` (write_RLum2CSV) the input folder will become the output folder.

**Usage**

```r
convert_PSL2CSV(file, ...)
```

**Arguments**

- **file** character *(required)*: name of the PSL-file to be converted to CSV-files
- **...** further arguments that will be passed to the function read_PSL2R and write_RLum2CSV
Value

The function returns either a CSV-file (or many of them) or for the option `export = FALSE` a list comprising objects of type `data.frame` and `matrix`.

Function version

0.1.0

How to cite


Author(s)

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

See Also

`RLum.Analysis`, `RLum.Data`, `RLum.Results`, `utils::write.table`, `write_RLum2CSV`, `read_PSL2R`

Examples

```r
## Not run:
##select your BIN-file
file <- file.choose()

##convert
convert_PSL2CSV(file)

## End(Not run)
```

---

**convert_RLum2Risoe.BINfileData**

*Converts RLum.Analysis-objects and RLum.Data.Curve-objects to RLum2Risoe.BINfileData-objects*

Description

The function converts `RLum.Analysis` and `RLum.Data.Curve` objects and a list of those to `Risoe.BINfileData` objects. The function intends to provide a minimum of compatibility between both formats. The created `RLum.Analysis` object can be later exported to a BIN-file using the function `write_R2BIN`. 
convert_RLum2Risoe.BINfileData

Usage

convert_RLum2Risoe.BINfileData(object, keep.position.number = FALSE)

Arguments

object RLum.Analysis or RLum.Data.Curve (required): input object to be converted
keep.position.number logical (with default): keeps the original position number or re-calculate the
numbers to avoid doubling

Value

The function returns a Risoe.BINfileData object.

Function version

0.1.3

How to cite

Kreutzer, S., 2019. convert_RLum2Risoe.BINfileData(): Converts RLum.Analysis-objects and
RLum.Data.Curve-objects to RLum2Risoe.BINfileData-objects. Function version 0.1.3. In: Kreutzer,
S., Burow, C., Dietze, M., Fuchs, M.C., Schmidt, C., Fischer, M., Friedrich, J., 2019. Lumines-
cence: Comprehensive Luminescence Dating Data Analysis R package version 0.9.5. https://CRAN.R-
project.org/package=Luminescence

Note

The conversion can be never perfect. The RLum objects may contain information which are not part
of the Risoe.BINfileData definition.

Author(s)

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

See Also

RLum.Analysis, RLum.Data.Curve, write_R2BIN

Examples

# simple conversion using the example dataset
data(ExampleData.RLum.Analysis, envir = environment())
convert_RLum2Risoe.BINfileData(IRSAR.RF.Data)
**convert_Wavelength2Energy**

*Emission Spectra Conversion from Wavelength to Energy Scales*

**Description**

The function provides a convenient and fast way to convert emission spectra wavelength to energy scales. The function works on `RLum.Data.Spectrum`, `data.frame` and `matrix` and a list of such objects. The function was written to smooth the workflow while analysing emission spectra data. This is in particular useful if you want to further treat your data and apply, e.g., a signal deconvolution.

**Usage**

```r
catalogue_Wavelength2Energy(object, digits = 3L, order = FALSE)
```

**Arguments**

- **object**: `RLum.Data.Spectrum`, `data.frame`, `matrix` (*required*): input object to be converted. If the input is not a `RLum.Data.Spectrum`, the first column is always treated as the wavelength column. The function supports a list of allowed input objects.
- **digits**: integer (*with default*): set the number of digits on the returned energy axis
- **order**: logical (*with default*): enables/disables sorting of the values in ascending energy order. After the conversion the longest wavelength has the lowest energy value and the shortest wavelength the highest. While this is correct, some R functions expect increasing x-values.

**Details**

The intensity of the spectrum is re-calculated using the following approach to recalculate wavelength and corresponding intensity values (e.g., Appendix 4 in Blasse and Grabmeier, 1994; Mooney and Kambhampati, 2013):

\[
\phi_E = \phi_\lambda \times \frac{\lambda^2}{hc}
\]

with \(\phi_E\) the intensity per interval of energy \(E\) (1/eV), \(\phi_\lambda\) the intensity per interval of wavelength \(\lambda\) (1/nm) and \(h\) (eV * s) the Planck constant and \(c\) (nm/s) the velocity of light.

For transforming the wavelength axis (x-values) the equation as follow is used

\[
E = \frac{hc}{\lambda}
\]

**Value**

The same object class as provided as input is returned.
Function version

0.1.1

How to cite


Note

This conversion works solely for emission spectra. In case of absorption spectra only the x-axis has to be converted.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

References


Further reading


See Also

RLum.Data.Spectrum, plot_RLum

Examples

```R
##=====================##
##(1) Literature example after Mooney et al. (2013)
##(1.1) create matrix
m <- matrix(
  data = c(seq(400, 800, 50), rep(1, 9)), ncol = 2)
```
## (1.2) set plot function to reproduce the literature figure

```r
p <- function(m) {
  plot(x = m[, 1], y = m[, 2])
  polygon(
    x = c(m[, 1], rev(m[, 1])),
    y = c(m[, 2], rep(0, nrow(m))))
  for (i in 1:nrow(m)) {
    lines(x = rep(m[i, 1], 2), y = c(0, m[i, 2]))
  }
}
```

## (1.3) plot curves

```r
par(mfrow = c(1,2))
p(m)
p(convert_Wavelength2Energy(m))
```

## (2) Another example using density curves

```r
# create dataset
xy <- density(
  c(rnorm(n = 100, mean = 500, sd = 20),
    rnorm(n = 100, mean = 800, sd = 20)))
xy <- data.frame(xy$x, xy$y)

# plot
par(mfrow = c(1,2))
plot(
  xy,
  type = "l",
  xlim = c(150, 1000),
  xlab = "Wavelength [nm]",
  ylab = "Luminescence [a.u.]"
)
plot(
  convert_Wavelength2Energy(xy),
  xy$y,
  type = "l",
  xlim = c(1.23, 8.3),
  xlab = "Energy [eV]",
  ylab = "Luminescence [a.u.]"
)
```

---

**convert_XSYG2CSV**

Export XSYG-file(s) to CSV-files
convert_XSYG2CSV

Description

This function is a wrapper function around the functions `read_XSYG2R` and `write_RLum2CSV` and it imports an XSYG-file and directly exports its content to CSV-files. If nothing is set for the argument `path` (`write_RLum2CSV`) the input folder will become the output folder.

Usage

```r
convert_XSYG2CSV(file, ...)
```

Arguments

- `file` character (required): name of the XSYG-file to be converted to CSV-files
- `...` further arguments that will be passed to the function `read_XSYG2R` and `write_RLum2CSV`

Value

The function returns either a CSV-file (or many of them) or for the option `export = FALSE` a list comprising objects of type `data.frame` and `matrix`

Function version

0.1.0

How to cite


Author(s)

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

See Also

- `RLum.Analysis`, `RLum.Data`, `RLum.Results`, `utils::write.table`, `write_RLum2CSV`, `read_XSYG2R`

Examples

```r
##transform XSYG-file values to a list
data(ExampleData.XSYG, envir = environment())
convert_XSYG2CSV(OSL.SARMeasurement$Sequence.Object[1:10], export = FALSE)

## Not run:
##select your BIN-file
file <- file.choose()
```
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

```r
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:
   \[ t' = t - \left( \frac{1}{\delta} \right) \times \log(1 + \delta \times t) \]
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 \( t' > \max(t) \)

**NOTE:** 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.

**RLum.Data.Curve**

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

**data.frame**

\$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

**How to cite**


**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 \texttt{approx} may produce some \texttt{Inf} and \texttt{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, RLum Developer Team

References

Further Reading

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

```r
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

```r
curve.transformed <- CW2pHMi(curve)
```

## plot curve

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

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

## load data

```r
data(ExampleData.CW_OSL_Curve, envir = environment())
values <- CW_Curve.BosWallinga2012
```

## open plot area

```r
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)"
)
```

```r
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)
```

```r
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)
```

```r
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 R Lum. Data. Curve or data.frame (required): R Lum. 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):

\[ P = 2 \times \text{max}(t) \]
\[ u = \sqrt{(2 \times t \times P)} \]

The new count values are then calculated by

\[ \text{ctsNEW} = \text{cts}(u/P) \]

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

The output of the function can be further used for LM-OSL fitting.

Value

The function returns the same data type as the input data type with the transformed curve values (data.frame or RLum.Data.Curve).

Function version

0.4.1

How to cite


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), RLum Developer Team

References

Further Reading

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

Examples

```r
##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],"HIGH"],
    by = CWOSL.SAR.Data@METADATA[id.OSL[1],"HIGH"],
    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:

\[ t’ = \frac{1}{2} \times \frac{1}{P} \times t^2 \]

(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

\[ pLM(t) = \frac{t}{P} \times CW(t’) \]

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

NOTE: 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.

RLum.Data.Curve
Function version

0.3.1

How to cite


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 , RLum Developer Team

References


Further Reading


See Also

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

Examples

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

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

```r
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:
   
   \[ t’ = \frac{1}{3} \times \left( \frac{1}{P^2} \right) t^3 \]

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
   
   \[ pLM(t) = t^2 / P^2 \times CW(t’) \]

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

**Value**

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

`RLum.Data.Curve`

- `$CW2pPMi.x.t`: transformed time values
- `$CW2pPMi.method`: used method for the production of the new data points
data.frame

  $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

How to cite


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, RLum Developer Team

References


Further Reading


See Also


Examples

```r
##(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.Al2O3C  

**Example Al2O3:C Measurement Data**

**Description**

Measurement data obtained from measuring Al2O3:C chips at the IRAMAT-CRP2A, Université Bordeaux Montaingue in 2017 on a Freiberg Instruments lexsyg SMART reader. The example data used in particular to allow test of the functions developed in framework of the work by Kreutzer et al., 2018.
ExampleData.BINfileData

Format

Two datasets comprising **RLum.Analysis** data imported using the function `read_XSYG2R`:
- `data_ITC`: Measurement data to determine the irradiation time correction, the data can be analysed with the function `analyse_Al2O3C_ITC`
- `data_CrossTalk`: Measurement data obtained while estimating the irradiation cross-talk of the reader used for the experiments. The data can be analysed either with the function `analyse_Al2O3C_CrossTalk` or `analyse_Al2O3C_Measurement`

Note

From both datasets unneeded curves have been removed and the number of aliquots have been reduced to a required minimum to keep the file size small, but still being able to run the corresponding functions.

References


See Also

`analyse_Al2O3C_ITC`, `analyse_Al2O3C_CrossTalk`, `analyse_Al2O3C_Measurement`

Examples

```r
#(1) curves
data(ExampleData.Al2O3C, envir = environment())
plot_RLum(data_ITC[1:2])
```

ExampleData.BINfileData

Example data from a SAR OSL and SAR TL measurement for 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`
ExampleData.BINfileData

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,]
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


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

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 reader
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]"

Description

Example data set for fading measurements of the IR50, IR100, IR150 and IR225 feldspar signals of sample UNIL/NB123. It further contains regular equivalent dose measurement data of the same sample, which can be used to apply a fading correction to.

Format

A list with two elements, each containing a further list of data.frames containing the data on the fading and equivalent dose measurements:

$fading.data: A named list of data.frames, each having three named columns (LxTx,LxTx.error,timeSinceIrradiation).
  ..$IR50: Fading data of the IR50 signal.
  ..$IR100: Fading data of the IR100 signal.
  ..$IR150: Fading data of the IR150 signal.
  ..$IR225: Fading data of the IR225 signal.
$equivalentDose.data: A named of data.frames, each having three named columns (dose,LxTx,LxTx.error).
  ..$IR50: Equivalent dose measurement data of the IR50 signal.
  ..$IR100: Equivalent dose measurement data of the IR100 signal.
  ..$IR150: Equivalent dose measurement data of the IR150 signal.
  ..$IR225: Equivalent dose measurement data of the IR225 signal.

Source

These data were kindly provided by Georgina E. King. Detailed information on the sample UNIL/NB123 can be found in the reference given below. The raw data can be found in the accompanying supplementary information.
## References


## Details

Lab: University of Lausanne  
Lab-Code: UNIL/NB123  
Location: Namche Barwa (eastern Himalaya)  
Material: Coarse grained (180-212 microns) potassium feldspar  
Units: Values are given in seconds  
Lab Dose Rate: Dose rate of the beta-source at measurement ca. 0.1335 +/- 0.004 Gy/s  
Environmental Dose Rate: 7.00 +/- 0.92 Gy/ka (includes internal dose rate)

## Examples

```r
## Load example data
data("ExampleData.Fading", envir = environment())

## Get fading measurement data of the IR50 signal
IR50_fading <- ExampleData.Fading$fading.data$IR50
head(IR50_fading)

## Determine g-value and rho' for the IR50 signal
IR50_fading.res <- analyse_FadingMeasurement(IR50_fading)

## Show g-value and rho' results
gval <- get_RLum(IR50_fading.res)
rhop <- get_RLum(IR50_fading.res, "rho_prime")
gval
rhop

## Get LxTx values of the IR50 DE measurement
IR50_De.LxTx <- ExampleData.Fading$equivalentDose.data$IR50

## Calculate the De of the IR50 signal
IR50_De <- plot_GrowthCurve(IR50_De.LxTx,
                            mode = "interpolation",
                            fit.method = "EXP")

## Extract the calculated De and its error
IR50_De.res <- get_RLum(IR50_De)
De <- c(IR50_De.res$De, IR50_De.res$De.Error)

## Apply fading correction (age conversion greatly simplified)
IR50_Age <- De / 7.00
IR50_Age.corr <- calc_FadingCorr(IR50_Age, g_value = IR50_fading.res)
```
ExampleData.FittingLM  Example data for fit_LMCurve() in the package Luminescence

Description

Linearly 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


Examples

```r
##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.
**ExampleData.LxTxOSLData**

**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**

```r
## 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

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

# plot data
plot(Lx.data)
plot(Tx.data)
```

### ExampleData.portableOSL

*Example portable OSL curve data for the package Luminescence*

**Description**

A list of `RLum.Analysis` objects, each containing the same number of `RLum.Data.Curve` objects representing individual OSL, IRSL and dark count measurements of a sample.

**Source**

**ExampleData.portableOSL**

- **Lab:** Cologne Luminescence Laboratory
- **Lab-Code:** <none>
- **Location:** Nievenheim/Germany
- **Material:** Fine grain quartz
- **Reference:** unpublished data

**Examples**

```r
data(ExampleData.portableOSL, envir = environment())
plot_RLum(ExampleData.portableOSL)
```

### ExampleData.RLum.Analysis

*Example data as `RLum.Analysis` objects*

**Description**

Collection of different `RLum.Analysis` objects for protocol analysis.
ExampleData.RLum.Data.Image

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


Examples

```r
#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 Princeton Instruments camera imported with the function read_SPE2R to R to produce an RLum.Data.Image object.

