RLumModel - Fitting model parameters to experimental data

Johannes Friedrich

2016-03-07

Contents

1 Introduction **1**

2 Preparing **1**

3 Fitting TL data **2**

3.1 Sensitivity Analysis 2

3.1.1 Global Sensitivity Analysis 2

3.1.2 Local Sensitivity Analysis 3

3.2 Fit model parameters to experimental TL data 5

3.3 Plot results 6

3.4 Fitting OSL data 7

1 Introduction

This vignette gives an overview how to fit quartz luminescence model parameters to experimental data, TL and OSL, respectively. There will be a sensitivity analysis of single parameters or the whole parameter set as well as a fit for model parameters to experimental data.

The sensitivity analysis as well as the fitting procedure is done with the R-package FME by Karline Soetaert and Thomas Petzoldt (https://cran.r-project.org/web/packages/FME/index.html). For further information the reader is referred to the FME manual and the FME vignettes.

2 Preparing

Before starting any analysis the data had to be prepared to fit to the FME and RLumModel package. For creating a sequence or choosing a model the reader is preferred to read the RLumModel vignette "RLumModel - Getting started with RLumModel".

A new feature from RLumModel version $\geq$ 1.1 is the function "fit_RLumModel2data()". The result from this function is the main function for all sensitivity analysis (local and global). The main arguments of the "fit_RLumModel2data()" function are a sequence to be simulated, a quartz luminescence model and the sequence step in "sequence" to be analysed. Note that only sequence steps, which produce a signal output, are allowed. This are: TL, OSL, LM-OSL and RF.

3 Fitting TL data

First the global and local sensitivity of single parameters and then the procedure to fit model parameters to experimental TL data will be shown.

3.1 Sensitivity Analysis

The following example shows a TL simulation from 20 °C to 400 °C with heating rate 5 °C/s following an irradiation step at 20 °C, a dose of 5 Gy and a doserate of 0.5 Gy/s. The step to be analysed is the last TL step from 0 °C to 400°C with 5 °C/s and therefor the third step in the created sequence. As quartz luminescence model the parameter from Pagonis et al. 2007 are chosen. For sensitivity analysis the function "func_FME" will be created with all mentioned arguments.

```
sequence <-  list(
    TL = c(20, 450, 5),
    IRR = c(20, 5, 0.5),
    TL = c(0, 450, 5))

model <- "Pagonis2007"

func_FME <- fit_RLumModel2data(
  sequence = sequence,
  model = model,
  seq.step2fit = 3)
```

3.1.1 Global Sensitivity Analysis

In global sensitivity analysis, certain parameters are changed over a large range, and the effect on certain model ouput variables assessed. A few preparations are necessary for use of the FME function "sensRange":

- create parameters, which are able to fit to function "sensRange". This is done via RLumModel function "extract_pars2FME". The user has to choose only the used luminescence model.
- create a data.frame to specify the minimum and maximum values of parameters
- name the rowes of this data.frame like the used parameters

```
parms <- extract_pars2FME(model = model)


parRanges <- data.frame(min = c(5.1e7,  1e5, 1e9,  2.5e6,  5e8,   3e6,   1e8,  1e6,  5e7),
                        max = c(5.1e11, 1e9, 1e13, 2.5e10, 5e12,  3e10,  1e12, 1e10, 5e11))

rownames(parRanges) <- c("N1","N2","N3","N4","N5","N6","N7","N8","N9")
```

