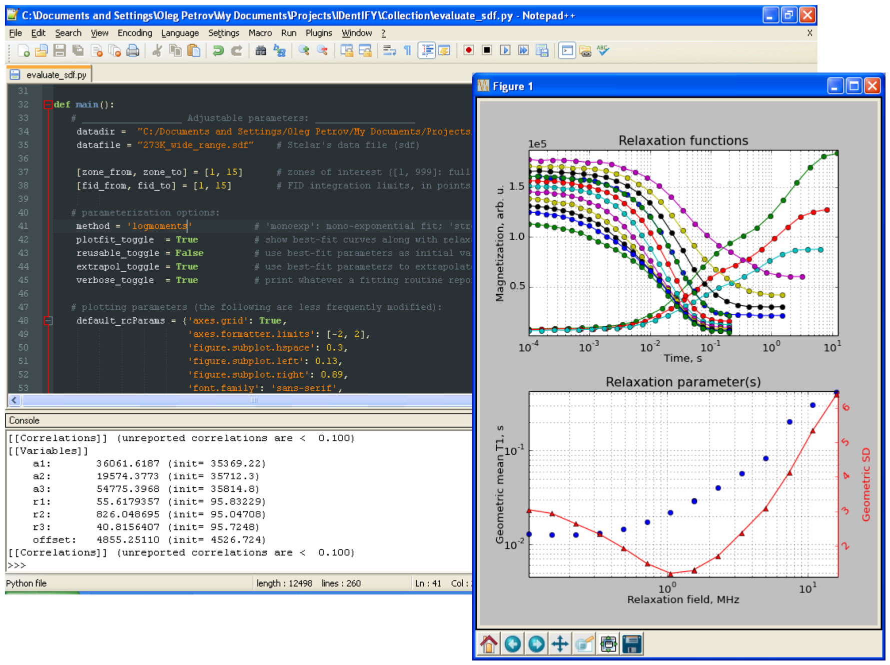
A User's Guide

Python modules for parameterization of non-exponential relaxation functions featuring Logarithmic Moment Analysis

Version 1.0, August 2017

Oleg V. Petrov | Arbeitsgruppe Prof. Dr. S. Stapf | Technische Universität Ilmenau


C:\Documents and Settings\Oleg Petrov\My Documents\Projects\IDentIFY\Collection\evaluate_sdf.py - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
evaluate_sdf.py
Figure 1
Relaxation functions
1e5
Magnetization, arb. u.
Time, s
Relaxation parameter(s)
Geometric mean T1, s
Geometric SD
Relaxation field, MHz
Console
Python file
length : 12498 lines : 260
Ln : 41 Col :

```
def main():
    # _________________ Adjustable parameters: _________________
     datadir =  "C:/Documents and Settings/Oleg Petrov/My Documents/Projects
     datafile = "273K_wide_range.sdf"    # Stelar's data file (sdf)

     [zone_from, zone_to] = [1, 15]      # zones of interest ([1, 999]: full
     [fid_from, fid_to] = [1, 15]        # FID integration limits, in points

    # parameterization options:
     method = 'logmoments'              # 'monoexp': mono-exponential fit; 'str
     plotfit_toggle  = True             # show best-fit curves along with relax
     reusable_toggle = False            # use best-fit parameters as initial va
     extrapol_toggle = True             # use best-fit parameters to extrapolat
     verbose_toggle  = True             # print whatever a fitting routine repo

    # plotting parameters (the following are less frequently modified):
     default_rcParams = {'axes.grid': True,
                         'axes.formatter.limits': [-2, 2],
                         'figure.subplot.hspace': 0.3,
                         'figure.subplot.left': 0.13,
                         'figure.subplot.right': 0.89,
                         'font.family': 'sans-serif',
```

```
[[Correlations]] (unreported correlations are <  0.100)
[[Variables]]
    a1:        36061.6187 (init= 35369.22)
    a2:        19574.3773 (init= 35712.3)
    a3:        54775.3968 (init= 35814.8)
    r1:        55.6179357 (init= 95.83229)
    r2:        826.048695 (init= 95.04708)
    r3:        40.8156407 (init= 95.7248)
    offset:    4855.25110 (init= 4526.724)
[[Correlations]] (unreported correlations are <  0.100)
>>>
```