Format

Object of class RLum.Data.Image
**ExampleData.ScaleGammaDose**

**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**

```r
# load data
data(ExampleData.RLum.Data.Image, envir = environment())

# plot data
plot_RLum(ExampleData.RLum.Data.Image)
```

**ExampleData.ScaleGammaDose**

*Example data for scale_GammaDose()*

**Description**

An example data set for the function `scale_GammaDose()` containing layer specific information to scale the gamma dose rate considering variations in soil radioactivity.

**Format**

A `data.frame`. Please see `?scale_GammaDose()` for a detailed description of its structure.

**Version**

0.1
## Load data

data("ExampleData.ScaleGammaDose")

### ExampleData.SurfaceExposure

**Example OSL surface exposure dating data**

### Description

A set of synthetic OSL surface exposure dating data to demonstrate the `fit_SurfaceExposure` functionality. See examples to reproduce the data interactively.

### Format

A list with 4 elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sample_1</td>
<td>A data.frame with 3 columns (depth, intensity, error)</td>
</tr>
<tr>
<td>$sample_2</td>
<td>A data.frame with 3 columns (depth, intensity, error)</td>
</tr>
<tr>
<td>$set_1</td>
<td>A list of 4 data.frames, each representing a sample with different ages</td>
</tr>
<tr>
<td>$set_2</td>
<td>A list of 5 data.frames, each representing a sample with different ages</td>
</tr>
</tbody>
</table>

### Details

#### $sample_1

<table>
<thead>
<tr>
<th>mu</th>
<th>sigmaphi</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>5e-10</td>
<td>10000</td>
</tr>
</tbody>
</table>

#### $sample_2

<table>
<thead>
<tr>
<th>mu</th>
<th>sigmaphi</th>
<th>age</th>
<th>Dose rate</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>5e-10</td>
<td>10000</td>
<td>2.5</td>
<td>40</td>
</tr>
</tbody>
</table>

#### $set_1

<table>
<thead>
<tr>
<th>mu</th>
<th>sigmaphi</th>
<th>ages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>5e-10</td>
<td>1e3, 1e4, 1e5, 1e6</td>
</tr>
</tbody>
</table>
ExampleData.SurfaceExposure

$set_2

<table>
<thead>
<tr>
<th>mu</th>
<th>sigmaphi</th>
<th>ages</th>
<th>Dose rate</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>5e-10</td>
<td>1e2, 1e3, 1e4, 1e5, 1e6</td>
<td>1.0</td>
<td>40</td>
</tr>
</tbody>
</table>

Source
See examples for the code used to create the data sets.

References
Unpublished synthetic data

Examples

```r
## ExampleData.SurfaceExposure$sample_1
sigmaphi <- 5e-10
age <- 10000
mu <- 0.9
x <- seq(0, 10, 0.1)
fun <- exp(-sigmaphi * age * 365.25*24*3600 * exp(-mu * x))
set.seed(666)
synth_1 <- data.frame(depth = x,
                      intensity = jitter(fun, 1, 0.1),
                      error = runif(length(x), 0.01, 0.2))

## VALIDATE sample_1
fit_SurfaceExposure(synth_1, mu = mu, sigmaphi = sigmaphi)

## ExampleData.SurfaceExposure$sample_2
sigmaphi <- 5e-10
age <- 10000
mu <- 0.9
x <- seq(0, 10, 0.1)
Ddot <- 2.5 / 1000 / 365.25 / 24 / 60 / 60 # 2.5 Gy/ka in Seconds
D0 <- 40
fun <- (sigmaphi * exp(-mu * x) *
        exp(-(age * 365.25*24*3600) *
        (sigmaphi * exp(-mu * x) + Ddot/D0)) + Ddot/D0) /
        (sigmaphi * exp(-mu * x) + Ddot/D0)
set.seed(666)
synth_2 <- data.frame(depth = x,
                      intensity = jitter(fun, 1, 0.1),
                      error = runif(length(x), 0.01, 0.2))
```
## VALIDATE sample_2

```r
fit_SurfaceExposure(synth_2, mu = mu, sigmaphi = sigmaphi, Ddot = 2.5, D0 = D0)
```

## ExampleData.SurfaceExposure$set_1

```r
sigmaphi <- 5e-10
mu <- 0.9
x <- seq(0, 15, 0.2)
age <- c(1e3, 1e4, 1e5, 1e6)
set.seed(666)
synth_3 <- vector("list", length = length(age))
for (i in 1:length(age)) {
  fun <- exp(-sigmaphi * age[i] * 365.25*24*3600 * exp(-mu * x))
  synth_3[[i]] <- data.frame(depth = x,
                           intensity = jitter(fun, 1, 0.05))
}

## VALIDATE set_1

```r
fit_SurfaceExposure(synth_3, age = age, sigmaphi = sigmaphi)
```

## ExampleData.SurfaceExposure$set_2

```r
sigmaphi <- 5e-10
mu <- 0.9
x <- seq(0, 15, 0.2)
age <- c(1e2, 1e3, 1e4, 1e5, 1e6)
Ddot <- 1.0 / 1000 / 365.25 / 24 / 60 / 60 # 2.0 Gy/ka in Seconds
D0 <- 40
set.seed(666)
synth_4 <- vector("list", length = length(age))
for (i in 1:length(age)) {
  fun <- (sigmaphi * exp(-mu * x) * 
          exp(-age[i] * 365.25*24*3600) * 
          (sigmaphi * exp(-mu * x) + Ddot/D0)) + Ddot/D0 / 
          (sigmaphi * exp(-mu * x) + Ddot/D0)
  synth_4[[i]] <- data.frame(depth = x,
                           intensity = jitter(fun, 1, 0.05))
}

## VALIDATE set_2

```r
fit_SurfaceExposure(synth_4, age = age, sigmaphi = sigmaphi, D0 = D0, Ddot = 1.0)
```

## Not run:

ExampleData.SurfaceExposure <- list(
ExampleData.TR_OSL

Example TR-OSL data

Description

Single TR-OSL curve obtained by Schmidt et al. (under review) for quartz sample BT729 (origin: Trebgast Valley, Germay, quartz, 90-200 µm, unpublished data).

Format

One \texttt{RLum.Data.Curve} dataset imported using the function \texttt{read_XSYG2R}

\texttt{ExampleData.TR_OSL}: A single \texttt{RLum.Data.Curve} object with the TR-OSL data

References


See Also

\texttt{fit_OSLLifeTimes}

Examples

```r
##(1) curves
data(ExampleData.TR_OSL, envir = environment())
plot_RLum(ExampleData.TR_OSL)
```
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 research 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:


See Also


Examples

```r
##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)
```


Description

Description and listing of data provided in the folder data/extdata

Details

The R package Luminescence includes a number of raw data files, which are mostly used in the example sections of appropriate functions. They are also used internally for testing corresponding functions using the testthat package (see files in tests/testthat/) to ensure their operational reliability.

Accessibility

If the R package Luminescence is installed correctly the preferred way to access and use these data from within R is as follows:

```
 system.file("extdata/<FILENAME>",package = "Luminescence")
```

Individual file descriptions

»Daybreak_TestFile.DAT/.txt«

Type: raw measurement data
Device: Daybreak OSL/TL reader
Measurement date: unknown
Location: unknown
Provided by: unknown
Related R function(s): read_Daybreak2R()
Reference: unknown

»DorNie_0016.psl«

Type: raw measurement data
Device: SUERC portable OSL reader
Measurement date: 19/05/2016
Location: Dormagen-Nievenheim, Germany
Provided by: Christoph Burow (University of Cologne)
Related R function(s): read_PSL2R()
Reference: unpublished
Additional information: Sample measured at an archaeological site near Dormagen-Nievenheim (Germany) during a practical course on Luminescence dating in 2016.

»QNL84_2_bleached.txt, QNL84_2_unbleached.txt«

Type: Test data for exponential fits

»STRB87_1_bleached.txt, STRB87_1_unbleached.txt«
**Type:** Test data for exponential fits  

**XYSG_file.xysg**

**Type:** XSYG-file stump  
**Info:** XSYG-file with some basic curves to test functions  
**Reference:** no reference available

---

**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**

```r
evaluate_IrradiationTimes(object, file.BINX,  
recordType = c("irradiation (NA)", "IRSL (UVVIS)", "OSL (UVVIS)",  
"TL (UVVIS)"), compatibility.mode = TRUE, txtProgressBar = TRUE)
```

**Arguments**

- **object** character, RLum.Analysis or list (required): 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
extract_IrradiationTimes

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 to store 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.

Besides 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, instead of a link to only one file. In this case the argument input for file.BINX is ignored.

In the self call mode (input is a list of RLum.Analysis objects a list of RLum.Results is returned.

Function version
0.3.1

How to cite

Note
The produced output object contains still the irradiation steps to keep the output transparent. However, for the BINX-file export these steps are removed as the BINX-file format description does not allow irradiations as separat sequences steps.

BINX-file 'Time Since Irradiation' value differs from the table output?
The way the value 'Time Since Irradiation' is defined differs. In the BINX-file the 'Time Since Irradiation' is calculated as the 'Time Since Irradiation' plus the 'Irradiation Time'. The table output returns only the real 'Time Since Irradiation', i.e. time between the end of the irradiation and the next step.

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: 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), RLum Developer Team

References

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

See Also

RLum.Analysis, RLum.Results, Risoe.BINfileData, read_XSYG2R, read_BIN2R, write_R2BIN

Examples

```r
## (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)
```
**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**

```r
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 using the function 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.
Details

Fitting function
The function for the CW-OSL fitting has the general form: \[ y = I_0 \lambda_1 \exp(-\lambda_1 x) + \ldots + I_0 \lambda_i \exp(-\lambda_i x) \]
where 0 < i < 8
and \( \lambda \) is the decay constant
and \( I_0 \) the initial number of trapped electrons.

(for the used equation cf. Boetter-Jensen et al., 2003, Eq. 2.31)

Start values
Start values are estimated automatically by fitting a linear function to the logarithmed 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:

\[ \text{pseudoR}^2 = 1 - \frac{\text{RSS}}{\text{TSS}} \]

where \( \text{RSS} = \text{Residual Sum of Squares} \)
and \( \text{TSS} = \text{Total Sum of Squares} \)

Error of fitted component parameters
The 1-sigma error for the components is calculated using the function \text{confint}. Due to considerable calculation time, this option is deactivated by default. In addition, the error for the components can be estimated by using internal R functions like \text{summary}. See the \text{nls} help page for more information.

For details on the nonlinear regression in R, see Ritz & Streibig (2008).

Value

\text{plot (optional)}
the fitted CW-OSL curves are returned as plot.

\text{table (optional)}
an output table (*.csv) with parameters of the fitted components is provided if the output.path is set.
**fit_CWCurve**

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**

0.5.2

**How to cite**


**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!
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 function 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 forintensity 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 conft.
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:
$y = (\exp(0.5) * Im_1 * x / x_m) * \exp(-x^2 / (2 * x_m^2)) + \ldots + \exp(0.5) * Im_i * x / x_m) * \exp(-x^2 / (2 * x_m^2))$

where $1 < i < 8$

This function and the equations for the conversion to $b$ (detrapping probability) and $n_0$ (proportional to initially trapped charge) have been taken from Kitis et al. (2008):

$x_m_i = \sqrt{\text{max}(t) / b_i}$

$Im_i = \exp(-0.5) n_0 / x_m_i$

Background subtraction

Three methods for background subtraction are provided for a given background signal ($\text{values.bg}$).

- **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 ($\text{start_values}$) 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 $x_m$ and $Im$ are recalculated for the given data set. In all cases, the fitting starts with the ultra-fast component and (depending on $\text{n.components}$) steps through the following values. If no fit could be achieved, an error plot (for $\text{plot} = \text{TRUE}$) 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 $\text{fit.advanced = TRUE}$ is chosen, an advanced start parameter 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:

$$\text{pseudoR}^2 = 1 - \text{RSS/TSS}$$
where \( RSS = \text{Residual Sum of Squares} \) and \( TSS = \text{Total Sum of Squares} \)

**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 deactivated 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:
.. $data : data.frame with fitting results
.. $fit : nls (nls object)
.. $component.contribution.matrix : list component distribution matrix

info:
.. $call : call the original function call

Matrix structure for the distribution matrix:

Column 1 and 2: time and \( \text{rev(time)} \) values

Additional columns are used for the components, two for each component, containing \( I_0 \) and \( n_0 \). The last columns provide information on the relative component contribution for each time interval including the row sum for this values.

**Function version**

0.3.2

**How to cite**


**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), RLum Developer Team
References


See Also

fit_CWCurve, plot, nls, minpack.lm::nlsLM, get_RLum

Examples

```r
##(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)))
```

Fitting and Deconvolution of OSL Lifetime Components

Description

Fitting and Deconvolution of OSL Lifetime Components
Usage

