Package ‘Luminescence’

August 19, 2024

Type Package

Title Comprehensive Luminescence Dating Data Analysis

Version 0.9.25.9000-43

Date 2024-08-19

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] (<https://orcid.org/0000-0001-7210-1132>),
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 [aut] (<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 [aut] (<https://orcid.org/0000-0002-5331-5087>),
Guillaume Guerin [ctb] (<https://orcid.org/0000-0001-6298-5579>),
Svenja Riedesel [aut] (<https://orcid.org/0000-0003-2936-8776>),
Martin Autzen [aut] (<https://orcid.org/0000-0001-6249-426X>),
Pierre Guibert [ctb] (<https://orcid.org/0000-0001-8969-8684>),
Dirk Mittelstrass [aut] (<https://orcid.org/0000-0002-9567-8791>),
Harrison J. Gray [aut] (<https://orcid.org/0000-0002-4555-7473>),
Jean-Michel Galharret [aut] (<https://orcid.org/0000-0003-2219-8727>),
Marco Colombo [aut] (<https://orcid.org/0000-0001-6672-0623>),
Markus Fuchs [ths] (<https://orcid.org/0000-0003-4669-6528>)

Maintainer Sebastian Kreutzer <maintainer_luminescence@r-luminescence.org>

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 (>= 4.3),
utils

LinkingTo Rcpp (>= 1.0.12),
RcppArmadillo (>= 0.12.8.4.0)

Imports bbmle (>= 1.0.25.1),
data.table (>= 1.15.4),
DEoptim (>= 2.2-8),
httr (>= 1.4.7),
interp (>= 1.1-6),
lamW (>= 2.2.3),
matrixStats (>= 1.3.0),
methods,
minpack.lm (>= 1.2-4),
mclust (>= 6.1),
readxl (>= 1.4.3),
Rcpp (>= 1.0.12),
shape (>= 1.4.6),
parallel,
XML (>= 3.99-0.16),
zoo (>= 1.8-12)

Suggests spelling (>= 2.3.0),
plotly (>= 4.10.4),
rmarkdown (>= 2.27),
rstudioapi (>= 0.16.0),
rjags (>= 4-15),
coda (>= 0.19-4),
pander (>= 0.6.5),
testthat (>= 3.2.1),
tiff (>= 0.1-12),
devtools (>= 2.4.5),
R.rsp (>= 0.46.0)

VignetteBuilder R.rsp

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

Encoding UTF-8

Language en-US

Collate 'Analyse_SAR.OSLdata.R'
'CW2pHMi.R'
'CW2pLM.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.Curve-class.R'
'RLum.Data.Image-class.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'
'Risoe.BINfileData2RLum.Data.Curve.R'
'Second2Gray.R'
'addins_RLum.R'
'analyse_Al2O3C_CrossTalk.R'
'analyse_Al2O3C_ITC.R'
'analyse_Al2O3C_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'
'apply_CosmicRayRemoval.R'
'apply_EfficiencyCorrection.R'
'calc_AliquotSize.R'
'calc_AverageDose.R'
'calc_CentralDose.R'
'calc_CobbleDoseRate.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_OSLLxTxDecomposed.R'
'calc_OSLLxTxRatio.R'
'calc_SourceDoseRate.R'
'calc_Statistics.R'
'calc_TLLxTxRatio.R'
'calc_ThermalLifetime.R'
'calc_WodaFuchs2008.R'
'calc_gSGC.R'
'calc_gSGC_feldspar.R'
'combine_De_Dr.R'

'convert_Activity2Concentration.R'
'convert_BIN2CSV.R'
'convert_Concentration2DoseRate.R'
'convert_Daybreak2CSV.R'
'convert_PSL2CSV.R'
'convert_RLum2Risoe.BINfileData.R'
'convert_SG2MG.R'
'convert_Wavelength2Energy.R'
'convert_XSYG2CSV.R'
'extract_IrradiationTimes.R'
'extract_ROI.R'
'fit_CWCurve.R'
'fit_EmissionSpectra.R'
'fit_LMCurve.R'
'fit_OSLLifeTimes.R'
'fit_SurfaceExposure.R'
'fit_ThermalQuenching.R'
'get_Layout.R'
'get_Quote.R'
'get_rightAnswer.R'
'github.R'
'import_Data.R'
'install_DevelopmentVersion.R'
'internal_as.latex.table.R'
'internals_RLum.R'
'internals_Thermochronometry.R'
'merge_RLum.Analysis.R'
'merge_RLum.Data.Curve.R'
'merge_RLum.R'
'merge_RLum.Results.R'
'merge_Risoe.BINfileData.R'
'methods_DRAC.R'
'methods_RLum.R'
'plot_AbanicoPlot.R'
'plot_DRCSummary.R'
'plot_DRTResults.R'
'plot_DetPlot.R'
'plot_FilterCombinations.R'
'plot_GrowthCurve.R'
'plot_Histogram.R'
'plot_KDE.R'
'plot_NRt.R'
'plot_OSLAgeSummary.R'
'plot_RLum.Analysis.R'
'plot_RLum.Data.Curve.R'
'plot_RLum.Data.Image.R'
'plot_RLum.Data.Spectrum.R'
'plot_RLum.R'
'plot_RLum.Results.R'
'plot_ROI.R'
'plot_RadialPlot.R'
'plot_Risoe.BINfileData.R'

'plot_ViolinPlot.R'
'read_BIN2R.R'
'read_Daybreak2R.R'
'read_HeliosOSL2R.R'
'read_PSL2R.R'
'read_RF2R.R'
'read_SPE2R.R'
'read_TIFF2R.R'
'read_XSYG2R.R'
'report_RLum.R'
'scale_GammaDose.R'
'subset_SingleGrainData.R'
'template_DRAC.R'
'trim_RLum.Data.R'
'tune_Data.R'
'use_DRAC.R'
'utils_DRAC.R'
'verify_SingleGrainData.R'
'write_R2BIN.R'
'write_R2TIFF.R'
'write_RLum2CSV.R'
'zzz.R'

RoxygenNote 7.3.2

Contents

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

Description

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

Details

Supervisor of the initial version in 2012

Markus Fuchs, Justus-Liebig-University Giessen, Germany

Support contact

- <developers@r-luminescence.org>
- https://github.com/R-Lum/Luminescence/discussions

Bug reporting

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

Project website

- https://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
- https://cran.r-project.org/package=RLumCarlo
- https://cran.r-project.org/package=RCarb

Funding

- 2011-2013: The initial version of the package was developed, while Sebastian Kreutzer was funded through the DFG programme "Rekonstruktion der Umweltbedingungen des Spätpleistozäns in Mittelsachsen anhand von Löss-Paläobodensequenzen" (DFG id: 46526743)
- 2014-2018: 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)
- 05/2014-12/2019: The work of Sebastian Kreutzer as maintainer of the package was supported by LabEx LaScArBx (ANR - n. ANR-10-LABX-52).
- 01/2020-04/2022: Sebastian Kreutzer as maintainer of the package has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 844457 (CREDit), and could continue maintaining the package.
- since 03/2023: Sebastian Kreutzer as maintainer of the package receives funding from the DFG Heisenberg programme No 505822867.
- All other authors gratefully received additional funding from various public funding bodies.

Author(s)

Maintainer: Sebastian Kreutzer <maintainer_luminescence@r-luminescence.org> (ORCID) [translator, data contributor]

Authors:

- Christoph Burow (ORCID) [translator, data contributor]
- Michael Dietze (ORCID)
- Margret C. Fuchs (ORCID)
- Christoph Schmidt (ORCID)
- Manfred Fischer [translator]
- Johannes Friedrich (ORCID)
- Norbert Mercier (ORCID)
- Anne Philippe (ORCID)
- Svenja Riedesel (ORCID)
- Martin Autzen (ORCID)
- Dirk Mittelstrass (ORCID)
- Harrison J. Gray (ORCID)
- Jean-Michel Galharret (ORCID)
- Marco Colombo (ORCID)

Other contributors:

- Rachel K. Smedley (ORCID) [contributor]

- Claire Christophe [contributor]
- Antoine Zink (ORCID) [contributor]
- Julie Durcan (ORCID) [contributor]
- Georgina E. King (ORCID) [contributor, data contributor]
- Guillaume Guerin (ORCID) [contributor]
- Pierre Guibert (ORCID) [contributor]
- Markus Fuchs (ORCID) [thesis advisor]

References

Dietze, M., Kreutzer, S., Fuchs, M.C., Burow, C., Fischer, M., Schmidt, C., 2013. A practical guide to the R package Luminescence. Ancient TL, 31 (1), 11-18.

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

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

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

Mercier, N., Kreutzer, S., Christophe, C., Guérin, G., Guibert, P., Lahaye, C., Lanos, P., Philippe, A., Tribolo, C., 2016. Bayesian statistics in luminescence dating: The 'baSAR'-model and its implementation in the R package 'Luminescence'. Ancient TL 34 (2), 14-21.

Mercier, N., Galharret, J.-M., Tribolo, C., Kreutzer, S., Philippe, A., 2022. Luminescence age calculation through Bayesian convolution of equivalent dose and dose-rate distributions: the De_Dr model. Geochronology 4, 297–310. https://doi.org/10.5194/gchron-4-297-2022

Smedley, R.K., 2015. A new R function for the Internal External Uncertainty (IEU) model. Ancient TL, 33 (1), 16-21.

King, E.G., Burow, C., Roberts, H., Pearce, N.J.G., 2018. Age determination using feldspar: evaluating fading-correction model performance. Radiation Measurements 119, 58-73. https://doi.org/10.1016/j.radmeas.2018

See Also

Useful links:

- `https://CRAN.R-project.org/package=Luminescence`
- Report bugs at `https://github.com/R-Lum/Luminescence/issues`

`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

```
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

Value

Function returns results numerically and graphically:

`[ NUMERICAL OUTPUT ]`

`RLum.Results`-object

slot: `@data`

Element	**Type**	**Description**
`$data`	`data.frame`	summed apparent dose table
`$data_full`	`data.frame`	full apparent dose table
`$fit`	`lm`	the linear model obtained from fitting
`$col.seq`	`numeric`	the used colour vector

slot: `@info`

The original function call

```
[ PLOT OUTPUT ]
```

- An overview of the obtained apparent dose values

Function version

0.1.3

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Kreutzer, S., Martin, L., Guérin, G., Tribolo, C., Selva, P., Mercier, N., 2018. Environmental Dose Rate Determination Using a Passive Dosimeter: Techniques and Workflow for alpha-Al2O3:C Chips. Geochronometria 45, 56-67. doi: 10.1515/geochr-2015-0086

See Also

analyse_Al2O3C_ITC

Examples

```
##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

```
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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Kreutzer, S., Martin, L., Guérin, G., Tribolo, C., Selva, P., Mercier, N., 2018. Environmental Dose Rate Determination Using a Passive Dosimeter: Techniques and Workflow for alpha-Al2O3:C Chips. Geochronometria 45, 56-67. doi: 10.1515/geochr-2015-0086

See Also

plot_GrowthCurve

Examples

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

##run analysis
analyse_Al2O3C_ITC(data_ITC)
```

`analyse_Al2O3C_Measurement` *Al2O3:C Passive Dosimeter Measurement Analysis*

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

```
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. I 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 a 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
`...`	further arguments that can be passed to the plot output, supported are `norm`, `main`, `mtext`, `title` (for self-call mode to specify, e.g., sample names)

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 fall back 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.

Calculating the palaeodose is possible without **any TL** curve in the sequence!

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 delivered 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

0.2.6

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Kreutzer, S., Martin, L., Guérin, G., Tribolo, C., Selva, P., Mercier, N., 2018. Environmental Dose Rate Determination Using a Passive Dosimeter: Techniques and Workflow for alpha-Al2O3:C Chips. Geochronometria 45, 56-67.

See Also

analyse_Al2O3C_ITC

Examples

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

##run analysis
analyse_Al2O3C_Measurement(data_CrossTalk)
```

---

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

Description

This function allows the application of Bayesian models on luminescence data, measured with the single-aliquot regenerative-dose (SAR, Murray and Wintle, 2000) protocol. In particular, it follows the idea proposed by Combès 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 adoption of this approach for the R environment and (II) an extension and a technical refinement of the published code.

Usage

```
analyse_baSAR(
  object,
  XLS_file = NULL,
  aliquot_range = NULL,
  source_doserate = NULL,
  signal.integral,
  signal.integral.Tx = NULL,
  background.integral,
  background.integral.Tx = NULL,
  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. I f 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 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:

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

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

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

Input / output scenarios

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

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

Finally it does not matter how the information of the BIN/BINX file are provided. The function supports **(a)** either a path to a file or directory or a `list` of file names or paths or **(b)** a Risoe.BINfileData object or a list of these objects. The latter one can be produced by using the function read_BIN2R, but this function is called automatically if only a file name 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:

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

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):

```
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.

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

Value

Function returns results numerically and graphically:

`[ NUMERICAL OUTPUT ]`

`RLum.Results`-object

slot: `@data`

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

slot: `@info`

The original function call

`[ PLOT OUTPUT ]`

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

- (E) the final plot is the De distribution as calculated using the conventional (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

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, Université Bordeaux Montaigne (France)
Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
The underlying Bayesian model based on a contribution by Combès et al., 2015.

References

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

Mercier, N., Kreutzer, S., Christophe, C., Guerin, G., Guibert, P., Lahaye, C., Lanos, P., Philippe, A., Tribolo, C., 2016. Bayesian statistics in luminescence dating: The 'baSAR'-model and its implementation in the R package 'Luminescence'. Ancient TL 34, 14-21.

Further reading

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

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

Plummer, M., 2017. JAGS Version 4.3.0 user manual. `https://sourceforge.net/projects/mcmc-jags/files/Manu`

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",
  plot = FALSE,
  n.MCMC = 200

)

print(results)


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

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


## End(Not run)
```

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

Description

The function analysis 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

```
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 channels for the signal integral (e.g., `c(1:10)`). Not required if a `data.frame` with `LxTx` values is provided.

`background.integral` — vector (**required**): vector with channels for the background integral (e.g., `c(90:100)`). Not required if a `data.frame` with `LxTx` values is provided.

`t_star` — character (*with default*): method for calculating the time elapsed since irradiation if input is **not** a `data.frame`. Options are: `'half'` (the default), `'half_complex`, which uses the long equation in Auclair et al. 2003, and and `'end'`, which takes the time between irradiation and the measurement step. Alternatively, `t_star` can be a function with one parameter which works on `t1`. For more information see details.

`t_star` *has no effect if the input is a data.frame, because this input comes without irradiation times.*

`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. Supported arguments: `xlab`, `log`, `mtext`, `plot.trend` (enable/disable trend blue line), and `xlim` for the two first curve plots, and `ylim` for the fading curve plot. For further plot customization please use the numerical output of the functions for own plots.

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.

The options for `t_star`

- `t_star = "half"` (the default) The calculation follows the simplified version in Auclair et al. (2003), which reads
$$t_{star} := t_1 + (t_2 - t_1)/2$$
- `t_star = "half_complex"` This option applies the complex function shown in Auclair et al. (2003), which is derived from Aitken (1985) appendix F, equations 9 and 11. It reads
$$t_{star} = t0 * 10^[(t_2 log(t_2/t_0) - t_1 log(t_1/t_0) - 0.43(t_2 - t_1))/(t_2 - t_1)]$$
where 0.43 = $1/ln(10)$. t0, which is an arbitrary constant, is set to 1. Please note that the equation in Auclair et al. (2003) is incorrect insofar that it reads $10exp(...)$, where the base should be 10 and not the Euler's number. Here we use the correct version (base 10).
- `t_star = "end"` This option uses the simplest possible form for `t_star` which is the time since irradiation without taking into account any addition parameter and it equals t1 in Auclair et al. (2003)
- `t_star = <function>` This last option allows you to provide an R function object that works on t1 and gives you all possible freedom. For instance, you may want to define the following function `fun <- function(x) {x^2}`, this would square all values of t1, because internally it calls `fun(t1)`. The name of the function does not matter.

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 the `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.

Shine-down curve plots Please note that the shine-down curve plots are for information only. As such not all pause steps are plotted to avoid graphically overloaded plots. However, *all* pause times are taken into consideration for the analysis.

Value

An RLum.Results object is returned:

Slot: **@data**

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

Slot: **@info**

OBJECT	`TYPE`	`COMMENT`
`call`	`call`	the original function call

Function version

0.1.22

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Christoph Burow, University of Cologne (Germany)

References

Aitken, M.J., 1985. Thermoluminescence dating, Studies in archaeological science. Academic Press, London, Orlando.

Auclair, M., Lamothe, M., Huot, S., 2003. Measurement of anomalous fading for feldspar IRSL using SAR. Radiation Measurements 37, 487-492. doi:10.1016/S13504487(03)000180

Huntley, D.J., Lamothe, M., 2001. Ubiquity of anomalous fading in K-feldspars and the measurement and correction for it in optical dating. Canadian Journal of Earth Sciences 38, 1093-1106. doi: `10.1139/cjes-38-7-1093`

Kars, R.H., Wallinga, J., Cohen, K.M., 2008. A new approach towards anomalous fading correction for feldspar IRSL dating-tests on samples in field saturation. Radiation Measurements 43, 786-790. doi:10.1016/j.radmeas.2008.01.021

See Also

calc_OSLLxTxRatio, read_BIN2R, read_XSYG2R, extract_IrradiationTimes, calc_FadingCorr

Examples

```
## 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

```
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*): select method applied for the data analysis. Possible options are `"FIT"`, `"SLIDE"`, `"VSLIDE"`.

`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` (s. Details)

  (see Details for further information)

`n.MC`
: numeric (*with default*): set number of Monte Carlo runs for start parameter estimation (`method = "FIT"`) or error estimation (`method = "SLIDE"`). 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`

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

Details

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

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

1. Measuring IR-RF intensity of the natural dose for a few seconds (RF_{nat})
2. Bleach the samples under solar conditions for at least 30 min without changing the geometry
3. Waiting for at least one hour
4. Regeneration of the IR-RF signal to at least the natural level (measuring (RF_{reg})

5. Fitting data with a stretched exponential function
6. Calculate the the palaeodose D_e using the parameters from the fitting

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

`method = "FIT"`

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

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

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

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

To obtain the palaeodose D_e the function is changed to:

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

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

`method = "SLIDE"`

For this method, the natural curve is slid 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 = "VSLIDE"`

Same as `"SLIDE"` but searching also vertically for the best match (i.e. in xy-direction.) See Kreutzer et al. (2017) and Murari et al. (2021). By default the vertical sliding range will is set to `"auto"` (see `method.control`). This setting can be still changed with `method.control`.

`method.control`

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

ARGUMENT	**METHOD**	**DESCRIPTION**
`trace`	`FIT`, `SLIDE` or `VSLIDE`	as in nls; shows sum of squared residuals
`trace_vslide`	`SLIDE` or `VSLIDE`	logical argument to enable or disable the tracing of the vertical sliding
`maxiter`	`FIT`	as in nls
`warnOnly`	`FIT`	as in nls
`minFactor`	`FIT`	as in nls
`correct_onset`	`SLIDE` or `VSLIDE`	The logical argument shifts the curves along the x-axis by the first channel, as
`show_density`	`SLIDE` or `VSLIDE`	logical (*with default*) enables or disables KDE plots for MC run results. If the
`show_fit`	`SLIDE` or `VSLIDE`	logical (*with default*) enables or disables the plot of the fitted curve routinely o
`n.MC`	`SLIDE` or `VSLIDE`	integer (*with default*): This controls the number of MC runs within the sliding
`vslide_range`	`SLIDE` or `VSLIDE`	logical or numeric or character (*with default*): This argument sets the boundar
`cores`	`SLIDE` or `VSLIDE`	`number` or `character` (*with default*): set number of cores to be allocated for a

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

Test parameters

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

curves_ratio numeric (default: 1.001):

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

intersection_ratio numeric (default: NA):

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

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

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

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

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

dynamic_ratio numeric (default: NA):

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

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

The stretched exponential function suggested by Erfurt et al. (2003) describing the decay of the RF signal, comprises several parameters that might be useful to evaluate the shape of the curves. For method = "FIT" this parameter is obtained during the fitting, for method = "SLIDE" a rather rough estimation is made using the function 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]`

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

`[.. $De.MC : numeric]`

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

Column	Type	Description
`POSITION`	`numeric`	aliquot position
`PARAMETER`	`character`	test parameter name
`THRESHOLD`	`numeric`	set test parameter threshold value
`VALUE`	`numeric`	the calculated test parameter value (to be compared with the threshold)
`STATUS`	`character`	test parameter status either `"OK"` or `"FAILED"`
`SEQUENCE_NAME`	`character`	name of the sequence, so far available
`UID`	`character`	unique data set ID

`[.. $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:

Element	Type	Description
`De`	`numeric`	the final De obtained with the sliding approach
`De.MC`	`numeric`	all De values obtained by the MC runs
`residuals`	`numeric`	the obtained residuals for each channel of the curve
`trend.fit`	`lm`	fitting results produced by the fitting of the residuals
`RF_nat.slid`	`matrix`	the slid `RF_nat` curve
`t_n.id`	`numeric`	the index of the t_n offset
`I_n`	`numeric`	the vertical intensity offset if a vertical slide was applied

`algorithm_error`	`numeric`	the vertical sliding suffers from a systematic effect induced by the used algorithm. The
`vslide_range`	`numeric`	the range used for the vertical sliding
`squared_residuals`	`numeric`	the squared residuals (horizontal sliding)

slot: `@info`

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

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

---

`[ PLOT OUTPUT ]`

The slid IR-RF curves with the finally obtained De

Function version

0.7.10

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, Institute of Geography, Heidelberg University (Germany)

References

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

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

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

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

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

Frouin, M., Huot, S., Kreutzer, S., Lahaye, C., Lamothe, M., Philippe, A., Mercier, N., 2017. An improved radiofluorescence single-aliquot regenerative dose protocol for K-feldspars. Quaternary Geochronology 38, 13-24. doi:10.1016/j.quageo.2016.11.004

Kreutzer, S., Murari, M.K., Frouin, M., Fuchs, M., Mercier, N., 2017. Always remain suspicious: a case study on tracking down a technical artefact while measuring IR-RF. Ancient TL 35, 20–30.

Murari, M.K., Kreutzer, S., Fuchs, M., 2018. Further investigations on IR-RF: Dose recovery and correction. Radiation Measurements 120, 110–119. doi: 10.1016/j.radmeas.2018.04.017

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

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

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

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

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

** Further reading**

Murari, M.K., Kreutzer, S., King, G.E., Frouin, M., Tsukamoto, S., Schmidt, C., Lauer, T., Klasen, N., Richter, D., Friedrich, J., Mercier, N., Fuchs, M., 2021. Infrared radiofluorescence (IR-RF) dating: A review. Quaternary Geochronology 64, 101155. doi: 10.1016/j.quageo.2021.101155

See Also

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

Examples

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

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

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

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

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

## End(Not run)
```

`analyse_pIRIRSequence` *Analyse post-IR IRSL 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`, `IRSL`, 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:

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

The output should be accessed using the function get_RLum.

Function version

0.2.4

Note

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

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Murray, A.S., Wintle, A.G., 2000. Luminescence dating of quartz using an improved single-aliquot regenerative-dose protocol. Radiation Measurements 32, 57-73. doi:10.1016/S13504487(99)00253-X

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

See Also

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

Examples

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

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

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

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

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

  object[[sequence.structure[x]]]

})

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

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


##(3) Perform pIRIR analysis (for this example with quartz OSL data!)
## Alternative for PDF output, uncomment and complete for usage
## Not run:
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)
```

---

analyse_portableOSL *Analyse portable CW-OSL measurements*

Description

The function analyses CW-OSL curve data produced by a SUERC portable OSL reader and produces a combined plot of OSL/IRSL signal intensities, OSL/IRSL depletion ratios and the IRSL/OSL ratio.

Usage

```
analyse_portableOSL(
  object,
  signal.integral = NULL,
  invert = FALSE,
  normalise = FALSE,
  mode = "profile",
  coord = NULL,
  plot = TRUE,
  ...
)
```

Arguments

`object`	RLum.Analysis (**required**): RLum.Analysis object produced by read_PSL2R. The input can be a list of such objects, in such case each input is treated as a separate sample and the results are merged.
`signal.integral`	numeric (**required**): A vector of two values specifying the lower and upper channel used to calculate the OSL/IRSL signal. Can be provided in form of `c(1, 5)` or `1:5`.
`invert`	logical (*with default*): `TRUE` flip the plot the data in reverse order.
`normalise`	logical (*with default*): `TRUE` to normalise the OSL/IRSL signals to the *mean* of all corresponding data curves.
`mode`	character (*with default*): defines the analysis mode, allowed are `"profile"` (the default) and `"surface"` for surface interpolation. If you select something else, nothing will be plotted (similar to `plot = FALSE`).
`coord`	list matrix (*optional*): a list or matrix of the same length as number of samples measured with coordinates for the sampling positions. Coordinates are expected to be provided in meter (unit: m). Expected are x and y coordinates, e.g., `coord = list(samp1 = c(0.1, 0.2)`. If you have not measured x coordinates, please x should be 0.
`plot`	logical (*with default*): enable/disable plot output

... other parameters to be passed to modify the plot output. Supported are `run` to provide the run name , if the input is a `list`, this is set automatically. Further plot parameters are `surface_values` (character with value to plot), `legend` (`TRUE/FALSE`), `col_ramp` (for surface mode), `contour` (contour lines `TRUE/FALSE` in surface mode), `grid` (`TRUE/FALSE`), `col`, `pch` (for profile mode), `xlim` (a name list for profile mode), `ylim`, `zlim` (surface mode only), `ylab`, `xlab`, `zlab` (here x-axis labelling), `main`, `bg_img` (for profile mode background image, usually a profile photo; should be a raster object), `bg_img_positions` (a vector with the four corner positions, cf. graphics::rasterImage)

Details

This function only works with RLum.Analysis objects produced by read_PSL2R. It further assumes (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`).

Note: The function assumes the following sequence pattern: `DARK COUNT`**,** `IRSL`**,** `DARK COUNT`**,** `BSL`**,** `DARK COUNT`**. If you have written a different sequence, the analysis function will (likely) not work!**.

Signal processing The function processes the signals as follows: `BSL` and `IRSL` signals are extracted using the chosen signal integral, dark counts are taken in full.

Working with coordinates Usually samples are taken from a profile with a certain stratigraphy. In the past the function calculated an index. With this newer version, you have two option of passing on xy-coordinates to the function:

- (1) Add coordinates to the sample name during measurement. The form is rather strict and has to follow the scheme `_x:<number>|y:<number>`. Example: `sample_x:0.2|y:0.4`.
- (2) Alternatively, you can provide a list or matrix with the sample coordinates. Example: `coord = list(c(0.2, 1), c(0.3,1.2))`

Please note that the unit is meter (m) and the function expects always xy-coordinates. The latter one is useful for surface interpolations. If you have measured a profile where the x-coordinates to not measure, x-coordinates should be 0.

Value

Returns an S4 RLum.Results object with the following elements:
`$data`
`.. $summary`: data.frame with the results
`.. $data`: list with the RLum.Analysis objects
`.. $args`: list the input arguments

Function version

0.1.1

Author(s)

Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, Institute of Geography, Ruprecht-Karl University of Heidelberg, Germany

See Also

RLum.Analysis, RLum.Data.Curve, read_PSL2R

Examples

```
## example profile plot
# (1) load example data set
data("ExampleData.portableOSL", envir = environment())

# (2) merge and plot all RLum.Analysis objects
merged <- merge_RLum(ExampleData.portableOSL)
plot_RLum(
 object = merged,
 combine = TRUE,
 records_max = 5,
 legend.pos = "outside")
merged

# (3) analyse and plot
results <- analyse_portableOSL(
  merged,
  signal.integral = 1:5,
  invert = FALSE,
  normalise = TRUE)
get_RLum(results)
```

---

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 = NA,
  signal.integral.max = NA,
  background.integral.min = NA,
  background.integral.max = NA,
  OSL.component = NULL,
  rejection.criteria = list(),
  dose.points = NULL,
  trim_channels = FALSE,
  mtext.outer = "",
  plot = TRUE,
  plot_onePage = FALSE,
  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. The object should contain **only** curves considered part of the SAR protocol (see Details.)

`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. Can be set to `NA`, in this case no integrals are taken into account.

`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. Can be set to `NA`, in this case no integrals are taken into account.

`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. Can be set to `NA`, in this case no integrals are taken into account.

`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. Can be set to `NA`, in this case no integrals are taken into account.