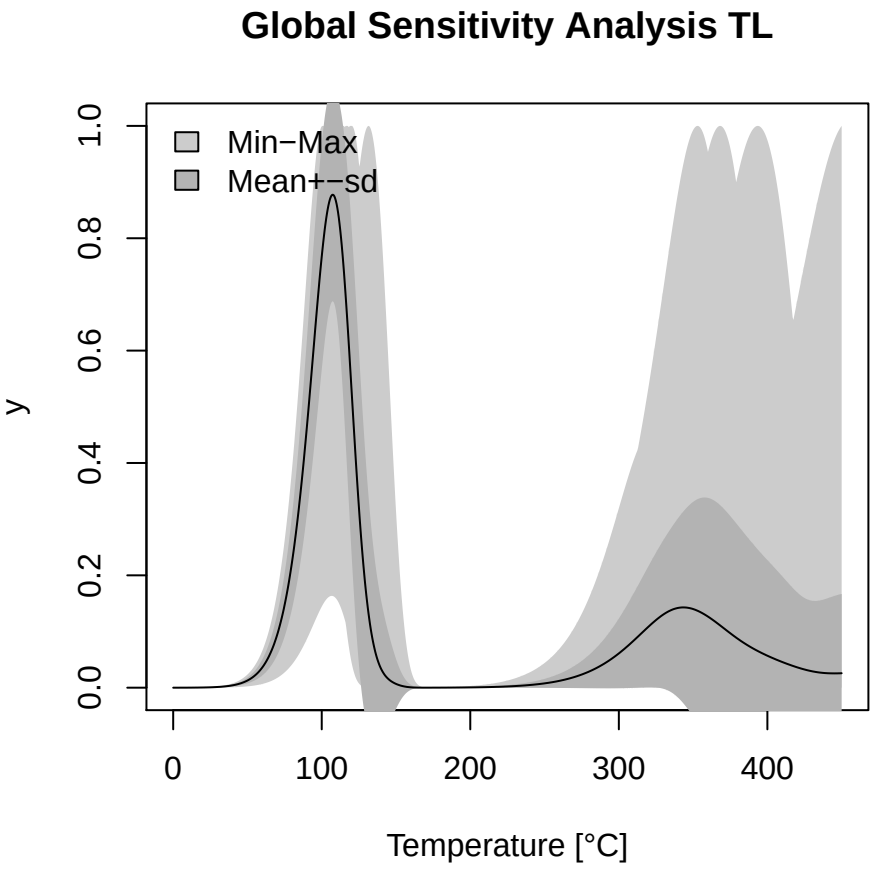
Now the user is able to calculate the global sensitivity of the chosen sequence-step and the parameters mentioned in "parRanges". In FME this is done via function "sensRange". A short explanation of the parameters in sensRage in the example below:

- func: The above created function "func_FME"
- parms: unlisted parameters (done via "extract_pars2FME")
- dist: The values of parRanges vary according to a regual grid

- The effect of the sensitivity “signal” is desired. Note: According to inner structures the parameter sensvar always has to be “signal”
- parRange is the minimum and maximum variation of the parameters chosen
- num: Times the model is run

For a detailed description the user is referred to the FME package. Subsequent a plot shows the range of possible TL curves.

```
global_Sens <- FME::sensRange(func = func_FME,
                    parms = parms,
                    dist = "latin",
                    sensvar = "signal",
                    parRange = parRanges,
                    num = 50)

global_Sens_sum <- summary(global_Sens)
```