```r
fit_OSLLifeTimes(object, tp = 0, signal_range = NULL,
                 n.components = NULL, method_control = list(), plot = TRUE,
                 verbose = TRUE, ...)
```

Arguments

**object**  
RLum.Data.Curve, RLum.Analysis, data.frame or matrix **(required)**: Input object containing the data to be analysed. All objects can be provided also as list for an automated processing. Please note: NA values are automatically removed and the dataset should comprise at least 5 data points.

**tp**  
numeric **(with default)**: option to account for the stimulation pulse width. For off-time measurements the default value is 0. tp has the same unit as the measurement data, e.g., µs. Please set this parameter carefully if it all, otherwise you may heavily bias your fit results.

**signal_range**  
numeric **(optional)**: allows to set a channel range, by default all channels are used, e.g., `signal_range = c(2,100)` considers only channels 2 to 100 and `signal_range = c(2)` considers only channels from channel 2 onwards.

**n.components**  
numeric **(optional)**: Fix the number of components. If set the algorithm will try to fit the number of predefined components. If nothing is set, the algorithm will try to find the best number of components.

**method_control**  
list **(optional)**: Named to allow a more fine control of the fitting process. See details for allowed options.

**plot**  
logical **(with default)**: Enable/disable plot output

**verbose**  
logical **(with default)**: Enable/disable terminal feedback

**...**  
parameters passed to `plot.default` to control the plot output. Please note that not all parameters are supported.

Details

The function intends to provide an easy access to pulsed optically stimulated luminescence (POSL) data, in order determine signal lifetimes. The fitting is currently optimised to work with the off-time flank of POSL measurements only. For the signal deconvolution, a differential evolution optimisation is combined with nonlinear least-square fitting following the approach by Bluszcz & Adamiec (2006).

**Component deconvolution algorithm**

The component deconvolution consists of two steps:

1. **Adaption phase**

   In the adaption phase the function tries to figure out the optimal and statistically justified number of signal components following roughly the approach suggested by Bluszcz & Adamiec (2006). In contrast to their work, for the optimisation by differential evolution here the package 'DEoptim' is used.

   The function to be optimized has the form:

   $$
   \chi^2 = \sum (w * (n_i/c - \sum (A_i * exp(-x/(tau_i + tp))))^2)
   $$
with \( w = 1 \) for unweighted regression analysis (\texttt{method_control = list(weights = FALSE)}) or \( w = c^2/n_i \) for weighted regression analysis. The default values is \texttt{TRUE}.

\[
F = (\Delta \chi^2 / 2) / (\chi^2 / (N - 2 \ast m - 2))
\]

(2) Final fitting

\begin{tabular}{ll}
\textbf{Parameter} & \textbf{Type} & \textbf{Description} \\
p & numeric & controls the probability for the F statistic reference values. For a significance level of 5% a value of 0.95 (the default) should be added, for 1%, a value of 0.99 is sufficient: 1 > p > 0 (cf. \texttt{stats::FDist}) \\
seed & numeric & set the seed for the random number generator, provide a value here to get reproducible results \\
DEoptim.trace & logical & enables/disables the tracing of the differential evolution (cf. \texttt{DEoptim::DEoptim.control}) \\
DEoptim.itermax & logical & controls the number of the allowed generations (cf. \texttt{DEoptim::DEoptim.control}) \\
weights & logical & enables/disables the weighting for the start parameter estimation and fitting (see equations above). \\
nlsLM.trace & logical & enables/disables trace mode for the nls fitting (\texttt{minpack.lm::nlsLM}), can be used to identify convergence problems. \\
nlsLM.upper & logical & enables/disables upper parameter boundary, default is \texttt{TRUE} \\
nlsLM.lower & logical & enables/disables lower parameter boundary, default is \texttt{TRUE}
\end{tabular}

Value

---

### [ NUMERICAL OUTPUT ]

RLum.Results-object

\texttt{slot: @data}

\begin{tabular}{ll}
\textbf{Element} & \textbf{Type} & \textbf{Description} \\
$data$ & matrix & the final fit matrix \\
$start\_matrix$ & matrix & the start matrix used for the fitting \\
$total\_counts$ & integer & Photon count sum \\
$fit$ & nls & the fit object returned by \texttt{minpack.lm::nls.lm}
\end{tabular}

\texttt{slot: @info}

The original function call

---

### [ TERMINAL OUTPUT ]

Terminal output is only shown of the argument \texttt{verbose = TRUE}.

(1) \textit{Start parameter and component adaptation}

Trave of the parameter adaption process

(2) \textit{Fitting results (sorted by ascending tau)}

The fitting results sorted by ascending tau value. Please note that if you access the \texttt{nls} fitting object, the values are not sorted.
(3) **Further information**

- The photon count sum
- Durbin-Watson residual statistic to assess whether the residuals are correlated, ideally the residuals should be not correlated at all. Rough measures are:
  - \( D = 0 \): the residuals are systematically correlated
  - \( D = 2 \): the residuals are randomly distributed
  - \( D = 4 \): the residuals are systematically anticorrelated

You should be suspicious if \( D \) differs largely from 2.

---

**[ PLOT OUTPUT ]**

---

A plot showing the original data and the fit so far possible. The lower plot shows the residuals of the fit.

**Function version**

0.1.4

**How to cite**


**Author(s)**

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS-Université Bordeaux Montaigne (France), Christoph Schmidt, University of Bayreuth (Germany) , RLum Developer Team

**References**


**Further reading**


fit_SurfaceExposure

Nonlinear Least Squares Fit for OSL surface exposure data

Description
This function determines the (weighted) least-squares estimates of the parameters of either eq. 1 in Sohbati et al. (2012a) or eq. 12 in Sohbati et al. (2012b) for a given OSL surface exposure data set (BETA).

Usage

fit_SurfaceExposure(data, sigmaphi = NULL, mu = NULL, age = NULL, Ddot = NULL, D0 = NULL, weights = FALSE, plot = TRUE, legend = TRUE, error_bars = TRUE, coord_flip = FALSE, ...)

Arguments
data data.frame or list (required): Measured OSL surface exposure data with the following structure:

<table>
<thead>
<tr>
<th>depth (a.u.)</th>
<th>intensity</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ,1]</td>
<td>[ ,2]</td>
<td>[ ,3]</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>[1, ]</td>
<td>~~~~</td>
<td>~~~~</td>
</tr>
<tr>
<td>[2, ]</td>
<td>~~~~</td>
<td>~~~~</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Alternatively, a list of data.frames can be provided, where each data.frame has the same structure as shown above, with the exception that they must not include the optional error column. Providing a list as input automatically activates the global fitting procedure (see details).

**sigmaphi** numeric *(optional)*: A numeric value for sigmaphi, i.e., the charge detrapping rate. Example: `sigmaphi = 5e-10`

**mu** numeric *(optional)*: A numeric value for mu, i.e., the light attenuation coefficient. Example: `mu = 0.9`

**age** numeric *(optional)*: The age (a) of the sample, if known. If data is a list of x samples, then age must be a numeric vector of length x. Example: `age = 10000`, or `age = c(1e4,1e5,1e6)`.

**Ddot** numeric *(optional)*: A numeric value for the environmental dose rate (Gy/ka). For this argument to be considered a value for D0 must also be provided; otherwise it will be ignored.

**D0** numeric *(optional)*: A numeric value for the characteristic saturation dose (Gy). For this argument to be considered a value for Ddot must also be provided; otherwise it will be ignored.

**weights** logical *(optional)*: If TRUE the fit will be weighted by the inverse square of the error. Requires data to be a data.frame with three columns.

**plot** logical *(optional)*: Show or hide the plot.

**legend** logical *(optional)*: Show or hide the equation inside the plot.

**error_bars** logical *(optional)*: Show or hide error bars (only applies if errors were provided).

**coord_flip** logical *(optional)*: Flip the coordinate system.

... Further parameters passed to plot. Custom parameters include:

- **verbose** (logical): show or hide console output
- **line_col**: Color of the fitted line
- **line_lty**: Type of the fitted line (see lty in ?par)
- **line_lwd**: Line width of the fitted line (see lwd in ?par)

**Details**

**Weighted fitting**

If weights = TRUE the function will use the inverse square of the error \(1/\sigma^2\) as weights during fitting using minpack.lm::nlsLM. Naturally, for this to take effect individual errors must be provided in the third column of the data.frame for data. Weighted fitting is not supported if data is a list of multiple data.frames, i.e., it is not available for global fitting.

**Dose rate**

If any of the arguments Ddot or D0 is at its default value (NULL), this function will fit eq. 1 in Sohbati et al. (2012a) to the data. If the effect of dose rate (i.e., signal saturation) needs to be considered, numeric values for the dose rate (Ddot) (in Gy/ka) and the characteristic saturation dose (D0) (in Gy) must be provided. The function will then fit eq. 12 in Sohbati et al. (2012b) to the data.
NOTE: Currently, this function does not consider the variability of the dose rate with sample depth (x)!. In the original equation the dose rate D is an arbitrary function of x (term D(x)), but here D is assumed constant.

**Global fitting**

If data is list of multiple data.frames, each representing a separate sample, the function automatically performs a global fit to the data. This may be useful to better constrain the parameters sigmaphi or mu and requires that known ages for each sample is provided (e.g., age = c(100,1000) if data is a list with two samples).

**Value**

Function returns results numerically and graphically:

---

**[ NUMERICAL OUTPUT ]**

---

RLum.Results-object

**slot: @data**

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$summary</td>
<td>data.frame</td>
<td>summary of the fitting results</td>
</tr>
<tr>
<td>$data</td>
<td>data.frame</td>
<td>the original input data</td>
</tr>
<tr>
<td>$fit</td>
<td>nls</td>
<td>the fitting object produced by minpack.lm::nlsLM</td>
</tr>
<tr>
<td>$args</td>
<td>character</td>
<td>arguments of the call</td>
</tr>
<tr>
<td>$call</td>
<td>call</td>
<td>the original function call</td>
</tr>
</tbody>
</table>

**slot: @info**

Currently unused.

---

**[ PLOT OUTPUT ]**

---

A scatter plot of the provided depth-intensity OSL surface exposure data with the fitted model.

**Function version**

0.1.0

**How to cite**

Note

This function has BETA status. If possible, results should be cross-checked.

Author(s)

Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


See Also

ExampleData.SurfaceExposure, minpack.lm::nlsLM

Examples

```r
## Load example data
data("ExampleData.SurfaceExposure")

## Example 1 - Single sample
# Known parameters: 10000 a, mu = 0.9, sigmaphi = 5e-10
sample_1 <- ExampleData.SurfaceExposure$sample_1
head(sample_1)
results <- fit_SurfaceExposure(sample_1, mu = 0.9, sigmaphi = 5e-10)
get_RLum(results)

## Example 2 - Single sample and considering dose rate
# Known parameters: 10000 a, mu = 0.9, sigmaphi = 5e-10,
# dose rate = 2.5 Gy/ka, D0 = 40 Gy
sample_2 <- ExampleData.SurfaceExposure$sample_2
head(sample_2)
results <- fit_SurfaceExposure(sample_2, mu = 0.9, sigmaphi = 5e-10,
                                Ddot = 2.5, D0 = 40)
get_RLum(results)

## Example 3 - Multiple samples (global fit) to better constrain 'mu'
# Known parameters: ages = 1e3, 1e4, 1e5, 1e6 a, mu = 0.9, sigmaphi = 5e-10
set_1 <- ExampleData.SurfaceExposure$set_1
str(set_1, max.level = 2)
results <- fit_SurfaceExposure(set_1, age = c(1e3, 1e4, 1e5, 1e6),
                                sigmaphi = 5e-10)
get_RLum(results)
```
## Example 4 - Multiple samples (global fit) and considering dose rate

Known parameters: ages = 1e2, 1e3, 1e4, 1e5, 1e6 a, mu = 0.9, sigmaphi = 5e-10,
# dose rate = 1.0 Ga/ka, D0 = 40 Gy

```
set_2 <- ExampleData.SurfaceExposure$set_2
str(set_2, max.level = 2)
results <- fit_SurfaceExposure(set_2, age = c(1e2, 1e3, 1e4, 1e5, 1e6),
                              sigmaphi = 5e-10, Ddot = 1, D0 = 40)
get_RLum(results)
```

---

### fit_ThermalQuenching

#### Fitting Thermal Quenching Data

**Description**

Applying a nls-fitting to thermal quenching data.

**Usage**

```r
d东营 fit_ThermalQuenching(data, start_param = list(),
method_control = list(), n.MC = 100, verbose = TRUE, plot = TRUE,
...)
```

**Arguments**

- **data** `data.frame` **(required)**: input data with three columns, the first column contains temperature values in deg. C, columns 2 and 3 the dependent values with its error
- **start_param** `list` **(optional)**: option to provide own start parameters for the fitting, see details
- **method_control** `list` **(optional)**: further options to fine tune the fitting, see details for further information
- **n.MC** `numeric` **(with default)**: number of Monte Carlo runs for the error estimation. If `n.MC` is NULL or <=1, the error estimation is skipped
- **verbose** `logical` **(with default)**: enables/disables terminal output
- **plot** `logical` **(with default)**: enables/disables plot output
- **...** further arguments that can be passed to control the plotting, support are `main`, `pch`, `col_fit`, `col_points`, `lty`, `lwd`, `xlab`, `ylab`, `xlim`, `ylim`, `xaxt`

**Details**

**Used equation**

The equation used for the fitting is

\[
y = \frac{A}{1 + C \times (exp(-W/(k \times x)))} + c
\]
**fit_ThermalQuenching**

$W$ is the energy depth in eV and $C$ is dimensionless constant. $A$ and $c$ are used to adjust the curve for the given signal. $k$ is the Boltzmann in eV/K and $x$ is the absolute temperature in K.

**Error estimation**

The error estimation is done by varying the input parameters using the given uncertainties in a Monte Carlo simulation. Errors are assumed to follow a normal distribution.

**start_param**

The function allows the injection of own start parameters via the argument `start_param`. The parameters needs to be provided as names list. The names are the parameters to be optimised. Examples: `start_param = list(A = 1, C = 1e+5, W = 0.5, c = 0)`

**method_control**

The following arguments can be provided via `method_control`. Please note that arguments provided via `method_control` are not further tested, i.e., if the function crashes your input was probably wrong.

<table>
<thead>
<tr>
<th>ARGUMENT</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper</td>
<td>named vector</td>
<td>sets upper fitting boundaries, if provided boundaries for all arguments are required, e.g., c(A = 0, C = 0, W = 0, c = 0)</td>
</tr>
<tr>
<td>lower</td>
<td>names vector</td>
<td>sets lower fitting boundaries (see upper for details)</td>
</tr>
<tr>
<td>trace</td>
<td>logical</td>
<td>enables/disables progression trace for <code>minpack.lm::nlsLM</code></td>
</tr>
<tr>
<td>weights</td>
<td>numeric</td>
<td>option to provide own weights for the fitting, the length of this vector needs to be equal to the number of rows in the input data frame</td>
</tr>
</tbody>
</table>

**Value**

The function returns numerical output and an *(optional)* plot.

---

### [ NUMERICAL OUTPUT ]

**RLum.Results-object**

**slot**: `@data`

`[.. $data : data.frame]`

A table with all fitting parameters and the number of Monte Carlo runs used for the error estimation.

`[.. $fit : nls object]`

The nls `stats::nls` object returned by the function `minpack.lm::nlsLM`. This object can be further passed to other functions supporting an nls object (cf. details section in `stats::nls`)

**slot**: `@info`

`[.. $call : call]`

The original function call.
Plotted are temperature against the signal and their uncertainties. The fit is shown as dashed-line (can be modified). Please note that for the fitting the absolute temperature values are used but are re-calculated to deg. C for the plot.

Function version

0.1.0

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR5060, CNRS - Université Bordeaux Montaigne (Fragne) , RLum Developer Team

References


See Also

minpack.lm::nlsLM

Examples