`OSL.component` character or integer (*optional*): s single index or a character defining the signal component to be evaluated. It requires that the object was processed by `[OSLdecomposition::RLum.OSL_decomposition]`. This argument can either be the name of the OSL component assigned by `[OSLdecomposition::RLum.OSL_global_fitting]` or the index in the descending order of decay rates. Then `"1"` selects the fastest decaying component, `"2"` the second fastest and so on. Can be a list of integers or strings (or mixed) If object is a list and this parameter is provided as list it alternates over the elements (aliquots) of the object list, e.g., `list(1,2)` processes the first aliquot with component `1` and the second aliquot with component `2`. `NULL` does not process any component.

`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. Note: If an *unnamed* list is provided the new settings are ignored!

Allowed arguments are `recycling.ratio`, `recuperation.rate`, `palaeodose.error`, `testdose.error`, `exceed.max.regpoint = TRUE/FALSE`, `recuperation_reference = "Natural"` (or any other dose point, e.g., `"R1"`). Example: `rejection.criteria = list(recycling.ratio = 10)`. Per default all numerical values are set to 10, `exceed.max.regpoint = TRUE`. Every criterion 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 extracted from other data. Can be a list of numeric vectors, if `object` is of type list

`trim_channels` logical (*with default*): trim channels per record category to the lowest number of channels in the category by using trim_RLum.Data. Applies only to `OSL` and `IRSL` curves. For a more granular control use trim_RLum.Data before passing the input object.

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_onePage	logical (*with default*): enables or disables on page 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.

What is part of a SAR sequence?

The function is rather picky when it comes down to accepted curve input (OSL,IRSL,...) and structure. A SAR sequence is basically a set of L_x/T_x curves. Hence, every 2nd curve is considered a shine-down curve related to the test dose. It also means that the number of curves for L_x has to be equal to the number of T_x curves, and that hot-bleach curves **do not** belong into a SAR sequence; at least not for the analysis. Other curves allowed and processed are preheat curves, or preheat curves measured as TL, and irradiation curves. The later one indicates the duration of the irradiation, the dose and test dose points, e.g., as part of XSYG files.

Argument object is of type list

If the argument object is of type list containing **only** RLum.Analysis objects, the function re-calls 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 an efficient data analysis, but the function may also break with an unclear error message, due to wrong input data.

Working with IRSL data

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

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

Supported rejection criteria

[recycling.ratio]: calculated for every repeated regeneration dose point.
[recuperation.rate]: recuperation rate calculated by comparing the Lx/Tx values of the zero regeneration point with the Ln/Tn value (the Lx/Tx ratio of the natural signal). For methodological background see Aitken and Smith (1988). As a variant with the argument recuperation_reference another dose point can be selected as reference instead of Ln/Tn.
[testdose.error]: set the allowed error for the test dose, which per default should not exceed 10%. The test dose error is calculated as Tx_net.error/Tx_net. The calculation of the T_n error is detailed in calc_OSLLxTxRatio.
[palaeodose.error]: set the allowed error for the De value, which per default should not exceed 10%.

Irradiation times

The function makes two attempts to extra irradiation data (dose points) automatically from the input object, if the argument dose.points was not set (aka set to NULL).

1. It searches in every curve for an info object called IRR_TIME. If this was set, any value set here is taken as dose point.
2. If the object contains curves of type irradiation, the function tries to use this information to assign these values to the curves. However, the function does **not** overwrite values preset in IRR_TIME.

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 by used as rejection criteria. NA is produced if no R0 dose point exists.
Formula	formula formula that have been used for the growth curve fitting

The output should be accessed using the function get_RLum.

Function version

0.10.3

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, Institute of Geography, Heidelberg University (Germany)

References

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

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

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

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

See Also

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

Examples

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

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

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

##show De results
get_RLum(results)

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

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

Description

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

Usage

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

Arguments

`input.data` Risoe.BINfileData (**required**): input data from a Risø BIN file, produced by the function read_BIN2R.

`signal.integral` vector (**required**): channels used for the signal integral, e.g. `signal.integral=c(1:2)`

`background.integral` vector (**required**): channels used for the background integral, e.g. `background.integral=c(85:100`

`position` vector (*optional*): reader positions that want to be analysed (e.g. `position=c(1:48)`. Empty positions are automatically omitted. If no value is given all positions are analysed by default.

`run` vector (*optional*): range of runs used for the analysis. If no value is given the range of the runs in the sequence is deduced from the `Risoe.BINfileData` object.

`set` vector (*optional*): range of sets used for the analysis. If no value is given the range of the sets in the sequence is deduced from the `Risoe.BINfileData` object.

`dtype` character (*optional*): allows to further limit the curves by their data type (`DTYPE`), e.g., `dtype = c("Natural", "Dose")` limits the curves to this two data types. By default all values are allowed. See Risoe.BINfileData 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
`[recyling ratio]`: calculated for every repeated regeneration dose point.
`[recuperation]`: recuperation rate calculated by comparing the `Lx/Tx` values of the zero regeneration point with the `Ln/Tn` value (the `Lx/Tx` ratio of the natural signal). For methodological background see Aitken and Smith (1988)
`[IRSL/BOSL]`: the integrated counts (`signal.integral`) of an IRSL curve are compared to the integrated counts of the first regenerated dose point. It is assumed that IRSL curves got the same dose as the first regenerated dose point. **Note:** This is not the IR depletion 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

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, Institute of Geography, Heidelberg University (Germany)
Margret C. Fuchs, HZDR, Freiberg (Germany)

References

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

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

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

See Also

calc_OSLLxTxRatio, Risoe.BINfileData, read_BIN2R, plot_GrowthCurve

Examples

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

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

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

`analyse_SAR.TL` *Analyse SAR TL measurements*

Description

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

Usage

```
analyse_SAR.TL(
  object,
  object.background,
  signal.integral.min,
  signal.integral.max,
  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`)

`integral_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

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

Value

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

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

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

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

note: the output should be accessed using the function get_RLum

Function version

0.3.0

Note

THIS IS A BETA VERSION

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

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

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

See Also

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

Examples

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

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

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

apply_CosmicRayRemoval

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

Description

The function provides several methods for cosmic-ray removal and spectrum smoothing 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,
  ...
)
```

Arguments

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

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

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

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

`MARGIN` integer (*with default*): on which part the function cosmic ray removal should be applied on:

- 1 = along the time axis (line by line),
- 2 = along the wavelength axis (column by column).

Note: This argument currently only affects the methods `smooth` and `smooth.spline`

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

`plot`	logical (*with default*): If `TRUE` the histograms used for the cosmic-ray removal are returned as plot including the used threshold. Note: A separate plot is returned for each frame! Currently only for `method = "Pych"` a graphical output is provided.
`...`	further arguments and graphical parameters that will be passed to the smooth function.

Details

`method = "Pych"`

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

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

For further details see references below.

`method = "smooth"`

Method uses the function smooth to remove cosmic rays.

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

`method = "smooth.spline"`

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

Arguments that can be passed are: `spar`

How to combine methods?

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

Value

Returns same object as input.

Function version

0.3.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Pych, W., 2004. A Fast Algorithm for Cosmic-Ray Removal from Single Images. The Astronomical Society of the Pacific 116 (816), 148-153. doi:10.1086/381786

See Also

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

Examples

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

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

Description

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

Usage

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

Arguments

`object` RLum.Data.Spectrum or RLum.Analysis (**required**): S4 object of class `RLum.Data.Spectrum`, `RLum.Analysis`or a list of such objects. Other objects in the list are skipped.

`spectral.efficiency` 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 stats::approx

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

Value

Returns same object as provided as input

Function version

0.2.0

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)

See Also

RLum.Data.Spectrum, RLum.Analysis

Examples

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

---

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**
data.frame	data.frame
matrix	matrix

RLum.Data.Spectrum

from	**to**
data.frame	data.frame
matrix	matrix

RLum.Results

from	**to**
list	list

Given that the list consists 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
Cresswelletal2018:	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

0.2.0

Source

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

References

Adamiec, G., Aitken, M.J., 1998. Dose-rate conversion factors: update. Ancient TL 16, 37-46.

Cresswell., A.J., Carter, J., Sanderson, D.C.W., 2018. Dose rate conversion parameters: Assessment of nuclear data. Radiation Measurements 120, 195-201.

Guerin, G., Mercier, N., Adamiec, G., 2011. Dose-rate conversion factors: update. Ancient TL, 29, 5-8.

Liritzis, I., Stamoulis, K., Papachristodoulou, C., Ioannides, K., 2013. A re-evaluation of radiation dose-rate conversion factors. Mediterranean Archaeology and Archaeometry 13, 1-15.

Examples

```
## Load data
data("BaseDataSet.ConversionFactors", envir = environment())
```

`BaseDataSet.CosmicDoseRate` *Base data set for cosmic dose rate calculation*

Description

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

Format

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

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

Version

0.1

Source

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

values.cosmic.Softcomp

Program:	"AGE"
Reference:	Gruen (2009)

Fit: Polynomials in the form of

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

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

(For depths <40 g cm^-2)

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

`values.factor.Altitude`

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

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

`values.par.FJH`

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

F (non-linear part, λ < 36.5 deg.):

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

F (linear part, λ > 36.5 deg.):

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

J (non-linear part, λ < 34 deg.):

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

J (linear part, λ > 34 deg.):

$$y = 0.0005 * x + 0.7388$$

H (non-linear part, λ < 36 deg.):

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

H (linear part, λ > 36 deg.):

$$y = 0.0002 * x + 4.0914$$

References

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

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

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

Examples

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

`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

Source

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

References

Aitken, M.J., 1985. Thermoluminescence Dating. Academic Press, London.

Examples

```
## Load data
data("BaseDataSet.FractionalGammaDose", envir = environment())
```

`BaseDataSet.GrainSizeAttenuation` *Base dataset for grain size attenuation data by Guérin et al. (2012)*

Description

Grain size correction data for beta-dose rates published by Guérin et al. (2012).

#' @format

A `data.frame` seven columns and sixteen rows. Column headers are `GrainSize`, `Q_K`, `FS_K`, `Q_Th`, `FS_Th`, `Q_U`, `FS_U`. Grain sizes are quoted in µm (e.g., 20, 40, 60 etc.)

Version

0.1.0

Source

Guérin, G., Mercier, N., Nathan, R., Adamiec, G., Lefrais, Y., 2012. On the use of the infinite matrix assumption and associated concepts: A critical review. Radiation Measurements, 47, 778-785.

Examples

```
## load data
data("BaseDataSet.GrainSizeAttenuation", envir = environment())
```

`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

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Data.Spectrum

Examples

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

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

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

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

Description

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

Usage

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

Arguments

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

Details

This function can be used to either estimate the number of grains on an aliquot or to compute the packing density depending on the the arguments provided.

The following function is used to estimate the number of grains `n`:

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

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

Packing density

The default value for `packing.density` is 0.65, which is the mean of empirical values determined by Heer et al. (2012) and unpublished data from the Cologne luminescence laboratory. If `packing.density = "Inf"` a maximum density of $\pi/\sqrt{12} = 0.9068\ldots$ is used. However, note that this value is not appropriate as the standard preparation procedure of aliquots resembles a PECC (*"Packing Equal Circles in a Circle"*) problem where the maximum packing density is asymptotic to about 0.87.

Monte Carlo simulation

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

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

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

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

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

Value

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

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

The output should be accessed using the function get_RLum.

Function version

0.31

Author(s)

Christoph Burow, University of Cologne (Germany)

References

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

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

Further reading

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

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

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

Examples

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

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

`calc_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

```
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.
`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.

`[ NUMERICAL OUTPUT ]`

`RLum.Results`-object

slot: `@data`

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

Column	Type	Description
AVERAGE_DOSE	numeric	the obtained average dose
AVERAGE_DOSE.SE	numeric	the average dose error
SIGMA_D	numeric	sigma
SIGMA_D.SE	numeric	standard error of the sigma
IC_AVERAGE_DOSE.LEVEL	character	confidence level average dose
IC_AVERAGE_DOSE.LOWER	character	lower quantile of average dose
IC_AVERAGE_DOSE.UPPER	character	upper quantile of average dose
IC_SIGMA_D.LEVEL	integer	confidence level sigma
IC_SIGMA_D.LOWER	character	lower sigma quantile
IC_SIGMA_D.UPPER	character	upper sigma quantile
L_MAX	character	maximum likelihood value

`[.. $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

0.1.5

Note

This function has beta status!

Author(s)

Claire Christophe, IRAMAT-CRP2A, Université de Nantes (France), Anne Philippe, Université de Nantes, (France), Guillaume Guérin, IRAMAT-CRP2A, Université Bordeaux Montaigne, (France), Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Guerin, G., Christophe, C., Philippe, A., Murray, A.S., Thomsen, K.J., Tribolo, C., Urbanova, P., Jain, M., Guibert, P., Mercier, N., Kreutzer, S., Lahaye, C., 2017. Absorbed dose, equivalent dose, measured dose rates, and implications for OSL age estimates: Introducing the Average Dose Model. Quaternary Geochronology 1-32. doi:10.1016/j.quageo.2017.04.002

Further reading

Efron, B., Tibshirani, R., 1986. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. Statistical Science 1, 54-75.

See Also

read.table, graphics::hist

Examples

```
##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 (*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 \ 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
`na.rm`	logical (*with default*): strip `NA` values before the computation proceeds
`plot`	logical (*with default*): plot output
`...`	further arguments (`trace`, `verbose`).

Details

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

Value

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

1.4.1

Author(s)

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

References

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

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

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

Further reading

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

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

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

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

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

See Also

plot, calc_CommonDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

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

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

`calc_CobbleDoseRate` *Calculate dose rate of slices in a spherical cobble*

Description

Calculates the dose rate profile through the cobble based on Riedesel and Autzen (2020).

Corrects the beta dose rate in the cobble for the grain size following results of Guérin et al. (2012). Sediment beta and gamma dose rates are corrected for the water content of the sediment using the correction factors of Aitken (1985). Water content in the cobble is assumed to be 0.

Usage

```
calc_CobbleDoseRate(input, conversion = "Guerinetal2011")
```

Arguments

`input`	data.frame (**required**): A table containing all relevant information for each individual layer. For the table layout see details.
`conversion`	Which dose rate conversion factors to use. For accepted values see BaseDataSet.ConversionFactors

Details

The input table layout

COLUMN	DATA TYPE	DESCRIPTION
`Distance`	`numeric`	distance from the surface of the cobble to the top of each rock slice in mm. The dista
`DistanceError`	`numeric`	Error on the distance in mm
`Thickness`	`numeric`	Thickness of each slice in mm
`TicknessError`	`numeric`	uncertainty of the thickness in mm.
`Mineral`	`character`	`'FS'` for feldspar, `'Q'` for quartz, depending which mineral in the cobble is used for
`Cobble_K`	`numeric`	K nuclide content in % of the bulk cobble
`Cobble_K_SE`	`numeric`	error on K nuclide content in % of the bulk cobble
`Cobble_Th`	`numeric`	Th nuclide content in ppm of the bulk cobble
`Cobble_Th_SE`	`numeric`	error on Th nuclide content in ppm of the bulk cobble
`Cobble_U`	`numeric`	U nuclide content in ppm of the bulk cobble
`CobbleU_SE`	`numeric`	error on U nuclide content in ppm of the bulk cobble
`GrainSize`	`numeric`	average grain size in µm of the grains used for dating
`Density`	`numeric`	Density of the cobble. Default is 2.7 g cm^-3
`CobbleDiameter`	`numeric`	Diameter of the cobble in cm.
`Sed_K`	`numeric`	K nuclide content in % of the sediment matrix
`Sed_K_SE`	`numeric`	error on K nuclide content in % of the sediment matrix
`Sed_Th`	`numeric`	Th nuclide content in ppm of the sediment matrix
`Sed_Th_SE`	`numeric`	error on Th nuclide content in ppm of the sediment matrix
`Sed_U`	`numeric`	U nuclide content in ppm of the sediment matrix
`Sed_U_SE`	`numeric`	error on U nuclide content in ppm of the sediment matrix
`GrainSize`	`numeric`	average grain size of the sediment matrix
`WaterContent`	`numeric`	mean water content of the sediment matrix in %
`WaterContent_SE`	`numeric`	relative error on water content

Water content The water content provided by the user should be calculated according to:

$$(Wet_{w}eight - Dry_{w}eight)/Dry_{w}eight * 100$$

Value

The function returns an RLum.Results object for which the first element is a matrix (`DataIndividual`) that gives the dose rate results for each slice for each decay chain individually, for both, the cobble dose rate and the sediment dose rate. The second element is also a matrix (`DataComponent`) that gives the total beta and gamma-dose rates for the cobble and the adjacent sediment for each slice of the cobble.

Function version

0.1.0

Author(s)

Svenja Riedesel, Aberystwyth University (United Kingdom)
Martin Autzen, DTU NUTECH Center for Nuclear Technologies (Denmark)

References

Riedesel, S., Autzen, M., 2020. Beta and gamma dose rate attenuation in rocks and sediment. Radiation Measurements 133, 106295.

See Also

convert_Concentration2DoseRate

Examples

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

## run function
calc_CobbleDoseRate(ExampleData.CobbleData)
```

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

Description

Function to calculate the common dose of a De distribution.

Usage

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

Arguments

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

Author(s)

Christoph Burow, University of Cologne (Germany)

References

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

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

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

Further reading

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

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

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

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

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

See Also

calc_CentralDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

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

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

`calc_CosmicDoseRate` *Calculate the cosmic dose rate*

Description

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

Usage

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

```
  error = 10,
  ...
)
```

Arguments

`depth`	numeric (**required**): depth of overburden (m). For more than one absorber use `c(depth_1, depth_2, ..., depth_n)`
`density`	numeric (**required**): average overburden density (g/cm^3). For more than one absorber use `c(density_1, density_2, ..., density_n)`
`latitude`	numeric (**required**): latitude (decimal degree), N positive
`longitude`	numeric (**required**): longitude (decimal degree), E positive
`altitude`	numeric (**required**): altitude (m above sea-level)
`corr.fieldChanges`	logical (*with default*): correct for geomagnetic field changes after Prescott & Hutton (1994). Apply only when justified by the data.
`est.age`	numeric (*with default*): estimated age range (ka) for geomagnetic field change correction (0-80 ka allowed)
`half.depth`	logical (*with default*): How to overcome with varying overburden thickness. If `TRUE` only half the depth is used for calculation. Apply only when justified, i.e. when a constant sedimentation rate can safely be assumed.
`error`	numeric (*with default*): general error (percentage) to be implemented on corrected cosmic dose rate estimate
`...`	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)

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

(2) If `half.depth` = `TRUE`

$$absorber = absorber/2$$

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

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

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

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

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

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

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

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

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

$$Dc' = Dc * correctionFactor$$

Usage of `depth` and `density`

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

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

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

Value

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

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

The output should be accessed using the function get_RLum

Function version

0.5.2

Note

Despite its universal use the equation to calculate the cosmic dose rate provided by Prescott & Hutton (1994) is falsely stated to be valid from the surface to 10^4 hg/cm^2 of standard rock. The original expression by Barbouti & Rastin (1983) only considers the muon flux (i.e. hard-component) and is by their own definition only valid for depths between 10-10^4 hg/cm^2.

Thus, for near-surface samples (i.e. for depths < 167 g/cm^2) the equation of Prescott & Hutton (1994) underestimates the total cosmic dose rate, as it neglects the influence of the soft-component of the cosmic ray flux. For samples at zero depth and at sea-level the underestimation can be as large as ~0.1 Gy/ka. In a previous article, Prescott & Hutton (1988) give another approximation of Barbouti & Rastin's equation in the form of

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

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

As a result, this function employs the equation of Prescott & Hutton (1994) only for depths > 167 g/cm^2, i.e. only for the hard-component of the cosmic ray flux. Cosmic dose rate values for depths < 167 g/cm^2 were obtained from the "AGE" 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)

References

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

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

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

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

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

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

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

See Also

BaseDataSet.CosmicDoseRate

Examples

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

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

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


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

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

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

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

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

---

calc_FadingCorr *Fading Correction after Huntley & Lamothe (2001)*

Description

Apply a fading correction according to Huntley & Lamothe (2001) for a given g-value and a given t_c

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

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):

$$\frac{A_f}{A} = 1 - \kappa * \left[ln(\frac{A}{t_c}) - 1\right]$$

with κ defined as

$$\kappa = \frac{\frac{\text{g\_value}}{ln(10)}}{100}$$

A and A_f are given in ka. t_c is given in s, however, it is internally recalculated to ka.

As the g-value slightly depends on the time between irradiation and the prompt measurement, this is t_c, always a t_c 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 t_c value for the g-value needs to be provided (argument tc.g_value and then the g-value is recalculated to t_c of the measurement used for estimating the age applying the following equation:

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

where

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

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

`n.MC = 'auto'`

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

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

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

`seed`

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

FAQ

Q: Which t_c value is expected?

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

Q: The function could not find a solution, what should I do?

A: This usually happens for model parameters exceeding the boundaries of the fading correction model (e.g., very high g-value). Please check whether another fading correction model might be more appropriate.

Value

Returns an S4 object of type RLum.Results.

Slot: `@data`

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

Slot: `@info`

Object	Type	Comment
info	character	the original function call

Function version

0.4.3

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

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

See Also

RLum.Results, analyse_FadingMeasurement, get_RLum, uniroot

Examples

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

##(1) faded age: 100 a
results <- calc_FadingCorr(
   age.faded = c(0.1,0),
   g_value = c(5.0, 1.0),
   tc = 2592000,
   tc.g_value = 172800,
   n.MC = 100)

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

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

```
  n.MC = 100)

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

---

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

Description

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

Usage

```
calc_FastRatio(
  object,
  stimulation.power = 30.6,
  wavelength = 470,
  sigmaF = 2.6e-17,
  sigmaM = 4.28e-18,
  Ch_L1 = 1,
  Ch_L2 = NULL,
  Ch_L3 = NULL,
  x = 1,
  x2 = 0.1,
  dead.channels = c(0, 0),
  fitCW.sigma = FALSE,
  fitCW.curve = FALSE,
  plot = TRUE,
  ...
)
```

Arguments

`object`	RLum.Analysis, RLum.Data.Curve or data.frame (**required**): x, y data of measured values (time and counts).
`stimulation.power`	numeric (*with default*): Stimulation power in mW/cm^2
`wavelength`	numeric (*with default*): Stimulation wavelength in nm
`sigmaF`	numeric (*with default*): Photoionisation cross-section (cm^2) of the fast component. Default value after Durcan & Duller (2011).
`sigmaM`	numeric (*with default*): Photoionisation cross-section (cm^2) of the medium component. Default value after Durcan & Duller (2011).
`Ch_L1`	numeric (*with default*): An integer specifying the channel for L1.
`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*): \ Used to define the location of L2 and L3 (start).
`x2`	numeric (*with default*): \ 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` \ and end). The fast ratio is then calculated from: $(L1 - L3)/(L2 - L3)$.

Value

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

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

Function version

0.1.1

Author(s)

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

References

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

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

Further reading

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

See Also

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

Examples

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

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

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

---

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

Description

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

Usage

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

Arguments

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

`n.components`	numeric (**required**): number of components to be fitted. If a vector is provided (e.g. `c(2:8)`) the finite mixtures for 2, 3 ... 8 components are calculated and a plot and a statistical evaluation of the model performance (BIC score and maximum log-likelihood) is provided.
`grain.probability`	logical (*with default*): prints the estimated probabilities of which component each grain is in
`dose.scale`	numeric: manually set the scaling of the y-axis of the first plot with a vector in the form of `c(min, max)`
`pdf.weight`	logical (*with default*): weight the probability density functions by the components proportion (applies only when a vector is provided for `n.components`)
`pdf.sigma`	character (*with default*): if `"sigmab"` the components normal distributions are plotted with a common standard deviation (i.e. `sigmab`) as assumed by the FFM. Alternatively, `"se"` takes the standard error of each component for the sigma parameter of the normal distribution
`pdf.colors`	character (*with default*): colour 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 graphics::barplot showing the proportions of components if `n.components` a vector with a length > 1 (e.g., `n.components = c(2:3)`)
`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 `NaN`
- 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 graphics::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

0.4.2

Author(s)

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

References

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

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

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

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

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

Further reading

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

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

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

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

See Also

calc_CentralDose, calc_CommonDose, calc_FuchsLang2001, calc_MinDose

Examples

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

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

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

## show structure of the results
FMM

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

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

Description

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

Usage

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

Arguments

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

Details

Used values

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

Basic steps of the approach

1. Estimate natural relative variation of the sample using a dose recovery test
2. Sort the input values in ascending order
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

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Christoph Burow, University of Cologne (Germany)

References

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

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

See Also

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

Examples

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

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

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

Description

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

Usage

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

Arguments

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

`gSGC.parameters`
: list (*optional*): option to provide own function parameters used for fitting as named list. Nomenclature follows Li et al., 2015, i.e. `list(A,A.error,D0,D0.error,c,c.error,Y`
range requires a vector for the range the function is considered as valid, e.g. `range = c(0,250)`
Using this option overwrites the default parameter list of the gSGC, meaning the argument `gSGC.type` will be without effect

`n.MC`
: integer (*with default*): number of Monte Carlo simulation runs for error estimation, see details.

`verbose`
: logical: enable or disable terminal output

`plot`
: logical: enable or disable graphical feedback as plot

`...`
: parameters will be passed to the plot output

Details

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

Value

Returns an S4 object of type RLum.Results.
`@data`
`$ De.value` (data.frame)
`.. $ De`
`.. $ De.error`
`.. $ Eta`
`$ De.MC` (list) contains the matrices from the error estimation.
`$ uniroot` (list) contains the uniroot outputs of the De estimations

`@info`
'$ call" (call) the original function call

Function version

0.1.1

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

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

See Also

RLum.Results, get_RLum, uniroot

Examples

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

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

---

calc_gSGC_feldspar   *Calculate Global Standardised Growth Curve (gSGC) for Feldspar MET-pIRIR*

Description

Implementation of the gSGC approach for feldspar MET-pIRIR by Li et al. (2015)

Usage

```
calc_gSGC_feldspar(
  data,
  gSGC.type = "50LxTx",
  gSGC.parameters,
  n.MC = 100,
  plot = FALSE
)
```

Arguments

`data`	data.frame (**required**): data frame with five columns per sample `c("LnTn", "LnTn.error", "Lr1Tr1", "Lr1Tr1.error","Dr1")`
`gSGC.type`	character (*with default*): growth curve type to be selected according to Table 3 in Li et al. (2015). Allowed options are `"50LxTx"`, `"50Lx"`, `"50Tx"`, `"100LxTx"`, `"100Lx"`, `"100Tx"`, `"150LxTx"`, `"150Lx"`, `"150Tx"`, `"200LxTx"`, `"200Lx"`, `"200Tx"`, `"250LxTx"`, `"250Lx"`, `"250Tx"`
`gSGC.parameters`	data.frame (*optional*): an own parameter set for the gSGC with the following columns `y1`, `y1_err`, `D1` `D1_err`, `y2`, `y2_err`, `y0`, `y0_err`.
`n.MC`	numeric (*with default*): number of Monte-Carlo runs for the error calculation
`plot`	logical (*with default*): enables/disables the control plot output

Details

##TODO

Value

Returns an S4 object of type RLum.Results.
@data
$ df (data.frame)
.. $DE the calculated equivalent dose
.. $DE.ERROR error on the equivalent dose, which is the standard deviation of the MC runs
.. $HPD95_LOWER lower boundary of the highest probability density (95%)
.. $HPD95_UPPER upper boundary of the highest probability density (95%)
$ m.MC (list) numeric vector with results from the MC runs.

@info
'$ call" (call) the original function call

Function version

0.1.0

Author(s)

Harrison Gray, USGS (United States), Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Li, B., Roberts, R.G., Jacobs, Z., Li, S.-H., Guo, Y.-J., 2015. Construction of a "global standardised growth curve" (gSGC) for infrared stimulated luminescence dating of K-feldspar 27, 119–130. doi:10.1016/j.quageo.2015.02.010

See Also

RLum.Results, get_RLum, uniroot, calc_gSGC

Examples

```
##test on a generated random sample
n_samples <- 10
data <- data.frame(
  LnTn = rnorm(n=n_samples, mean=1.0, sd=0.02),
  LnTn.error = rnorm(n=n_samples, mean=0.05, sd=0.002),
  Lr1Tr1 = rnorm(n=n_samples, mean=1.0, sd=0.02),
  Lr1Tr1.error = rnorm(n=n_samples, mean=0.05, sd=0.002),
  Dr1 = rep(100,n_samples))

results <- calc_gSGC_feldspar(
  data = data, gSGC.type = "50LxTx",
  plot = FALSE)

plot_AbanicoPlot(results)
```

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

Function version

0.3.0

Author(s)

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

References

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

See Also

pchisq

Examples

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

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

## 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)
```

`calc_Huntley2006` *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"),
  lower.bounds = c(-Inf, -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 \
- 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.