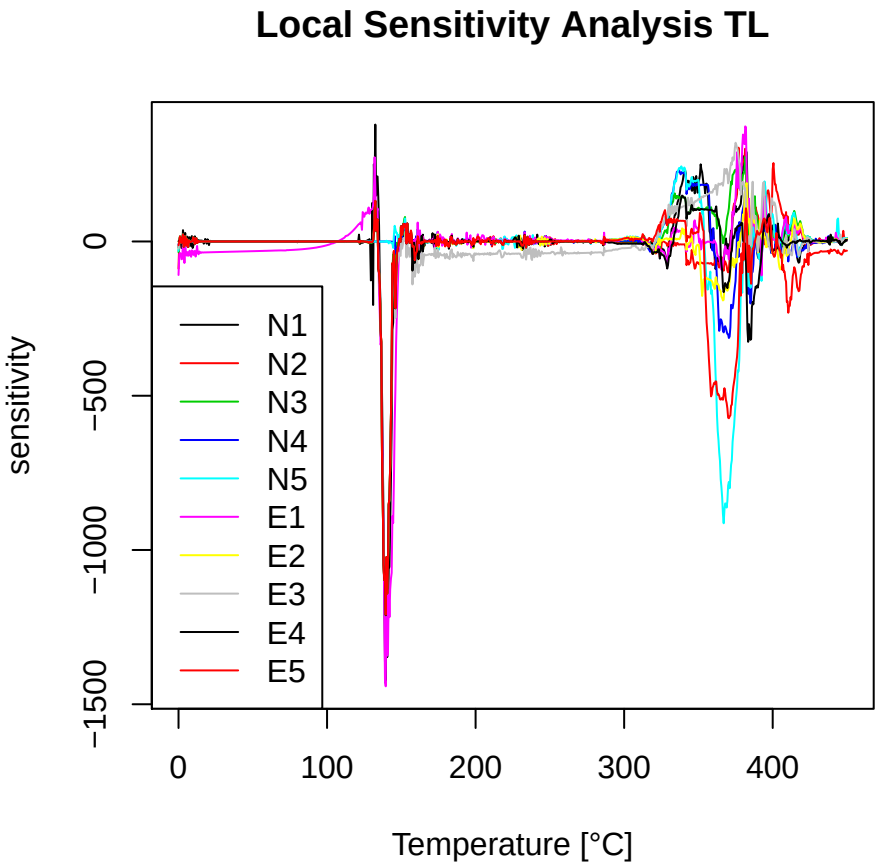
3.1.2 Local Sensitivity Analysis

In local sensitivity, the effect of a parameter value in a very small region near its nominal value is estimated.

With the above defined “parms” the user is now able run a local sensitivity analysis with the FME function “sensFun”. In parameter “senspar” the parameters to be analysed are written. In the example below the a few parameters of N and E are observed. It is also possible to observe all parameters, then the argument “sensvar” is omit.

```
SensR <- FME::sensFun(func = func_FME,
                      parms = parms,
                      senspar = c("N1","N2","N3","N4","N5","E1","E2", "E3", "E4", "E5"))

summary(SensR)
```

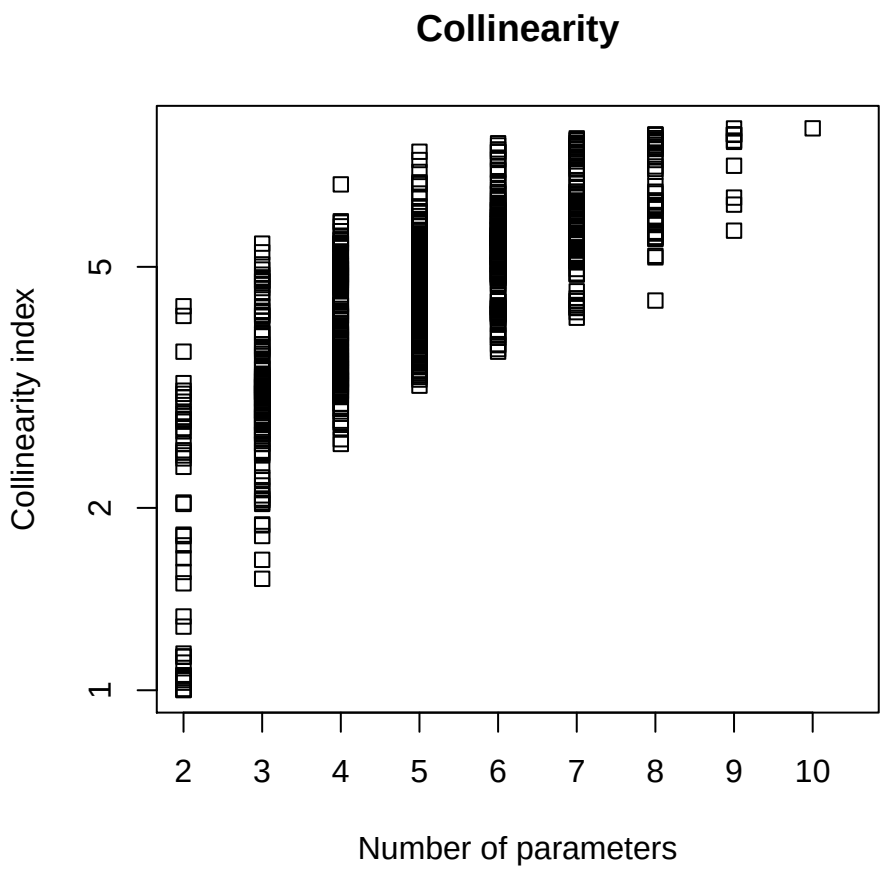
```
##      value   scale L1  L2  Mean   Min Max   N
## N1 5.1e+09 5.1e+09 42 4.8  -3.5 -1432 379 901
## N2 1.0e+07 1.0e+07 36 4.2  -9.6 -1210 304 901
## N3 1.0e+11 1.0e+11 37 4.2   3.7 -1214 274 901
## N4 2.5e+08 2.5e+08 27 2.2   3.2  -313 230 901
## N5 5.0e+10 5.0e+10 47 4.8 -14.6  -913 244 901
## E1 9.7e-01 9.7e-01 49 5.4 -17.8 -1442 373 901
## E2 1.6e+00 1.6e+00 32 4.1 -20.1 -1210 189 901
## E3 1.7e+00 1.7e+00 56 4.5  -7.4 -1212 320 901
## E4 1.8e+00 1.8e+00 36 4.2 -15.7 -1212 149 901
## E5 2.0e+00 2.0e+00 56 5.4 -40.9 -1210 132 901
```