```r
##create short example dataset
data <- data.frame(
  T = c(25, 40, 50, 60, 70, 80, 90, 100, 110),
  V = c(0.06, 0.058, 0.052, 0.051, 0.041, 0.034, 0.035, 0.033, 0.032),
  V_X = c(0.012, 0.009, 0.008, 0.008, 0.007, 0.006, 0.005, 0.005, 0.004))

##fit
fit_ThermalQuenching(
data = data,
n.MC = NULL)
```
*get/Layout*

**Collection of layout definitions**

**Description**

This helper function returns a list with layout definitions for homogeneous plotting.

**Usage**

```r
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

**How to cite**


**Author(s)**

Michael Dietze, GFZ Potsdam (Germany), RLum Developer Team
Examples

```r
## 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, separated = FALSE)`

Arguments

- **ID**  character *(optional)*: quote ID to be returned.
- **separated**  logical *(with default)*: return result in separated form.

Value

Returns a character with quote and respective (false) author.

Function version

0.1.4
get_rightAnswer

How to cite


Author(s)

Michael Dietze, GFZ Potsdam (Germany), Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

Examples

## ask for an arbitrary quote

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

How to cite

get_Risoe.BINfileData

Author(s)
inspired by R.G., RLum Developer Team

Examples

```r
## 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
```r
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

How to cite
get_RLum

Author(s)

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

```r
get_RLum(object, ...)
```

```r
## S4 method for signature 'list'
get_RLum(object, class = NULL, null.rm = FALSE, ...)
```

```r
## S4 method for signature `NULL`
get_RLum(object, ...)
```

**Arguments**

- **object**
  - `RLum (required)`: S4 object of class `RLum` or an object of type `list` containing only objects of type `RLum`.
  - `...`: further arguments that will be passed to the object specific methods. For further details on the supported arguments please see the class documentation: `RLum.Data.Curve`, `RLum.Data.Spectrum`, `RLum.Data.Image`, `RLum.Analysis` and `RLum.Results`.

- **class**
  - `character (optional)`: allows to define the class that gets selected if applied to a list, e.g., if a list consists of different type of RLum-class objects, this arguments allows to make selection. If nothing is provided, all RLum-objects are treated.

- **null.rm**
  - `logical (with default)`: option to get rid of empty and NULL objects

**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
- NULL: Returns NULL

Function version

0.3.3

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

See Also


Examples

```r
# 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)

g# get results and store them in a new object
temp.get <- get_RLum(object = temp1)
```

GitHub-API

GitHub API

Description

R Interface to the GitHub API v3.
Usage

```r
github_commits(user = "r-lum", repo = "luminescence", 
               branch = "master", n = 5)

github_branches(user = "r-lum", repo = "luminescence")

github_issues(user = "r-lum", repo = "luminescence", verbose = TRUE)
```

Arguments

- **user** character (*with default*: GitHub user name (defaults to 'r-lum')).
- **repo** character (*with default*: name of a GitHub repository (defaults to 'luminescence')).
- **branch** character (*with default*: branch of a GitHub repository (defaults to 'master')).
- **n** integer (*with default*: number of commits returned (defaults to 5)).
- **verbose** logical (*with default*: print the output to the console (defaults to TRUE)).

Details

These functions can be used to query a specific repository hosted on GitHub.

`github_commits` lists the most recent `n` commits of a specific branch of a repository.

`github_branches` can be used to list all current branches of a repository and returns the corresponding SHA hash as well as an installation command to install the branch in R via the 'devtools' package.

`github_issues` lists all open issues for a repository in valid YAML.

Value

- **github_commits**: data.frame with columns:

  ```
  [,1]  SHA
  [,2]  AUTHOR
  [,3]  DATE
  [,4]  MESSAGE
  ```

- **github_branches**: data.frame with columns:

  ```
  [,1]  BRANCH
  [,2]  SHA
  [,3]  INSTALL
  ```

- **github_commits**: Nested list with `n` elements. Each commit element is a list with elements:
install_DevelopmentVersion

Function version

0.1.0

How to cite


Author(s)

Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


Examples

```r
# Not run:
github_branches(user = "r-lum", repo = "luminescence")
github_issues(user = "r-lum", repo = "luminescence")
github_commits(user = "r-lum", repo = "luminescence", branch = "master", n = 10)
```

## End(Not run)

install_DevelopmentVersion

Attempts to install the development version of the 'Luminescence'
package

Description

This function is a convenient method for installing the development version of the R package 'Luminescence' directly from GitHub.
Usage

install_DevelopmentVersion(force_install = FALSE)

Arguments

force_install  logical (optional): If FALSE (the default) the function produces and prints the required code to the console for the user to run manually afterwards. When TRUE and all requirements are fulfilled (see details) this function attempts to install the package itself.

Details

This function uses Luminescence::github_branches to check which development branches of the R package ‘Luminescence’ are currently available on GitHub. The user is then prompted to choose one of the branches to be installed. It further checks whether the R package ‘devtools’ is currently installed and available on the system. Finally, it prints R code to the console that the user can copy and paste to the R console in order to install the desired development version of the package.

If force_install=TRUE the functions checks if ‘devtools’ is available and then attempts to install the chosen development branch via devtools::remote-reexports.

Value

This function requires user input at the command prompt to choose the desired development branch to be installed. The required R code to install the package is then printed to the console.

Examples

```r
## Not run:
install_DevelopmentVersion()
## End(Not run)
```

---

### 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

How to cite


Author(s)

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

See Also


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)
merge_Risoe.BINfileData

Arguments

input.objects character with Risoe.BINfileData objects (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)). Alternatively a list is supported.

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

0.2.7

How to cite

Note

The validity of the output objects is not further checked.

Author(s)

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

References


See Also

Risoe.BINfileData, read_BIN2R, write_R2BIN

Examples

```r
## 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 RLucm 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.
merge_RLum

<table>
<thead>
<tr>
<th>object</th>
<th>corresponding merge function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLum.Analysis</td>
<td>merge_RLum.Analysis</td>
</tr>
<tr>
<td>RLum.Results</td>
<td>merge_RLum.Results</td>
</tr>
</tbody>
</table>

Value

Return is the same as input objects as provided in the list.

Function version

0.1.2

How to cite


Note

So far not for every RLum object a merging function exists.

Author(s)

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

See Also


Examples

```r
##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)))
```
names_RLum  

*S4-names function for RLum S4 class objects*

Description

Function calls object-specific names functions for RLum S4 class objects.

Usage

```r
names_RLum(object)
```

```r
## S4 method for signature 'list'
names_RLum(object)
```

Arguments

```r
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

Methods (by class)

- `list`: Returns a list of RLum objects that had been passed to `names_RLum`

Function version

0.1.0

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), RLum Developer Team
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 = TRUE, 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),
The 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.

See Also
plot.ratio  numeric: Relative space, given to the radial versus the cartesian plot part, default 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 strecthing 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 TRUE.

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 themselves.

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 modified 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**

0.1.11
How to cite


Author(s)

Michael Dietze, GFZ Potsdam (Germany)
Sebastian Kreutzer, RAMAT-CRP2A, Universite Bordeaux Montaigne (France)
Inspired by a plot introduced by Galbraith & Green (1990), RLum Developer Team

References


See Also

plot_RadialPlot, plot_KDE, plot_Histogram

Examples

```r
## 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",  
legend.position = "bottom")
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)
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 R Lum.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): selected 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)
plot_DetPlot

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 consecutively expanding 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.

analyse_function.control

The argument `analyse_function.control` currently supports the following arguments `sequence.structure`, `dose.points`, `mtext.outer`, `fit.method`, `fit.force_through_origin`, `plot`, `plot.single`

Value

A plot and an `RLum.Results` object with the produced De values

@data:

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>De.values</td>
<td>data.frame</td>
<td>table with De values</td>
</tr>
<tr>
<td>signal_integral.seq</td>
<td>numeric</td>
<td>integral sequence used for the calculation</td>
</tr>
</tbody>
</table>

@info:

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>call</td>
<td>the original function call</td>
</tr>
</tbody>
</table>

Function version

0.1.3

How to cite


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, UMR 5060, CNRS - Université Bordeaux Montaigne (France) , RLum Developer Team

References

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_DRCSummary**

Create a Dose-Response Curve Summary Plot

Description
While analysing OSL SAR or pIRIR data the view on the data is limited usually to one dose-response curve (DRC) at the time for one aliquot. This function overcomes this limitation by plotting all DRC from an RLum.Results object created by the function analyse_SAR.CWOSL in one single plot.
Usage

plot_DRCSummary(object, source_dose_rate = NULL, sel_curves = NULL,
show_dose_points = FALSE, show_natural = FALSE, n = 51L, ...)

Arguments

object  RLum.Results object (required): input object created by the function analysyse_SAR.CWOSL. The input object can be provided as list.
source_dose_rate  numeric (optional): allows to modify the axis and show values in Gy, instead seconds. Only a single numerical values is allowed.
sel_curves  numeric (optional): id of the curves to be plotting in its occurring order. A sequence can be provided for selecting, e.g., only every 2nd curve from the input object
show_dose_points  logical (with default): enable or disable plot of dose points in the graph
show_natural  logical (with default): enable or disable the plot of the natural Lx/Tx values
n  integer (with default): the number of x-values used to evaluate one curve object. Large numbers slow down the plotting process and are usually not needed

Details

If you want plot your DRC on an energy scale (dose in Gy), you can either use the option source_dose_rate provided below or your can SAR analysis with the dose points in Gy (better axis scaling).

Value

An RLum.Results object is returned:

Slot: @data

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>results</td>
<td>data.frame</td>
<td>with dose and Lx/Tx values</td>
</tr>
<tr>
<td>data</td>
<td>RLum.Results</td>
<td>original input data</td>
</tr>
</tbody>
</table>

Slot: @info

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>call</td>
<td>the original function call</td>
</tr>
<tr>
<td>args</td>
<td>list</td>
<td>arguments of the original function call</td>
</tr>
</tbody>
</table>

Note: If the input object is a list a list of RLum.Results objects is returned.
Function version

0.2.1

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France)
Christoph Burow, University of Cologne, RLum Developer Team

See Also

RLum.Results, analyse_SAR.CWOSL

Examples

#load data example 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)

results <- analyse_SAR.CWOSL(
  object = object,
  signal.integral.min = 1,
  signal.integral.max = 2,
  background.integral.min = 900,
  background.integral.max = 1000,
  plot = FALSE
)

##plot only DRC
plot_DRCSummary(results)

plot_DRTResults

Visualise dose recovery test results

Description

The function provides a standardised plot output for dose recovery test measurements.
plot_DRTResults

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 given.dose in NULL the 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),
• "weighted$mean" (error-weighted mean),
• "median" (median of the De values),
• "sd.rel" (relative standard deviation in percent),
• "sd.abs" (absolute standard deviation),
• "se.rel" (relative standard error) and
• "se.abs" (absolute standard error)

and all other measures returned by the function calc_Statistics.

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 whether NA values are removed before plotting from the input data set

... further arguments and graphical parameters passed to *plot*, supported are: *xlab*, *ylab*, *xlim*, *ylim*, *main*, *cex*, *las* and *pch*

**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**

0.1.13

**How to cite**


**Note**

Further data and plot arguments can be added by using the appropriate R commands.

**Author(s)**

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060 - Université Bordeaux Montaigne (France)
Michael Dietze, GFZ Potsdam (Germany), RLum Developer Team

**References**


**See Also**

plot
Examples

```r
## 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
```
### Description

The function allows to plot transmission windows for different filters. Missing data for specific wavelengths 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

```r
plot_FilterCombinations(filters, wavelength_range = 200:1000,
                        show_net_transmission = TRUE, interactive = FALSE, 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.
- **interactive**: logical (with default): enable/disable interactive plot
- **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

Calculations

Net transmission window

The net transmission window of two filters is approximated by

\[ T_{\text{final}} = T_1 \times T_2 \]

Optical density

\[ OD = -\log(T) \]

Total optical density

\[ OD_{\text{total}} = OD_1 + OD_2 \]

Please consider using own calculations for more precise values.

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 columns 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 reflection 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, P = 0.9))`. The final transmission becomes:

\[ Transmission = Transmission(d) \times P \]

Advanced plotting parameters

The following further non-common plotting parameters can be passed to the function:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>legend</td>
<td>logical</td>
<td>enable/disable legend</td>
</tr>
</tbody>
</table>
plot_FilterCombinations

legend.pos character change legend position (graphics::legend)
legend.text character same as the argument legend in (graphics::legend)
net_transmission.col col colour of net transmission window polygon
net_transmission.col_lines col colour of net transmission window polygon lines
net_transmission.density numeric specify line density in the transmission polygon
grid list full list of arguments that can be passed to the function graphics::grid

For further modifications standard additional R plot functions are recommended, e.g., the legend can be fully customised by disabling the standard legend and use the function graphics::legend instead.

Value

Returns an S4 object of type RLum.Results.

@data

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net_transmission_window</td>
<td>matrix</td>
<td>the resulting net transmission window</td>
</tr>
<tr>
<td>OD_total</td>
<td>matrix</td>
<td>the total optical density</td>
</tr>
<tr>
<td>filter_matrix</td>
<td>matrix</td>
<td>the filter matrix used for plotting</td>
</tr>
</tbody>
</table>

@info

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>call</td>
<td>the original function call</td>
</tr>
</tbody>
</table>

Function version

0.3.1

How to cite


Author(s)

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

See Also

RLum.Results, approx

Examples

```r
## (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

## Example 3 show optical density
plot(results$OD_total)

## Not run:
##Example 4
##show the filters using the interactive mode
plot_FilterCombinations(filters = list(filter1, filter2), interactive = TRUE)

## End(Not run)

---

**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 or additive protocol. The function supports interpolation and extrapolation to calculate the equivalent dose.