```
                          (optional)
     | dose (s)| LxTx | LxTx error |
     |  [ ,1]  | [ ,2]|    [ ,3]   |
     |---------|------|------------|
[1, ]|  0      | LnTn | LnTn error | (optional, see arg 'LnTn')
[2, ]|  R1     | L1T1 | L1T1 error |
 ... |    ...  |  ... |     ...    |
[x, ]|  Rx     | LxTx | LxTx error |
```

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:

```
     |  LnTn  |  LnTn error  |
     |  [ ,1] |      [ ,2]   |
     |--------|--------------|
[1, ]| LnTn_1 | LnTn_1 error |
[2, ]| LnTn_2 | LnTn_2 error |
 ... |   ...  |      ...     |
[x, ]| LnTn_x | LnTn_x error |
```

The function will calculate a **mean** `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 `LnTn`-values with this argument the data frame for the `data`-argument **must not** contain any `LnTn`-values!

`rhop` numeric (**required**): The density of recombination centres (ρ') and its error (see Huntley 2006), given as numeric vector of length two. Note that ρ' must **not** be provided as the common logarithm. Example: `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: `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: `readerDdot = c(0.08, 0.01)`.

`normalise` logical (*with default*): If `TRUE` (the default) all measured and computed $\frac{L_x}{T_x}$ 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 `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 adaptation of the model and should be used with great care.

`lower.bounds` numeric (*with default*): Only applicable for `fit.method = 'GOK'`. A vector of length 3 that contains the lower bound values for fitting the general-order kinetics function using minpack.lm::nlsLM. In most cases, the default values (`c(-Inf, -Inf, -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 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 plot and plot_GrowthCurve (in particular `mode` for the fit mode and `fit.force_through_origin`)

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. ρ' (`rhop`), 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 ρ' 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 $\frac{L_x}{T_x}$ data as a function of irradiation time, and fitting the data with a single saturating exponential of the form:

$$\frac{L_x}{T_x}(t^*) = A\phi(t^*)\{1 - exp(-\frac{t^*}{D_0}))\}$$

where

$$\phi(t^*) = exp(-\rho' ln(1.8\tilde{s}t^*)^3)$$

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 $\tilde{s}$ (3×10^{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, $\dot{D}$ (Gy/s) and equations `[1]` and `[2]`. Computed $\frac{L_x}{T_x}$ 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 D_e 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:

$$\frac{L_x}{T_x}(t^*) = A\phi(t^*)(1 - (1 + (\frac{1}{D_0})t^*c)^{-1/c})$$

where A, ϕ, 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 ($\frac{n}{N}$) and the field saturation (i.e. athermal steady state, (n/N)_SS) value for the sample under investigation using the sample specific ρ', unfaded D_0 and $\dot{D}$ values, following the approach of Kars et al. (2008).

Uncertainties

Uncertainties are reported at 1σ and are assumed to be normally distributed and are estimated using Monte-Carlo re-sampling (`n.MC = 1000`) of ρ' and $\frac{L_x}{T_x}$ during dose response curve fitting, and of ρ' in the derivation of (n/N) and (n/N)_SS.

Age calculated from 2D0 of the simulated natural DRC

In addition to the age calculated from the equivalent dose derived from $\frac{L_n}{T_n}$ 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 $\frac{L_n}{T_n}$ 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**

OBJECT	**TYPE**	**COMMENT**
`results`	data.frame	results of the of Kars et al. 2008 model
`data`	data.frame	original input data
`Ln`	numeric	Ln and its error
`LxTx_tables`	`list`	A `list` of `data.frames` containing data on dose, LxTx and LxTx error for each of the dose
`fits`	`list`	A `list` of `nls` objects produced by minpack.lm::nlsLM when fitting the dose response curv

Slot: **@info**

OBJECT	**TYPE**	**COMMENT**
`call`	`call`	the original function call
`args`	`list`	arguments of the original function call

Function version

0.4.5

Note

This function has BETA status, in particular for the GOK implementation. Please verify your results carefully

Author(s)

Georgina E. King, University of Lausanne (Switzerland)
Christoph Burow, University of Cologne (Germany)
Sebastian Kreutzer, Ruprecht-Karl University of Heidelberg (Germany)

References

Kars, R.H., Wallinga, J., Cohen, K.M., 2008. A new approach towards anomalous fading correction for feldspar IRSL dating-tests on samples in field saturation. Radiation Measurements 43, 786-790. doi:10.1016/j.radmeas.2008.01.021

Guralnik, B., Li, B., Jain, M., Chen, R., Paris, R.B., Murray, A.S., Li, S.-H., Pagonis, P., Herman, F., 2015. Radiation-induced growth and isothermal decay of infrared-stimulated luminescence from feldspar. Radiation Measurements 81, 224-231.

Huntley, D.J., 2006. An explanation of the power-law decay of luminescence. Journal of Physics: Condensed Matter 18, 1359-1365. doi:10.1088/0953-8984/18/4/020

King, G.E., Herman, F., Lambert, R., Valla, P.G., Guralnik, B., 2016. Multi-OSL-thermochronometry of feldspar. Quaternary Geochronology 33, 76-87. doi:10.1016/j.quageo.2016.01.004

Further reading

Morthekai, P., Jain, M., Cunha, P.P., Azevedo, J.M., Singhvi, A.K., 2011. An attempt to correct for the fading in million year old basaltic rocks. Geochronometria 38(3), 223-230.

Examples

```
## 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),
 nTn.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	RLum.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

Author(s)

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

References

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

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

See Also

plot, calc_CommonDose, calc_CentralDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

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

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

`calc_Kars2008` *Apply the Kars et al. (2008) model (deprecated)*

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. ρ' (`rhop`), 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 ρ' value.

Value

An RLum.Results object is returned:

Function version

0.4.0

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)

References

Kars, R.H., Wallinga, J., Cohen, K.M., 2008. A new approach towards anomalous fading correction for feldspar IRSL dating-tests on samples in field saturation. Radiation Measurements 43, 786-790. doi:10.1016/j.radmeas.2008.01.021

Huntley, D.J., 2006. An explanation of the power-law decay of luminescence. Journal of Physics: Condensed Matter 18, 1359-1365. doi:10.1088/0953-8984/18/4/020

King, G.E., Herman, F., Lambert, R., Valla, P.G., Guralnik, B., 2016. Multi-OSL-thermochronometry of feldspar. Quaternary Geochronology 33, 76-87. doi:10.1016/j.quageo.2016.01.004

Further reading

Morthekai, P., Jain, M., Cunha, P.P., Azevedo, J.M., Singhvi, A.K., 2011. An attempt to correct for the fading in million year old basaltic rocks. Geochronometria 38(3), 223-230.

Examples

```
## 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,
                ddot = ddot,
                readerDdot = readerDdot,
                n.MC = 25)
)
```

---

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 \$L_n/T_n\$ values and fits 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. Allow are (1) data.frame with three columns (`dose`, `LxTx`, `LxTx error`; see details), (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, which is, according to Lamothe et al. (2003) De/t*.
- `g_value` — numeric vector of length 2 (**required**): g_value in \ 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 the **end** of the irradiation and the prompt measurement used in the equivalent dose 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 estimating the *g*-value. If the *g*-value was normalised to, e.g., 2 days, this time in seconds (i.e., `172800`) should be entered here along with the time used for the equivalent dose estimation. If nothing is provided the time is set to `tc`, which is the 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

Details

Format of `object` if `data.frame`

If `object` is of type data.frame, all input values most be of type numeric. Dose values are excepted in seconds (s) not Gray (Gy). No `NA` values are allowed and the value for the natural dose (first row) should be `0`. Example for three dose points, column names are arbitrary:

```
object <- data.frame(
dose = c(0,25,50),
LxTx = c(4.2, 2.5, 5.0),
LxTx_error = c(0.2, 0.1, 0.2))
```

Note on the g-value and `tc`

Users new to R and fading measurements are often confused about what to enter for `tc` and why it may differ from `tc.g_value`. The `tc` value is, by convention (Huntley & Lamothe 2001), the time elapsed between the end of the irradiation and the prompt measurement. Usually there is no reason for having a `tc` value different for the equivalent dose measurement and the *g*-value measurement, except if different equipment was used. However, if, for instance, the *g*-value measurement sequence was analysed with the *Analyst* (Duller 2015) and the `'Luminescence` is used to correct for fading, there is a high chance that the value returned by the *Analyst* comes normalised to 2-days; even the `tc` values of the measurement were identical. In such cases, the fading correction cannot be correct until the `tc.g_value` was manually set to 2-days (`172800` s) because the function will internally recalculate values to an identical `tc` value.

Value

The function returns are graphical output produced by the function plot_GrowthCurve and an RLum.Results.

`[ NUMERICAL OUTPUT ]`

`RLum.Results`-object

slot: `@data`

Element	Type	Description
`$data`	`data.frame`	the fading corrected values
`$fit`	`nls`	the object returned by the dose response curve fitting

'**slot:** `@info`

The original function call

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany), Norbert Mercier, IRAMAT-CRP2A, Université Bordeaux Montaigne (France)

References

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

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

Lamothe, M., Auclair, M., Hamzaoui, C., Huot, S., 2003. Towards a prediction of long-term anomalous fading of feldspar IRSL. Radiation Measurements 37, 493-498.

See Also

plot_GrowthCurve, calc_FadingCorr, analyse_SAR.CWOSL, analyse_pIRIRSequence

Examples

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

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

##perform SAR analysis and set rejection criteria
results <- analyse_SAR.CWOSL(
object = object,
signal.integral.min = 1,
signal.integral.max = 2,
background.integral.min = 900,
background.integral.max = 1000,
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

```
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 & Walling 2012, p. 100). **NOTE**: For the logged model (`log = TRUE`) this value must be a fraction, e.g. 0.2 (= 20 \ 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 create 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

Author(s)

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

References

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

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

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

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

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

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

Further reading

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

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

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

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

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

See Also

calc_CentralDose, calc_CommonDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MinDose

Examples

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

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

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

Description

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

Usage

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

`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 \

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

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

Multicore support

This function supports parallel computing and can be activated by `multicore=TRUE`. By default, the number of available logical CPU cores is determined automatically, but can be changed with `cores`. The multicore support is only available when `bootstrap=TRUE` and spawns `n` R instances for each core to get MAM estimates for each of the N and M 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 performance 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:

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

`.$profile`	stats::profile the log likelihood profiles
`.$bootstrap`	list bootstrap results

The output should be accessed using the function get_RLum

Function version

0.4.4

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

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

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

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

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

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

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

Further reading

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

Bolker, B., 2016. Maximum likelihood estimation analysis with the bbmle package. In: Bolker, B., R Development Core Team, 2016. bbmle: Tools for General Maximum Likelihood Estimation. R package version 1.0.18. https://CRAN.R-project.org/package=bbmle

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

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

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

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

See Also

calc_CentralDose, calc_CommonDose, calc_FiniteMixture, calc_FuchsLang2001, calc_MaxDose

Examples

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

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

# Show structure of the RLum.Results object
mam

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

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

# Plot the dose distribution in an abanico plot and draw a line
# at the minimum dose estimate
plot_AbanicoPlot(data = ExampleData.DeValues$CA1,
                 main = "3-parameter Minimum Age Model",
                 line = mam,polygon.col = "none",
                 hist = TRUE,
                 rug = TRUE,
                 summary = c("n", "mean", "mean.weighted", "median", "in.ci"),
                 centrality = res$de,
                 line.col = "red",
                 grid.col = "none",
                 line.label = paste0(round(res$de, 1), "\U00B1",
                                     round(res$de_err, 1), " Gy"),
```

```
                  bw = 0.1,
                  ylim = c(-25, 18),
                  summary.pos = "topleft",
                  mtext = bquote("Parameters: " ~
                                 sigma[b] == .(get_RLum(mam, "args")$sigmab) ~ ", " ~
                                 gamma == .(round(log(res$de), 1)) ~ ", " ~
                                 sigma == .(round(res$sig, 1)) ~ ", " ~
                                 rho == .(round(res$p0, 2))))



# (3) Run the minimum age model with bootstrap
# NOTE: Bootstrapping is computationally intensive
# (3.1) run the minimum age model with default values for bootstrapping
calc_MinDose(data = ExampleData.DeValues$CA1,
             sigmab = 0.15,
             bootstrap = TRUE)

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

# Plot the results
plot_RLum(mam)

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

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

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

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

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

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

## End(Not run)
```

---

calc_OSLLxTxDecomposed — *Calculate Lx/Tx ratio for decomposed CW-OSL signal components*

Description

Calculate Lx/Tx ratios from a given set of decomposed CW-OSL curves decomposed by [OSLdecomposition::RLum.OS

Usage

```
calc_OSLLxTxDecomposed(
  Lx.data,
  Tx.data = NULL,
  OSL.component = 1L,
  sig0 = 0,
  digits = NULL
)
```

Arguments

Lx.data	data.frame (**required**): Component table created by [OSLdecomposition::RLum.OSL_decompositi and per default located at object@records[[...]]@info$COMPONENTS.The value of $n[OSL.component] is set as LnLx. The value of $n.error[OSL.component] is set as LnLx.error
Tx.data	data.frame (*optional*): Component table created by [OSLdecomposition::RLum.OSL_decompositio and per default located at object@records[[...]]@info$COMPONENTS. The value of $n[OSL.component] is set as TnTx. The value of $n.error[OSL.component] is set as TnTx.error
OSL.component	integer or character (*optional*): a single index or a name describing which OSL signal component shall be evaluated. This argument can either be the name of the OSL component assigned by [OSLdecomposition::RLum.OSL_global_fitting] or the index of component. Then '1' selects the fastest decaying component, '2' the second fastest and so on. If not defined, the fastest decaying component is selected.
sig0	numeric (*with default*): allows adding an extra error component to the final Lx/Tx error value (e.g., instrumental error).
digits	integer (*with default*): round numbers to the specified digits. If digits is set to NULL nothing is rounded.

Value

Returns an S4 object of type RLum.Results.

Slot data contains a list with the following structure:

@data

```
$LxTx.table (data.frame)
.. $ LnLx
.. $ TnTx
.. $ Net_LnLx
.. $ Net_LnLx.Error
.. $ Net_TnTx
.. $ Net_TnTx.Error
.. $ LxTx
.. $ LxTx.relError
.. $ LxTx.Error
```

Function version

0.1.0

Author(s)

Dirk Mittelstrass

References

Mittelstrass D., Schmidt C., Beyer J., Straessner A., 2019. Automated identification and separation of quartz CW-OSL signal components with R. talk presented at DLED 2019, Bingen, Germany http://luminescence.de/OSLdecomp_talk.pdf

See Also

RLum.Data.Curve, plot_GrowthCurve, analyse_SAR.CWOSL

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

Description

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

Usage

```
calc_OSLLxTxRatio(
  Lx.data,
  Tx.data = 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)
`Tx.data`	RLum.Data.Curve or data.frame (*optional*): requires a CW-OSL shine down curve (x = time, y = counts). If no input is given the `Tx.data` will be treated as `NA` and no `Lx/Tx` ratio is calculated.

`signal.integral`
: numeric (**required**): vector with the limits for the signal integral. Can be set to `NA` than now integrals are considered and all other integrals are set to `NA` as well.

`signal.integral.Tx`
: numeric (*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`
: numeric (**required**): vector with the bounds for the background integral. Can be set to `NA` than now integrals are considered and all other integrals are set to `NA` as well.

`background.integral.Tx`
: numeric (*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. `sigmab = c(300,300)`. **Note:** If only one value is provided this value is taken for both (`LnTx` and `TnTx`) signals.

`sig0`
: numeric (*with default*): allow adding an extra component of error to the final `Lx/Tx` error value (e.g., instrumental 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 subtraction method.

sig0

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

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

background.count.distribution

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

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

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

Value

Returns an S4 object of type RLum.Results.

Slot `data` contains a list with the following structure:

@data

```
$LxTx.table (data.frame)
.. $ LnLx
.. $ LnLx.BG
.. $ TnTx
.. $ TnTx.BG
.. $ Net_LnLx
.. $ Net_LnLx.Error
.. $ Net_TnTx
.. $ Net_TnTx.Error
.. $ LxTx
.. $ LxTx.Error
$ calc.parameters (list)
.. $ sigmab.LnTx
.. $ sigmab.TnTx
.. $ k
```

@info

```
$ call (original function call)
```

Function version

0.8.0

Note

The results of this function have been cross-checked with the Analyst (version 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, Institute of Geography, Heidelberg University (Germany)

References

Duller, G., 2018. Analyst v4.57 - User Manual. https://users.aber.ac.uk/ggd

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

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

See Also

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

Examples

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

##calculate Lx/Tx ratio
results <- calc_OSLLxTxRatio(
 Lx.data = Lx.data,
 Tx.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.

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

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

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

Allowed source types and related values

#	**Source type**	**T.1/2**	**Reference**
[1]	Sr-90	28.90 y	NNDC, Brookhaven National Laboratory
[2]	Am-214	432.6 y	NNDC, Brookhaven National Laboratory
[3]	Co-60	5.274 y	NNDC, Brookhaven National Laboratory
[4	Cs-137	30.08 y	NNDC, Brookhaven National Laboratory

Value

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

```
$ dose.rate (data.frame)
.. $ dose.rate
.. $ dose.rate.error
.. $ date (corresponding measurement date)
$ parameters (list)
.. $ source.type
.. $ halflife
.. $ dose.rate.unit
$ call (the original function call)
```

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

Function version

0.3.2

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, Institute of Geography, Heidelberg University (Germany)

References

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

See Also

Second2Gray, get_RLum, plot_RLum

Examples

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

##show results
get_RLum(dose.rate)

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

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

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

```
plot_RLum(dose.rate)


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


## End(Not run)
```

---

calc_Statistics *Function to calculate statistic measures*

Description

This function calculates a number of descriptive statistics for 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

Author(s)

Michael Dietze, GFZ Potsdam (Germany)

Examples

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

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

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

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

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

## End(Not run)
```

---

calc_ThermalLifetime *Calculates the Thermal Lifetime using the Arrhenius equation*

Description

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

Usage

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

Arguments

`E`	numeric (**required**): vector of trap depths in eV, if `profiling = TRUE` only the first two elements are considered
`s`	numeric (**required**): vector of frequency factor in 1/s, if `profiling = TRUE` only the first two elements are considered
`T`	numeric (*with default*): temperature in deg. C for which the lifetime(s) will be calculated. A vector can be provided.
`output_unit`	character (*with default*): output unit of the calculated lifetimes, accepted entries are: `"Ma"`, `"ka"`, `"a"`, `"d"`, `"h"`, `"min"`, `"s"`
`profiling`	logical (*with default*): this option allows to estimate uncertainties based on given E and s parameters and their corresponding standard error (cf. details and examples section)
`profiling_config`	list (*optional*): allows to set configuration 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 re-sampling for E) and • `s.distribution` (distribution used for the re-sampling 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)

$$\tau = 1/s exp(E/kT)$$

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

Value

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

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

`@info`

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

Function version

0.1.0

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

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

See Also

graphics::matplot, stats::rnorm, get_RLum

Examples

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

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

---

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

Description

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

Usage

```
calc_TLLxTxRatio(
  Lx.data.signal,
  Lx.data.background = 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.

`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:

$$SE_{signal} = abs(Signal_{net} * BG_f / BG_{signal})$$

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

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, Institute of Geography, Heidelberg University (Germany)
Christoph Schmidt, University of Bayreuth (Germany)

See Also

RLum.Results, analyse_SAR.TL

Examples

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

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


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

```
signal.integral.max <- 230

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

---

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

```
calc_WodaFuchs2008(data, breaks = NULL, plot = 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)`).
`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.
`...`	Further plot arguments passed to the function.

Function version

0.2.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany),
Michael Dietze, GFZ Potsdam (Germany)

References

Woda, C., Fuchs, M., 2008. On the applicability of the leading edge method to obtain equivalent doses in OSL dating and dosimetry. Radiation Measurements 43, 26-37.

See Also

calc_FuchsLang2001, calc_CentralDose

Examples

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

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

---

combine_De_Dr *Combine Dose Rate and Equivalent Dose Distribution*

Description

A Bayesian statistical analysis of OSL age requiring dose rate sample. Estimation contains a preliminary step for detecting outliers in the equivalent dose sample.

Usage

```
combine_De_Dr(
  De,
  s,
  Dr,
  int_OD,
  Age_range = c(1, 300),
  outlier_threshold = 0.05,
  outlier_method = "default",
  outlier_analysis_plot = FALSE,
  method_control = list(),
  par_local = TRUE,
  verbose = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

De	numeric (**required**): a equivalent dose sample
s	numeric (**required**): a vector of measurement errors on the equivalent dose
Dr	numeric (**required**): a dose rate sample
int_OD	numeric (**required**): the intrinsic overdispersion, typically the standard deviation characterizing a dose-recovery test distribution
Age_range	numeric (*with default*): the age range to be investigated by the algorithm, the larger the value the more iterations are needed and the longer it takes. Should not be set too narrow, cut the algorithm some slack.

`outlier_threshold`
: numeric (*with default*): the required significance level used for the outlier detection. If set to `1`, no outliers are removed. If `outlier_method = "RousseeuwCroux1993"`, the median distance is used as outlier threshold. Please see details for further information.

`outlier_method`
: character (*with default*): select the outlier detection method, either `"default"` or `"RousseeuwCroux1993"`. See details for further information.

`outlier_analysis_plot`
: logical (*with default*): enables/disables the outlier analysis plot. Note: the outlier analysis will happen with or without plot output

`method_control`
: list (*with default*): named list of further parameters passed down to the rjags::rjags modelling

`par_local`
: logical (*with default*): if set to `TRUE` the function uses its own graphics::par settings (which will end in two plots next to each other)

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

`plot`
: logical (*with default*): enable/disable plot output

`...`
: a few further arguments to fine-tune the plot output such as `cdf_ADr_quantiles` (`TRUE`/`FALSE`), `legend.pos`, `legend` (`TRUE`/`FALSE`)

Details

Outlier detection

Two different outlier detection methods are implemented (full details are given in the cited literature).

1. The *default* and recommend method, uses quantiles to compare prior and posterior distributions of the individual variances of the equivalent doses. If the corresponding quantile in the corresponding posterior distribution is larger than the quantile in the prior distribution, the value is marked as outlier (cf. Galharret et al., preprint)
2. The alternative method employs the method suggested by Rousseeuw and Croux (1993) using the absolute median distance.

Parameters available for `method_control`

The parameters listed below are used to granular control Bayesian modelling using rjags::rjags. Internally the functions `.calc_IndividualAgeModel()` and `.calc_BayesianCentraAgelModel()`. The parameter settings affect both models. Note: `method_control` expects a **named** list of parameters

PARAMETER	**TYPE**	**DEFAULT**	**REMARKS**
`variable.names_IAM`	character	`c('A', 'a', 'sig_a')`	variables names to be monitored in the modelling proces
`variable.names_BCAM`	character	`c('A', 'D_e')`	variables names to be monitored in the modelling proces
`n.chains`	integer	`4`	number of MCMC chains
`n.adapt`	integer	`1000`	number of iterations for the adaptation
`n.iter`	integer	`5000`	number of iterations to monitor cf. rjags::coda.samples
`thin`	numeric	`1`	thinning interval for the monitoring cf. rjags::coda.samp
`diag`	logical	`FALSE`	additional terminal convergence diagnostic. `FALSE` if ver
`progress.bar`	logical	`FALSE`	enable/disable progress bar. `FALSE` if `verbose = FALSE`
`quiet`	logical	`TRUE`	silence terminal output. Set to `TRUE` if `verbose = FALSE`
`return_mcmc`	logical	`FALSE`	return additional MCMC diagnostic information

Value

The function returns a plot if `plot = TRUE` and an RLum.Results object with the following slots:
`@data`
.. `$Ages`: a numeric vector with the modelled ages to be further analysed or visualised
.. `$Ages_stats`: a data.frame with sum HPD, CI 68% and CI 95% for the ages
.. `$outliers_index`: the index with the detected outliers
.. `$cdf_ADr_mean` : empirical cumulative density distribution A * Dr (mean)
.. `$cdf_ADr_quantiles` : empirical cumulative density distribution A * Dr (quantiles .025,.975)
.. `$cdf_De_no_outlier` : empirical cumulative density distribution of the De with no outliers
.. `$cdf_De_initial` : empirical cumulative density distribution of the initial De
.. `$mcmc_IAM` : the MCMC list of the Individual Age Model, only of `method_control = list(return_mcmc = TRUE)` otherwise `NULL`
.. `$mcmc_BCAM` : the MCMC list of the Bayesian Central Age Model, only of `method_control = list(return_mcmc = TRUE)` otherwise `NULL`

`@info`
.. `$call`: the original function call
.. `$model_IAM`: the BUGS model used to derive the individual age
.. `$model_BCAM`: the BUGS model used to calculate the Bayesian Central Age

Function version

0.1.0

Author(s)

Anne Philippe, Université de Nantes (France), Jean-Michel Galharret, Université de Nantes (France), Norbert Mercier, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Mercier, N., Galharret, J.-M., Tribolo, C., Kreutzer, S., Philippe, A., preprint. Luminescence age calculation through Bayesian convolution of equivalent dose and dose-rate distributions: the De_Dr model. Geochronology, 1-22.

Galharret, J-M., Philippe, A., Mercier, N., preprint. Detection of outliers with a Bayesian hierarchical model: application to the single-grain luminescence dating method. Electronic Journal of Applied Statistics

Further reading

Rousseeuw, P.J., Croux, C., 1993. Alternatives to the median absolute deviation. Journal of the American Statistical Association 88, 1273–1283. doi:10.2307/2291267

Rousseeuw, P.J., Debruyne, M., Engelen, S., Hubert, M., 2006. Robustness and outlier detection in chemometrics. Critical Reviews in Analytical Chemistry 36, 221–242. doi:10.1080/10408340600969403

See Also

plot_OSLAgeSummary, rjags::rjags, mclust::mclust-package

Examples

```
## set parameters
Dr <- stats::rlnorm (1000, 0, 0.3)
De <- 50*sample(Dr, 50, replace = TRUE)
s <- stats::rnorm(50, 10, 2)

## run modelling
## note: modify parameters for more realistic results
## Not run:
results <- combine_De_Dr(
 Dr = Dr,
 int_OD = 0.1,
 De,
 s,
 Age_range = c(0,100),
 method_control = list(
  n.iter = 100,
  n.chains = 1))

## show models used
writeLines(results@info$model_IAM)
writeLines(results@info$model_BCAM)

## End(Not run)
```

`convert_Activity2Concentration`
Convert Nuclide Activities to Abundance and Vice Versa

Description

The function performs the conversion of the specific activities into mass abundance and vice versa for the radioelements U, Th, and K 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

```
convert_Activity2Concentration(data, input_unit = "activity", verbose = TRUE)
```

Arguments

`data`	data.frame (**required**): provide dose rate data (activity or concentration) in three columns. The first column indicates the nuclide, 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 `"activity"` (considered as given Bq/kg) and `"abundance"` (considered as given in mug/g or mass. %). The default value is `"activity"`
`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 nuclide.

Constants used in this function were obtained from `https://physics.nist.gov/cuu/Constants/` all atomic weights and composition values from `https://www.nist.gov/pml/atomic-weights-and-isotopic-compo` and the nuclide data from `https://www.iaea.org/resources/databases/livechart-of-nuclides-advanced-vers`

The factors can be calculated using the equation:

$$A = N_A \frac{N_{abund}}{N_{mol.mass}} ln(2)/N.half.life$$

to convert in µg/g we further use:

$$f = A/10^6$$

where:

- `N_A` - Avogadro constant in 1/mol
- `A` - specific activity of the nuclide in Bq/kg
- `N.abund` - relative natural abundance of the isotope
- `N.mol.mass` molar mass in kg/mol
- `N.half.life` half-life of the nuclide in s

example for calculating the activity of the radionuclide U-238:

- `N_A` = 6.02214076e+23 (1/mol)
- `T_0.5` = 1.41e+17 (s)
- `m_U_238` = 0.23802891 (kg/mol)
- `U_abund` = 0.992745 (unitless)

$$A_U = N_A * U_{abund}/m_{U_238} * ln(2)/T_{1/2} = 2347046$$

(Bq/kg)

$$f.U = A_U/10^6$$

Value

Returns an RLum.Results object with a data.frame containing input and newly calculated values. Please not that in the column header µg/g is written as `mug/g` due to the R requirement to maintain packages portable using ASCII characters only.

Function version

0.1.2

Note

Although written otherwise for historical reasons. Input values must be element values. For instance, if a value is provided for U-238 the function assumes that this value represents the sum (activity or abundance) of U-238, U-235 and U-234. In other words, 1 µg/g of U means that this is the composition of 0.992 parts of U-238, 0.000054 parts of U-234, and 0.00072 parts of U-235.