Based on the sensitivity functions of model variables to selection of parameters, function collin calculates the collinearity or identifiability of sets of parameters.

The larger the collinearity value, the less identifiable the parameter based on the data. In general a collinearity value less than about 20 is ”identifiable”. Below we plot the collinarity as a function of the number of parameters selected. In this case all parameter sets are below this “magic number”.

```
Coll <- FME::collin(SensR)

plot(Coll, log = "y")
```



3.2 Fit model parameters to experimental TL data

Now the fit from model parameters to experimental data is shown. A data set from R-package "RlumModel" is loaded with a set of SAR-TL curves. The second TL curve (see subsection 3.1) is choosen and normalized. This normalization is recommended because it is difficult for the luminescence models to simulate absolute values but normalized signals are much easier to simualte. From function "func_FME" automatically normalized data output is provided. It is important, that the name of the columns are "time" and "signal". Otherwise the function call will break.

```
data("ExampleData.FittingTL", envir = environment())
exp_data <- get_RLum(TL_fitting_data, record.id = 2)
exp_data <- get_RLum(exp_data)

## normalize data
exp_data[,2] <- exp_data[,2]/max(exp_data[,2])
## rename columns
colnames(exp_data) <- c("time","signal")
```

The R-package FME offers the function "modCost", which estimates the "model cost", which is the sum of (weighted) squared residuals of the model versus the data. This function is central to parameter identifiability analysis, model fitting or running a Markov chain Monte Carlo.

We first define an objective function "Fit_func" that returns the residuals of the model versus the data, as estimated by modcost. It includes several fitting procedures; the default one is the Levenberg-Marquardt algorithm. Input to the function are the current values of the parameters that need to be finetuned and their names. In this case all parameters $\neq 0$ are chosen to fit the experimental data. Note that this is not allways possible, because the collinearitiy has to be less than about 20.