**Usage**

```r
plot_GrowthCurve(sample, na.rm = TRUE, mode = "interpolation", 
  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.

dos0 character (with default): selects calculation mode of the function.
• "interpolation" (default) calculates the De by interpolation,
• "extrapolation" calculates the De by extrapolation and
• "alternate" calculates no De and just fits the data points.
Please note that for option "regenerative" the first point is considered as natural
dose

fit.method character (with default): function used for fitting. Possible options are:
• LIN,
• QDR,
• EXP,
• EXP OR LIN,
• EXP+LIN,
• EXP+EXP or
• GOK.
See details.

fit.force_through_origin logical (with default) allow to force the fitted function through the origin. For
method = "EXP+EXP" and method = "GOK" the function will go through the ori-
gin 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, EXP OR LIN and GOK. 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:
plot_GrowthCurve

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 `minpack.lm::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`:

\[ y = m \times x + n \]

**QDR:** fits a linear function to the data using `lm`:

\[ y = a + b \times x + c \times x^2 \]

**EXP:** try to fit a function of the form

\[ y = a \times \left(1 - \exp\left(-\frac{x + c}{b}\right)\right) \]

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:

\[ y = a \times \left(1 - \exp\left(-\frac{x + c}{b}\right) + (g \times x)\right) \]

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

\[ y = (a_1 \times (1 - \exp((-x)/b1))) + (a_2 \times (1 - \exp((-x)/b2))) \]
This fitting procedure is not robust against wrong start parameters and should be further improved. GOK tries to fit the general-order kinetics function after Guralnik et al. (2015) of the form of

\[ y = a \cdot (1 - (1 + (1/b) \cdot x \cdot c)^{(-1/c)}) \]

where \( c > 0 \) is a kinetic order modifier (not to be confused with \( c \) in EXP or EXP+LIN!).

**Fit weighting**

If the option `fit.weights = TRUE` is chosen, weights are calculated using provided signal errors (Lx/Tx error):

\[ \text{fit.weights} = 1 / \text{error} / (\text{sum}(1/\text{error})) \]

**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 sampled 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 RLucResults object is returned containing, the slot data contains the following elements:

<table>
<thead>
<tr>
<th>DATA.OBJECT</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>..$De</td>
<td>data.frame</td>
<td>Table with De values</td>
</tr>
<tr>
<td>..$De.MC</td>
<td>numeric</td>
<td>Table with De values from MC runs</td>
</tr>
<tr>
<td>..$Fit</td>
<td>nls or lm</td>
<td>object from the fitting for EXP, EXP+LIN and EXP+EXP. In case of a resulting linear fit when using LIN, QDR or EXP OR LIN.</td>
</tr>
<tr>
<td>..$Formula</td>
<td>expression</td>
<td>Fitting formula as R expression</td>
</tr>
<tr>
<td>..$call</td>
<td>call</td>
<td>The original function call</td>
</tr>
</tbody>
</table>

**Function version**

1.10.8

**How to cite**

plot_GrowthCurve

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France)
Michael Dietze, GFZ Potsdam (Germany), RLum Developer Team

References

See Also
nls, RLum.Results, get_RLum, minpack.lm::nlsLM, lm, uniroot

Examples

```r
#(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"
)

#(5) plot using the 'extrapolation' mode
LxTxData[1,2:3] <- c(0.5, 0.001)
print(plot_GrowthCurve(LxTxData,mode = "extrapolation"))

#(6) plot using the 'alternate' mode
LxTxData[1,2:3] <- c(0.5, 0.001)
print(plot_GrowthCurve(LxTxData,mode = "alternate"))
```
##(7) import and fit test data set by Berger & Huntley 1989

```r
QNL84_2_unbleached <- read.table(system.file("extdata/QNL84_2_unbleached.txt", package = "Luminescence"))

results <- plot_GrowthCurve(
  QNL84_2_unbleached,
  mode = "extrapolation",
  plot = FALSE,
  verbose = FALSE)

# calculate confidence interval for the parameters
# as alternative error estimation
confint(results$Fit, level = 0.68)
```

## Not run:

```r
QNL84_2_bleached <- read.table(system.file("extdata/QNL84_2_bleached.txt", package = "Luminescence"))
STRB87_1_unbleached <- read.table(system.file("extdata/STRB87_1_unbleached.txt", package = "Luminescence"))
STRB87_1_bleached <- read.table(system.file("extdata/STRB87_1_bleached.txt", package = "Luminescence"))

print(
  plot_GrowthCurve(
    QNL84_2_bleached,
    mode = "alternate",
    plot = FALSE,
    verbose = FALSE)$Fit)

print(
  plot_GrowthCurve(
    STRB87_1_unbleached,
    mode = "alternate",
    plot = FALSE,
    verbose = FALSE)$Fit)

print(
  plot_GrowthCurve(
    STRB87_1_bleached,
    mode = "alternate",
    plot = FALSE,
    verbose = FALSE)$Fit)

## End(Not run)
```

---

**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

How to cite


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), RLum Developer Team

See Also

hist, plot
Examples

```r
## 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**

```r
plot_KDE(data, na.rm = TRUE, values.cumulative = TRUE, order = TRUE,
    boxplot = TRUE, rug = TRUE, summary, summary.pos,
    summary.method = "MCM", bw = "nrd0", output = TRUE, ...)
```

**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)).
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)
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
3.6.0

How to cite

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, UMR 5060, CNRS-Université Bordeaux Montaigne, France, RLum Developer Team

See Also
density, plot

Examples

```r
## 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")
```
## Visualise natural/regenerated signal ratios

This function creates a Natural/Regenerated signal vs. time (NR(t)) plot as shown in Steffen et al. 2009.
plot_NRt

Usage

plot_NRt(data, log = FALSE, smooth = c("none", "spline", "rmean"), k = 3, legend = TRUE, legend.pos = "topright", ...)  

Arguments

data list, data.frame, matrix or RLum.Analysis (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.

How to cite


Author(s)

Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


See Also

plot
Examples

```r
## 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")
```
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, 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.

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
0.5.5

How to cite
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, RLum Developer Team

References


See Also

plot, plot_KDE, plot_Histogram

Examples

```r
## 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
```
```r
# 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"逝,
                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 (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.


dose_rate numeric (optional): dose rate of the irradiation 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

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

How to cite

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), RLum Developer Team

References

See Also
Risoe.BINfileData, read_BIN2R, CW2pLM, CW2pLMi, CW2pPMi, CW2pHMi

Examples

```r
## 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.

<table>
<thead>
<tr>
<th>object</th>
<th>corresponding plot function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLuM.Data.Curve</td>
<td>plot_RLuM.Data.Curve</td>
</tr>
<tr>
<td>RLuM.Data.Spectrum</td>
<td>plot_RLuM.Data.Spectrum</td>
</tr>
<tr>
<td>RLuM.Data.Image</td>
<td>plot_RLuM.Data.Image</td>
</tr>
<tr>
<td>RLuM.Analysis</td>
<td>plot_RLuM.Analysis</td>
</tr>
<tr>
<td>RLuM.Results</td>
<td>plot_RLuM.Results</td>
</tr>
</tbody>
</table>

**Value**

Returns a plot.

**Function version**

0.4.3

**How to cite**


**Note**

The provided plot output depends on the input object.

**Author(s)**

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

**See Also**

Examples

```r
# 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 RLuc.Analysis S4 class object*

### Description

The function provides a standardised plot output for curve data of an RLuc.Analysis S4 class object

### Usage

```r
plot_RLum.Analysis(object, subset = NULL, nrows, ncols, abline = NULL,
combine = FALSE, curve.transformation, plot.single = FALSE, ...)
```

### Arguments

- **object**: `RLuc.Analysis (required)`: S4 object of class RLuc.Analysis
- **subset**: named list (optional): subsets elements for plotting. The arguments in the named list will be directly passed to the function `get_RLuc` (e.g., `subset = list(curveType = "measured")`)
- **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 forwarded 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 RLuc.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, CW2pHM and CW2pPMi. See details.
plot.single logical (with default): global par settings are considered, normally this should end in one plot per page...

plot function, default 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

0.3.11

How to cite


Note

Not all arguments available for plot will be passed and they partly do not behave in the way you might expect them to work. This function was designed to serve as an overview plot, if you want to have more control, extract the objects and plot them individually.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLM Developer Team
See Also

plot, plot_RLum, plot_RLum.Data.Curve

Examples

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

# convert values for position 1
temp <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)

#(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))
)

#(2) same as example (1) but using
## the argument smooth = TRUE
plot_RLum.Analysis(
  temp,
  subset = list(recordType = "TL"),
  combine = TRUE,
  norm = TRUE,
  smooth = 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

```r
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 `zoo::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**

0.2.3

**How to cite**


**Note**

Not all arguments of `plot` will be passed!

**Author(s)**

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

**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.Image S4 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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>RLum.Data.Image <em>(required)</em>: S4 object of class RLum.Data.Image</td>
</tr>
<tr>
<td>par.local</td>
<td>logical <em>(with default)</em>: 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.</td>
</tr>
<tr>
<td>plot.type</td>
<td>character <em>(with default)</em>: plot types. Supported types are plot.raster, plotRGB or contour</td>
</tr>
<tr>
<td>...</td>
<td>further arguments and graphical parameters that will be passed to the specific plot functions.</td>
</tr>
</tbody>
</table>

**Details**

**Details on the plot functions**

Image is visualised as 2D plot using 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::raster: 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 `raster::plotRGB` from the `raster::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 (`raster::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

**How to cite**


**Note**

This function has been created to facilitate 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), RLum Developer Team

**See Also**

`RLum.Data.Image`, `plot`, `plot_RLum`, `raster::raster`

**Examples**

```r
# load data
data(ExampleData.RLum.Data.Image, envir = environment())

# plot data
```
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.spectrum = NULL, bg.channels = NULL, bin.rows = 1, bin.cols = 1, norm = NULL, 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`. If you provide already binned spectra, the colour assignment is likely to be wrong, since the colour gradients are calculated using the bin number.
- **bg.spectrum**: RLum.Data.Spectrum or matrix (optional): Spectrum used for the background subtraction. By definition, the background spectrum should have been measured with the same setting as the signal spectrum. If a spectrum is provided, the argument `bg.channels` works only on the provided background spectrum.
- **bg.channels**: vector (optional): defines channel for background subtraction. If a vector is provided the mean of the channels is used for subtraction. If a spectrum is provided via `bg.spectrum`, this argument only works on the `bg.spectrum`. **Note**: Background subtraction is applied prior to channel binning.
**bin.rows** integer (with default): allow summing-up wavelength channels (horizontal binning), e.g. `bin.rows = 2` two channels are summed up. Binning is applied after the background subtraction.

**bin.cols** integer (with default): allow summing-up channel counts (vertical binning) for plotting, e.g. `bin.cols = 2` two channels are summed up. Binning is applied after the background subtraction.

**norm** character (optional): Normalise data to the maximum (`norm = "max"`) or minimum (`norm = "min"`) count values. The normalisation is applied after the binning.

**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 helpful especially in case of TL-spectra.

**xaxis.energy** logical (with default): enables or disables energy instead of wavelength axis. For the conversion the function `convert_Wavelength2Energy` is used.

**Note:** This option means not only simply redrawing the axis, instead the spectrum in terms of intensity is recalculated, s. details.

**legend.text** character (with default): possibility 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

**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
plot_RLum.Data.Spectrum

- 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 ("single", "multiple.lines", "interactive"), col, border, box lwd, bty, showscale ("interactive")

Value
Returns a plot.

Function version
0.6.2

How to cite

Note
Not all additional arguments (…) will be passed similarly!

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

See Also
RLum.Data.Spectrum, convert_Wavelength2Energy, plot, plot_RLum, persp, plotly::plot_ly, contour
Examples

```r
# 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 spectrum on energy axis
# please note the background subtraction
plot_RLum.Data.Spectrum(TL.Spectrum,
    plot.type = "persp",
    ylim = c(0, 200),
    bg.channels = 10,
    bin.cols = 1,
    xaxis.energy = TRUE)

## (4) 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 ("surface")
plot_RLum.Data.Spectrum(TL.Spectrum, plot.type = "interactive",
                        xlim = c(310, 750), ylim = c(0, 300), bin.rows = 10,
                        bin.cols = 1)

## (5) interactive plot using the package plotly ("contour")
plot_RLum.Data.Spectrum(TL.Spectrum, plot.type = "interactive",
                        xlim = c(310, 750), ylim = c(0, 300), bin.rows = 10,
                        bin.cols = 1,
                        type = "contour",
                        )
```

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.
...