Author(s)

Margret C. Fuchs, Helmholtz-Institute Freiberg for Resource Technology (Germany)

References

Debertin, K., Helmer, R.G., 1988. Gamma- and X-ray Spectrometry with Semiconductor Detectors, Elsevier Science Publishers, p.283

Wiechen, A., Ruehle, H., Vogl, K., 2013. Bestimmung der massebezogenen Aktivitaet von Radionukliden. AEQUIVAL/MASSAKT, ISSN 1865-8725, https://www.bmuv.de/fileadmin/Daten_BMU/Download_PDF/Strahlenschutz/aequival-massakt_v2013-07_bf.pdf

Examples

```
##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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

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

Examples

```
##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_Concentration2DoseRate`

Dose-rate conversion function

Description

This function converts radionuclide concentrations (K in %, Th and U in ppm) into dose rates (Gy/ka). Beta-dose rates are also attenuated for the grain size. Beta and gamma-dose rates are corrected for the water content. This function converts concentrations into dose rates (Gy/ka) and corrects for grain size attenuation and water content

Dose rate conversion factors can be chosen from Adamiec and Aitken (1998), Guerin et al. (2011), Liritzis et al. (201) and Cresswell et al. (2018). Default is Guerin et al. (2011).

Grain size correction for beta dose rates is achieved using the correction factors published by Guérin et al. (2012).

Water content correction is based on factors provided by Aitken (1985), with the factor for beta dose rate being 1.25 and for gamma 1.14.

Usage

```
convert_Concentration2DoseRate(input, conversion = "Guerinetal2011")
```

Arguments

`input`	data.frame (*optional*): a table containing all relevant information for each individual layer if nothing is provided, the function returns a template data.frame Please note that until one dataset per input is supported!
`conversion`	character (*with default*): which dose rate conversion factors to use, defaults uses Guérin et al. (2011). For accepted values see BaseDataSet.ConversionFactors

Details

The input data

COLUMN	DATA TYPE	DESCRIPTION
`Mineral`	`character`	`'FS'` for feldspar, `'Q'` for quartz
`K`	`numeric`	K nuclide content in %
`K_SE`	`numeric`	error on K nuclide content in %
`Th`	`numeric`	Th nuclide content in ppm
`Th_SE`	`numeric` error on Th nuclide content in ppm	
`U`	`numeric` U nuclide content in ppm	
`U_SE`	`numeric`	error on U nuclide content in ppm
`GrainSize`	`numeric`	average grain size in µm
`WaterContent`	`numeric`	mean water content in %
`WaterContent_SE`	`numeric`	relative error on water content

Water content The water content provided by the user should be calculated according to:

$$(Wet_weight - Dry_weight)/Dry_weight * 100$$

The unit for the weight is gram (g).

Value

The function returns an RLum.Results object for which the first element is matrix with the converted values. If no input is provided, the function returns a template data.frame that can be used as input.

Function version

0.1.0

Author(s)

Svenja Riedesel, Aberystwyth University (United Kingdom)
Martin Autzen, DTU NUTECH Center for Nuclear Technologies (Denmark)

References

Adamiec, G., Aitken, M.J., 1998. Dose-rate conversion factors: update. Ancient TL 16, 37-46.

Cresswell., A.J., Carter, J., Sanderson, D.C.W., 2018. Dose rate conversion parameters: Assessment of nuclear data. Radiation Measurements 120, 195-201.

Guerin, G., Mercier, N., Adamiec, G., 2011. Dose-rate conversion factors: update. Ancient TL, 29, 5-8.

Guerin, G., Mercier, N., Nathan, R., Adamiec, G., Lefrais, Y., 2012. On the use of the infinite matrix assumption and associated concepts: A critical review. Radiation Measurements, 47, 778-785.

Liritzis, I., Stamoulis, K., Papachristodoulou, C., Ioannides, K., 2013. A re-evaluation of radiation dose-rate conversion factors. Mediterranean Archaeology and Archaeometry 13, 1-15.

Examples

```
## create input template
input <- convert_Concentration2DoseRate()

## fill input
input$Mineral <- "FS"
input$K <- 2.13
input$K_SE <- 0.07
input$Th <- 9.76
input$Th_SE <- 0.32
input$U <- 2.24
input$U_SE <- 0.12
input$GrainSize <- 200
input$WaterContent <- 30
input$WaterContent_SE <- 5

## convert
convert_Concentration2DoseRate(input)
```

---

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

```
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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

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

Examples

```
## 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

```
convert_PSL2CSV(file, extract_raw_data = FALSE, single_table = FALSE, ...)
```

Arguments

`file` character (**required**): name of the PSL-file to be converted to CSV-files

`extract_raw_data` logical (*with default*): enable/disable raw data extraction. The PSL files imported into R contain an element `$raw_data`, which provides a few more information (e.g., count errors), sometimes it makes sense to use this data of the more compact standard values created by read_PSL2R

`single_table` logical (*with default*): enable/disable the creation of single table with n rows and n columns, instead of separate data.frame objects. Each curve will be represented by two columns for time and counts

`...` 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

0.1.2

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

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

Examples

```
## export into single data.frame
file <- system.file("extdata/DorNie_0016.psl", package="Luminescence")
convert_PSL2CSV(file, export = FALSE, single_table = TRUE)


## 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 functions 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.

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

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, Institute of Geography, Heidelberg University (Germany)

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_SG2MG` *Converts Single-Grain Data to Multiple-Grain Data*

Description

Conversion of single-grain data to multiple-grain data by adding signals from grains belonging to one disc (unique pairs of position, set and run).

Usage

```
convert_SG2MG(object, write_file = FALSE, ...)
```

Arguments

`object`	Risoe.BINfileData character (**required**): Risoe.BINfileData object or BIN/BINX-file name
`write_file`	logical (*with default*): if the input was a path to a file, the output can be written to a file if `TRUE`. The multiple grain file will be written into the same folder and with extension `-SG` to the file name.
`...`	further arguments passed down to read_BIN2R if input is file path

Value

Risoe.BINfileData object and if `write_file = TRUE` and the input was a file path, a file is written to origin folder.

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany), Norbert Mercier, IRAMAT-CRP2A, UMR 5060, CNRS-Université Bordeaux Montaigne (France);

See Also

Risoe.BINfileData, read_BIN2R, write_R2BIN

Examples

```
## simple run
## (please not that the example is not using SG data)
data(ExampleData.BINfileData, envir = environment())
convert_SG2MG(CWOSL.SAR.Data)
```

---

convert_Wavelength2Energy

Emission Spectra Conversion from Wavelength to Energy Scales (Jacobian Conversion)

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

```
convert_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 an 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):

$$\phi_E = \phi_\lambda * \lambda^2/(hc)$$

with ϕ_E the intensity per interval of energy E (1/eV), ϕ_λ the intensity per interval of wavelength λ (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

$$E = hc/\lambda$$

Value

The same object class as provided as input is returned.

Function version

0.1.1

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, Institute of Geography, Heidelberg University (Germany)

References

Blasse, G., Grabmaier, B.C., 1994. Luminescent Materials. Springer.

Mooney, J., Kambhampati, P., 2013. Get the Basics Right: Jacobian Conversion of Wavelength and Energy Scales for Quantitative Analysis of Emission Spectra. J. Phys. Chem. Lett. 4, 3316–3318. doi:10.1021/jz401508t

Mooney, J., Kambhampati, P., 2013. Correction to "Get the Basics Right: Jacobian Conversion of Wavelength and Energy Scales for Quantitative Analysis of Emission Spectra." J. Phys. Chem. Lett. 4, 3316–3318. doi:10.1021/jz401508t

Further reading

Angulo, G., Grampp, G., Rosspeintner, A., 2006. Recalling the appropriate representation of electronic spectra. Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy 65, 727–731. doi:10.1016/j.saa.2006.01.007

Wang, Y., Townsend, P.D., 2013. Potential problems in collection and data processing of luminescence signals. Journal of Luminescence 142, 202–211. doi:10.1016/j.jlumin.2013.03.052

See Also

RLum.Data.Spectrum, plot_RLum

Examples

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

##=====================##
##(2) Another example using density curves
##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*

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

```
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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

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

Examples

```
##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()

##convert
convert_XSYG2CSV(file)


## End(Not run)
```

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

Description

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

Usage

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

Arguments

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

Details

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

Internal transformation steps

(1) log(CW-OSL) values

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

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

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

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

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

(6) Transform values using

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

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

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

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

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

Note

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

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Based on comments and suggestions from:
Adrie J.J. Bos, Delft University of Technology, The Netherlands

References

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

Further Reading

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

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

See Also

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

Examples

```
##(1) - simple transformation

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

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

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

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

##load BINfile
#BINfileData<-readBIN2R("[path to BIN-file]")
data(ExampleData.BINfileData, envir = environment())

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

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

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

##combine curve to data set

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


##transform values

curve.transformed <- CW2pHMi(curve)

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


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

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

##open plot area
plot(NA, NA,
     xlim=c(0.001,10),
     ylim=c(0,8000),
     ylab="pseudo OSL (cts/0.01 s)",
     xlab="t [s]",
```

```
     log="x",
     main="Fig. 4 - Bos & Wallinga (2012)")

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

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

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

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

Description

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

Usage

```
CW2pLM(values)
```

Arguments

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

Details

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

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

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

The new count values are then calculated by

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

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

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

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

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

Further Reading

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

See Also

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

Examples

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

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

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

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

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

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

Description

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

Usage

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

Arguments

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

Details

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

Nomenclature

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

Internal transformation steps

(1) log(CW-OSL) values

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

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

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

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

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

(6) Transform values using

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

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

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`

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

Function version

0.3.1

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, Institute of Geography, Heidelberg University (Germany)

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

References

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

Further Reading

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

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

See Also

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

Examples

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

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

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

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

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


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

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

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

CW2pPMi	*Transform a CW-OSL curve into a pPM-OSL curve via interpolation under parabolic modulation conditions*

Description

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

Usage

```
CW2pPMi(values, P)
```

Arguments

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

Details

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

Nomenclature

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

Internal transformation steps

(1) log(CW-OSL) values

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

$$t' = (1/3) * (1/P^2)t^3$$

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

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

(5) Extrapolate values for t' < `min(t)` based on the previously obtained fit parameters. The extrapolation is limited to two values. Other values at the beginning of the transformed curve are set to 0.

(6) Transform values using

$$pLM(t) = t^2/P^2 * CW(t')$$

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

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

Value

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

`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

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, Institute of Geography, Heidelberg University (Germany)

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

References

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

Further Reading

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

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

See Also

CW2pLM, CW2pLMi, CW2pHMi, fit_LMCurve, RLum.Data.Curve

Examples

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

##transform values
values.transformed <- CW2pPMi(ExampleData.CW_OSL_Curve)

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

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

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

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

values.t <- CW2pLMi(values, P = 1/20)
lines(values[1:length(values.t[,1]),1],CW2pLMi(values, P = 1/20)[,2],
```

```
      col = "red",lwd = 1.3)
text(0.03,4500,"LM", col = "red", cex = .8)

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

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

ExampleData.Al2O3C *Example Al2O3:C Measurement Data*

Description

Measurement data obtained from measuring Al2O3:C chips at the IRAMAT-CRP2A, Université Bordeaux Montaigne 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.

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

Kreutzer, S., Martin, L., Guérin, G., Tribolo, C., Selva, P., Mercier, N., 2018. Environmental Dose Rate Determination Using a Passive Dosimeter: Techniques and Workflow for alpha-Al2O3:C Chips. Geochronometria 45, 56–67.

See Also

analyse_Al2O3C_ITC, analyse_Al2O3C_CrossTalk, analyse_Al2O3C_Measurement

Examples

```
##(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 the package Luminescence

---

Description

Example data from a SAR OSL and TL measurement for package Luminescence directly extracted from a Risoe BIN-file and provided in an object of type Risoe.BINfileData

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 aluminium cups on a Risø TL/OSL DA-15 reader
Reference:	unpublished

TL.SAR.Data

Lab:	Luminescence Laboratory of Cologne
Lab-Code:	LP1_5
Location:	Spain
Material:	Flint

Setup:	Risoe TL/OSL DA-20 reader (Filter: Semrock Brightline, HC475/50, N2, unpolished steel discs)
Reference:	unpublished
Remarks:	dataset limited to one position

References

CWOSL.SAR.Data: unpublished data

TL.SAR.Data: unpublished data

Examples

```
## show first 5 elements of the METADATA and DATA elements in the terminal
data(ExampleData.BINfileData, envir = environment())
CWOSL.SAR.Data@METADATA[1:5,]
CWOSL.SAR.Data@DATA[1:5]
```

`ExampleData.CobbleData` *Example data for calc_CobbleDoseRate()*

Description

An example data set for the function calc_CobbleDoseRate containing layer specific information for the cobble to be used in the function.

Format

A `data.frame`. Please see calc_CobbleDoseRate for detailed information on the structure of the data.frame.

Version

0.1.0

Examples

```
## Load data
data("ExampleData.CobbleData", envir = environment())
```

`ExampleData.CW_OSL_Curve`

Example CW-OSL curve data for the package Luminescence

Description

`data.frame` containing CW-OSL curve data (time, counts)

Format

Data frame with 1000 observations on the following 2 variables:

- `list("x")`: a numeric vector, time
- `list("y")`: a numeric vector, counts

Source

ExampleData.CW_OSL_Curve

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT607
Location:	Saxony/Germany
Material:	Middle grain quartz measured on aluminium cups on a Risø TL/OSL DA-15 reader.
Reference:	unpublished data

CW_Curve.BosWallinga2012

Lab:	Netherlands Centre for Luminescence Dating (NCL)
Lab-Code:	NCL-2108077
Location:	Guadalentin Basin, Spain
Material:	Coarse grain quartz
Reference:	Bos & Wallinga (2012) and Baartman et al. (2011)

References

Baartman, J.E.M., Veldkamp, A., Schoorl, J.M., Wallinga, J., Cammeraat, L.H., 2011. Unravelling Late Pleistocene and Holocene landscape dynamics: The Upper Guadalentin Basin, SE Spain. Geomorphology, 125, 172-185.

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

Examples

```
data(ExampleData.CW_OSL_Curve, envir = environment())
plot(ExampleData.CW_OSL_Curve)
```

`ExampleData.DeValues` *Example De data sets for the package Luminescence*

Description

Equivalent dose (De) values measured for a fine grain quartz sample from a loess section in Rottewitz (Saxony/Germany) and for a coarse grain quartz sample from a fluvial deposit in the rock shelter of Cueva Anton (Murcia/Spain).

Format

A list with two elements, each containing a two column data.frame:

`$BT998`: De and De error values for a fine grain quartz sample from a loess section in Rottewitz.

`$CA1`: Single grain De and De error values for a coarse grain quartz sample from a fluvial deposit in the rock shelter of Cueva Anton

References

BT998

Unpublished data

CA1

Burow, C., Kehl, M., Hilgers, A., Weniger, G.-C., Angelucci, D., Villaverde, V., Zapata, J. and Zilhao, J. (2015). Luminescence dating of fluvial deposits in the rock shelter of Cueva Anton, Spain. Geochronometria 52, 107-125.

BT998

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT998
Location:	Rottewitz (Saxony/Germany)
Material:	Fine grain quartz measured on aluminium discs on a Risø TL/OSL DA-15 reader
Units:	Values are given in seconds
Dose Rate:	Dose rate of the beta-source at measurement ca. 0.0438 Gy/s +/- 0.0019 Gy/s
Measurement Date:	2012-01-27

CA1

Lab:	Cologne Luminescence Laboratory (CLL)
Lab-Code:	C-L2941
Location:	Cueva Anton (Murcia/Spain)
Material:	Coarse grain quartz (200-250 microns) measured on single grain discs on a Risoe TL/OSL DA-20 r
Units:	Values are given in Gray
Measurement Date:	2012

Examples

```
##(1) plot values as histogram
data(ExampleData.DeValues, envir = environment())
plot_Histogram(ExampleData.DeValues$BT998, xlab = "De [s]")
```

```
##(2) plot values as histogram (with second to gray conversion)
data(ExampleData.DeValues, envir = environment())

De.values <- Second2Gray(ExampleData.DeValues$BT998,
                         dose.rate = c(0.0438, 0.0019))


plot_Histogram(De.values, xlab = "De [Gy]")
```

`ExampleData.Fading` *Example data for feldspar fading measurements*

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:

`$fading.data`: A named list of data.frames, each having three named columns (`LxTx, LxTx.error, timeSinceIr`
`..$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

King, G.E., Herman, F., Lambert, R., Valla, P.G., Guralnik, B., 2016. Multi-OSL-thermochronometry of feldspar. Quaternary Geochronology 33, 76-87. doi:10.1016/j.quageo.2016.01.004

Details

Lab:	University of Lausanne
Lab-Code:	UNIL/NB123
Location:	Namche Barwa (eastern Himalayas)
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

```
## 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 aluminium discs on a Risø TL/OSL DA-15 reader

References

Fuchs, M., Kreutzer, S., Fischer, M., Sauer, D., Soerensen, R., 2012. OSL and IRSL dating of raised beach sand deposits along the south-eastern coast of Norway. Quaternary Geochronology, 10, 195-200.

Examples

```
##show LM data
data(ExampleData.FittingLM, envir = environment())
plot(values.curve,log="x")
```

`ExampleData.LxTxData` *Example Lx/Tx data from CW-OSL SAR measurement*

Description

LxTx data from a SAR measurement for the package Luminescence.

Format

A `data.frame` with 4 columns (Dose, LxTx, LxTx.Error, TnTx).

Source

Lab:	Luminescence Laboratory Bayreuth
Lab-Code:	BT607
Location:	Ostrau (Saxony-Anhalt/Germany)
Material:	Middle grain (38-63 μm) quartz measured on a Risoe TL/OSL DA-15 reader.

References

unpublished data

Examples

```
## plot Lx/Tx data vs dose [s]
data(ExampleData.LxTxData, envir = environment())
plot(LxTxData$Dose,LxTxData$LxTx)
```

`ExampleData.LxTxOSLData` *Example Lx and Tx curve data from an artificial OSL measurement*

Description

`Lx` and `Tx` data of continuous wave (CW-) OSL signal curves.

Format

Two `data.frame`s containing time and count values.

Source

Arbitrary OSL measurement.

References

unpublished data

Examples

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

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

`ExampleData.MortarData` *Example equivalent dose data from mortar samples*

Description

Arbitrary data to test the function `calc_EED_Model`

Format

Two `data.frame`s containing De and De error

Source

Arbitrary measurements.

References

unpublished data

Examples

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

##plot data
plot(MortarData)
```

`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

```
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.

Format

`IRSAR.RF.Data`: IRSAR.RF.Data on coarse grain feldspar

Each object contains data needed for the given protocol analysis.

Version

0.1

Source

IRSAR.RF.Data

These data were kindly provided by Tobias Lauer and Matthias Krbetschek.

Lab:	Luminescence Laboratory TU Bergakademie Freiberg
Lab-Code:	ZEU/SA1
Location:	Zeuchfeld (Zeuchfeld Sandur; Saxony-Anhalt/Germany)
Material:	K-feldspar (130-200 μm)
Reference:	Kreutzer et al. (2014)

References

IRSAR.RF.Data

Kreutzer, S., Lauer, T., Meszner, S., Krbetschek, M.R., Faust, D., Fuchs, M., 2014. Chronology of the Quaternary profile Zeuchfeld in Saxony-Anhalt / Germany - a preliminary luminescence dating study. Zeitschrift fuer Geomorphologie 58, 5-26. doi: 10.1127/0372-8854/2012/S-00112

Examples

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

##plot data
plot_RLum(IRSAR.RF.Data)
```

`ExampleData.RLum.Data.Image` *Example data as RLum.Data.Image objects*

Description

Measurement of Princton Instruments camera imported with the function read_SPE2R to R to produce an RLum.Data.Image object.

Format

Object of class RLum.Data.Image

Version

0.1

Source

ExampleData.RLum.Data.Image

These data were kindly provided by Regina DeWitt.

Lab.:	Department of Physics, East-Carolina University, NC, USA
Lab-Code:	-

Location:	-
Material:	-
Reference:	-

Image data is a measurement of fluorescent ceiling lights with a cooled Princeton Instruments (TM) camera fitted on Risø DA-20 TL/OSL reader.

Examples

```
##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

Examples

```
## Load data
data("ExampleData.ScaleGammaDose", envir = environment())
```

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:

Element	Content
`$sample_1`	A data.frame with 3 columns (depth, intensity, error)
`$sample_2`	A data.frame with 3 columns (depth, intensity, error)
`$set_1`	A list of 4 data.frames, each representing a sample with different ages
`$set_2`	A list of 5 data.frames, each representing a sample with different ages

Details

`$sample_1`

mu	`sigmaphi`	age
0.9	5e-10	10000

`$sample_2`

mu	`sigmaphi`	age	Dose rate	D0
0.9	5e-10	10000	2.5	40

`$set_1`

mu	`sigmaphi`	ages
0.9	5e-10	1e3, 1e4, 1e5, 1e6

`$set_2`

mu	`sigmaphi`	ages	Dose rate	D0
0.9	5e-10	1e2, 1e3, 1e4, 1e5, 1e6	1.0	40

Source

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

References

Unpublished synthetic data

Examples

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



## ExampleData.SurfaceExposure$set_1
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
fit_SurfaceExposure(synth_3, age = age, sigmaphi = sigmaphi)



## ExampleData.SurfaceExposure$set_2
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
fit_SurfaceExposure(synth_4, age = age, sigmaphi = sigmaphi, D0 = D0, Ddot = 1.0)

## Not run:
ExampleData.SurfaceExposure <- list(
  sample_1 = synth_1,
  sample_2 = synth_2,
  set_1 = synth_3,
  set_2 = synth_4
)

## End(Not run)
```

`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, Germany, quartz, 90-200 µm, unpublished data).

Format

One RLum.Data.Curve dataset imported using the function read_XSYG2R

`ExampleData.TR_OSL`: A single RLum.Data.Curve object with the TR-OSL data

References

Schmidt, C., Simmank, O., Kreutzer, S., under review. Time-Resolved Optically Stimulated Luminescence of Quartz in the Nanosecond Time Domain. Journal of Luminescence, 1-90

See Also

fit_OSLLifeTimes

Examples

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

ExampleData.XSYG   *Example data for a SAR OSL measurement and a TL spectrum using a lexsyg reader*

Description

Example data from a SAR OSL measurement and a TL spectrum for package Luminescence imported from a Freiberg Instruments XSYG file using the function read_XSYG2R.

Format

`OSL.SARMeasurement`: SAR OSL measurement data

The data contain two elements: (a) `$Sequence.Header` is a data.frame with metadata from the measurement,(b) `Sequence.Object` contains an RLum.Analysis object for further analysis.

`TL.Spectrum`: TL spectrum data

RLum.Data.Spectrum object for further analysis. The spectrum was cleaned from cosmic-rays using the function

`apply_CosmicRayRemoval`. Note that no quantum efficiency calibration was performed.

Version

0.1

Source

OSL.SARMeasurement

Lab:	Luminescence Laboratory Giessen
Lab-Code:	no code
Location:	not specified
Material:	Coarse grain quartz on steel cups on lexsyg research reader
Reference:	unpublished

TL.Spectrum

Lab:	Luminescence Laboratory Giessen
Lab-Code:	BT753
Location:	Dolni Vestonice/Czech Republic
Material:	Fine grain polymineral on steel cups on lexsyg 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:

Fuchs, M., Kreutzer, S., Rousseau, D.D., Antoine, P., Hatte, C., Lagroix, F., Moine, O., Gauthier, C., Svoboda, J., Lisa, L., 2013. The loess sequence of Dolni Vestonice, Czech Republic: A new OSL-based chronology of the Last Climatic Cycle. Boreas, 42, 664–677.

See Also

read_XSYG2R, RLum.Analysis, RLum.Data.Spectrum, plot_RLum, plot_RLum.Analysis, plot_RLum.Data.Spectrum

Examples

```
##show data
data(ExampleData.XSYG, envir = environment())

## =========================================
##(1) OSL.SARMeasurement
OSL.SARMeasurement

##show $Sequence.Object
OSL.SARMeasurement$Sequence.Object

##grep OSL curves and plot the first curve
OSLcurve <- get_RLum(OSL.SARMeasurement$Sequence.Object,
recordType="OSL")[[1]]
plot_RLum(OSLcurve)

## =========================================
##(2) TL.Spectrum
TL.Spectrum

##plot simple spectrum (2D)
plot_RLum.Data.Spectrum(TL.Spectrum,
                        plot.type="contour",
                        xlim = c(310,750),
                        ylim = c(0,300),
                        bin.rows=10,
                        bin.cols = 1)

##plot 3d spectrum (uncomment for usage)
# plot_RLum.Data.Spectrum(TL.Spectrum, plot.type="persp",
# xlim = c(310,750), ylim = c(0,300), bin.rows=10,
# bin.cols = 1)
```

extdata *Collection of External Data*

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
Reference: Berger, G.W., Huntley, D.J., 1989. Test data for exponential fits. Ancient TL 7, 43-46.

»STRB87_1_bleached.txt, *STRB87_1_unbleached.txt«*

Type: Test data for exponential fits
Reference: Berger, G.W., Huntley, D.J., 1989. Test data for exponential fits. Ancient TL 7, 43-46.

»XSYG_file.xsyg

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

```
extract_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

Details

The function was written to compensate missing information in the BINX-file output of Freiberg Instruments lexsyg readers. As all information are available within the XSYG-file anyway, these information can be extracted and used for further analysis or/and to stored in a new BINX-file, which can be further used by other software, e.g., Analyst (Geoff Duller).

Typical application example: g-value estimation from fading measurements using the Analyst or any other self written script.

Beside the some simple data transformation steps the function applies the functions read_XSYG2R, read_BIN2R, write_R2BIN for data import and export.

Value

An RLum.Results object is returned with the following structure:

```
.. $irr.times (data.frame)
```

If a BINX-file path and name is set, the output will be additionally transferred into a new BINX-file with the function name as suffix. For the output the path of the input BINX-file itself is used. Note that this will not work if the input object is a file path to an XSYG-file, 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

0.3.3

Note

The function can be also used to extract irradiation times from RLum.Analysis objects previously imported via read_BIN2R (`fastForward = TRUE`) or in combination with Risoe.BINfileData2RLum.Analysis. Unfortunately the timestamp might not be very precise (or even invalid), but it allows to essentially treat different formats in a similar manner.

The produced output object contains still the irradiation steps to keep the output transparent. However, for the BINX-file export this steps are removed as the BINX-file format description does not allow irradiations as separate 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, Institute of Geography, Heidelberg University (Germany)

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

```
## (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)
```

---

extract_ROI — *Extract Pixel Values through Circular Region-of-Interests (ROI) from an Image*

Description

Light-weighted function to extract pixel values from pre-defined regions-of-interest (ROI) from RLum.Data.Image, array or matrix objects and provide simple image processing capacity. The function is limited to circular ROIs.

Usage

```
extract_ROI(object, roi, roi_summary = "mean", plot = FALSE)
```

Arguments

`object`	RLum.Data.Image, array or matrix (**required**): input image data
`roi`	matrix (**required**): matrix with three columns containing the centre coordinates of the ROI (first two columns) and the diameter of the circular ROI. All numbers must by of type integer and will forcefully coerced into such numbers using `as.integer()` regardless.

roi_summary	(**with default**): if "mean" (the default) defines what is returned in the element roi_summary; alternatively "mean", "median", "sd" or "sum" can be chosen. Pixel values are conveniently summarised using the above defined keyword.
plot	logical (*optional*): enables/disables control plot. Only the first image frame is shown

Details

The function uses a cheap approach to decide whether a pixel lies within a circle or not. It assumes that pixel coordinates are integer values and that a pixel centring within the circle is satisfied by:

$$x^2 + y^2 <= (d/2)^2$$

where x and y are integer pixel coordinates and d is the integer diameter of the circle in pixel.

Value

RLum.Results object with the following elements: ..$roi_signals: a named list with all ROI values and their coordinates ..$roi_summary: an matrix where rows are frames from the image, and columns are different ROI The element has two attributes: summary (the method used to summarise pixels) and area (the pixel area) ..$roi_coord: a matrix that can be passed to plot_ROI

If plot = TRUE a control plot is returned.

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Image

Examples

```
m <- matrix(runif(100,0,255), ncol = 10, nrow = 10)
roi <- matrix(c(2.,4,2,5,6,7,3,1,1), ncol = 3)
extract_ROI(object = m, roi = roi, plot = TRUE)
```

---

fit_CWCurve *Nonlinear Least Squares Fit for CW-OSL curves -beta version-*

Description

The function determines the weighted least-squares estimates of the component parameters of a CW-OSL signal for a given maximum number of components and returns various component parameters. The fitting procedure uses the nls function with the port algorithm.