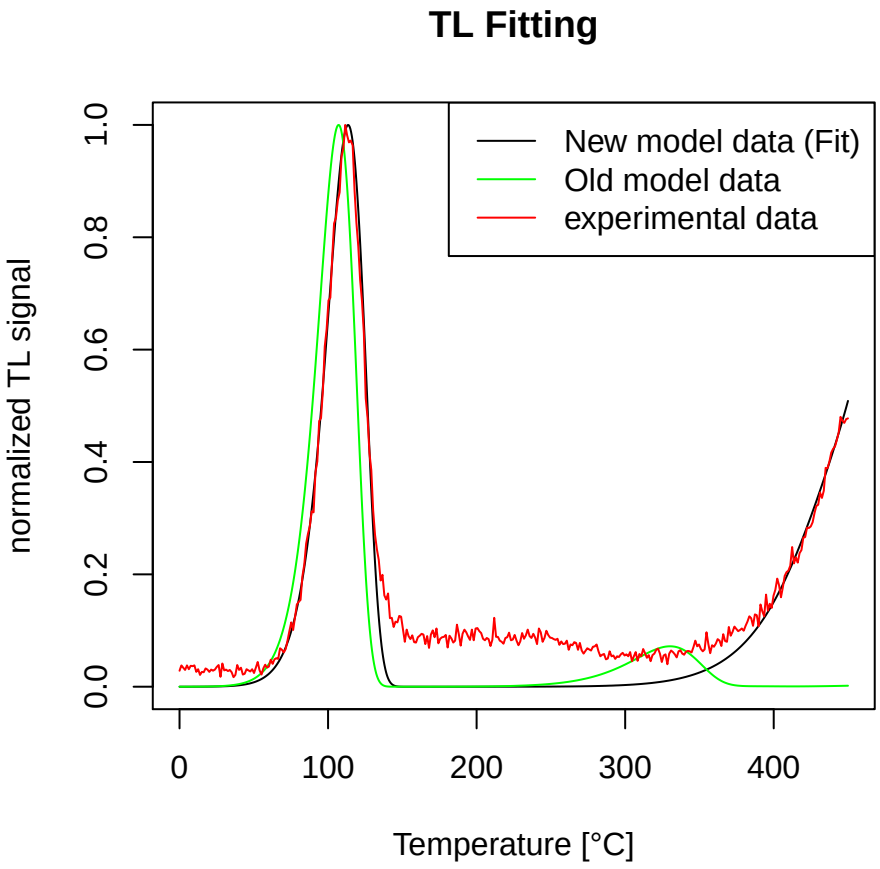
```
Fit_func <- function(x, parset = names(x)){

  parms[parset] <- exp(x)
  out <- func_FME(parms)
  return(modCost(model = out, obs = exp_data))
}

Fit_TL <- modFit(f = Fit_func,
                 p = log(c(parms[parms != 0])),
                 method = "Port")
```

Comparing old and new model parameters show only small differences between the single model parameters.

```
##        old          new   percent
## N1 5.1e+09   5071617816  99.44349
## N2 1.0e+07     10052917 100.52917
## N3 1.0e+11 102671533320 102.67153
## N4 2.5e+08    253751808 101.50072
## N5 5.0e+10  48445838452  96.89168
## N6 3.0e+08    306444068 102.14802
```

3.3 Plot results

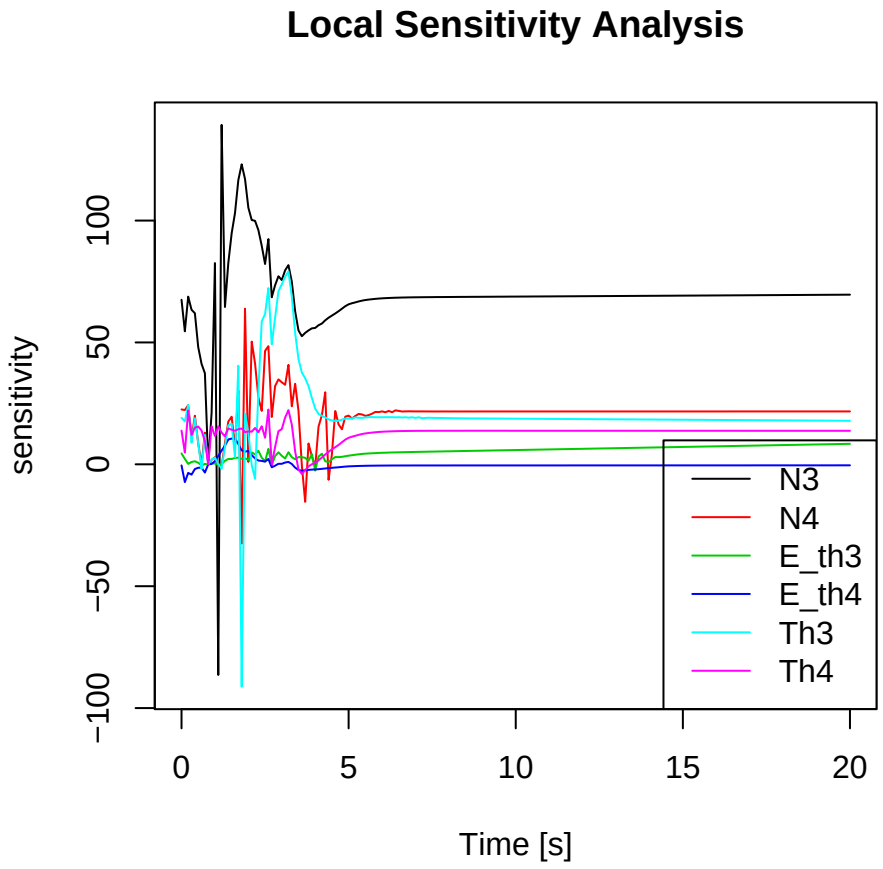
At least the simulations with old and new parameters and the experimental data are plotted.