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
How to cite


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), RLum Developer Team

See Also

plot, plot_RLum

Examples

```R
###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
plot_ViolinPlot

Description

Draws a kernel density 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 too small to be visualised. The idea for this plot is based on the 'volcano plot' in the ggplot2 package by Hadley 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 (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, stats::density and boxplot. See details for further information.

Details

The function is passing several arguments to the function plot, stats::density, graphics::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

0.1.4

How to cite

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), RLum Developer 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=vioplot


See Also

stats::density, plot, boxplot, rug, calc_Statistics

Examples

```r
## 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)
```

---

**PSL2Risoe.BINfileData**  
*Convert portable OSL data to an Risoe.BINfileData object*

**Description**

Converts an RLum.Analysis object produced by the function read_PSL2R() to an Risoe.BINfileData object (BETA).

**Usage**

```r
PSL2Risoe.BINfileData(object, ...)
```
Arguments


... currently not used.

Details

This function converts an `RLum.Analysis` object that was produced by the `read_PSL2R` function to an `Risoe.BINfileData`. The `Risoe.BINfileData` can be used to write a Risoe BIN file via `write_R2BIN`.

Value

Returns an S4 `Risoe.BINfileData` object that can be used to write a BIN file using `write_R2BIN`.

Function version

0.0.1

How to cite


Author(s)

Christoph Burow, University of Cologne (Germany), RLum Developer Team

See Also

`RLum.Analysis`, `RLum.Data.Curve`, `Risoe.BINfileData`

Examples

```r
# (1) load and plot example data set
data("ExampleData.portableOSL", envir = environment())
plot_RLum(ExampleData.portableOSL)

# (2) merge all RLum.Analysis objects into one
merged <- merge_RLum(ExampleData.portableOSL)
merged

# (3) convert to RisoeBINfile object
bin <- PSL2Risoe.BINfileData(merged)
bin

# (4) write Risoe BIN file
## Not run:
```
read_BIN2R

Import Risø BIN/BINX-files into R

Description

Import a *.bin or a *.binx file produced by a Risoe DA15 and DA20 TL/OSL reader into R.

Usage

```r
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,
           ignore.RECTYPE = FALSE, pattern = NULL, verbose = TRUE, ...)
```

Arguments

- **file**: character or list (required): path and file name of the BIN/BINX file (URLs are supported). 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 usually 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 predecessor 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 object. Can be provided as list if file is a list.
### 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/BINX-file structure, the reader is referred to the Risoe website: [http://www.nutech.dtu.dk/](http://www.nutech.dtu.dk/)

### Value

Returns an S4 `Risoe.BINfileData` 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

0.16.1

### How to cite

Note

The function works for BIN/BINX-format versions 03, 04, 05, 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 during import.**

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France)
Margret C. Fuchs, HZDR Freiberg, (Germany)
based on information provided by Torben Lapp and Karsten Bracht Nielsen (Risø DTU, Denmark)
, RLum Developer Team

References


See Also

write_R2BIN, Risoe.BINfileData, base::readBin, merge_Risoe.BINfileData, RLum.Analysis utils::txtProgressBar, list.files

Examples

```r
##(1) import Risoe BIN-file to R (uncomment for usage)

#FILE <- file.choose()
#temp <- read_BIN2R(FILE)
#temp
```

### read_Daybreak2R

Import measurement data produced by a Daybreak TL/OSL reader into R

**Description**

Import a TXT-file (ASCII file) or a DAT-file (binary file) produced by a Daybreak reader into R. The import of the DAT-files is limited to the file format described for the software TLAPllic v.3.2 used for a Daybreak, model 1100.

**Usage**

```r
read_Daybreak2R(file, raw = FALSE, 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.

raw logical (with default): if the input is a DAT-file (binary) a data.table::data.table instead of the RLum.Analysis object can be returned for debugging purposes.

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

0.3.1

How to cite


Note

[BETA VERSION] This function still needs to be tested properly. In particular the function has underwent only very rough rests using a few files.

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
Anotine Zink, C2RMF, Palais du Louvre, Paris (France)
The ASCII-file import is based on a suggestion by William Amidon and Andrew Louis Gorin , RLum Developer Team

See Also

RLum.Analysis, RLum.Data.Curve, data.table::data.table

Examples

```r
## Not run:
file <- file.choose()
temp <- read_Daybreak2R(file)
## End(Not run)
```
Description

Imports PSL files produced by a SUERC portable OSL reader into R (BETA).

Usage

```r
read_PSL2R(file, drop_bg = FALSE, as_decay_curve = TRUE,
            smooth = FALSE, merge = FALSE, ...)
```

Arguments

- `file` character (required): path and file name of the PSL file. If input is a vector it should comprise only characters representing valid paths and PSL file names. Alternatively the input character can be just a directory (path). In this case the function tries to detect and import all PSL files found in the directory.
- `drop_bg` logical (with default): TRUE to automatically remove all non-OSL/IRSL curves.
- `as_decay_curve` logical (with default): Portable OSL Reader curves are often given as cumulative light sum curves. Use TRUE (default) to convert the curves to the more usual decay form.
- `smooth` logical (with default): TRUE to apply Tukey’s Running Median Smoothing for OSL and IRSL decay curves. Smoothing is encouraged if you see random signal drops within the decay curves related to hardware errors.
- `merge` logical (with default): TRUE to merge all RLum.Analysis objects. Only applicable if multiple files are imported.
- `...` currently not used.

Details

This function provides an import routine for the SUERC portable OSL Reader PSL format. PSL files are just plain text and can be viewed with any text editor. Due to the formatting of PSL files this import function relies heavily on regular expression to find and extract all relevant information. See note.

Value

Returns an S4 RLum.Analysis object containing RLum.Data.Curve objects for each curve.

Function version

0.0.1
How to cite


Note

Because this function relies heavily on regular expressions to parse PSL files it is currently only in beta status. If the routine fails to import a specific PSL file please report to christoph.burow@gmx.net so the function can be updated.

Author(s)

Christoph Burow, University of Cologne (Germany), RLum Developer Team

See Also


Examples

# (1) Import PSL file to R

code <- system.file("extdata", "DorNie_0016.psl", package = "Luminescence")
psl <- read_PSL2R(file, drop_bg = FALSE, as_decay_curve = TRUE, smooth = TRUE, merge = FALSE)
print(str(psl, max.level = 3))
plot(psl, combine = TRUE)

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, verbose = 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"). Additionally internet connections are supported.

- **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.

- **verbose** logical (with default): enables or disables verbose mode

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

0.1.2

How to cite

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, Université Bordeaux Montaigne (France), RLum Developer Team

References


Hall, C., 2012: readSPE.m. http://www.mathworks.com/matlabcentral/fileexchange/35940-readspe/content/readSPE.m

See Also

readBin, RLum.Data.Spectrum, raster::raster

Examples

```r
## 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", verbose = TRUE,
              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

- **verbose**  
  logical (with default): enable or disable verbose mode. If verbose is FALSE the txtProgressBar is also switched off

- **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
<?xml?>
<Sample>
  <Sequence>
```
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 lexsysy 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:

1. HE: Heating element
2. PMT: Photomultiplier tube
3. 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 multiple 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:
read_XSYG2R

- data.frame with information on file.
- 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**

0.6.7

**How to cite**


**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, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

**References**


**Further reading**


**See Also**

xml, RLum.Analysis, RLum.Data.Curve, approx
Examples

```r
##(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` *RLum (required)*: an RLum object
- `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
How to cite


Author(s)

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

See Also

RLum

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, show_report = 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).
show_report logical (with default): If set to TRUE the function tries to display the report output in the local viewer, e.g., within RStudio after rendering.
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 rmarkdown::render and has the following structure:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>A summary of general characteristics of the object</td>
</tr>
<tr>
<td>Object content</td>
<td>A comprehensive list of the complete structure and content of the provided object.</td>
</tr>
<tr>
<td>Object structure</td>
<td>Summary of the objects structure given as a table</td>
</tr>
<tr>
<td>File</td>
<td>Information on the saved RDS file</td>
</tr>
<tr>
<td>Session Info</td>
<td>Captured output from sessionInfo()</td>
</tr>
<tr>
<td>Plots</td>
<td>(optional) For RLum-class objects a variable number of plots</td>
</tr>
</tbody>
</table>

The structure of the report can be controlled individually by providing one or more of the following arguments (all logical):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>Hide or show general information on the object</td>
</tr>
<tr>
<td>main</td>
<td>Hide or show the object’s content</td>
</tr>
<tr>
<td>structure</td>
<td>Hide or show object’s structure</td>
</tr>
<tr>
<td>rds</td>
<td>Hide or show information on the saved RDS file</td>
</tr>
<tr>
<td>session</td>
<td>Hide or show the session info</td>
</tr>
<tr>
<td>plot</td>
<td>Hide or show the plots (depending on object)</td>
</tr>
</tbody>
</table>

Note that these arguments have higher precedence than compact.

Further options that can be provided via the ... argument:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short_table</td>
<td>If TRUE only show the first and last 5 rows of lang tables.</td>
</tr>
<tr>
<td>theme</td>
<td>Specifies the Bootstrap theme to use for the report. Valid themes include &quot;default&quot;, &quot;cerulean&quot;, &quot;journal&quot;, &quot;flatly&quot;, &quot;readable&quot;, &quot;spacelab&quot;, &quot;united&quot;, &quot;cosmo&quot;, &quot;lumen&quot;, &quot;paper&quot;, &quot;sandstone&quot;, &quot;simplex&quot;, and &quot;yeti&quot;.</td>
</tr>
<tr>
<td>highlight</td>
<td>Specifies the syntax highlighting style. Supported styles include &quot;default&quot;, &quot;tango&quot;, &quot;pygments&quot;, &quot;kate&quot;, &quot;monokai&quot;, &quot;rainbow&quot;, &quot;pl��&quot;, &quot;tomorrow&quot;, &quot;snow&quot;, &quot;cisco&quot;, &quot;toke&quot;, &quot;zenburn&quot;, and &quot;haddock&quot;.</td>
</tr>
<tr>
<td>css</td>
<td>TRUE or FALSE to enable/disable custom CSS styling</td>
</tr>
</tbody>
</table>

The following arguments can be used to customise the report via CSS (Cascading Style Sheets):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
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
 0.1.1

How to cite

Note
 This function requires the R packages ’rmarkdown’, ’pander’ and ’rstudioapi’.

Author(s)
 Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

See Also
 rmarkdown::render, pander::pander_return, pander::openFileInOS, rstudioapi::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.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", keep.empty = TRUE, 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.

keep.empty  
logical (with default): If TRUE (default) an RLum.Analysis object is returned even if it does not contain any records. Set to FALSE to discard all empty objects.

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.
RLum-class

Value

Returns an RLum.Analysis object.

Function version

0.4.2

How to cite


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), RLum Developer Team

See Also

Risoe.BINfileData, RLum.Analysis, read_BIN2R

Examples

```r
# 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 Sublasses are:

Usage

```r
## S4 method for signature 'RLum'
replicate_RLum(object, times = NULL)
```
Arguments

object \textbf{RLum (required):} an object of class \textbf{RLum}
times \textbf{integer (optional):} number for times each element is repeated element

Details

\textbf{RLum-class}

\begin{itemize}
\item RLum.Data
\item RLum.Data.Curve
\item RLum.Data.Spectrum
\item RLum.Data.Image
\item RLum.Analysis
\item RLum.Results
\end{itemize}

Methods (by generic)

\begin{itemize}
\item \textit{replicate\_RLum:} Replication method RLum-objects
\end{itemize}

Slots

\begin{itemize}
\item originator Object of class character containing the name of the producing function for the object.
  Set automatically by using the function \texttt{set\_RLum}.
\item info Object of class list for additional information on the object itself
\item .uid Object of class character for a unique object identifier. This id is usually calculated using
  the internal function \texttt{create\_UID()} if the function \texttt{set\_RLum} is called.
\item .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.
\end{itemize}

Objects from the Class

A virtual Class: No objects can be created from it.

Class version

0.4.0

How to cite


Note

\texttt{RLum} is a virtual class.
Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

See Also


Examples

showClass("RLum")

---

table

scale_GammaDose

Calculate the gamma dose deposited within a sample taking layer-to-layer variations in radioactivity into account (according to Aitken, 1985)

Description

This function calculates the gamma dose deposited in a luminescence sample taking into account layer-to-layer variations in sediment radioactivity. The function scales user inputs of Uranium, Thorium and Potassium based on input parameters for sediment density, water content and given layer thicknesses and distances to the sample.

Usage

scale_GammaDose(data, conversion_factors = c("Cresswelletal2019", "Guerinetal2011", "AdamiecAitken1998", "Liritzisetal2013")[1], fractional_gamma_dose = c("Aitken1985")[1], verbose = TRUE, plot = TRUE, plot_single = TRUE, ...)

Arguments

data data.frame (required): A table containing all relevant information for each individual layer. The table must have the following named columns:
  - id (character): an arbitrary id or name of each layer
  - thickness (numeric): vertical extent of each layer in cm
  - sample_offset (logical): distance of the sample in cm, measured from the BOTTOM OF THE TARGET LAYER. Except for the target layer all values must be NA.
  - K (numeric): K nuclide content in
  - K_se (numeric): error on the K content
  - Th (numeric): Th nuclide content in ppm
scale_GammaDose

- \( \text{Th}\_\text{se} \) (numeric): error on the Th content
- \( \text{U} \) (numeric): U nuclide content in ppm
- \( \text{U}\_\text{se} \) (numeric): error on the U content
- \( \text{water\_content} \) (numeric): water content of each layer in
- \( \text{water\_content\_se} \) (numeric): error on the water content
- \( \text{density} \) (numeric): bulk density of each layer in g/cm\(^3\)

conversion_factors

- character (optional): The conversion factors used to calculate the dose rate from sediment nuclide contents. Valid options are:
  - "Cresswelletal2019" (default)
  - "Liritzisetal2013"
  - "Guerinetal2011"
  - "AdamiecAitken1998"

fractional_gamma_dose

- character (optional): Factors to scale gamma dose rate values. Valid options are:
  - "Aitken1985" (default): Table H1 in the appendix

verbose logical (optional): Show or hide console output (defaults to TRUE).

plot logical (optional): Show or hide the plot (defaults to TRUE).

plot_single logical (optional): Show all plots in one panel (defaults to TRUE).

Details

User Input

To calculate the gamma dose which is deposited in a sample, the user needs to provide information on those samples influencing the luminescence sample. As a rule of thumb, all sediment layers within at least 30 cm radius from the luminescence sample taken should be taken into account when calculating the gamma dose rate. However, the actual range of gamma radiation might be different, depending on the emitting radioelement, the water content and the sediment density of each layer (Aitken, 1985). Therefore the user is advised to provide as much detail as possible and physically sensible.

The function requires a data.frame that is to be structured in columns and rows, with samples listed in rows. The first column contains information on the layer/sample ID, the second on the thickness (in cm) of each layer, whilst column 3 should contain NA for all layers that are not sampled for OSL/TL. For the layer the OSL/TL sample was taken from a numerical value must be provided, which is the distance (in cm) measured from bottom of the layer of interest. If the whole layer was sampled insert 0. If the sample was taken from within the layer, insert a numerical value >0, which describes the distance from the middle of the sample to the bottom of the layer in cm. Columns 4 to 9 should contain radionuclide concentrations and their standard errors for Potassium (in give information on the water content and its uncertainty (standard error) in should be left blank. Please ensure to keep the column titles as given in the example dataset (data("ExampleData.ScaleGammaDose"), see examples).