Usage

```
fit_CWCurve(
  values,
  n.components.max,
  fit.failure_threshold = 5,
  fit.method = "port",
  fit.trace = FALSE,
  fit.calcError = FALSE,
  LED.power = 36,
  LED.wavelength = 470,
  cex.global = 0.6,
  sample_code = "Default",
  output.path,
  output.terminal = TRUE,
  output.terminalAdvanced = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

`values` RLum.Data.Curve or data.frame (**required**): x, y data of measured values (time and counts). See examples.

`n.components.max` vector (*optional*): maximum number of components that are to be used for fitting. The upper limit is 7.

`fit.failure_threshold` vector (*with default*): limits the failed fitting attempts.

`fit.method` character (*with default*): select fit method, allowed values: `'port'` and `'LM'`. `'port'` uses the 'port' routine from 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 stats::confint

`LED.power` numeric (*with default*): LED power (max.) used for intensity ramping in mW/cm^2. **Note:** The value is used for the calculation of the absolute photoionisation cross section.

`LED.wavelength` numeric (*with default*): LED wavelength used for stimulation in nm. **Note:** The value is used for the calculation of the absolute photoionisation cross section.

`cex.global` numeric (*with default*): global scaling factor.

`sample_code` character (*optional*): sample code used for the plot and the optional output table (`mtext`).

`output.path` character (*optional*): output path for table output containing the results of the fit. The file name is set automatically. If the file already exists in the directory, the values are appended.

`output.terminal` logical (*with default*): terminal output with fitting results.

`output.terminalAdvanced` logical (*with default*): enhanced terminal output. Requires `output.terminal = TRUE`. If `output.terminal = FALSE` no advanced output is possible.

`plot` logical (*with default*): returns a plot of the fitted curves.

`...` further arguments and graphical parameters passed to plot.

Details

Fitting function

The function for the CW-OSL fitting has the general form:

$$y = I0_1 * \lambda_1 * exp(-\lambda_1 * x) + , \ldots , + I0_i * \lambda_i * exp(-\lambda_i * x)$$

where $0 < i < 8$

and λ is the decay constant
and $I0$ 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 logarithmized input data set. Currently, there is no option to manually provide start parameters.

Goodness of fit

The goodness of the fit is given as pseudoR^2 value (pseudo coefficient of determination). According to Lave (1970), the value is calculated as:

$$pseudoR^2 = 1 - RSS/TSS$$

where $RSS = Residual~Sum~of~Squares$
and $TSS = Total~Sum~of~Squares$

Error of fitted component parameters

The 1-sigma error for the components is calculated using the function stats::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 details on the nonlinear regression in R, see Ritz & Streibig (2008).

Value

plot (*optional*)

the fitted CW-OSL curves are returned as plot.

table (*optional*)

an output table (`*.csv`) with parameters of the fitted components is provided if the `output.path` is set.

RLum.Results

Beside the plot and table output options, an RLum.Results object is returned.

`fit`: an `nls` object (`$fit`) for which generic R functions are provided, e.g. summary, stats::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

Note

Beta version - This function has not been properly tested yet and should therefore not be used for publication purposes!

The pseudo-R^2 may not be the best parameter to describe the goodness of the fit. The trade off between the `n.components` and the pseudo-R^2 value is currently not considered.

The function **does not** ensure that the fitting procedure has reached a global minimum rather than a local minimum!

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Boetter-Jensen, L., McKeever, S.W.S., Wintle, A.G., 2003. Optically Stimulated Luminescence Dosimetry. Elsevier Science B.V.

Lave, C.A.T., 1970. The Demand for Urban Mass Transportation. The Review of Economics and Statistics, 52 (3), 320-323.

Ritz, C. & Streibig, J.C., 2008. Nonlinear Regression with R. In: R. Gentleman, K. Hornik, G. Parmigiani, eds., Springer, p. 150.

See Also

fit_LMCurve, plot,nls, RLum.Data.Curve, RLum.Results, get_RLum, minpack.lm::nlsLM

Examples

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

##fit data
fit <- fit_CWCurve(values = ExampleData.CW_OSL_Curve,
                   main = "CW Curve Fit",
                   n.components.max = 4,
                   log = "x")
```

`fit_EmissionSpectra` *Luminescence Emission Spectra Deconvolution*

Description

Luminescence spectra deconvolution on RLum.Data.Spectrum and matrix objects on an **energy scale**. The function is optimised for emission spectra typically obtained in the context of TL, OSL and RF measurements detected between 200 and 1000 nm. The function is not prepared to deconvolve TL curves (counts against temperature; no wavelength scale). If you are interested in such analysis, please check, e.g., the package `'tgcd'`.

Usage

```
fit_EmissionSpectra(
  object,
  frame = NULL,
  n_components = NULL,
  start_parameters = NULL,
  sub_negative = 0,
  input_scale = NULL,
  method_control = list(),
  verbose = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

`object` RLum.Data.Spectrum, matrix (**required**): input object. Please note that an energy spectrum is expected

`frame` numeric (*optional*): defines the frame to be analysed

`n_components` numeric (*optional*): allows a number of the aimed number of components. However, it defines rather a maximum than than a minimum. Can be combined with other parameters.

`start_parameters` numeric (*optional*): allows to provide own start parameters for a semi-automated procedure. Parameters need to be provided in eV. Every value provided replaces a value from the automated peak finding algorithm (in ascending order).

`sub_negative` numeric (*with default*): substitute negative values in the input object by the number provided here (default: `0`). Can be set to `NULL`, i.e. negative values are kept.

`input_scale`	character (*optional*): defines whether your x-values define wavelength or energy values. For the analysis an energy scale is expected, allowed values are `'wavelength'` and `'energy'`. If nothing (`NULL`) is defined, the function tries to understand the input automatically.
`method_control`	list (*optional*): options to control the fit method, see details
`verbose`	logical (*with default*): enable/disable verbose mode
`plot`	logical (*with default*): enable/disable plot output
`...`	further arguments to be passed to control the plot output (supported: `main`, `xlab`, `ylab`, `xlim`, `ylim`, `log`, `mtext`, `legend` (`TRUE` or `FALSE`), `legend.text`, `legend.pos`)

Details

Used equation

The emission spectra (on an energy scale) can be best described as the sum of multiple Gaussian components:

,

$$y = \Sigma Ci * 1/(\sigma_i * \sqrt(2 * \pi)) * exp(-1/2 * ((x - \mu_i)/\sigma_i))^2)$$

with the parameters σ (peak width) and μ (peak centre) and C (scaling factor).

Start parameter estimation and fitting algorithm

The spectrum deconvolution consists of the following steps:

1. Peak finding
2. Start parameter estimation
3. Fitting via minpack.lm::nls.lm

The peak finding is realised by an approach (re-)suggested by Petr Pikal via the R-help mailing list (`https://stat.ethz.ch/pipermail/r-help/2005-November/thread.html`) in November 2005. This goes back to even earlier discussion in 2001 based on Prof Brian Ripley's idea. It smartly uses the functions stats::embed and max.col to identify peaks positions. For the use in this context, the algorithm has been further modified to scale on the input data resolution (cf. source code).

The start parameter estimation uses random sampling from a range of meaningful parameters and repeats the fitting until 1000 successful fits have been produced or the set `max.runs` value is exceeded.

Currently the best fit is the one with the lowest number for squared residuals, but other parameters are returned as well. If a series of curves needs to be analysed, it is recommended to make few trial runs, then fix the number of components and run at least 10,000 iterations (parameter `method_control = list(max.runs = 10000)`).

Supported `method_control` settings

Parameter	Type	Default	Description
`max.runs`	integer	`10000`	maximum allowed search iterations, if exceed the searching stops
`graining`	numeric	`15`	gives control over how coarse or fine the spectrum is split into search intervals for the

norm	logical	TRUE	normalises data to the highest count value before fitting
trace	logical	FALSE	enables/disables the tracing of the minimisation routine

Value

[NUMERICAL OUTPUT]

RLum.Results-object

slot: @data

Element	Type	Description
$data	matrix	the final fit matrix
$fit	nls	the fit object returned by minpack.lm::nls.lm
$fit_info	list	a few additional parameters that can be used to asses the quality of the fit

slot: @info

The original function call

[TERMINAL OUTPUT]

The terminal output provides brief information on the deconvolution process and the obtained results. Terminal output is only shown of the argument verbose = TRUE.

[PLOT OUTPUT]

The function returns a plot showing the raw signal with the detected components. If the fitting failed, a basic plot is returned showing the raw data and indicating the peaks detected for the start parameter estimation. The grey band in the residual plot indicates the 10% deviation from 0 (means no residual).

Function version

0.1.1

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Spectrum, RLum.Results, plot_RLum, convert_Wavelength2Energy, minpack.lm::nls.lm

Examples

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

##subtract background
TL.Spectrum@data <- TL.Spectrum@data[] - TL.Spectrum@data[,15]

results <- fit_EmissionSpectra(
 object = TL.Spectrum,
 frame = 5,
 method_control = list(max.runs = 10)
)

##deconvolution of a TL spectrum
## Not run:

##load example data

##replace 0 values
results <- fit_EmissionSpectra(
 object = TL.Spectrum,
 frame = 5, main = "TL spectrum"
)


## End(Not run)
```

`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 from 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 for intensity ramping in mW/cm^2. **Note:** This value is used for the calculation of the absolute photoionisation cross section.
`LED.wavelength`	numeric (*with default*): LED wavelength in nm used for stimulation. **Note:** This value is used for the calculation of the absolute photoionisation cross section.
`fit.trace`	logical (*with default*): traces the fitting process on the terminal.
`fit.advanced`	logical (*with default*): enables advanced fitting attempt for automatic start parameter recognition. Works only if no start parameters are provided. **Note:** It may take a while and it is not compatible with `fit.method = "LM"`.
`fit.calcError`	logical (*with default*): calculate 1-sigma error range of components using stats::confint.
`bg.subtraction`	character (*with default*): specifies method for background subtraction (`polynomial`, `linear`, `channel`, see Details). **Note:** requires input for `values.bg`.
`verbose`	logical (*with default*): terminal output with fitting results.
`plot`	logical (*with default*): returns a plot of the fitted curves.
`plot.BG`	logical (*with default*): returns a plot of the background values with the fit used for the background subtraction.
`...`	Further arguments that may be passed to the plot output, e.g. `xlab`, `xlab`, `main`, `log`.

Details

Fitting function

The function for the fitting has the general form:

$$y = (exp(0.5)*Im_1*x/xm_1)*exp(-x^2/(2*xm_1^2))+,\dots,+exp(0.5)*Im_i*x/xm_i)*exp(-x^2/(2*xm_i^2))$$

where $1 < i < 8$

This function and the equations for the conversion to b (detrapping probability) and n0 (proportional to initially trapped charge) have been taken from Kitis et al. (2008):

$$xm_i = \sqrt{max(t)/b_i}$$
$$Im_i = exp(-0.5)n0/xm_i$$

Background subtraction

Three methods for background subtraction are provided for a given background signal (`values.bg`).

- `polynomial`: default method. A polynomial function is fitted using glm and the resulting function is used for background subtraction:
$$y = a*x^4 + b*x^3 + c*x^2 + d*x + e$$
- `linear`: a linear function is fitted using glm and the resulting function is used for background subtraction:
$$y = a*x + b$$
- `channel`: the measured background signal is subtracted channel wise from the measured signal.

Start values

The choice of the initial parameters for the `nls`-fitting is a crucial point and the fitting procedure may mainly fail due to ill chosen start parameters. Here, three options are provided:

(a) If no start values (`start_values`) are provided by the user, a cheap guess is made by using the detrapping values found by Jain et al. (2003) for quartz for a maximum of 7 components. Based on these values, the pseudo start parameters `xm` and `Im` are recalculated for the given data set. In all cases, the fitting starts with the ultra-fast component and (depending on `n.components`) steps through the following values. If no fit could be achieved, an error plot (for `plot = TRUE`) with the pseudo curve (based on the pseudo start parameters) is provided. This may give the opportunity to identify appropriate start parameters visually.

(b) If start values are provided, the function works like a simple nls fitting approach.

(c) If no start parameters are provided and the option `fit.advanced = TRUE` is chosen, an advanced start parameter estimation is applied using a stochastic 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 pseudo-R^2 value (pseudo coefficient of determination). According to Lave (1970), the value is calculated as:

$$pseudoR^2 = 1 - RSS/TSS$$

where $RSS = Residual\ Sum\ of\ Squares$ and $TSS = Total\ Sum\ of\ Squares$

Error of fitted component parameters

The 1-sigma error for the components is calculated using the function stats::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)
`..` `$component_matrix` : matrix with numerical xy-values of the single fitted components with the resolution of the input data `..` `$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 `rev(time)` values
Additional columns are used for the components, two for each component, containing I0 and n0. The last columns `cont.` provide information on the relative component contribution for each time interval including the row sum for this values.

Function version

0.3.4

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, Institute of Geography, Heidelberg University (Germany)

References

Bulur, E., 1996. An Alternative Technique For Optically Stimulated Luminescence (OSL) Experiment. Radiation Measurements, 26, 5, 701-709.

Jain, M., Murray, A.S., Boetter-Jensen, L., 2003. Characterisation of blue-light stimulated luminescence components in different quartz samples: implications for dose measurement. Radiation Measurements, 37 (4-5), 441-449.

Kitis, G. & Pagonis, V., 2008. Computerized curve deconvolution analysis for LM-OSL. Radiation Measurements, 43, 737-741.

Lave, C.A.T., 1970. The Demand for Urban Mass Transportation. The Review of Economics and Statistics, 52 (3), 320-323.

Ritz, C. & Streibig, J.C., 2008. Nonlinear Regression with R. R. Gentleman, K. Hornik, & G. Parmigiani, eds., Springer, p. 150.

See Also

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

Examples

```
##(1) fit LM data without background subtraction
data(ExampleData.FittingLM, envir = environment())
fit_LMCurve(values = values.curve, n.components = 3, log = "x")

##(2) fit LM data with background subtraction and export as JPEG
## -alter file path for your preferred system
##jpeg(file = "~/Desktop/Fit_Output\%03d.jpg", quality = 100,
## height = 3000, width = 3000, res = 300)
data(ExampleData.FittingLM, envir = environment())
fit_LMCurve(values = values.curve, values.bg = values.curveBG,
            n.components = 2, log = "x", plot.BG = TRUE)
##dev.off()

##(3) fit LM data with manual start parameters
data(ExampleData.FittingLM, envir = environment())
fit_LMCurve(values = values.curve,
            values.bg = values.curveBG,
            n.components = 3,
            log = "x",
            start_values = data.frame(Im = c(170,25,400), xm = c(56,200,1500)))
```

`fit_OSLLifeTimes` *Fitting and Deconvolution of OSL Lifetime Components*

Description

Fitting and Deconvolution of OSL Lifetime Components

Usage

```
fit_OSLLifeTimes(
  object,
  tp = 0,
  signal_range = NULL,
  n.components = NULL,
  method_control = list(),
  plot = TRUE,
  plot_simple = FALSE,
  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
`plot_simple`	logical (*with default*): Enable/disable reduced plot output. If `TRUE`, no residual plot is shown, however, plot output can be combined using the standard R layout options, such as `par(mfrow = c(2,2))`.
`verbose`	logical (*with default*): Enable/disable terminal feedback
`...`	parameters passed to plot.default to control the plot output, supported are: `main`, `xlab`, `ylab`, `log`, `xlim`, `ylim`, `col`, `lty`, `legend.pos`, `legend.text`. If the input object is of type RLum.Analysis this arguments can be provided as a list.

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) Adaptation phase

In the adaptation 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:

$$\chi^2 = \sum(w * (n_i/c - \sum(A_i * exp(-x/(tau_i + t_p))))^2)$$

with $w = 1$ for unweighted regression analysis (`method_control = list(weights = FALSE)`) or $w = c^2/n_i$ for weighted regression analysis. The default values is `TRUE`.

$$F = (\Delta\chi^2/2)/(\chi^2/(N - 2 * m - 2))$$

(2) Final fitting

`method_control`

Parameter	Type	Description
`p`	numeric	controls the probability for the F statistic reference values. For a significance level of 5 %
`seed`	numeric	set the seed for the random number generator, provide a value here to get reproducible res
`DEoptim.trace`	logical	enables/disables the tracing of the differential evolution (cf. DEoptim::DEoptim.control)
`DEoptim.itermax`	logical	controls the number of the allowed generations (cf. DEoptim::DEoptim.control)
`weights`	logical	enables/disables the weighting for the start parameter estimation and fitting (see equation
`nlsLM.trace`	logical	enables/disables trace mode for the nls fitting (minpack.lm::nlsLM), can be used to identi
`nlsLM.upper`	logical	enables/disables upper parameter boundary, default is `TRUE`
`nlsLM.lower`	logical	enables/disables lower parameter boundary, default is `TRUE`

Value

`[ NUMERICAL OUTPUT ]`

`RLum.Results`-object

slot: `@data`

Element	Type	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 minpack.lm::nls.lm

slot: `@info`

The original function call

`[ TERMINAL OUTPUT ]`

Terminal output is only shown of the argument `verbose = TRUE`.

(1) Start parameter and component adaption
Trave of the parameter adaptation process

(2) Fitting results (sorted by ascending tau)
The fitting results sorted by ascending tau value. Please note that if you access the `nls` fitting object, the values are not sorted.

(3) Further information

- The photon count sum
- Durbin-Watson residual statistic to asses 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 anti-correlated

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

0.1.5

Author(s)

Sebastian Kreutzer, Geography & Earth Sciences, Aberystwyth University, Christoph Schmidt, University of Bayreuth (Germany)

References

Bluszcz, A., Adamiec, G., 2006. Application of differential evolution to fitting OSL decay curves. Radiation Measurements 41, 886-891. doi:10.1016/j.radmeas.2006.05.016

Durbin, J., Watson, G.S., 1950. Testing for Serial Correlation in Least Squares Regression: I. Biometrika 37, 409-21. doi:10.2307/2332391

Further reading

Hughes, I., Hase, T., 2010. Measurements and Their Uncertainties. Oxford University Press.

Storn, R., Price, K., 1997. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359.

See Also

minpack.lm::nls.lm, DEoptim::DEoptim

Examples

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

##fit lifetimes (short run)
fit_OSLLifeTimes(
 object = ExampleData.TR_OSL,
 n.components = 1)

##long example
## Not run:
fit_OSLLifeTimes(
object = ExampleData.TR_OSL)

## End(Not run)
```

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 equation 1 in *Sohbati et al. (2012a)* or equation 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:

```
                                 (optional)
      | depth (a.u.)| intensity | error |
      |     [ ,1]   |    [ ,2]  | [ ,3] |
      |-------------|-----------|-------|
[1, ]|    ~~~~     |    ~~~~   | ~~~~  |
[2, ]|    ~~~~     |    ~~~~   | ~~~~  |
 ... |     ...     |    ...    |  ...  |
[x, ]|    ~~~~     |    ~~~~   | ~~~~  |
```

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: Colour 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 equation 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 equation 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

Element	**Type**	**Description**
$summary	data.frame	summary of the fitting results
$data	data.frame	the original input data
$fit	nls	the fitting object produced by minpack.lm::nlsLM
$args	character	arguments of the call
$call	call	the original function call

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

Note

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

Author(s)

Christoph Burow, University of Cologne (Germany)

References

Sohbati, R., Murray, A.S., Chapot, M.S., Jain, M., Pederson, J., 2012a. Optically stimulated luminescence (OSL) as a chronometer for surface exposure dating. Journal of Geophysical Research 117, B09202. doi: doi:10.1029/2012JB009383

Sohbati, R., Jain, M., Murray, A.S., 2012b. Surface exposure dating of non-terrestrial bodies using optically stimulated luminescence: A new method. Icarus 221, 160-166.

See Also

ExampleData.SurfaceExposure, minpack.lm::nlsLM

Examples

```
## 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(
```

```
 data = 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(
 data = 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(
  data = 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(
 data = 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

```
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

$$y = (A/(1 + C * (exp(-W/(k * x))))) + c$$

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 be 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.

ARGUMENT	TYPE	DESCRIPTION
`upper`	named vector	sets upper fitting boundaries, if provided boundaries for all arguments are required, e.g.,
`lower`	names vector	sets lower fitting boundaries (see `upper` for details)
`trace`	logical	enables/disables progression trace for minpack.lm::nlsLM
`weights`	numeric	option to provide own weights for the fitting, the length of this vector needs to be equal t

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.

`[ GAPHICAL OUTPUT ]`

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Wintle, A.G., 1975. Thermal Quenching of Thermoluminescence in Quartz. Geophys. J. R. astr. Soc. 41, 107–113.

See Also

minpack.lm::nlsLM

Examples

```
##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

```
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

Author(s)

Michael Dietze, GFZ Potsdam (Germany)

Examples

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

## show structure of the default layout definition
layout.default <- get_Layout(layout = "default")
str(layout.default)

## show colour definitions for Abanico plot, only
layout.default$abanico$colour

## set Abanico plot title colour to orange
layout.default$abanico$colour$main <- "orange"

## create Abanico plot with modofied layout definition
plot_AbanicoPlot(data = ExampleData.DeValues,
                 layout = layout.default)

## create Abanico plot with predefined layout "journal"
plot_AbanicoPlot(data = ExampleData.DeValues,
                 layout = "journal")
```

`get_Quote` *Function to return essential quotes*

Description

This function returns one of the collected essential quotes in the growing library. If called without any parameters, a random quote is returned.

Usage

```
get_Quote(ID, 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

0.1.5

Author(s)

Quote credits: Michael Dietze, GFZ Potsdam (Germany), Sebastian Kreutzer, Geography & Earth Science, Aberystwyth University (United Kingdom), Dirk Mittelstraß, TU Dresden (Germany), Jakob Wallinga (Wageningen University, Netherlands)

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

Author(s)

inspired by R.G.

Examples

```
## you really want to know?
get_rightAnswer()
```

---

get_Risoe.BINfileData *General accessor function for RLum S4 class objects*

Description

Function calls object-specific get functions for RisoeBINfileData S4 class objects.

Usage

```
get_Risoe.BINfileData(object, ...)
```

Arguments

`object`	Risoe.BINfileData (**required**): S4 object of class `RLum`
`...`	further arguments that one might want to pass to the specific get function

Details

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

Value

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

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

Risoe.BINfileData

`get_RLum` *General accessors function for RLum S4 class objects*

Description

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

Usage

```
get_RLum(object, ...)

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

## 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.

Functions

- `get_RLum(list)`: Returns a list of RLum objects that had been passed to get_RLum
- ``get_RLum(`NULL`)``: Returns NULL

Function version

0.3.3

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##Example based using data and from the calc_CentralDose() function

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

##apply the central dose model 1st time
temp1 <- calc_CentralDose(ExampleData.DeValues$CA1)

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

`GitHub-API` *GitHub API*

Description

R Interface to the GitHub API v3.

Usage

```
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:

[[1]]	NUMBER
[[2]]	TITLE
[[3]]	BODY
[[4]]	CREATED
[[5]]	UPDATED
[[6]]	CREATOR
[[7]]	URL
[[8]]	STATUS

Function version

0.1.0

Author(s)

Christoph Burow, University of Cologne (Germany)

References

GitHub Developer API v3. https://docs.github.com/v3/, last accessed: 10/01/2017.

Examples

```
## 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)
```

---

import_Data *Import Luminescence Data into R*

Description

Convenience wrapper function to provide a quicker and more standardised way of reading data into R by looping through all in the package available data import functions starting with read_.

Usage

```
import_Data(file, ..., fastForward = TRUE, verbose = FALSE)
```

Arguments

file	character (**required**): file to be imported, can be a list
...	arguments to be further passed down to supported functions (please check the functions to determine the correct arguments)
fastForward	logical (*with default*): option to create RLum objects during import or a list of such objects
verbose	logical (*with default*): enable/disable verbose mode

Function version

0.1.1

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

read_BIN2R, read_XSYG2R, read_PSL2R, read_SPE2R, read_TIFF2R, read_RF2R, read_Daybreak2R

Examples

```
## import BINX/BIN
file <- system.file("extdata/BINfile_V8.binx", package = "Luminescence")
temp <- import_Data(file)

## RF data
file <- system.file("extdata", "RF_file.rf", package = "Luminescence")
temp <- import_Data(file)
```

---

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::install_github.

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

```
## 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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany) (France)

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

---

`merge_Risoe.BINfileData`

Merge Risoe.BINfileData objects or Risoe BIN-files

Description

Function allows merging Risoe BIN/BINX files or Risoe.BINfileData objects.

Usage

```
merge_Risoe.BINfileData(
  input.objects,
  output.file,
  keep.position.number = FALSE,
  position.number.append.gap = 0
)
```

Arguments

`input.objects` character 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

0.2.9

Note

The validity of the output objects is not further checked.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Duller, G.A.T., 2007. Analyst (Version 3.24) (manual). Aberystwyth University, Aberystwyth.

See Also

Risoe.BINfileData, read_BIN2R, write_R2BIN

Examples

```
##merge two objects
data(ExampleData.BINfileData, envir = environment())

object1 <- CWOSL.SAR.Data
object2 <- CWOSL.SAR.Data

object.new <- merge_Risoe.BINfileData(c(object1, object2))
```

`merge_RLum` *General merge function for RLum S4 class objects*

Description

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

Usage

```
merge_RLum(objects, ...)
```

Arguments

`objects`	list of RLum (**required**): list of S4 object of class `RLum`
`...`	further arguments that one might want to pass to the specific merge function

Details

The function provides a generalised access point for merge specific RLum objects. Depending on the input object, the corresponding merge function will be selected. Allowed arguments can be found in the documentations of each merge function. Empty list elements (NULL) are automatically removed from the input list.

object		corresponding merge function
RLum.Data.Curve	:	merge_RLum.Data.Curve
RLum.Analysis	:	merge_RLum.Analysis
RLum.Results	:	merge_RLum.Results

Value

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

Function version

0.1.3

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##Example based using data and from the calc_CentralDose() function

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

##apply the central dose model 1st time
temp1 <- calc_CentralDose(ExampleData.DeValues$CA1)

##apply the central dose model 2nd time
temp2 <- calc_CentralDose(ExampleData.DeValues$CA1)

##merge the results and store them in a new object
temp.merged <- get_RLum(merge_RLum(objects = list(temp1, temp2)))
```

`names_RLum` *S4-names function for RLum S4 class objects*

Description

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

Usage

```
names_RLum(object)

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

Arguments

`object`	RLum (**required**): S4 object of class `RLum`

Details

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

Value

Returns a character

Functions

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

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

plot_AbanicoPlot *Function to create an Abanico Plot.*

Description

A plot is produced which allows comprehensive presentation of data precision and its dispersion around a central value as well as illustration of a kernel density estimate, histogram and/or dot plot of the dose values.

Usage

```
plot_AbanicoPlot(
  data,
  na.rm = TRUE,
  log.z = TRUE,
  z.0 = "mean.weighted",
  dispersion = "qr",
  plot.ratio = 0.75,
  rotate = FALSE,
  mtext,
  summary,
  summary.pos,
  summary.method = "MCM",
  legend,
  legend.pos,
  stats,
  rug = FALSE,
  kde = TRUE,
  hist = FALSE,
  dots = FALSE,
  boxplot = FALSE,
  y.axis = TRUE,
  error.bars = FALSE,
  bar,
  bar.col,
  polygon.col,
  line,
  line.col,
  line.lty,
  line.label,
  grid.col,
  frame = 1,
  bw = "SJ",
  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 centring 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 centre the polygon.

`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 standard y-axis labels and show 0 only. Useful for data with small scatter. If you want to suppress the y-axis entirely please use `yaxt == 'n'` (the standard graphics::par setting) instead.
`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) centred 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 stretching 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.
`interactive`	logical (*with default*): create an interactive abanico plot (requires the `'plotly'` package)
`...`	Further plot arguments to pass (see graphics::plot.default). Supported are: `main`, `sub`, `ylab`, `xlab`, `zlab`, `zlim`, `ylim`, `cex`, `lty`, `lwd`, `pch`, `col`, `tck`, `tcl`, `at`, `breaks`. `xlab` must be a vector of length two, 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 thought provoking 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 displaying 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 \ 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 e.g., 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

0.1.17

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Inspired by a plot introduced by Galbraith & Green (1990)

References

Galbraith, R. & Green, P., 1990. Estimating the component ages in a finite mixture. International Journal of Radiation Applications and Instrumentation. Part D. Nuclear Tracks and Radiation Measurements, 17 (3), 197-206.