3.4 Fitting OSL data

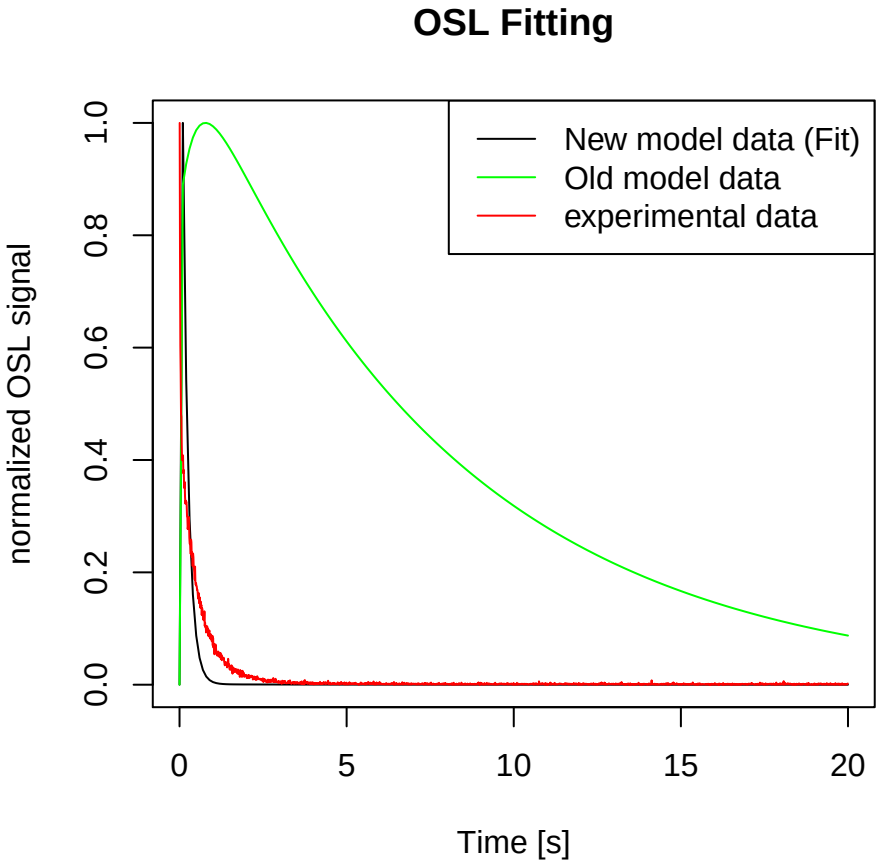
```
sequence <- list(OSL = c(125, 20, 200))

model <- "Pagonis2007"

func_FME <- fit_RLumModel2data(
  sequence = sequence,
  model = model,
  seq.step2fit = 1)

parms <- extract_pars2FME(model = model)
```

```
SensR <- FME::sensFun(func = func_FME,
                      parms = parms,
                      sensvar = c("signal"),
                      senspar = c("N3", "N4", "E_th3", "E_th4", "Th3", "Th4"))
```




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

exp_data[,2] <- exp_data[,2]/max(exp_data[,2])
colnames(exp_data) <- c("time","signal")

Fit_func_OSL <- function(x, parset = names(x)){

  parms[parset] <- exp(x)
  out <- func_FME(parms)
  return(modCost(model = out, obs = exp_data))
}

Fit_OSL <- modFit(f = Fit_func,
              p = log(c(parms[parms != 0])),
              method = "Port")
```

```
##        old          new   percent
## N1 5.1e+09   4977335595  97.59482
## N2 1.0e+07      9352918  93.52918
## N3 1.0e+11  98389267042  98.38927
## N4 2.5e+08    280326320 112.13053
## N5 5.0e+10 118281876386 236.56375
## N6 3.0e+08    219576636  73.19221
```


OSL Fitting
New model data (Fit)
Old model data
experimental data
normalized OSL signal
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
Time [s]