The user can decide which dose rate conversion factors should be used to calculate the gamma dose rates. The options are:
"Cresswell et al. 2019" (Cresswell et al., in press; the default)
"Liritzis et al. 2013" (Liritzis et al., 2013)
"Guerin et al. 2011" (Guerin et al., 2011)

Water content
The water content provided by the user should be calculated according to:

\[
\frac{(Wetweight[g] - Dryweight[g])}{Dryweight[g]} \times 100
\]

Calculations
After converting the radionuclide concentrations into dose rates, the function will scale the dose rates based on the thickness of the layers, the distances to the sample, the water content and the density of the sediment. The calculations are based on Aitken (1985, Appendix H). As an example (equivalent to Aitken, 1985), assuming three layers of sediment, where \( L \) is inert and positioned in between the infinite thick and equally active layers \( A \) and \( B \), the dose in \( L \) and \( B \) due to \( A \) is given by

\[
1 - f(x)D_A
\]

Where \( x \) is the distance into the inert medium, so \( f(x) \) is the weighted average fractional dose at \( x \) and \( D_A \) denotes that the dose is delivered by \( A \). \( f(x) \) is derived from table H1 (Aitken, 1985), when setting \( z = x \). Consequently, the dose in \( A \) and \( L \) due to \( B \) is given by

\[
1 - f(t-x)D_B
\]

Here \( t \) is the thickness of \( L \) and the other parameters are denoted as above, just for the dose being delivered by \( B \). \( f(t-x) \) is derived from table H1 (Aitken, 1985), when setting \( z \) equal to \( t-x \). Following this, the dose in \( L \) delivered by \( A \) and \( B \) is given by

\[
2 - f(x) - f(t-x)D_{AB}
\]

Since \( A \) and \( B \) are equally active \( D_{AB} = D_A = D_B \).

The function uses the value of the fractional dose rate at the layer boundary to start the calculation for the next layer. This way, the function is able to scale the gamma dose rate accurately for distant layers when the density and water content is not constant for the entire section.

Value
After performing the calculations the user is provided with different outputs.

1. The total gamma dose rate received by the sample (+/– uncertainties) as a print in the console.
2. A plot showing the sediment sequence, the user input sample information and the contribution to total gamma dose rate.
3. RLum Results. If the user wishes to save these results, writing a script to run the function and to save the results would look like this:
mydata <- read.table("c:/path/to/input/file.txt")
results <- scale_GammaDose(mydata)
table <- get_RLum(results)
write.csv(table, "c:/path/to/results.csv")

[ NUMERICAL OUTPUT ]

RLum.Results-object

slot: @data

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$summary</td>
<td>data.frame</td>
<td>summary of the model results</td>
</tr>
<tr>
<td>$data</td>
<td>data.frame</td>
<td>the original input data</td>
</tr>
<tr>
<td>$dose_rates</td>
<td>list</td>
<td>two data.frames for the scaled and infinite matrix dose rates</td>
</tr>
<tr>
<td>$tables</td>
<td>list</td>
<td>several data.frames containing intermediate results</td>
</tr>
<tr>
<td>$args</td>
<td>character</td>
<td>arguments of the call</td>
</tr>
<tr>
<td>$call</td>
<td>call</td>
<td>the original function call</td>
</tr>
</tbody>
</table>

slot: @info
Currently unused.

[ PLOT OUTPUT ]

Three plots are produced:

- A visualisation of the provided sediment layer structure to quickly assess whether the data was provided and interpreted correctly.
- A scatter plot of the nuclide contents per layer (K, Th, U) as well as the water content. This may help to correlate the dose rate contribution of specific layers to the layer of interest.
- A barplot visualising the contribution of each layer to the total dose rate received by the sample in the target layer.

Function version

0.1.1

Acknowledgements

We thank Dr Ian Bailiff for the provision of an excel spreadsheet, which has been very helpful when writing this function.
How to cite


Note

This function has BETA status. If possible, results should be cross-checked.

Author(s)

Svenja Riedesel, Aberystwyth University (United Kingdom)
Martin Autzen, DTU NUTECH Center for Nuclear Technologies (Denmark)
Christoph Burow, University of Cologne (Germany)
Based on an excel spreadsheet and accompanying macro written by Ian Bailiff. , RLum Developer Team

References


See Also

ExampleData.ScaleGammaDose, approx, barplot

Examples

# Load example data
data("ExampleData.ScaleGammaDose", envir = environment())
x <- ExampleData.ScaleGammaDose

# Scale gamma dose rate
results <- scale_GammaDose(data = x,
                          conversion_factors = "Cresswelletal2019",
                          fractional_gamma_dose = "Aitken1985",
                          verbose = TRUE,
                          plot = TRUE)

going RLum(results)
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, data.frame or numeric (required): RLum.Results needs to be originated 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)

\[
De\text{[Gy]} = De\text{[s]} \times DoseRate\text{[Gy/s]}
\]

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)

\[
se(De)\text{[Gy]} = se(De)\text{[s]} \times DR\text{[Gy/s]}
\]

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

\[
se(De)\text{[Gy]} = \sqrt{(DR\text{[Gy/s]} \times se(De)\text{[s]})^2 + (De\text{[s]} \times se(DR)\text{[Gy/s]})^2)}
\]
Applicable under the assumption that errors of De and se are uncorrelated.

(3) absolute error propagation

\[ se(De)[Gy] = abs(DR[Gy/s] \times se(De)[s]) + abs(De[s] \times se(DR)[Gy/s]) \]

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

How to cite


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)
RLum Developer Team

References


See Also

calc_SourceDoseRate

Examples

```r
##(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**

```r
set_Risoe.BINfileData(METADATA = data.frame(), DATA = list(),
                      .RESERVED = list())
```

**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.
set_RLum

Function version
0.1

How to cite

Author(s)
Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France), RLum Developer 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>RLum (required): name of the S4 class to create</td>
</tr>
<tr>
<td>originator</td>
<td>character (automatic): contains the name of the calling function (the function that produces this object); can be set manually.</td>
</tr>
<tr>
<td>.uid</td>
<td>character (automatic): sets an unique ID for this object using the internal C++ function create_UID.</td>
</tr>
<tr>
<td>.pid</td>
<td>character (with default): option to provide a parent id for nesting at will.</td>
</tr>
<tr>
<td>...</td>
<td>further arguments that one might want to pass to the specific set method</td>
</tr>
</tbody>
</table>
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`,
- `RLum.Results`

**Value**

Returns an object of the specified class.

**Function version**

0.3.0

**How to cite**


**Author(s)**

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

**See Also**


**Examples**

```r
## 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)

smooth_RLum  Smoothing of data

Description

Function calls the object-specific smooth functions for provided RLUm S4-class objects.

Usage

smooth_RLum(object, ...)

# S4 method for signature 'list'
smooth_RLum(object, ...)

Arguments

object RLUm (required): S4 object of class RLUm
...

further arguments passed to the specific class method

Details

The function provides a generalised access point for specific RLUm objects. Depending on the input object, the corresponding function will be selected. Allowed arguments can be found in the documentations of the corresponding RLUm class. The smoothing is based on an internal function called .smoothing.

Value

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

Methods (by class)

- list: Returns a list of RLUm objects that had been passed to smooth_RLum

Function version

0.1.0
How to cite


Note

Currently only RLum objects of class RLum.Data.Curve and RLum.Analysis (with curve data) are supported!

Author(s)

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

See Also

RLum.Data.Curve, RLum.Analysis

Examples

```r
## 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 smoothing
plot_RLum(curve)
plot_RLum(smooth_RLum(curve))
```
Arguments

- **n_frames**  \(\)  \text{integer (with default): n frames}
- **t_animation**  \(\)  \text{integer (with default): t animation}
- **n.tree**  \(\)  \text{integer (with default): how many trees do you want to cut?}
- **type**  \(\)  \text{integer (optional): Make a decision: 1, 2 or 3}

Details

This amazing sophisticated function validates your data seriously.

Value

Validates your data.

How to cite


Note

This function should not be taken too seriously.

Author(s)

R Luminescence Team, 2012-2046, RLum Developer Team

See Also

plot_KDE

Examples

## no example available
Description

Function calls object-specific get functions for RLum S4 class objects.

Usage

structure_RLum(object, ...)

## S4 method for signature 'list'
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.

Methods (by class)

- list: Returns a list of RLum objects that had been passed to structure_RLum

Function version

0.2.0

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team
template_DRAC

Create a DRAC input data template (v1.2)

Description

This function returns a DRAC input template (v1.2) to be used in conjunction with the use_DRAC() function.

Usage

```r
template_DRAC(nrow = 1, preset = NULL, 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).
- `preset` `character` *(optional)*: By default, all values of the template are set to NA, which means that the user needs to fill in all data first before submitting to DRAC using use_DRAC(). To reduce the number of values that need to be provided, preset can be used to create a template with at least a minimum of reasonable preset values. preset can be one of the following:
  - `quartz_coarse`
  - `quartz_fine`
  - `feldspar_coarse`
  - `polymineral_fine`
  - `DRAC-example_quartz`
  - `DRAC-example_feldspar`
  - `DRAC-example_polymineral`

Note that the last three options can be used to produce a template with values directly taken from the official DRAC input .csv file.
- `notification` `logical` *(with default)*: show or hide the notification
Template_DRAC

Value

A list.

How to cite


Author(s)

Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), RLum Developer Team

References


See Also

as.data.frame, list

Examples

# create a new DRAC input input <- template_DRAC(preset = "DRAC-example_quartz")

# 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$'Conversion factors' <- "AdamiecAitken1998"
input$'External U (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"
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 increased, ranges between 0 and inf.

Value

Returns a data.frame with tuned values.

Function version

0.5.0
use_DRAC

How to cite


Note

You should not use this function to improve your poor data set!

Author(s)

Michael Dietze, GFZ Potsdam (Germany), RLum Developer Team

Examples

```r
## 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"))
```

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

```r
use_DRAC(file, name, print_references = TRUE, citation_style = "text",
          ...)```
use_DRAC

Arguments

file character (required): spreadsheet to be passed to the DRAC website for calculation. Can also be a DRAC template object obtained from template_DRAC().

name character (with default): Optional user name submitted to DRAC. If omitted, a random name will be generated

print_references (with default): Print all references used in the input data table to the console.

citation_style (with default): If print_references = TRUE this argument determines the output style of the used references. Valid options are "Bibtex", "citation", "html", "latex" or "R". Default is "text".

Further arguments.

• url character: provide an alternative URL to DRAC
• verbose logical: show or hide console output

Value

Returns an RLum.Results object containing the following elements:

DRAC list: a named list containing the following elements in slot @data:

$hhighlights 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
$references list A list of bibentries of used references

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

0.1.3

How to cite

use_DRAC

Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, Universite Bordeaux Montaigne (France)
Michael Dietze, GFZ Potsdam (Germany)
Christoph Burow, University of Cologne (Germany), RLum Developer Team

References


Examples

## (1) Method using the DRAC spreadsheet

```r
file <- "/PATH/TO/DRAC_Input_Template.csv"

# send the actual IO template spreadsheet to DRAC
## Not run:
use_DRAC(file = file)
## End(Not run)
```

## (2) Method using an R template object

```r
# Create a template
input <- template_DRAC(preset = "DRAC-example_quartz")

# Fill the template with values
input$`Project ID` <- "DRAC-Example"
input$`Sample ID` <- "Quartz"
input$`Conversion factors` <- "AdamiecAitken1998"
input$`External U (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
```
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 identified as zero light level curves are automatically removed. Output 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 the terminal feedback
- `plot`: logical (with default): enables or disables the graphical feedback
Details

How does the method work?
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:

\[
|E(x) - Var(x)| \geq \Theta
\]

With \( \Theta \) an arbitrary, user defined, threshold. Values above the threshold indicating curves comprising a signal.

Note: the absolute difference of \( E(X) \) and \( Var(X) \) instead of the ratio was chosen as both terms can become 0 which would result in 0 or \( Inf \), if the ratio is calculated.

Value

The function returns

```
[ NUMERICAL OUTPUT ]
```

RLum.Results-object

slot:****@data

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$unique_pairs</td>
<td>data.frame</td>
<td>the unique position and grain pairs</td>
</tr>
<tr>
<td>$selection_id</td>
<td>numeric</td>
<td>the selection as record ID</td>
</tr>
<tr>
<td>$selection_full</td>
<td>data.frame</td>
<td>implemented models used in the baSAR-model core</td>
</tr>
</tbody>
</table>

slot:****@info

The original function call

Output variation

For \( \text{cleanup} = \text{TRUE} \) the same object as the input is returned, but cleaned up (invalid curves were removed). This means: Either an \texttt{Riso.e.BINfileData} or an \texttt{RLum.Analysis} object is returned in such cases. An \texttt{Riso.e.BINfileData} object can be exported to a BIN-file by using the function \texttt{write_R2BIN}.

Function version

0.2.1
How to cite


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, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

See Also

Risoe.BINfileData, RLum.Analysis, write_R2BIN, read_BIN2R

Examples

```r
##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

write_R2BIN(object, paste0(dirname(file),"/", basename(file), "_CLEANED.BIN")))

## End(Not run)

---

**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**

```r
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. It further limits the maximum number of points per curve to 9,999. If a curve contains more data the curve data got binned using the smallest possible bin width.

- **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, DTYPEx and LIGHTSOURCE are not of type *character*, no transformation into numeric values is done.

**Value**

Write a binary file.
Function version

0.5.1

How to cite


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 05 to 04 or 03).

Warning

Although the coding was done carefully it seems that the BIN/BINX-files produced by Risø 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, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

References


See Also

read_BIN2R, Risoe.BINfileData, writeBin

Examples

```r
## Not run:
# load exampled dataset
data(ExampleData.BINfileData, envir = environment())

# create temporary filepath
# (for usage replace by own path)
```
write_RLum2CSV

```r
temp_file <- tempfile(pattern = "output", fileext = ".bin")

##export to temporary file path
write_R2BIN(CWOSL.SAR.Data, file = temp_file)

## End(Not run)
```

---

write_RLum2CSV  Export RLum-objects to CSV

**Description**

This function exports RLum-objects to CSV-files using the R function `utils::write.table`. All RLum-objects are supported, but the export is lossy, i.e. the pure numerical values are exported only. Information that cannot be coerced to a `data.frame` or a `matrix` are discarded as well as metadata.