Dietze, M., Kreutzer, S., Burow, C., Fuchs, M.C., Fischer, M., Schmidt, C., 2015. The abanico plot: visualising chronometric data with individual standard errors. Quaternary Geochronology. doi:10.1016/j.quageo.2015.09.003

See Also

plot_RadialPlot, plot_KDE, plot_Histogram, plot_ViolinPlot

Examples

```
## load example data and recalculate to Gray
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <- ExampleData.DeValues$CA1

## plot the example data straightforward
plot_AbanicoPlot(data = ExampleData.DeValues)

## now with linear z-scale
plot_AbanicoPlot(data = ExampleData.DeValues,
                 log.z = FALSE)

## now with output of the plot parameters
plot1 <- plot_AbanicoPlot(data = ExampleData.DeValues,
                          output = TRUE)
str(plot1)
plot1$zlim

## now with adjusted z-scale limits
plot_AbanicoPlot(data = ExampleData.DeValues,
                 zlim = c(10, 200))

## now with adjusted x-scale limits
plot_AbanicoPlot(data = ExampleData.DeValues,
                 xlim = c(0, 20))

## now with rug to indicate individual values in KDE part
plot_AbanicoPlot(data = ExampleData.DeValues,
```

```
                 rug = TRUE)

## now with a smaller bandwidth for the KDE plot
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bw = 0.04)

## now with a histogram instead of the KDE plot
plot_AbanicoPlot(data = ExampleData.DeValues,
                 hist = TRUE,
                 kde = FALSE)

## now with a KDE plot and histogram with manual number of bins
plot_AbanicoPlot(data = ExampleData.DeValues,
                 hist = TRUE,
                 breaks = 20)

## now with a KDE plot and a dot plot
plot_AbanicoPlot(data = ExampleData.DeValues,
                 dots = TRUE)

## now with user-defined plot ratio
plot_AbanicoPlot(data = ExampleData.DeValues,
                 plot.ratio = 0.5)
## now with user-defined central value
plot_AbanicoPlot(data = ExampleData.DeValues,
                 z.0 = 70)

## now with median as central value
plot_AbanicoPlot(data = ExampleData.DeValues,
                 z.0 = "median")

## now with the 17-83 percentile range as definition of scatter
plot_AbanicoPlot(data = ExampleData.DeValues,
                 z.0 = "median",
                 dispersion = "p17")

## now with user-defined green line for minimum age model
CAM <- calc_CentralDose(ExampleData.DeValues,
                        plot = FALSE)

plot_AbanicoPlot(data = ExampleData.DeValues,
                 line = CAM,
                 line.col = "darkgreen",
                 line.label = "CAM")

## now create plot with legend, colour, different points and smaller scale
plot_AbanicoPlot(data = ExampleData.DeValues,
                 legend = "Sample 1",
                 col = "tomato4",
                 bar.col = "peachpuff",
                 pch = "R",
                 cex = 0.8)

## now without 2-sigma bar, polygon, grid lines and central value line
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bar.col = FALSE,
                 polygon.col = FALSE,
```

```
                 grid.col = FALSE,
                 y.axis = FALSE,
                 lwd = 0)

## now with direct display of De errors, without 2-sigma bar
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bar.col = FALSE,
                 ylab = "",
                 y.axis = FALSE,
                 error.bars = TRUE)

## now with user-defined axes labels
plot_AbanicoPlot(data = ExampleData.DeValues,
                 xlab = c("Data error (%)",
                          "Data precision"),
                 ylab = "Scatter",
                 zlab = "Equivalent dose [Gy]")

## now with minimum, maximum and median value indicated
plot_AbanicoPlot(data = ExampleData.DeValues,
                 stats = c("min", "max", "median"))

## now with a brief statistical summary as subheader
plot_AbanicoPlot(data = ExampleData.DeValues,
                 summary = c("n", "in.2s"))

## now with another statistical summary
plot_AbanicoPlot(data = ExampleData.DeValues,
                 summary = c("mean.weighted", "median"),
                 summary.pos = "topleft")

## now a plot with two 2-sigma bars for one data set
plot_AbanicoPlot(data = ExampleData.DeValues,
                 bar = c(30, 100))

## now the data set is split into sub-groups, one is manipulated
data.1 <- ExampleData.DeValues[1:30,]
data.2 <- ExampleData.DeValues[31:62,] * 1.3

## now a common dataset is created from the two subgroups
data.3 <- list(data.1, data.2)

## now the two data sets are plotted in one plot
plot_AbanicoPlot(data = data.3)

## now with some graphical modification
plot_AbanicoPlot(data = data.3,
                 z.0 = "median",
                 col = c("steelblue4", "orange4"),
                 bar.col = c("steelblue3", "orange3"),
                 polygon.col = c("steelblue1", "orange1"),
                 pch = c(2, 6),
                 angle = c(30, 50),
                 summary = c("n", "in.2s", "median"))

## create Abanico plot with predefined layout definition
plot_AbanicoPlot(data = ExampleData.DeValues,
```

```
                 layout = "journal")

## now with predefined layout definition and further modifications
plot_AbanicoPlot(
 data = data.3,
 z.0 = "median",
 layout = "journal",
 col = c("steelblue4", "orange4"),
 bar.col = adjustcolor(c("steelblue3", "orange3"),
                         alpha.f = 0.5),
 polygon.col = c("steelblue3", "orange3"))

## for further information on layout definitions see documentation
## of function get_Layout()

## now with manually added plot content
## create empty plot with numeric output
AP <- plot_AbanicoPlot(data = ExampleData.DeValues,
                       pch = NA,
                       output = TRUE)

## identify data in 2 sigma range
in_2sigma <- AP$data[[1]]$data.in.2s

## restore function-internal plot parameters
par(AP$par)

## add points inside 2-sigma range
points(x = AP$data[[1]]$precision[in_2sigma],
       y = AP$data[[1]]$std.estimate.plot[in_2sigma],
       pch = 16)

## add points outside 2-sigma range
points(x = AP$data[[1]]$precision[!in_2sigma],
       y = AP$data[[1]]$std.estimate.plot[!in_2sigma],
       pch = 1)
```

`plot_DetPlot` *Create De(t) plot*

Description

Plots the equivalent dose (D_e) 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,
  multicore = TRUE,
  verbose = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

`object` RLum.Analysis (**required**): input object containing data for analysis Can be provided as a list of such objects.

`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). The arguments must be provided as named list, e.g., `list(dose.points = c(0,10,20,30,0,10)` will set the regeneration dose points.

`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)

`multicore`	logical (*with default*) : enables/disables multi core calculation if `object` is a list of RLum.Analysis objects. Can be an integer specifying the number of cores
`verbose`	logical (*with default*): enables or disables terminal feedback
`plot`	logical (*with default*): enables/disables plot output Disabling the plot is useful in cases where the output need to be processed differently.
`...`	further arguments and graphical parameters passed to plot.default, analyse_SAR.CWOSL and analyse_pIRIRSequence (see details for further information). Plot control parameters are: `ylim`, `xlim`, `ylab`, `xlab`, `main`, `pch`, `mtext`, `cex`, `legend`, `legend.text`, `legend.pos`

Details

method

The original method presented by Bailey 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 D_e values
`@data`:

Object	**Type**	**Description**
`De.values`	`data.frame`	table with De values
`signal_integral.seq`	`numeric`	integral sequence used for the calculation

`@info`:

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

Function version

0.1.7

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 several hundreds of channels.

Author(s)

Sebastian Kreutzer, Institute of Geography, Ruprecht-Karl University of Heidelberg (Germany)

References

Bailey, R.M., Singarayer, J.S., Ward, S., Stokes, S., 2003. Identification of partial resetting using De as a function of illumination time. Radiation Measurements 37, 511-518. doi:10.1016/S1350-4487(03)00063-5

See Also

plot, analyse_SAR.CWOSL, analyse_pIRIRSequence

Examples

```
## Not run:
##load data
##ExampleData.BINfileData contains two BINfileData objects
##CWOSL.SAR.Data and TL.SAR.Data
data(ExampleData.BINfileData, envir = environment())

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

plot_DetPlot(
  object,
  signal.integral.min = 1,
  signal.integral.max = 3,
  background.integral.min = 900,
  background.integral.max = 1000,
  n.channels = 5)

## End(Not run)
```

---

plot_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 analyse_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
`...`	Further arguments and graphical parameters to be passed. In particular: `main`, `xlab`, `ylab`, `xlim`, `ylim`, `lty`, `lwd`, `pch`, `col.pch`, `col.lty`, `mtext`

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**

OBJECT	**TYPE**	**COMMENT**
`results`	data.frame	with dose and LxTx values
`data`	RLum.Results	original input data

Slot: **@info**

OBJECT	**TYPE**	**COMMENT**
`call`	`call`	the original function call
`args`	`list`	arguments of the original function call

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

Function version

0.2.3

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Christoph Burow, University of Cologne (Germany)

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.

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

0.1.14

Note

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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Michael Dietze, GFZ Potsdam (Germany)

References

Wintle, A.G., Murray, A.S., 2006. A review of quartz optically stimulated luminescence characteristics and their relevance in single-aliquot regeneration dating protocols. Radiation Measurements, 41, 369-391.

See Also

plot

Examples

```
## read example data set and misapply them for this plot type
data(ExampleData.DeValues, envir = environment())

## plot values
plot_DRTResults(
  values = ExampleData.DeValues$BT998[7:11,],
  given.dose = 2800,
  mtext = "Example data")

## plot values with legend
plot_DRTResults(
  values = ExampleData.DeValues$BT998[7:11,],
  given.dose = 2800,
  legend = "Test data set")

## create and plot two subsets with randomised values
x.1 <- ExampleData.DeValues$BT998[7:11,]
x.2 <- ExampleData.DeValues$BT998[7:11,] * c(runif(5, 0.9, 1.1), 1)

plot_DRTResults(
  values = list(x.1, x.2),
  given.dose = 2800)
```

```
## some more user-defined plot parameters
plot_DRTResults(
  values = list(x.1, x.2),
  given.dose = 2800,
  pch = c(2, 5),
  col = c("orange", "blue"),
  xlim = c(0, 8),
  ylim = c(0.85, 1.15),
  xlab = "Sample aliquot")

## plot the data with user-defined statistical measures as legend
plot_DRTResults(
  values = list(x.1, x.2),
  given.dose = 2800,
  summary = c("n", "weighted$mean", "sd.abs"))

## 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", "weighted$mean", "sd.abs"),
  summary.pos = "sub")

## plot the data grouped by preheat temperatures
plot_DRTResults(
  values = ExampleData.DeValues$BT998[7:11,],
  given.dose = 2800,
  preheat = c(200, 200, 200, 240, 240))

## read example data set and misapply them for this plot type
data(ExampleData.DeValues, envir = environment())

## plot values
plot_DRTResults(
  values = ExampleData.DeValues$BT998[7:11,],
  given.dose = 2800,
  mtext = "Example data")

## plot two data sets grouped by preheat temperatures
plot_DRTResults(
  values = list(x.1, x.2),
  given.dose = 2800,
  preheat = c(200, 200, 200, 240, 240))

## plot the data grouped by preheat temperatures as boxplots
plot_DRTResults(
  values = ExampleData.DeValues$BT998[7:11,],
  given.dose = 2800,
  preheat = c(200, 200, 200, 240, 240),
  boxplot = TRUE)
```

---

plot_FilterCombinations

Plot filter combinations along with the (optional) net transmission window

---

Description

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

```
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. Supported 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

$$T_{final} = T_1 * T_2$$

Optical density

$$OD = -log10(T)$$

Total optical density

$$OD_{total} = OD_1 + OD_2$$

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:

$$Transmission = Transmission^(d) * P$$

Advanced plotting parameters

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

Argument	Datatype	Description
`legend`	`logical`	enable/disable legend
`legend.pos`	`character`	change legend position (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 recommend, 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

Object	Type	Description
`net_transmission_window`	`matrix`	the resulting net transmission window
`OD_total`	`matrix`	the total optical density
`filter_matrix`	`matrix`	the filter matrix used for plotting

@info

Object	Type	Description
`call`	call	the original function call

Function version

0.3.2

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Results, approx

Examples

```
## (For legal reasons no real filter data are provided)

## Create filter sets
filter1 <- density(rnorm(100, mean = 450, sd = 20))
filter1 <- matrix(c(filter1$x, filter1$y/max(filter1$y)), ncol = 2)
filter2 <- matrix(c(200:799,rep(c(0,0.8,0),each = 200)), ncol = 2)

## Example 1 (standard)
plot_FilterCombinations(filters = list(filter1, filter2))

## Example 2 (with d and P value and name for filter 2)
results <- plot_FilterCombinations(
filters = list(filter_1 = filter1, Rectangle = list(filter2, d = 2, P = 0.6)))
results

## 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 dose-response 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

```
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. This argument is defunct and will be removed in a future version!

`mode` character (*with default*): selects calculation mode of the function.

- `"interpolation"` (default) calculates the De by interpolation,
- `"extrapolation"` calculates the equivalent dose by extrapolation (useful for MAAD measurements) and
- `"alternate"` calculates no equivalent dose 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`,
- `GOK`,
- `LambertW`

See details.

`fit.force_through_origin`
: logical (*with default*) allow to force the fitted function through the origin. For `method = "EXP+EXP"` the function will be fixed through the origin in either case, so this option will have no effect.

`fit.weights`
: logical (*with default*): option whether the fitting is done with or without weights. See details.

`fit.includingRepeatedRegPoints`
: logical (*with default*): includes repeated points for fitting (`TRUE`/`FALSE`).

`fit.NumberRegPoints`
: integer (*optional*): set number of regeneration points manually. By default the number of all (!) regeneration points is used automatically.

`fit.NumberRegPointsReal`
: integer (*optional*): if the number of regeneration points is provided manually, the value of the real, regeneration points = all points (repeated points) including reg 0, has to be inserted.

`fit.bounds`
: logical (*with default*): set lower fit bounds for all fitting parameters to 0. Limited for the use with the fit methods `EXP`, `EXP+LIN`, `EXP OR LIN`, `GOK`, `LambertW` Argument to be inserted for experimental application only!

`NumberIterations.MC`
: integer (*with default*): number of Monte Carlo simulations for error estimation. See details.

`output.plot`
: logical (*with default*): plot output (`TRUE`/`FALSE`).

`output.plotExtended`
: logical (*with default*): If' `TRUE`, 3 plots on one plot area are provided:

1. growth curve,
2. histogram from Monte Carlo error simulation and
3. a test dose response plot.

If `FALSE`, just the growth curve will be plotted. **Requires:** `output.plot = TRUE`.

`output.plotExtended.single`
: logical (*with default*): single plot output (`TRUE`/`FALSE`) to allow for plotting the results in single plot windows. Requires `output.plot = TRUE` and `output.plotExtended = TRUE`.

`cex.global`
: numeric (*with default*): global scaling factor.

`txtProgressBar`
: logical (*with default*): enables or disables `txtProgressBar`. If `verbose = FALSE` also no `txtProgressBar` is shown.

`verbose`
: logical (*with default*): enables or disables terminal feedback.

`...`
: Further arguments and graphical parameters to be passed. Note: Standard arguments will only be passed to the growth curve plot. Supported: `xlim`, `ylim`, `main`, `xlab`, `ylab`

Details

Fitting methods

For all options (except for the `LIN`, `QDR` and the `EXP OR LIN`), the 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:

$$y = mx + n$$

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

$$y = a + bx + cx^2$$

EXP: tries to fit a function of the form

$$y = a(1 - exp(-\frac{(x + c)}{b}))$$

Parameters b and c are approximated by a linear fit using lm. Note: b = D0
EXP OR LIN: works for some cases where an EXP fit fails. If the EXP fit fails, a LIN fit is done instead.

EXP+LIN: tries to fit an exponential plus linear function of the form:

$$y = a(1 - exp(-\frac{x + c}{b}) + (gx))$$

The D_e 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

$$y = (a_1(1 - exp(-\frac{x}{b_1}))) + (a_2(1 - exp(-\frac{x}{b_2})))$$

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

$$y = a(d - (1 + (\frac{1}{b})xc)^{(-1/c)})$$

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

LambertW: tries to fit a dose-response curve based on the Lambert W function according to Pagonis et al. (2020). The function has the form

$$y \ (1 + (W((R - 1) * exp(R - 1 - ((x + D_{int})/D_c)))/(1 - R))) * N$$

with W the Lambert W function, calculated using the package lamW::lambertW0, R the dimensionless retrapping ratio, N the total concentration of trappings states in cm^-3 and $D_c = N/R$ a constant. D_{int} is the offset on the x-axis. Please not that finding the root in mode = "extrapolation" is a non-easy task due to the shape of the function and the results might be unexpected.

Fit weighting

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

$$fit.weights = \frac{\frac{1}{error}}{\Sigma \frac{1}{error}}$$

Error estimation using Monte Carlo simulation

Error estimation is done using a parametric bootstrapping approach. A set of `Lx/Tx` values is constructed by randomly drawing curve data sampled from normal distributions. The normal distribution is defined by the input values (`mean = value`, `sd = value.error`). Then, a dose-response curve fit is attempted for each dataset resulting in a new distribution of single `De` values. The standard deviation of this distribution is becomes then the error of the `De`. With increasing iterations, the error value becomes more stable. However, naturally the error will not decrease with more MC runs.

Alternatively, the function returns highest probability density interval estimates as output, users may find more useful under certain circumstances.

Note: It may take some calculation time with increasing MC runs, especially for the composed functions (`EXP+LIN` and `EXP+EXP`).
Each error estimation is done with the function of the chosen fitting method.

Subtitle information

To avoid plotting the subtitle information, provide an empty user `mtext` `mtext = ""`. To plot any other subtitle text, use `mtext`.

Value

Along with a plot (so far wanted) an `RLum.Results` object is returned containing, the slot `data` contains the following elements:

DATA.OBJECT	TYPE	DESCRIPTION
`..$De :`	`data.frame`	Table with De values
`..$De.MC :`	`numeric`	Table with De values from MC runs
`..$Fit :`	nls or lm	object from the fitting for `EXP`, `EXP+LIN` and `EXP+EXP`. In case of a resulting linear fit w
`..$Formula :`	expression	Fitting formula as R expression
`..$call :`	`call`	The original function call

Function version

1.11.13

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Michael Dietze, GFZ Potsdam (Germany)

References

Berger, G.W., Huntley, D.J., 1989. Test data for exponential fits. Ancient TL 7, 43-46.

Guralnik, B., Li, B., Jain, M., Chen, R., Paris, R.B., Murray, A.S., Li, S.-H., Pagonis, P., Herman, F., 2015. Radiation-induced growth and isothermal decay of infrared-stimulated luminescence from feldspar. Radiation Measurements 81, 224-231.

Pagonis, V., Kitis, G., Chen, R., 2020. A new analytical equation for the dose response of dosimetric materials, based on the Lambert W function. Journal of Luminescence 225, 117333. doi:10.1016/j.jlumin.2020.117333

See Also

nls, RLum.Results, get_RLum, minpack.lm::nlsLM, lm, uniroot, lamW::lambertW0

Examples

```
##(1) plot growth curve for a dummy data.set and show De value
data(ExampleData.LxTxData, envir = environment())
temp <- plot_GrowthCurve(LxTxData)
get_RLum(temp)

##(1b) horizontal plot arrangement
layout(mat = matrix(c(1,1,2,3), ncol = 2))
plot_GrowthCurve(LxTxData, output.plotExtended.single = TRUE)

##(1c) 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
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:
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 standard deviation 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

0.4.5

Note

The input data is not restricted to a special type.

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

hist, plot

Examples

```
## load data
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <-
  Second2Gray(ExampleData.DeValues$BT998, dose.rate = c(0.0438,0.0019))

## plot histogram the easiest way
plot_Histogram(ExampleData.DeValues)

## plot histogram with some more modifications
plot_Histogram(ExampleData.DeValues,
               rug = TRUE,
               normal_curve = TRUE,
               cex.global = 0.9,
               pch = 2,
               colour = c("grey", "black", "blue", "green"),
               summary = c("n", "mean", "sdrel"),
               summary.pos = "topleft",
               main = "Histogram of De-values",
               mtext = "Example data set",
               ylab = c(expression(paste(D[e], " distribution")),
                        "Standard error"),
               xlim = c(100, 250),
               ylim = c(0, 0.1, 5, 20))
```

plot_KDE *Plot kernel density estimate with statistics*

Description

Plot a kernel density estimate of measurement values in combination with the actual values and associated error bars in ascending order. If enabled, the boxplot will show the usual distribution parameters (median as bold line, box delimited by the first and third quartile, whiskers defined by the extremes and outliers shown as points) and also the mean and standard deviation as pale bold line and pale polygon, respectively.

Usage

```
plot_KDE(
  data,
  na.rm = TRUE,
  values.cumulative = TRUE,
  order = TRUE,
  boxplot = TRUE,
  rug = TRUE,
  summary,
  summary.pos,
  summary.method = "MCM",
  bw = "nrd0",
  output = 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)`).
`na.rm`	logical (*with default*): exclude NA values from the data set prior to any further operation.
`values.cumulative`	logical (*with default*): show cumulative individual data.
`order`	logical: Order data in ascending order.
`boxplot`	logical (*with default*): optionally show a boxplot (depicting median as thick central line, first and third quartile as box limits, whiskers denoting +/- 1.5 interquartile ranges and dots further outliers).
`rug`	logical (*with default*): optionally add rug.
`summary`	character (*optional*): add statistic measures of centrality and dispersion to the plot. Can be one or more of several keywords. See details for available keywords.
`summary.pos`	numeric or character (*with default*): optional position coordinates or keyword (e.g. `"topright"`) for the statistical summary. Alternatively, the keyword `"sub"` may be specified to place the summary below the plot header. However, this latter option in only possible if `mtext` is not used. In case of coordinate specification, y-coordinate refers to the right y-axis.

summary.method	character (*with default*): keyword indicating the method used to calculate the statistic summary. One out of "unweighted", "weighted" and "MCM". See calc_Statistics for details.
bw	character (*with default*): bin-width, chose a numeric value for manual setting.
output	logical: Optional output of numerical plot parameters. These can be useful to reproduce similar plots. Default is TRUE.
...	further arguments and graphical parameters passed to plot.

Details

The function allows passing several plot arguments, such as `main`, `xlab`, `cex`. However, as the figure is an overlay of two separate plots, `ylim` must be specified in the order: c(ymin_axis1, ymax_axis1, ymin_axis2, ymax_axis2) when using the cumulative values plot option. See examples for some further explanations. For details on the calculation of the bin-width (parameter `bw`) see density.

A statistic summary, i.e. a collection of statistic measures of centrality and dispersion (and further measures) can be added by specifying one or more of the following keywords:

- `"n"` (number of samples)
- `"mean"` (mean De value)
- `"median"` (median of the De values)
- `"sd.rel"` (relative standard deviation in percent)
- `"sd.abs"` (absolute standard deviation)
- `"se.rel"` (relative standard error)
- `"se.abs"` (absolute standard error)
- `"in.2s"` (percent of samples in 2-sigma range)
- `"kurtosis"` (kurtosis)
- `"skewness"` (skewness)

Note that the input data for the statistic summary is sent to the function `calc_Statistics()` depending on the log-option for the z-scale. If `"log.z = TRUE"`, the summary is based on the logarithms of the input data. If `"log.z = FALSE"` the linearly scaled data is used.

Note as well, that `"calc_Statistics()"` calculates these statistic measures in three different ways: `unweighted`, `weighted` and `MCM-based` (i.e., based on Monte Carlo Methods). By default, the MCM-based version is used. If you wish to use another method, indicate this with the appropriate keyword using the argument `summary.method`.

Function version

3.6.0

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)
Geography & Earth Sciences, Aberystwyth University (United Kingdom)

See Also

density, plot

Examples

```
## read example data set
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <-
  Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))

## create plot straightforward
plot_KDE(data = ExampleData.DeValues)

## create plot with logarithmic x-axis
plot_KDE(data = ExampleData.DeValues,
         log = "x")

## create plot with user-defined labels and axes limits
plot_KDE(data = ExampleData.DeValues,
         main = "Dose distribution",
         xlab = "Dose (s)",
         ylab = c("KDE estimate", "Cumulative dose value"),
         xlim = c(100, 250),
         ylim = c(0, 0.08, 0, 30))

## create plot with boxplot option
plot_KDE(data = ExampleData.DeValues,
         boxplot = TRUE)

## create plot with statistical summary below header
plot_KDE(data = ExampleData.DeValues,
         summary = c("n", "median", "skewness", "in.2s"))

## create plot with statistical summary as legend
plot_KDE(data = ExampleData.DeValues,
         summary = c("n", "mean", "sd.rel", "se.abs"),
         summary.pos = "topleft")

## split data set into sub-groups, one is manipulated, and merge again
data.1 <- ExampleData.DeValues[1:15,]
data.2 <- ExampleData.DeValues[16:25,] * 1.3
data.3 <- list(data.1, data.2)

## create plot with two subsets straightforward
plot_KDE(data = data.3)

## create plot with two subsets and summary legend at user coordinates
plot_KDE(data = data.3,
         summary = c("n", "median", "skewness"),
         summary.pos = c(110, 0.07),
         col = c("blue", "orange"))

## example of how to use the numerical output of the function
## return plot output to draw a thicker KDE line
KDE_out <- plot_KDE(data = ExampleData.DeValues,
output = TRUE)
```

plot_NRt — *Visualise natural/regenerated signal ratios*

Description

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

Usage

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

Arguments

`data`	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 zoo::rollmean). `"spline"` applies a smoothing spline to each curve (see stats::smooth.spline)
`k`	integer (*with default*): integer width of the rolling window.
`legend`	logical (*with default*): show or hide the plot legend.
`legend.pos`	character (*with default*): keyword specifying the position of the legend (see legend).
`...`	further parameters passed to plot (also see par).

Details

This function accepts the individual curve data in many different formats. If `data` is a `list`, each element of the list must contain a two column `data.frame` or `matrix` containing the `XY` data of the curves (time and counts). Alternatively, the elements can be objects of class RLum.Data.Curve.

Input values can also be provided as a `data.frame` or `matrix` where the first column contains the time values and each following column contains the counts of each curve.

Value

Returns a plot and RLum.Analysis object.

Author(s)

Christoph Burow, University of Cologne (Germany)

References

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

See Also

plot

Examples

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

## EXAMPLE 1

## convert Risoe.BINfileData object to RLum.Analysis object
data <- Risoe.BINfileData2RLum.Analysis(object = CWOSL.SAR.Data, pos = 8, ltype = "OSL")

## extract all OSL curves
allCurves <- get_RLum(data)

## keep only the natural and regenerated signal curves
pos <- seq(1, 9, 2)
curves <- allCurves[pos]

## plot a standard NR(t) plot
plot_NRt(curves)

## re-plot with rolling mean data smoothing
plot_NRt(curves, smooth = "rmean", k = 10)

## re-plot with a logarithmic x-axis
plot_NRt(curves, log = "x", smooth = "rmean", k = 5)

## re-plot with custom axes ranges
plot_NRt(curves, smooth = "rmean", k = 5,
         xlim = c(0.1, 5), ylim = c(0.4, 1.6),
         legend.pos = "bottomleft")

## re-plot with smoothing spline on log scale
plot_NRt(curves, smooth = "spline", log = "x",
         legend.pos = "top")

## EXAMPLE 2

# you may also use this function to check whether all
# TD curves follow the same shape (making it a TnTx(t) plot).
posTD <- seq(2, 14, 2)
curves <- allCurves[posTD]

plot_NRt(curves, main = "TnTx(t) Plot",
         smooth = "rmean", k = 20,
```

```
          ylab = "TD natural / TD regenerated",
          xlim = c(0, 20), legend = FALSE)

## EXAMPLE 3

# extract data from all positions
data <- lapply(1:24, FUN = function(pos) {
   Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos = pos, ltype = "OSL")
})

# get individual curve data from each aliquot
aliquot <- lapply(data, get_RLum)

# set graphical parameters
par(mfrow = c(2, 2))

# create NR(t) plots for all aliquots
for (i in 1:length(aliquot)) {
   plot_NRt(aliquot[[i]][pos],
            main = paste0("Aliquot #", i),
            smooth = "rmean", k = 20,
            xlim = c(0, 10),
            cex = 0.6, legend.pos = "bottomleft")
}

# reset graphical parameters
par(mfrow = c(1, 1))
```

`plot_OSLAgeSummary` *Plot Posterior OSL-Age Summary*

Description

A graphical summary of the statistical inference of an OSL age

Usage

```
plot_OSLAgeSummary(object, level = 0.95, digits = 1L, verbose = TRUE, ...)
```

Arguments

`object`	RLum.Results, numeric (**required**): an object produced by combine_De_Dr. Alternatively, a numeric vector of a parameter from an MCMC process
`level`	numeric (*with default*): probability of shown credible interval
`digits`	integer (*with default*): number of digits considered for the calculation
`verbose`	logical (*with default*): enable/disable additional terminal output
`...`	further arguments to modify the plot, supported: `xlim`, `ylim`, `xlab`, `ylab`, `main`, `lwd`, `lty`, `col`, `polygon_col`, `polygon_density`, `rug`

Details

The function is called automatically by combine_De_Dr

Value

A posterior distribution plot and an RLum.Results object with the credible interval.

Function version

0.1.0

Author(s)

Anne Philippe, Université de Nantes (France), Jean-Michel Galharret, Université de Nantes (France), Norbert Mercier, IRAMAT-CRP2A, Université Bordeaux Montaigne (France), Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

combine_De_Dr, plot.default, rjags::rjags

Examples

```
##generate random data
set.seed(1234)
object <- rnorm(1000, 100, 10)
plot_OSLAgeSummary(object)
```

`plot_RadialPlot` *Function to create a Radial Plot*

Description

A Galbraith's radial plot is produced on a logarithmic or a linear scale.

Usage

```
plot_RadialPlot(
  data,
  na.rm = TRUE,
  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 centring of the z-axis.
centrality	character or numeric (*with default*): measure of centrality, used for automatically centring 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

0.5.9

Author(s)

Michael Dietze, GFZ Potsdam (Germany)
Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Based on a rewritten S script of Rex Galbraith, 2010

References

Galbraith, R.F., 1988. Graphical Display of Estimates Having Differing Standard Errors. Technometrics, 30 (3), 271-281.

Galbraith, R.F., 1990. The radial plot: Graphical assessment of spread in ages. International Journal of Radiation Applications and Instrumentation. Part D. Nuclear Tracks and Radiation Measurements, 17 (3), 207-214.

Galbraith, R. & Green, P., 1990. Estimating the component ages in a finite mixture. International Journal of Radiation Applications and Instrumentation. Part D. Nuclear Tracks and Radiation Measurements, 17 (3) 197-206.

Galbraith, R.F. & Laslett, G.M., 1993. Statistical models for mixed fission track ages. Nuclear Tracks And Radiation Measurements, 21 (4), 459-470.

Galbraith, R.F., 1994. Some Applications of Radial Plots. Journal of the American Statistical Association, 89 (428), 1232-1242.

Galbraith, R.F., 2010. On plotting OSL equivalent doses. Ancient TL, 28 (1), 1-10.

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

See Also

plot, plot_KDE, plot_Histogram, plot_AbanicoPlot

Examples

```
## load example data
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <- Second2Gray(
  ExampleData.DeValues$BT998, c(0.0438,0.0019))

## plot the example data straightforward
plot_RadialPlot(data = ExampleData.DeValues)

## now with linear z-scale
plot_RadialPlot(
  data = ExampleData.DeValues,
  log.z = FALSE)

## now with output of the plot parameters
```

```
plot1 <- plot_RadialPlot(
  data = ExampleData.DeValues,
  log.z = FALSE,
  output = TRUE)
plot1
plot1$zlim

## now with adjusted z-scale limits
plot_RadialPlot(
  data = ExampleData.DeValues,
  log.z = FALSE,
  zlim = c(100, 200))

## now the two plots with serious but seasonally changing fun
#plot_RadialPlot(data = data.3, fun = TRUE)

## now with user-defined central value, in log-scale again
plot_RadialPlot(
  data = ExampleData.DeValues,
  central.value = 150)

## now with a rug, indicating individual De values at the z-scale
plot_RadialPlot(
  data = ExampleData.DeValues,
  rug = TRUE)

## now with legend, colour, different points and smaller scale
plot_RadialPlot(
  data = ExampleData.DeValues,
  legend.text = "Sample 1",
  col = "tomato4",
  bar.col = "peachpuff",
  pch = "R",
  cex = 0.8)

## now without 2-sigma bar, y-axis, grid lines and central value line
plot_RadialPlot(
  data = ExampleData.DeValues,
  bar.col = "none",
  grid.col = "none",
  y.ticks = FALSE,
  lwd = 0)

## now with user-defined axes labels
plot_RadialPlot(
  data = ExampleData.DeValues,
  xlab = c("Data error (%)", "Data precision"),
  ylab = "Scatter",
  zlab = "Equivalent dose [Gy]")

## now with minimum, maximum and median value indicated
plot_RadialPlot(
  data = ExampleData.DeValues,
  central.value = 150,
  stats = c("min", "max", "median"))

## now with a brief statistical summary
```

```
plot_RadialPlot(
  data = ExampleData.DeValues,
  summary = c("n", "in.2s"))

## now with another statistical summary as subheader
plot_RadialPlot(
  data = ExampleData.DeValues,
  summary = c("mean.weighted", "median"),
  summary.pos = "sub")

## now the data set is split into sub-groups, one is manipulated
data.1 <- ExampleData.DeValues[1:15,]
data.2 <- ExampleData.DeValues[16:25,] * 1.3

## now a common dataset is created from the two subgroups
data.3 <- list(data.1, data.2)

## now the two data sets are plotted in one plot
plot_RadialPlot(data = data.3)

## now with some graphical modification
plot_RadialPlot(
  data = data.3,
  col = c("darkblue", "darkgreen"),
  bar.col = c("lightblue", "lightgreen"),
  pch = c(2, 6),
  summary = c("n", "in.2s"),
  summary.pos = "sub",
  legend = c("Sample 1", "Sample 2"))
```

---

plot_Risoe.BINfileData *Plot single luminescence curves from a BIN file object*

Description

Plots single luminescence curves from an object returned by the read_BIN2R function.

Usage

```
plot_Risoe.BINfileData(
  BINfileData,
  position,
  run,
  set,
  sorter = "POSITION",
  ltype = c("IRSL", "OSL", "TL", "RIR", "RBR", "RL"),
  curve.transformation,
  dose_rate,
  temp.lab,
  cex.global = 1,
  ...
)
```

Arguments

`BINfileData`	Risoe.BINfileData (**required**): requires an S4 object returned by the read_BIN2R function.
`position`	vector (*optional*): option to limit the plotted curves by position (e.g. `position = 1`, `position = c(1,3,5)`).
`run`	vector (*optional*): option to limit the plotted curves by run (e.g., `run = 1`, `run = c(1,3,5)`).
`set`	vector (*optional*): option to limit the plotted curves by set (e.g., `set = 1`, `set = c(1,3,5)`).
`sorter`	character (*with default*): the plot output can be ordered by "POSITION","SET" or "RUN". POSITION, SET and RUN are options defined in the Risoe Sequence Editor.
`ltype`	character (*with default*): option to limit the plotted curves by the type of luminescence stimulation. Allowed values: `"IRSL"`, `"OSL"`,`"TL"`, `"RIR"`, `"RBR"` (corresponds to LM-OSL), `"RL"`. All type of curves are plotted by default.
`curve.transformation`	character (*optional*): allows transforming CW-OSL and CW-IRSL curves to pseudo-LM curves via transformation functions. Allowed values are: `CW2pLM`, `CW2pLMi`, `CW2pHMi` and `CW2pPMi`. See details.
`dose_rate`	numeric (*optional*): dose rate of the 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

Note

The function has been successfully tested for the Sequence Editor file output version 3 and 4.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Michael Dietze, GFZ Potsdam (Germany)

References

Duller, G., 2007. Analyst. pp. 1-45.

See Also

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

Examples

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

##plot all curves from the first position to the desktop
#pdf(file = "~/Desktop/CurveOutput.pdf", paper = "a4", height = 11, onefile = TRUE)

##example - load from *.bin file
#BINfile<- file.choose()
#BINfileData<-read_BIN2R(BINfile)

#par(mfrow = c(4,3), oma = c(0.5,1,0.5,1))
#plot_Risoe.BINfileData(CWOSL.SAR.Data,position = 1)
#mtext(side = 4, BINfile, outer = TRUE, col = "blue", cex = .7)
#dev.off()
```

`plot_RLum` *General plot function for RLum S4 class objects*

Description

Function calls object specific plot functions for RLum S4 class objects.

Usage

```
plot_RLum(object, ...)
```

Arguments

`object`	RLum (**required**): S4 object of class `RLum`. Optional a list containing objects of class RLum can be provided. In this case the function tries to plot every object in this list according to its `RLum` class. Non-RLum objects are removed.

`...`	further arguments and graphical parameters that will be passed to the specific plot functions. The only argument that is supported directly is `main` (setting the plot title). In contrast to the normal behaviour `main` can be here provided as list and the arguments in the list will dispatched to the plots if the `object` is of type `list` as well.

Details

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

object		**corresponding plot function**
RLum.Data.Curve	:	plot_RLum.Data.Curve
RLum.Data.Spectrum	:	plot_RLum.Data.Spectrum
RLum.Data.Image	:	plot_RLum.Data.Image
RLum.Analysis	:	plot_RLum.Analysis
RLum.Results	:	plot_RLum.Results

Value

Returns a plot.

Function version

0.4.4

Note

The provided plot output depends on the input object.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

plot_RLum.Data.Curve, RLum.Data.Curve, plot_RLum.Data.Spectrum, RLum.Data.Spectrum, plot_RLum.Data.Image, RLum.Data.Image, plot_RLum.Analysis, RLum.Analysis, plot_RLum.Results, RLum.Results

Examples

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

#transform data.frame to RLum.Data.Curve object
temp <- as(ExampleData.CW_OSL_Curve, "RLum.Data.Curve")

#plot RLum object
plot_RLum(temp)
```

plot_RLum.Analysis *Plot function for an RLum.Analysis S4 class object*

Description

The function provides a standardised plot output for curve data of an RLum.Analysis S4 class object

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 considering whether it is measured with the PMT or not! However, for a fast look it might be helpful.

Usage

```
plot_RLum.Analysis(
  object,
  subset = NULL,
  nrows,
  ncols,
  abline = NULL,
  combine = FALSE,
  records_max = NULL,
  curve.transformation,
  plot.single = FALSE,
  ...
)
```

Arguments

`object`	RLum.Analysis (**required**): S4 object of class `RLum.Analysis`
`subset`	named list (*optional*): subsets elements for plotting. The arguments in the named list will be directly passed to the function get_RLum (e.g., `subset = list(curveType = "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 ab-lines to the plot. Argument are provided in a list and will be forward to the function abline, e.g., `list(v = c(10, 100))` adds two vertical lines add 10 and 100 to all plots. In contrast `list(v = c(10), v = c(100)` adds a vertical at 10 to the first and a vertical line at 100 to the 2nd plot.
`combine`	logical (*with default*): allows to combine all RLum.Data.Curve objects in one single plot.

records_max
: numeric (*optional*): limits number of records shown if `combine = TRUE`. Shown are always the first and the last curve, the other number of curves to be shown a distributed evenly, this may result in less number of curves plotted as specified. This parameter has only an effect for n > 2.

curve.transformation
: character (*optional*): allows transforming CW-OSL and CW-IRSL curves to pseudo-LM curves via transformation functions. Allowed values are: `CW2pLM`, `CW2pLMi`, `CW2pHMi` and `CW2pPMi`. See details.

plot.single
: logical (*with default*): global par settings are considered, normally this should end in one plot per page

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

  Supported arguments: `main`, `mtext`, `log`, `lwd`, `lty` `type`, `pch`, `col`, `norm` (see plot_RLum.Data.Curve), `xlim`,`ylim`, `xlab`, `ylab`, ...

  and for `combine = TRUE` also: `sub_title`, `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.

Value

Returns multiple plots.

Function version

0.3.15

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, Institute of Geography, Heidelberg University (Germany)

See Also

plot, plot_RLum, plot_RLum.Data.Curve

Examples

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

##convert values for position 1
temp <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos=1)

##(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

```
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 character (*with default*): allows curve normalisation to the highest count value ('default'). Alternatively, the function offers the modes `"max"`, `"min"` and `"huot"` for a background corrected normalisation, see details.
`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.

Curve normalisation

The argument `norm` normalises all count values. To date the following options are supported:

`norm = TRUE` or `norm = "max"`: Curve values are normalised to the highest count value in the curve

`norm = "last"`: Curves values are normalised to the last count value (this can be useful in particular for radiofluorescence curves)

`norm = "huot"`: Curve values are normalised as suggested by Sébastien Huot via GitHub:

$$y = (observed - median(background))/(max(observed) - median(background))$$

The background of the curve is defined as the last 20% of the count values of a curve.

Value

Returns a plot.

Function version

0.2.6

Note

Not all arguments of plot will be passed!

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

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 very basic plot functionality for image data of an RLum.Data.Image object. For more sophisticated plotting it is recommended to use other very powerful packages for image processing.

Details on the plot functions

Supported plot types:

`plot.type = "plot.raster"`

Uses the standard plot function of R graphics::image. If wanted, the image is enhanced, using the argument `stretch`. Possible values are `hist`, `lin`, and `NULL`. The latter does nothing. The argument `useRaster = TRUE` is used by default, but can be set to `FALSE`.

`plot.type = "contour"`

This uses the function graphics::contour

Usage

```
plot_RLum.Data.Image(
  object,
  frames = NULL,
  par.local = TRUE,
  plot.type = "plot.raster",
  ...
)
```

Arguments

`object`	RLum.Data.Image (**required**): S4 object of class `RLum.Data.Image`
`frames`	numeric (*optional*): sets the frames to be set, by default all frames are plotted. Can be sequence of numbers, as long as the frame number is valid.
`par.local`	logical (*with default*): use local graphical parameters for plotting, e.g. the plot is shown in one column and one row. If `par.local = FALSE` global parameters are inherited.
`plot.type`	character (*with default*): plot types. Supported types are `plot.raster`, `contour`
`...`	further arguments and graphical parameters that will be passed to the specific plot functions. Standard supported parameters are `xlim`, `ylim`, `zlim`, `xlab`, `ylab`, `main`, `legend` (`TRUE` or `FALSE`), `col`, `cex`, `axes` (`TRUE` or `FALSE`), `zlim_image` (adjust the z-scale over different images), `stretch`

Value

Returns a plot

Function version

0.2.1

Note

The axes limitations (`xlim`, `zlim`, `zlim`) work directly on the object, so that regardless of the chosen limits the image parameters can be adjusted for best visibility. However, in particular for z-scale limitations this is not always wanted, please use `zlim_image` to maintain a particular value range over a series of images.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Image, plot, plot_RLum, graphics::image, graphics::contour

Examples

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

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

---

plot_RLum.Data.Spectrum

Plot function for an RLum.Data.Spectrum S4 class object

Description

The function provides a standardised plot output for spectrum data of an RLum.Data.Spectrum class object. The purpose of this function is to provide easy and straight-forward spectra plotting, not provide a full customised access to all plot parameters. If this is wanted, standard R plot functionality should be used instead.

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 graphics::persp:

- shade: default is 0.4
- phi: default is 15
- theta: default is -30
- expand: default is 1
- axes: default is TRUE
- box: default is TRUE; accepts "alternate" for a custom plot design
- 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.

-frames: pick the frames to be plotted (depends on the binning!). Check without this setting before plotting.

plot.type = "multiple.lines"

All frames plotted in one frame.

-frames: pick the frames to be plotted (depends on the binning!). Check without this setting before plotting.

'**plot.type = "image" or ‘plot.type = "contour" **

These plot types use the R functions graphics::image or graphics::contour. The advantage is that many plots can be arranged conveniently using standard R plot functionality. If plot.type = "image" a contour is added by default, which can be disabled using the argument contour = FALSE to add own contour lines of choice.

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, box, zlim, main, mtext, pch, type ("single", "multiple.lines", "interactive"), col, border, lwd, bty, showscale ("interactive", "image") contour, contour.col ("image")

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,
  plot = TRUE,
  ...
)
```

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 row names and col names 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 `image`, `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"`

plot
: logical (*with default*): enables/disables plot output. If the plot output is disabled, the matrix used for the plotting and the calculated colour values (as attributes) are returned. This way, the (binned, transformed etc.) output can be used in other functions and packages, such as plotting with the package `'plot3D'`

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

Value

Returns a plot and the transformed `matrix` used for plotting with some useful attributes such as the `colour` and `pmat` (the transpose matrix from graphics::persp)

Function version

0.6.8

Note

Not all additional arguments (`...`) will be passed similarly!

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Spectrum, convert_Wavelength2Energy, plot, plot_RLum, graphics::persp, plotly::plot_ly, graphics::contour, graphics::image

Examples

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

##(1)plot simple spectrum (2D) - image
plot_RLum.Data.Spectrum(
 TL.Spectrum,
 plot.type="image",
 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),
bin.rows=10,
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",
 showscale = TRUE)

 ##(6) interactive plot using the package plotly ("heatmap")
 plot_RLum.Data.Spectrum(TL.Spectrum, plot.type="interactive",
 xlim = c(310,750), ylim = c(0,300), bin.rows=10,
 bin.cols = 1,
 type = "heatmap",
 showscale = TRUE)


## End(Not run)
```

---

plot_RLum.Results *Plot function for an RLum.Results S4 class object*

Description

The function provides a standardised plot output for data of an RLum.Results S4 class object

Usage

```
plot_RLum.Results(object, single = TRUE, ...)
```

Arguments

object	RLum.Results (**required**): S4 object of class `RLum.Results`
single	logical (*with default*): single plot output (`TRUE/FALSE`) to allow for plotting the results in as few plot windows as possible.
...	further arguments and graphical parameters will be passed to the `plot` function.

Details

The function produces a multiple plot output. A file output is recommended (e.g., pdf).

Value

Returns multiple plots.

Function version

0.2.1

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, Institute of Geography, Heidelberg University (Germany)

See Also

plot, plot_RLum

Examples

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

# apply the un-logged minimum age model
mam <- calc_MinDose(data = ExampleData.DeValues$CA1, sigmab = 0.2, log = TRUE, plot = FALSE)

##plot
plot_RLum.Results(mam)

# estimate the number of grains on an aliquot
grains<- calc_AliquotSize(grain.size = c(100,150), sample.diameter = 1, plot = FALSE, MC.iter = 100)

##plot
plot_RLum.Results(grains)
```

plot_ROI *Create Regions of Interest (ROI) Graphic*

Description

Create ROI graphic with data extracted from the data imported via read_RF2R. This function is used internally by analyse_IRSAR.RF but might be of use to work with reduced data from spatially resolved measurements. The plot dimensions mimic the original image dimensions

Usage

```
plot_ROI(
  object,
  exclude_ROI = c(1),
  dist_thre = -Inf,
  dim.CCD = NULL,
  bg_image = NULL,
  plot = TRUE,
  ...
)
```

Arguments

`object`	RLum.Analysis, RLum.Results or a list of such objects (**required**): data input. Please note that to avoid function errors, only input created by the functions read_RF2R or extract_ROI is accepted
`exclude_ROI`	numeric (*with default*): option to remove particular ROIs from the analysis. Those ROIs are plotted but not coloured and not taken into account in distance analysis. `NULL` excludes nothing.
`dist_thre`	numeric (*optional*): euclidean distance threshold in pixel distance. All ROI for which the euclidean distance is smaller are marked. This helps to identify ROIs that might be affected by signal cross-talk. Note: the distance is calculated from the centre of an ROI, e.g., the threshold should include consider the ROIs or grain radius.
`dim.CCD`	numeric (*optional*): metric x and y for the recorded (chip) surface in µm. For instance `c(8192,8192)`, if set additional x and y-axes are shown
`bg_image`	RLum.Data.Image (*optional*): background image object please note that the dimensions are not checked.
`plot`	logical (*with default*): enable or disable plot output to use the function only to extract the ROI data
`...`	further parameters to manipulate the plot. On top of all arguments of graphics::plot.default the following arguments are supported: `lwd.ROI`, `lty.ROI`, `col.ROI`, `col.pixel`, `text.labels`, `text.offset`, `grid` (`TRUE/FALSE`), `legend` (`TRUE/FALSE`), `legend.text`, `legend.pos`

Value

An ROI plot and an RLum.Results object with a matrix containing the extracted ROI data and a object produced by stats::dist containing the euclidean distance between the ROIs.

Function version

0.2.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

read_RF2R, analyse_IRSAR.RF

Examples

```
## simple example
file <- system.file("extdata", "RF_file.rf", package = "Luminescence")
temp <- read_RF2R(file)
plot_ROI(temp)

## in combination with extract_ROI()
m <- matrix(runif(100,0,255), ncol = 10, nrow = 10)
roi <- matrix(c(2.,4,2,5,6,7,3,1,1), ncol = 3)
t <- extract_ROI(object = m, roi = roi)
plot_ROI(t, bg_image = m)
```

---

plot_ViolinPlot *Create a violin plot*

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 to small to be visualised. The idea for this plot is based on the 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).

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'`

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

Function version

0.1.4

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, Institute of Geography, Heidelberg University (Germany)

References

Daniel Adler (2005). vioplot: A violin plot is a combination of a box plot and a kernel density plot. R package version 0.2 http://CRAN.R-project.org/package=violplot

Hintze, J.L., Nelson, R.D., 1998. A Box Plot-Density Trace Synergism. The American Statistician 52, 181-184.

Raphael W. Majeed (2012). violinmplot: Combination of violin plot with mean and standard deviation. R package version 0.2.1. http://CRAN.R-project.org/package=violinmplot

Wickham. H (2009). ggplot2: elegant graphics for data analysis. Springer New York.

See Also

stats::density, plot, boxplot, rug, calc_Statistics

Examples

```
## read example data set
data(ExampleData.DeValues, envir = environment())
ExampleData.DeValues <- Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))

## create plot straightforward
plot_ViolinPlot(data = ExampleData.DeValues)
```

PSL2Risoe.BINfileData *Convert portable OSL data to a Risoe.BINfileData object*

Description

Converts an `RLum.Analysis` object produced by the function `read_PSL2R()` to a `Risoe.BINfileData` object (**BETA**).

Usage

```
PSL2Risoe.BINfileData(object, ...)
```

Arguments

`object`	RLum.Analysis (**required**): `RLum.Analysis` object produced by read_PSL2R
`...`	currently not used.

Details

This function converts an RLum.Analysis object that was produced by the read_PSL2R function to a 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

Author(s)

Christoph Burow, University of Cologne (Germany)

See Also

RLum.Analysis, RLum.Data.Curve, Risoe.BINfileData

Examples

```
# (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:
write_R2BIN(bin, "~/portableOSL.binx")

## End(Not run)
```

---

read_BIN2R *Import Risø BIN/BINX-files into R*

Description

Import a `*.bin` or a `*.binx` file produced by a Risø DA15 and DA20 TL/OSL reader into R.

Usage

```
read_BIN2R(
  file,
  show.raw.values = FALSE,
  position = NULL,
  n.records = NULL,
  zero_data.rm = TRUE,
  duplicated.rm = FALSE,
  fastForward = FALSE,
  show.record.number = FALSE,
  txtProgressBar = TRUE,
  forced.VersionNumber = NULL,
  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`
: numeric (*optional*): limits the number of imported records to the provided record id (e.g., `n.records = 1:10` imports the first ten records, while `n.records = 3` imports only record number 3. 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`.

`show.record.number`
: logical (*with default*): shows record number of the imported record, for debugging usage only. Can be provided as `list` if `file` is a `list`.

`txtProgressBar`
: logical (*with default*): enables or disables txtProgressBar.

`forced.VersionNumber`
: integer (*optional*): allows to cheat the version number check in the function by own values for cases where the BIN-file version is not supported. Can be provided as `list` if `file` is a `list`.

  Note: The usage is at own risk, only supported BIN-file versions have been tested.

`ignore.RECTYPE`
: logical or numeric (*with default*): this argument allows to ignore values in the byte 'RECTYPE' (BIN-file version 08), in case there are not documented or faulty set. In this case the corrupted records are skipped. If the setting is numeric (e.g., `ignore.RECTYPE = 128`), records of those type are ignored for import.

`pattern`
: character (*optional*): argument that is used if only a path is provided. The argument will than be passed to the function list.files used internally to construct a `list` of wanted files

`verbose`
: logical (*with default*): enables or disables verbose mode

`...`
: further arguments that will be passed to the function Risoe.BINfileData2RLum.Analysis. Please note that any matching argument automatically sets `fastForward = TRUE`

Details

The binary data file is parsed byte by byte following the data structure published in the Appendices of the Analyst manual p. 42.

For the general BIN/BINX-file structure, the reader is referred to the Risø website: https://www.fysik.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

0.17.3

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.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Margret C. Fuchs, HZDR Freiberg, (Germany)
based on information provided by Torben Lapp and Karsten Bracht Nielsen (Risø DTU, Denmark)

References

DTU Nutech, 2016. The Sequence Editor, Users Manual, February, 2016. https://www.fysik.dtu.dk

See Also

write_R2BIN, Risoe.BINfileData, base::readBin, merge_Risoe.BINfileData, RLum.Analysis utils::txtProgressBar, list.files

Examples

```
file <- system.file("extdata/BINfile_V8.binx", package = "Luminescence")
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

```
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.
`...`	not in use, for compatibility reasons only

Value

A list of RLum.Analysis objects (each per position) is provided.

Function version

0.3.2

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, Institute of Geography, Heidelberg University (Germany)
Antoine Zink, C2RMF, Palais du Louvre, Paris (France)

The ASCII-file import is based on a suggestion by Willian Amidon and Andrew Louis Gorin

See Also

RLum.Analysis, RLum.Data.Curve, data.table::data.table

Examples

```
## Not run:
file <- system.file("extdata/Daybreak_TestFile.txt", package = "Luminescence")
temp <- read_Daybreak2R(file)

## End(Not run)
```

read_HeliosOSL2R *Import Luminescence Data from Helios Luminescence Reader*

Description

Straightforward import of files with the ending `.osl` produced by the zero rad Helios luminescence reader and conversion to RLum.Analysis objects.

Usage

```
read_HeliosOSL2R(file, verbose = TRUE, ...)
```

Arguments

`file`	character (**required**): path to file to be imported. Can be a list for further processing
`verbose`	logical: enable/disable terminal feedback
`...`	not in use, for compatibility reasons only

Value

RLum.Analysis object

Function version

0.1.0

Note

Thanks to Krzysztof Maternicki for providing example data.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Analysis

Examples

```
file <- system.file("extdata/HeliosOSL_Example.osl", package = "Luminescence")
read_HeliosOSL2R(file)
```

read_PSL2R *Import PSL files to R*

Description

Imports PSL files produced by a SUERC portable OSL reader into R.

Usage

```
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 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 (measurement data and sequence). 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

0.1.1

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), Sebastian Kreutzer, Institut of Geography, Heidelberg University (Germany)

See Also

RLum.Analysis, RLum.Data.Curve, RLum.Data.Curve

Examples

```
# (1) Import PSL file to R

file <- 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_RF2R` *Import RF-files to R*

Description

Import files produced by the IR-RF 'ImageJ' macro (`SR-RF.ijm`; Mittelstraß and Kreutzer, 2021) into R and create a list of RLum.Analysis objects

Usage

```
read_RF2R(file, ...)
```

Arguments

`file`	character (**required**): path and file name of the RF file. Alternatively a list of file names can be provided.
`...`	not used, only for compatible reasons

Details

The results of spatially resolved IR-RF data are summarised in so-called RF-files ((Mittelstraß and Kreutzer, 2021). This functions provides an easy import to process the data seamlessly with the R package 'Luminescence'. The output of the function can be passed to the function analyse_IRSAR.RF

Value

Returns an S4 RLum.Analysis object containing RLum.Data.Curve objects for each curve.

Function version

0.1.1

Author(s)

Sebastian Kreutzer, Geography & Earth Science, Aberystwyth University (United Kingdom)

References

Mittelstraß, D., Kreutzer, S., 2021. Spatially resolved infrared radiofluorescence: single-grain K-feldspar dating using CCD imaging. Geochronology 3, 299–319. doi:10.5194/gchron32992021

See Also

RLum.Analysis, RLum.Data.Curve, analyse_IRSAR.RF

Examples

```
##Import
file <- system.file("extdata", "RF_file.rf", package = "Luminescence")
temp <- read_RF2R(file)
```

`read_SPE2R` *Import Princeton Instruments (TM) SPE-file into R*

Description

Function imports Princeton Instruments (TM) SPE-files into R environment and provides RLum.Data.Image 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
`...`	not used, for compatibility reasons only

Details

Function provides an R only import routine for the Princeton Instruments SPE format. Import functionality is based on the file format description provided by Princeton 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 performance reasons the import is aborted for files containing more than 100 frames. This limitation can be overwritten manually by using the argument `frame.range`.