**Usage**

```r
write_RLum2CSV(object, path = NULL, prefix = "", export = TRUE,
compact = TRUE, ...)
```

**Arguments**

- `object`  
  RLum or a list of RLum objects (**required**): objects to be written

- `path`  
  character (**optional**): character string naming folder for the output to be written. If nothing is provided path will be set to the working directory. **Note:** this argument is ignored if the argument `export` is set to `FALSE`.

- `prefix`  
  character (**with default**): optional prefix to name the files. This prefix is valid for all written files

- `export`  
  logical (**with default**): enable or disable the file export. If set to `FALSE` nothing is written to the file connection, but a list comprising objects of type `data.frame` and `matrix` is returned instead

- `compact`  
  logical (**with default**): if `TRUE` (the default) the output will be more simple but less comprehensive, means not all elements in the objects will be fully broken down. This is in particular useful for writing RLum.Results objects to CSV-files, such objects can be rather complex and not all information are needed in a CSV-file or can be meaningful translated to it.

- `...`  
  further arguments that will be passed to the function `utils::write.table`. All arguments except the argument `file` are supported
Details

However, in combination with the implemented import functions, nearly every supported import data format can be exported to CSV-files, this gives a great deal of freedom in terms of compatibility with other tools.

Input is a list of objects

If the input is a list of objects all explicit function arguments can be provided as list.

Value

The function returns either a CSV-file (or many of them) or for the option export == FALSE a list comprising objects of type data.frame and matrix

Function version

0.2.0

How to cite


Author(s)

Sebastian Kreutzer, IRAMAT-CRP2A, UMR 5060, CNRS - Université Bordeaux Montaigne (France), RLum Developer Team

See Also

RLum.Analysis, RLum.Data, RLum.Results, utils::write.table

Examples

```r
##transform values to a list (and do not write)
data(ExampleData.BINfileData, envir = environment())
object <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data)[[1]]
write_RLum2CSV(object, export = FALSE)

## Not run:
##create temporary filepath
##(for usage replace by own path)
temp_file <- tempfile(pattern = "output", fileext = ".csv")

##write CSV-file to working directory
write_RLum2CSV(temp_file)
```
write_RLum2CSV

## End(Not run)
## Index

### Topic IO
- convert_Activity2Concentration, 123
- convert_BIN2CSV, 125
- convert_Daybreak2CSV, 126
- convert_PSL2CSV, 127
- convert_RLum2Risoedata.BINfileData, 128
- convert_Wavelength2Energy, 130
- extract_IrradiationTimes, 164
- merge_Risoedata.BINfileData, 194
- PSL2Risoedata.BINfileData, 255
- read_BIN2R, 257
- read_Daybreak2R, 259
- read_PSL2R, 261
- read_SPE2R, 262
- read_XSYG2R, 265
- write_R2BIN, 299
- write_RLum2CSV, 301

### Topic aplot
- plot_FilterCombinations, 216
- plot_RLum.Analysis, 242
- plot_RLum.Data.Curve, 244
- plot_RLum.Data.Image, 246
- plot_RLum.Data.Spectrum, 248
- plot_RLum.Results, 252

### Topic classes
- RLum-class, 274

### Topic datagen
- analyse_Al2O3C_CrossTalk, 8
- analyse_Al2O3C_ITC, 10
- analyse_Al2O3C_Measurement, 12
- analyse_baSAR, 15
- analyse_FadingMeasurement, 23
- analyse_IRSAR.RF, 26
- analyse_pIRIRSequence, 33
- analyse_portableOSL, 36
- analyse_SAR.CWOSL, 38
- Analyse_SAR.OSLdata, 42
- analyse_SAR.TL, 45
- calc_AverageDose, 61
- calc_FadingCorr, 73
- calc_gSGC, 84
- calc_Huntley2006, 88
- calc_Kars2008, 95
- calc_Lamothe2003, 97
- calc_OSLLxTxRatio, 108
- calc_Statistics, 114
- calc_ThermalLifetime, 116
- calc_TLLxTxRatio, 119
- fit_SurfaceExposure, 178
- plot_FilterCombinations, 216
- scale_GammaDose, 276
- verify_SingleGrainData, 296

### Topic datasets
- BaseDataSet.ConversionFactors, 53
- BaseDataSet.CosmicDoseRate, 54
- BaseDataSet.FractionalGammaDose, 56
- ExampleData.Al2O3C, 145
- ExampleData.BINfileData, 146
- ExampleData.CW_OSL_Curve, 148
- ExampleData.DeValues, 149
- ExampleData.Fading, 150
- ExampleData.portableOSL, 154
- ExampleData.RLum.Analysis, 154
- ExampleData.RLum.Data, 155
- ExampleData.ScaleGammaDose, 156
- ExampleData.SurfaceExposure, 157
- ExampleData.TR_OSL, 160
- ExampleData.XSYG, 161
- extdata, 163

### Topic dplot
- Analyse_SAR.OSLdata, 42
- calc_FuchsLang2001, 82
- fit_CWCurve, 167
- fit_LMCurve, 170
calc_OSSLxTxRatio, 17–20, 22, 26, 35, 39, 41, 43, 44, 108
calc_SourceDoseRate, 112, 281, 282
calc_Statistics, 114, 201, 213, 228, 255
calc_TerahmalLifetime, 116
calc_TLLxTxRatio, 46, 47, 119
calc_WodaFuchs2008, 121
call, 34, 60, 65, 67, 70, 80, 83, 85, 86, 94, 105, 173, 218, 294
coda::mcn.clist, 20
c conflnt, 167–169, 171, 173
c contour, 250
c convert_Activity2Concentration, 123
c convert_BIN2CSV, 125
c convert_Daybreak2CSV, 126
c convert_PSL2CSV, 127
c convert_RLum2Risoe.BINfileData, 128
c convert_Wavelength2Energy, 130, 249, 250
c convert_XSYG2CSV, 132
 CW2pHM1, 134, 139, 141, 144, 240
 CW2pLM, 136, 137, 141, 144, 171, 240
 CW2pLM1, 136, 139, 144, 240
 CW2pPMi, 136, 139, 141, 142, 240
 CW_Curve.BosWallinga2012 (ExampleData.CW_OSL_Curve), 148
 CWOSL.SAR.Data (ExampleData.BINfileData), 146
data.table::data.table, 260
data_CrossTalk(ExampleData.Al2O3C), 145
data_ITC(ExampleData.Al2O3C), 145
 Date, 112
density, 228, 229
DEoptim::DEoptim, 178
DEoptim::DEoptim.control, 176
devtools::remote-reexports, 193
ExampleData.Al2O3C, 145
ExampleData.BINfileData, 146
ExampleData.CW_OSL_Curve, 148
ExampleData.DevValues, 149
ExampleData.Fading, 150
ExampleData.FittingLM, 152
ExampleData.LxTxData, 152
ExampleData.LxTxOSLData, 153
ExampleData.portableOSL, 154
ExampleData.RLum.Analysis, 154
ExampleData.RLum.Data.Image, 155
ExampleData.ScaleGammaDose, 156, 280
ExampleData.SurfaceExposure, 157, 181
ExampleData.TR_OSL, 160
ExampleData.XSYG, 161
expression, 222
extdata, 163
extract_IrradiationTimes, 26, 164
fit_CWCurve, 77, 78, 167, 174
fit_LMCurve, 136, 139, 141, 144, 170, 170
fit_OSLLifeTimes, 160, 174
fit_SurfaceExposure, 157, 178
fit_TermalQuenching, 182
formula, 40
get_Layout, 185
g et Quote, 186
get_rightAnswer, 187
get_Risoe.BINfileData, 188
get_RLum, 8, 10, 13, 31, 32, 35, 40, 41, 46, 47, 60, 65, 67, 70, 75, 78, 81, 86, 94, 105, 111, 113, 114, 118, 164, 170, 174, 189, 190, 223, 242, 263, 294
get_RLum,list-method(get_RLum), 189
get_RLum,NULL-method(get_RLum), 189
GitHub-API, 190
github_branches(GitHub-API), 190
g ithub_commits(GitHub-API), 190
g ithub_issues(GitHub-API), 190
INDEX

glm, 172
graphics::boxplot, 254
graphics::grid, 218
graphics::hist, 62, 64
graphics::legend, 27, 218
graphics::matplot, 118

hist, 225, 226
install.DevelopmentVersion, 192
integer, 17, 19, 24, 28, 33, 38, 39, 45, 48, 62, 73, 85, 109, 112, 115, 120, 130, 171, 191, 195, 202, 208, 211, 220, 231, 242, 249, 258, 268, 275, 288, 290
IRSAR.RF.Data
  (ExampleData.RLum.Analysis), 154

legend, 231, 254
length_RLum, 193
list.files, 258, 259
lm, 134, 136, 139, 140, 143, 221–223
Luminescence (Luminescence-package), 6
Luminescence-package, 6
Luminescence::github_branches, 193
Luminescence::plot_GrowthCurve, 89
Lx.data (ExampleData.LxTxOSLData), 153
LxTxData (ExampleData.LxTxData), 152

matrix, 52, 80, 118, 125, 126, 128, 130, 133, 169, 175, 216, 231, 248, 254, 301, 302
merge_Risoe.BINfileData, 194, 259
merge_RLum, 196
methods::as, 53
methods::language-class, 31
methods_RLum, 276
minpack.lm::nls.lm, 176, 178
minpack.lm::nlsLM, 30, 32, 89, 91, 170, 174, 176, 179–181, 183, 184, 221, 223
mle2, 105
mtext, 225

names_RLum, 199, 199
names_RLum, list-method (names_RLum), 199
nlminb, 104
nls, 28, 30, 32, 167–174, 221–223

OSL.SARMeasurement (ExampleData.XSYG), 161

pander::openFileInOS, 271
pander::pander_return, 271
par, 231
pchisq, 87
pdf, 243, 252
persp, 249, 250
plot.default, 46, 117, 175, 209, 254
plot_AbanicoPlot, 115, 200
plot_DetPlot, 207
plot_DRCSummary, 210
plot_DRTResults, 212
plot_FilterCombinations, 216
plot_GrowthCurve, 11, 12, 17–20, 22, 34, 35, 39, 41, 44, 46, 47, 89, 90, 97–99, 111, 219
plot_histogram, 204, 224, 236  
plot_KDE, 204, 227, 236, 288  
plot_NRT, 230  
plot_RadialPlot, 204, 233  
plot_Risoe.BINfileData, 238  
plot_RLum, 113, 114, 131, 162, 240, 244, 245, 247, 250, 253  
plot_RLum.Analysis, 162, 241, 242  
plot_RLum.Data.Curve, 241, 244, 244  
plot_RLum.Data.Image, 241, 246  
plot_RLum.Data.Spectrum, 162, 241, 248  
plot_RLum.Results, 241, 252  
plot_ViolinPlot, 253  
plotly::plot_ly, 250  
profile, 169  
profile.mle2, 105  
PSL2Risoe.BINfileData, 255  
raster, 246, 247  
raster::contour, 247  
raster::plot, 246  
raster::plotRGB, 247  
raster::raster, 246, 247, 264  
raw, 257  
read.table, 64  
read_BIN2R, 18, 20, 22, 26, 42, 44, 125, 165, 166, 196, 238, 240, 257, 274, 298, 300  
read_Daybreak2R, 126, 127, 259  
read_PSL2R, 37, 127, 128, 256, 261  
read_SPEER, 155, 247, 262  
read_XSYG2R, 26, 133, 146, 160–162, 164–166, 265  
readBin, 264  
readxl::read_excel, 18, 20, 22  
regex, 265  
replicate_RLum, 268  
replicate_RLum,RLum-method (RLum-class), 274  
report_RLum, 269  
Risoe.BINfileData2RLum.Analysis, 258, 272  
rjags::coda.samples, 22  
rjags::jags.model, 19, 22  
RLum, 52, 189, 190, 193, 194, 196, 197, 199, 240, 241, 268, 269, 275, 284–286, 289, 301  
RLum-class, 274  
RLum.Data, 57, 125, 127, 128, 133, 275, 276, 302  
RLumShiny::app_RLum, 51  
RLumShiny::RLumShiny-package, 51  
rmarkdown::render, 270, 271  
rollmean, 231  
rstudio::viewer, 271  
rug, 255  
scale_GammaDose, 276  
sd, 222  
Second2Gray, 112, 114, 281  
set.seed, 73, 74  
set_Risoe.BINfileData, 283  
set_RLum, 275, 284
shiny::runApp, 51
smooth, 48, 49
smooth.spline, 48, 49, 231
smooth_RLum, 286, 286
smooth_RLum, list-method (smooth_RLum), 286
stats::approx, 50
stats::density, 254, 255
stats::FDist, 176
stats::lm, 24, 25
stats::nls, 24, 183
stats::rnorm, 118
stats::uniroot, 73
sTeve, 287
structure_RLum, 289, 289
structure_RLum, list-method
 (structure_RLum), 289
summary, 168, 169, 173

template_DRAC, 290
TL.SAR.Data (ExampleData.BINfileData), 146
TL.Spectrum (ExampleData.XSYG), 161
tune_Data, 292
Tx.data (ExampleData.LxTxOSLData), 153
txtProgressBar, 73, 258, 260, 263, 273, 299

uniroot, 74, 75, 85, 86, 221, 223
use_DRAC, 123, 293
utils::txtProgressBar, 259
utils::write.table, 125, 127, 128, 133, 301, 302

values.cosmic.Softcomp
 (BaseDataSet.CosmicDoseRate), 54
values.curve (ExampleData.FittingLM), 152
values.curveBG (ExampleData.FittingLM), 152
values.factor.Altitude
 (BaseDataSet.CosmicDoseRate), 54
values.par.FJH
 (BaseDataSet.CosmicDoseRate), 54
vector, 16, 17, 24, 27, 34, 37, 42, 43, 45, 73, 109, 134, 140, 143, 167, 183, 238, 248, 258, 263, 273

verify_SingleGrainData, 18–20, 22, 296

write_R2BIN, 128, 129, 164–166, 196, 256, 259, 297, 298, 299
write_RLum2CSV, 125–128, 133, 301
writeBin, 300

xml, 265, 267

zoo::rollmean, 245