`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

0.1.5

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, Institute of Geography, Heidelberg University (Germany)

References

Princeton Instruments, 2014. Princeton Instruments SPE 3.0 File Format Specification, Version 1.A (for document URL please use an internet search machine)

Hall, C., 2012: readSPE.m. `https://www.mathworks.com/matlabcentral/fileexchange/35940-readspe`

See Also

readBin, RLum.Data.Spectrum

Examples

```
## to run examples uncomment lines and run the code

##(1) Import data as RLum.Data.Spectrum object
#file <- file.choose()
#temp <- read_SPE2R(file)
#temp

##(2) Import data as RLum.Data.Image object
#file <- file.choose()
#temp <- read_SPE2R(file, output.object = "RLum.Data.Image")
#temp

##(3) Import data as matrix object
#file <- file.choose()
#temp <- read_SPE2R(file, output.object = "matrix")
#temp

##(4) Export raw data to csv, if temp is a RLum.Data.Spectrum object
# write.table(x = get_RLum(temp),
#             file = "[your path and filename]",
#             sep = ";", row.names = FALSE)
```

`read_TIFF2R` *Import TIFF Image Data into R*

Description

Simple wrapper around tiff::readTIFF to import TIFF images and TIFF image stacks to be further processed within the package 'Luminescence'

Usage

```
read_TIFF2R(file, ...)
```

Arguments

`file`	character (**required**): file name
`...`	not in use, for compatibility reasons only

Value

RLum.Data.Image object

Function version

0.1.2

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

tiff::readTIFF, RLum.Data.Image

Examples

```
## Not run:
file <- file.choose()
image <- read_TIFF2R(file)


## End(Not run)
```

---

read_XSYG2R *Import XSYG files to R*

Description

Imports XSYG-files produced by a Freiberg Instruments 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?>
<Sample>
  <Sequence>
    <Record>
      <Curve name="first curve" />
     <Curve name="curve with data">x0 , y0 ; x1 , y1 ; x2 , y2 ; x3 , y3</Curve>
    </Record>
  </Sequence>
</Sample>
```

So far, each XSYG file can only contain one `<Sample></Sample>`, but multiple sequences.

Each record may comprise several curves.

TL curve recalculation

On the FI lexsyg device TL curves are recorded as time against count values. Temperature values are monitored on the heating plate and stored in a separate curve (time vs. temperature). If the option `recalculate.TL.curves = TRUE` is chosen, the time values for each TL curve are replaced by temperature values.

Practically, this means combining two matrices (Time vs. Counts and Time vs. Temperature) with different row numbers by their time values. Three cases are considered:

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:

- 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

0.6.12

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, Institute of Geography, Heidelberg University (Germany)

References

Grehl, S., Kreutzer, S., Hoehne, M., 2013. Documentation of the XSYG file format. Unpublished Technical Note. Freiberg, Germany

Further reading

XML: https://en.wikipedia.org/wiki/XML

See Also

`'XML'`, RLum.Analysis, RLum.Data.Curve, approx

Examples

```
##(1) import XSYG file to R (uncomment for usage)

#FILE <- file.choose()
#temp <- read_XSYG2R(FILE)

##(2) additional examples for pure XML import using the package XML
##    (uncomment for usage)

  ##import entire XML file
  #FILE <- file.choose()
  #temp <- XML::xmlRoot(XML::xmlTreeParse(FILE))

  ##search for specific subnodes with curves containing 'OSL'
  #getNodeSet(temp, "//Sample/Sequence/Record[@recordType = 'OSL']/Curve")

##(2) How to extract single curves ... after import
data(ExampleData.XSYG, envir = environment())

##grep one OSL curves and plot the first curve
OSLcurve <- get_RLum(OSL.SARMeasurement$Sequence.Object, recordType="OSL")[[1]]

##(3) How to see the structure of an object?
structure_RLum(OSL.SARMeasurement$Sequence.Object)
```

---

replicate_RLum *General replication function for RLum S4 class objects*

Description

Function replicates RLum S4 class objects and returns a list for this objects

Usage

```
replicate_RLum(object, times = NULL)
```

Arguments

object	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

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum

---

report_RLum *Create a HTML-report for (RLum) objects*

Description

Create a HTML-report for (RLum) 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 suppress 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

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.

The HTML report is created with rmarkdown::render and has the following structure:

Section	**Description**
`Header`	A summary of general characteristics of the object
`Object content`	A comprehensive list of the complete structure and content of the provided object.
`Object structure`	Summary of the objects structure given as a table
`File`	Information on the saved RDS file
`Session Info`	Captured output from `sessionInfo()`
`Plots`	(*optional*) For `RLum-class` objects a variable number of plots

The structure of the report can be controlled individually by providing one or more of the following arguments (all `logical`):

Argument	**Description**
`header`	Hide or show general information on the object
`main`	Hide or show the object's content
`structure`	Hide or show object's structure
`rds`	Hide or show information on the saved RDS file
`session`	Hide or show the session info
`plot`	Hide or show the plots (depending on object)

Note that these arguments have higher precedence than `compact`.

Further options that can be provided via the `...` argument:

Argument	**Description**
`short_table`	If `TRUE` only show the first and last 5 rows of long tables.
`theme`	Specifies the Bootstrap theme to use for the report. Valid themes include `"default"`, `"cerulean"`, `"jour`
`highlight`	Specifies the syntax highlighting style. Supported styles include `"default"`, `"tango"`, `"pygments"`, `"kat`
`css`	`TRUE` or `FALSE` to enable/disable custom CSS styling

The following arguments can be used to customise the report via CSS (Cascading Style Sheets):

Argument	**Description**
`font_family`	Define the font family of the HTML document (default: `"arial"`)
`headings_size`	Size of the <h1> to <h6> tags used to define HTML headings (default: 166%).
`content_color`	Colour 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

0.1.5

Note

This function requires the R packages 'rmarkdown', 'pander' and 'rstudioapi'.

Author(s)

Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

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 chosen 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.

Value

Returns an RLum.Analysis object.

Function version

0.4.3

Note

The `protocol` argument of the RLum.Analysis object is set to 'unknown' if not stated otherwise.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

Risoe.BINfileData, RLum.Analysis, read_BIN2R

Examples

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

##convert values for position 1
Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos = 1)
```

`RLum-class` *Class* `"RLum"`

Description

Abstract class for data in the package Luminescence Subclasses are:

Usage

```
## S4 method for signature 'RLum'
replicate_RLum(object, times = NULL)
```

Arguments

`object`	RLum (**required**): an object of class RLum
`times`	integer (*optional*): number for times each element is repeated element

Details

RLum-class
|
|—–-RLum.Data
|—–-|– RLum.Data.Curve
|—–-|– RLum.Data.Spectrum
|—–-|– RLum.Data.Image
|—–-RLum.Analysis
|—–-RLum.Results

Methods (by generic)

- `replicate_RLum(RLum)`: Replication method RLum-objects

Slots

`originator` Object of class character containing the name of the producing function for the object. Set automatically by using the function set_RLum.

`info` Object of class list for additional information on the object itself

`.uid` Object of class character for a unique object identifier. This id is usually calculated using the internal function `create_UID()` if the function set_RLum is called.

`.pid` Object of class character for a parent id. This allows nesting RLum-objects at will. The parent id can be the uid of another object.

Objects from the Class

A virtual Class: No objects can be created from it.

Class version

0.4.0

Note

`RLum` is a virtual class.

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data, RLum.Data.Curve, RLum.Data.Spectrum, RLum.Data.Image, RLum.Analysis, RLum.Results, methods_RLum

Examples

```
showClass("RLum")
```

`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("Cresswelletal2018", "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
- `Th_se` (numeric): error on the Th content
- `U` (numeric): U nuclide content in ppm
- `U_se` (numeric): error on the U content
- `water_content` (numeric): water content of each layer in %
- `water_content_se` (numeric): error on the water content
- `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:

- `"Cresswelletal2018"` (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`).

`...` Further parameters passed to barplot.

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 %), thorium (in ppm) and uranium (in ppm). Columns 10 and 11 give information on the water content and its uncertainty (standard error) in %. The layer density (in g/cm3) should be given in column 12. No cell 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:

- `"Cresswelletal2018"` (Cresswell et al., 2018)
- `"Liritzisetal2013"` (Liritzis et al., 2013)
- `"Guerinetal2011"` (Guerin et al., 2011)
- `"AdamiecAitken1998"` (Adamiec and Aitken, 1998)

Water content

The water content provided by the user should be calculated according to:

$$(Wetweight[g] - Dryweight[g])/Dryweight[g] * 100$$

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`

Element	Type	Description
`$summary`	`data.frame`	summary of the model results
`$data`	`data.frame`	the original input data
`$dose_rates`	`list`	two `data.frames` for the scaled and infinite matrix dose rates
`$tables`	`list`	several `data.frames` containing intermediate results
`$args`	`character`	arguments of the call
`$call`	`call`	the original function call

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

0.1.2

Acknowledgements

We thank Dr Ian Bailiff for the provision of an excel spreadsheet, which has been very helpful when writing this function.

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.

References

Aitken, M.J., 1985. Thermoluminescence Dating. Academic Press, London.

Adamiec, G., Aitken, M.J., 1998. Dose-rate conversion factors: update. Ancient TL 16, 37-46.

Cresswell., A.J., Carter, J., Sanderson, D.C.W., 2018. Dose rate conversion parameters: Assessment of nuclear data. Radiation Measurements 120, 195-201.

Guerin, G., Mercier, N., Adamiec, G., 2011. Dose-rate conversion factors: update. Ancient TL, 29, 5-8.

Liritzis, I., Stamoulis, K., Papachristodoulou, C., Ioannides, K., 2013. A re-evaluation of radiation dose-rate conversion factors. Mediterranean Archaeology and Archaeometry 13, 1-15.

See Also

ExampleData.ScaleGammaDose, BaseDataSet.ConversionFactors, 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 = "Cresswelletal2018",
                           fractional_gamma_dose = "Aitken1985",
                           verbose = TRUE,
                           plot = TRUE)

get_RLum(results)
```

Second2Gray — *Converting equivalent dose values from seconds (s) to Gray (Gy)*

Description

Conversion of absorbed radiation dose in seconds (s) to the SI unit Gray (Gy) including error propagation. Normally used for equivalent dose data.

Usage

```
Second2Gray(data, dose.rate, error.propagation = "omit")
```

Arguments

`data`	data.frame (**required**): input values, structure: data (`values[,1]`) and data error (`values [,2]`) are required
`dose.rate`	RLum.Results, 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)

$$De[Gy] = De[s] * DoseRate[Gy/s])$$

Provided calculation error propagation methods for error calculation (with `'se'` as the standard error and `'DR'` of the dose rate of the beta-source):

(1) `omit` (default)

$$se(De)[Gy] = se(De)[s] * DR[Gy/s]$$

In this case the standard error of the dose rate of the beta-source is treated as systematic (i.e. non-random), it error propagation is omitted. However, the error must be considered during calculation of the final age. (cf. Aitken, 1985, pp. 242). This approach can be seen as method (2) (gaussian) for the case the (random) standard error of the beta-source calibration is 0. Which particular method is requested depends on the situation and cannot be prescriptive.

(2) `gaussian` error propagation

$$se(De)[Gy] = \sqrt((DR[Gy/s] * se(De)[s])^2 + (De[s] * se(DR)[Gy/s])^2)$$

Applicable under the assumption that errors of `De` and `se` are uncorrelated.

(3) `absolute` error propagation

$$se(De)[Gy] = abs(DR[Gy/s] * se(De)[s]) + abs(De[s] * se(DR)[Gy/s])$$

Applicable under the assumption that errors of `De` and `se` are correlated.

Value

Returns a data.frame with converted values.

Function version

0.6.0

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, Institute of Geography, Heidelberg University (Germany)
Michael Dietze, GFZ Potsdam (Germany)
Margret C. Fuchs, HZDR, Helmholtz-Institute Freiberg for Resource Technology (Germany)

References

Aitken, M.J., 1985. Thermoluminescence dating. Academic Press.

See Also

calc_SourceDoseRate

Examples

```
##(A) for known source dose rate at date of measurement
## - load De data from the example data help file
data(ExampleData.DeValues, envir = environment())
## - convert De(s) to De(Gy)
Second2Gray(ExampleData.DeValues$BT998, c(0.0438,0.0019))





##(B) for source dose rate calibration data
## - calculate source dose rate first
dose.rate <-  calc_SourceDoseRate(measurement.date = "2012-01-27",
                                  calib.date = "2014-12-19",
                                  calib.dose.rate = 0.0438,
                                  calib.error = 0.0019)
# read example data
data(ExampleData.DeValues, envir = environment())

# apply dose.rate to convert De(s) to De(Gy)
Second2Gray(ExampleData.DeValues$BT998, dose.rate)
```

set_Risoe.BINfileData *General accessor function for RLum S4 class objects*

Description

Function calls object-specific get functions for RisoeBINfileData S4 class objects.

Usage

```
set_Risoe.BINfileData(
  METADATA = data.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.

Function version

0.1

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

Risoe.BINfileData

set_RLum *General set function for RLum S4 class objects*

Description

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

Usage

```
set_RLum(class, originator, .uid = create_UID(), .pid = NA_character_, ...)
```

Arguments

class	RLum (**required**): name of the S4 class to create
originator	character (*automatic*): contains the name of the calling function (the function that produces this object); can be set manually.
.uid	character (*automatic*): sets an unique ID for this object using the internal C++ function `create_UID`.
.pid	character (*with default*): option to provide a parent id for nesting at will.
...	further arguments that one might want to pass to the specific set method

Details

The function provides a generalised access point for specific RLum objects.
Depending on the given class, the corresponding method to create an object from this class will be selected. Allowed additional arguments can be found in the documentations of the corresponding RLum class:

- RLum.Data.Curve,
- RLum.Data.Image,
- RLum.Data.Spectrum,
- RLum.Analysis,
- RLum.Results

Value

Returns an object of the specified class.

Function version

0.3.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

```
##produce empty objects from each class
set_RLum(class = "RLum.Data.Curve")
set_RLum(class = "RLum.Data.Spectrum")
set_RLum(class = "RLum.Data.Spectrum")
set_RLum(class = "RLum.Analysis")
set_RLum(class = "RLum.Results")

##produce a curve object with arbitrary curve values
object <- set_RLum(
class = "RLum.Data.Curve",
curveType = "arbitrary",
recordType = "OSL",
data = matrix(c(1:100,exp(-c(1:100))),ncol = 2))

##plot this curve object
plot_RLum(object)
```

---

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

Functions

- `smooth_RLum(list)`: Returns a list of RLum objects that had been passed to smooth_RLum

Function version

0.1.0

Note

Currently only `RLum` objects of class `RLum.Data.Curve` and `RLum.Analysis` (with curve data) are supported!

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Analysis

Examples

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

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

##plot data without and with smoothing
plot_RLum(curve)
plot_RLum(smooth_RLum(curve))
```

`sTeve` *sTeve - sophisticated tool for efficient data validation and evaluation*

Description

This function provides a sophisticated routine for comprehensive luminescence dating data analysis.

Usage

```
sTeve(n_frames = 10, t_animation = 2, n.tree = 7, type)
```

Arguments

`n_frames`	integer (*with default*): n frames
`t_animation`	integer (*with default*): t animation
`n.tree`	integer (*with default*): how many trees do you want to cut?
`type`	integer (*optional*): Make a decision: 1, 2 or 3

Details

This amazing sophisticated function validates your data seriously.

Value

Validates your data.

Note

This function should not be taken too seriously.

Author(s)

R Luminescence Team, 2012-2046

See Also

plot_KDE

Examples

```
##no example available
```

`structure_RLum` *General structure function for RLum S4 class objects*

Description

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

Usage

```
structure_RLum(object, ...)

## 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.

Functions

- `structure_RLum(list)`: Returns a list of RLum objects that had been passed to structure_RLum

Function version

0.2.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data.Curve, RLum.Data.Image, RLum.Data.Spectrum, RLum.Analysis, RLum.Results

Examples

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

##show structure
structure_RLum(OSL.SARMeasurement$Sequence.Object)
```

---

subset_SingleGrainData

Simple Subsetting of Single Grain Data from Risø BIN/BINX files

---

Description

Most measured single grains do not exhibit light and it makes usually sense to subset single grain datasets using a table of position and grain pairs

Usage

```
subset_SingleGrainData(object, selection)
```

Arguments

`object`	Risoe.BINfileData (**required**): input object with the data to subset
`selection`	data.frame (**required**): selection table with two columns for position (1st column) and grain (2nd column) (columns names do not matter)

Value

A subset Risoe.BINfileData object

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

Risoe.BINfileData, read_BIN2R, verify_SingleGrainData

Examples

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

## set POSITION/GRAIN pair dataset
selection <- data.frame(POSITION = c(1,5,7), GRAIN = c(0,0,0))

##subset
subset_SingleGrainData(object = CWOSL.SAR.Data, selection = selection)
```

---

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

```
template_DRAC(nrow = 1L, 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

Value

A list.

Author(s)

Christoph Burow, University of Cologne (Germany), Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

References

Durcan, J.A., King, G.E., Duller, G.A.T., 2015. DRAC: Dose Rate and Age Calculator for trapped charge dating. Quaternary Geochronology 28, 54-61. doi:10.1016/j.quageo.2015.03.012

See Also

as.data.frame, list

Examples

```
# create a new DRAC input input
input <- template_DRAC(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"
input$`Grain size min (microns)` <- 90
input$`Grain size max (microns)` <- 125
input$`Water content ((wet weight - dry weight)/dry weight) %` <- 5
input$`errWater content %` <- 2
input$`Depth (m)` <- 2.2
input$`errDepth (m)` <- 0.22
input$`Overburden density (g cm-3)` <- 1.8
input$`errOverburden density (g cm-3)` <- 0.1
input$`Latitude (decimal degrees)` <- 30.0000
input$`Longitude (decimal degrees)` <- 70.0000
input$`Altitude (m)` <- 150
input$`De (Gy)` <- 20
input$`errDe (Gy)` <- 0.2
```

```
# use DRAC
## Not run:
output <- use_DRAC(input)

## End(Not run)
```

trim_RLum.Data — *Trim Channels of RLum.Data-class Objects*

Description

Trim off the number of channels of RLum.Data objects of similar record type on the time domain. This function is useful in cases where objects have different lengths (short/longer measurement time) but should be analysed jointly by other functions.

Usage

```
trim_RLum.Data(object, recordType = NULL, trim_range = NULL)
```

Arguments

`object`	RLum.Data RLum.Analysis (**required**): input object, can be a list of objects. Please note that in the latter case the function works only isolated on each element of the list.
`recordType`	character (*optional*): type of the record where the trim should be applied. If not set, the types are determined automatically and applied for each record type classes. Can be provided as list.
`trim_range`	numeric (*optional*): sets the trim range (everything within the range + 1 is kept). If nothing is set all curves are trimmed to a similar maximum length. Can be provided as list.

Details

The function has two modes of operation:

1. Single RLum.Data objects or a list of such objects The function is applied separately over each object.
2. Multiple curves via RLum.Analysis or a list of such objects In this mode, the function first determines the minimum number of channels for each category of records and then jointly processes them. For instance, the object contains one TL curve with 100 channels and two OSL curves with 100 and 99 channels, respectively. Than the minimum for TL would be set to 100 channels and 99 for the OSL curves. If no further parameters are applied, the function will shorten all OSL curves to 99 channels, but leave the TL curve untouched.

Value

A trimmed object or list of such objects similar to the input objects

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

RLum.Data, RLum.Analysis

Examples

```
## trim all TL curves in the object to channels 10 to 20
data(ExampleData.BINfileData, envir = environment())
temp <- Risoe.BINfileData2RLum.Analysis(CWOSL.SAR.Data, pos = 1)

c <- trim_RLum.Data(
object = temp,
recordType = "TL",
trim_range = c(10,20))

plot_RLum.Analysis(
object = c,
combine = TRUE,
subset = list(recordType = "TL"))

## simulate a situation where one OSL curve
## in the dataset has only 999 channels instead of 1000
## all curves should be limited to 999
temp@records[[2]]@data <- temp@records[[2]]@data[-nrow(temp[[2]]@data),]

c <- trim_RLum.Data(object = temp)
nrow(c@records[[4]]@data)
```

---

tune_Data — *Tune data for experimental purpose*

Description

The error can be reduced and sample size increased for specific purpose.

Usage

```
tune_Data(data, decrease.error = 0, increase.data = 0)
```

Arguments

`data`	data.frame (**required**): input values, structure: data (`values[,1]`) and data error (`values [,2]`) are required
`decrease.error`	numeric: factor by which the error is decreased, ranges between 0 and 1.
`increase.data`	numeric: factor by which the error is decreased, ranges between 0 and `Inf`.

Value

Returns a data.frame with tuned values.

Function version

0.5.0

Note

You should not use this function to improve your poor data set!

Author(s)

Michael Dietze, GFZ Potsdam (Germany)

Examples

```
## load example data set
data(ExampleData.DeValues, envir = environment())
x <- ExampleData.DeValues$CA1

## plot original data
plot_AbanicoPlot(data = x,
                 summary = c("n", "mean"))

## decrease error by 10 %
plot_AbanicoPlot(data = tune_Data(x, decrease.error = 0.1),
                 summary = c("n", "mean"))

## increase sample size by 200 %
#plot_AbanicoPlot(data = tune_Data(x, increase.data = 2) ,
#                 summary = c("n", "mean"))
```

`use_DRAC` *Use DRAC to calculate dose rate data*

Description

The function provides an interface from R to DRAC. An R-object or a pre-formatted XLS/XLSX file is passed to the DRAC website and the results are re-imported into R.

Usage

```
use_DRAC(file, name, print_references = TRUE, citation_style = "text", ...)
```

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`:

`$highlights`	data.frame	summary of 25 most important input/output fields
`$header`	character	HTTP header from the DRAC server response
`$labels`	data.frame	descriptive headers of all input/output fields
`$content`	data.frame	complete DRAC input/output table
`$input`	data.frame	DRAC input table
`$output`	data.frame	DRAC output table
`references`	list	A list of bib entries 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

0.14

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)
Michael Dietze, GFZ Potsdam (Germany)
Christoph Burow, University of Cologne (Germany)

References

Durcan, J.A., King, G.E., Duller, G.A.T., 2015. DRAC: Dose Rate and Age Calculator for trapped charge dating. Quaternary Geochronology 28, 54-61. doi:10.1016/j.quageo.2015.03.012

Examples

```
## (1) Method using the DRAC spreadsheet

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

# 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$`Latitude (decimal degrees)` <- 30.0000
input$`Longitude (decimal degrees)` <- 70.0000
input$`Altitude (m)` <- 150
input$`De (Gy)` <- 20
input$`errDe (Gy)` <- 0.2

# use DRAC
## Not run:
output <- use_DRAC(input)

## End(Not run)
```

`verify_SingleGrainData` *Verify single grain data sets and check for invalid grains, i.e. zero-light level grains*

Description

This function tries to identify automatically zero-light level curves (grains) from single grain data measurements.

Usage

```
verify_SingleGrainData(
  object,
  threshold = 10,
  cleanup = FALSE,
  cleanup_level = "aliquot",
  verbose = TRUE,
  plot = FALSE,
  ...
)
```

Arguments

`object`	Risoe.BINfileData or RLum.Analysis (**required**): input object. The function also accepts a list with objects of allowed type.
`threshold`	numeric (*with default*): numeric threshold value for the allowed difference between the `mean` and the `var` of the count values (see details)
`cleanup`	logical (*with default*): if set to `TRUE` curves 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 clean-up 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
`...`	further parameters to control the plot output; if selected. Supported arguments `main`, `ylim`

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:

$$abs(E(x) - Var(x)) >= \Theta$$

With Θ 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`

Element	**Type**	**Description**
`$unique_pairs`	`data.frame`	the unique position and grain pairs
`$selection_id`	`numeric`	the selection as record ID
`$selection_full`	`data.frame`	implemented models used in the baSAR-model core

slot:`****@info`

The original function call

Output variation

For `cleanup = TRUE` the same object as the input is returned, but cleaned up (invalid curves were removed). This means: Either an Risoe.BINfileData or an RLum.Analysis object is returned in such cases. An Risoe.BINfileData object can be exported to a BIN-file by using the function write_R2BIN.

Function version

0.2.3

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, Institute of Geography, Heidelberg University (Germany)

See Also

Risoe.BINfileData, RLum.Analysis, write_R2BIN, read_BIN2R

Examples

```
##01 - basic example I
##just show how to apply the function
data(ExampleData.XSYG, envir = environment())

##verify and get data.frame out of it
```

```
verify_SingleGrainData(OSL.SARMeasurement$Sequence.Object)$selection_full

##02 - basic example II
data(ExampleData.BINfileData, envir = environment())
id <- verify_SingleGrainData(object = CWOSL.SAR.Data,
cleanup_level = "aliquot")$selection_id

## Not run:
##03 - advanced example I
##importing and exporting a BIN-file

##select and import file
file <- file.choose()
object <- read_BIN2R(file)

##remove invalid aliquots(!)
object <- verify_SingleGrainData(object, cleanup = TRUE)

##export to new BIN-file
write_R2BIN(object, paste0(dirname(file),"/", basename(file), "_CLEANED.BIN"))

## End(Not run)
```

`write_R2BIN` *Export Risoe.BINfileData into Risø BIN/BINX-file*

Description

Exports a `Risoe.BINfileData` object in a `*.bin` or `*.binx` file that can be opened by the Analyst software or other Risø software.

Usage

```
write_R2BIN(
  object,
  file,
  version,
  compatibility.mode = FALSE,
  txtProgressBar = TRUE
)
```

Arguments

`object` Risoe.BINfileData (**required**): input object to be stored in a bin file.

`file` character (**required**): file name and path of the output file

- [WIN]: `write_R2BIN(object, "C:/Desktop/test.bin")`
- [MAC/LINUX]: `write_R2BIN("/User/test/Desktop/test.bin")`

`version` character (*optional*): version number for the output file. If no value is provided, the highest version number from the Risoe.BINfileData is taken automatically.

Note: This argument can be used to convert BIN-file versions.

`compatibility.mode`
: logical (*with default*): this option recalculates the position values if necessary and set the max. value to 48. The old position number is appended as comment (e.g., 'OP: 70). This option accounts for potential compatibility problems with the Analyst software. It further limits the maximum number of points per curve to 9,999. If a curve contains more data the curve data get 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`, `DTYPE` and `LIGHTSOURCE` are not of type character, no transformation into numeric values is done.

Value

Write a binary file.

Function version

0.5.2

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 ones directly produced by the Risø 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, Institute of Geography, Heidelberg University (Germany)

References

DTU Nutech, 2016. The Sequence Editor, Users Manual, February, 2016. `https://www.fysik.dtu.dk`

See Also

read_BIN2R, Risoe.BINfileData, writeBin

Examples

```
##load example dataset
file <- system.file("extdata/BINfile_V8.binx", package = "Luminescence")
temp <- read_BIN2R(file)

##create temporary file path
##(for usage replace by own path)
temp_file <- tempfile(pattern = "output", fileext = ".binx")

##export to temporary file path
write_R2BIN(temp, file = temp_file)
```

---

write_R2TIFF — *Export RLum.Data.Image and RLum.Data.Spectrum objects to TIFF Images*

Description

Simple wrapper around tiff::writeTIFF to export suitable RLum-class objects to TIFF images. Per default 16-bit TIFF files are exported.

Usage

```
write_R2TIFF(object, file = tempfile(), norm = 65535, ...)
```

Arguments

`object`	RLum.Data.Image or RLum.Data.Spectrum object (**required**): input object, can be a list of such objects
`file`	character (**required**): the file name and path
`norm`	numeric (*with default*): normalisation values. Values in TIFF files must range between 0-1, however, usually in imaging applications the pixel values are real integer count values. The normalisation to the to the highest 16-bit integer values -1 ensures that the numerical values are retained in the exported image. If `1` nothing is normalised.
`...`	further arguments to be passed to tiff::writeTIFF.

Value

A TIFF file

Function version

0.1.0

Author(s)

Sebastian Kreutzer, Institute of Geography, Heidelberg University (Germany)

See Also

tiff::writeTIFF, RLum.Data.Image, RLum.Data.Spectrum

Examples

```
data(ExampleData.RLum.Data.Image, envir = environment())
write_R2TIFF(ExampleData.RLum.Data.Image, file = tempfile())
```

`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

```
write_RLum2CSV(
  object,
  path = NULL,
  prefix = "",
  export = TRUE,
  compact = TRUE,
  ...
)
```

Arguments

`object`	RLum or a list of `RLum` objects (**required**): objects to be written. Can be a data.frame if needed internally.
`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 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

0.2.2

Author(s)

Sebastian Kreutzer, Geography & Earth Science, Aberystwyth University (United Kingdom)

See Also

RLum.Analysis, RLum.Data, RLum.Results, utils::write.table

Examples

```
##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)


## End(Not run)
```

Index