Python modules for parameterization of non-exponential relaxation functions featuring Logarithmic Moment Analysis: A User's Guide

v1.0 August 2017

By Oleg V. Petrov

The group of Prof. Dr. Siegfried Stapf @ Technical University of Ilmenau

Please report comments and errors to: `oleg.petrov@tu-ilmenau.de`

Contents

1. Introduction

This document accompanies a collection of Python modules (py-files) for parameterization of non-exponential relaxation functions. The functions are time-domain functions of a general kind, either decaying or recovering, such as recorded in NMR T_1 relaxometry, diffusometry, fluorescence. The featured module is `logarithmic_moment_analysis.py` with a function `logmoments()` which returns three central lower-order logarithmic moments of a relaxation time distribution. The logarithmic moments refer to arithmetic mean relaxation time on the log-scale, the variance of relaxation time on the log-scale, and a skewness parameter of a relaxation time distribution (the latter is a combination of the 2nd and 3rd moments). To use `logmoments()`, the relaxation function must be sampled in logarithmically equidistant steps.

Other modules are `lmfit_exponential.py` and `lmfit_kohlrausch.py` which fit to the data exponential functions of up to 3rd order and a Kohlrausch (stretched exponential) function, respectively, both working in an automatic mode (no need for guessing about parameter values).

Two drivers have been provided in separate py-files. The driver in `profiles.py` is intended to work on explicit xy-data sets. It asks for an ASCII file with relaxation functions stored in separated XY columns. The driver in `evaluate_sdf.py` is intended to work with Stelar data files (sdf's), first evaluating relaxation functions from data blocks stored in a sdf-file.

The programs are written in Python 2.7. They depend on the `lmfit` package for "non-linear optimization and curve fitting problems" (lmfit.github.io/lmfit-py/), which can be downloaded for free. The programs are in source code rather than in .exe file. This assumes a Python interpreter with the `lmfit` module to have been installed in the system. Installing Python modules is out of scope of this instruction manual: one can find detailed instructions and recommendations for one's operating system elsewhere. In principle, the programs could be compiled in an .exe file, should it be really necessary. However, changes have been so frequent that one might want to have access to the source code.

Table 1. A list of functions

Function	*Module (source code)*	*Comments*
`logmoments()`	`logarithmic_moment_analysis.py`	Returns three lower-order logarithmic moments of a relaxation time distribution
`decay()`	`logarithmic_moment_analysis.py`	Auxiliary function used in data extrapolation
`recovery()`	`logarithmic_moment_analysis.py`	Auxiliary function used in data extrapolation
`minimize()`	`lmfit_kohlrausch.py`	Minimizes the objective function using Powell's method (by means of `lmfit`)
`initialize_params()`	`lmfit_kohlrausch.py`	Computes initial values of model function's parameters
`perturb_params()`	`lmfit_kohlrausch.py`	Randomizes parameters within few percents of their initial values
`residual()`	`lmfit_kohlrausch.py`	Defines the objective function for the minimizer
`minimize()`	`lmfit_exponential.py`	Same as above
`initialize_params()`	`lmfit_exponential.py`	Same as above
`perturb_params()`	`lmfit_exponential.py`	Same as above
`residual()`	`lmfit_exponential.py`	Same as above
`main()`	`profiles.py`	Main function (a driver) to parameterize explicitly defined data sets
`main()`	`evaluate_sdf.py`	Main function (a driver) to parameterize data stored in Stelar's data files

2. Logarithmic Moment Analysis

2.1. The general concept

Logarithmic moment analysis (LMA) is the main reason for using this bit of software. The method has been introduced in [1]. It is based on the moment rule of convolution and allows one to measure three central lower-order logarithmic moments (LM's) of a relaxation time distribution without first calculating the distribution function itself. More conveniently, the LM's are expressed in terms of the geometric mean time, the geometric standard deviation and a skewness parameter. Two prerequisites to the method are (i) to have the relaxation function sampled in logarithmically equidistant steps and (ii) to scale it to [1, 0]. With these preliminary steps done, the LM's are calculated straightforwardly through numerical integration of the relaxation function [2].

2.2. Practical aspects

Scaling the relaxation function to [1, 0] forms the principle computational problem of LMA. Apparently, the most feasible solution is to fit a model exponential function to data using the best-fit amplitude and offset parameters as approximation to extreme data points. We found a triple-exponential function sufficient to approximate relaxation functions at either plateau in all our tests, hence we use this model function by default.

In addition to a relatively accurate data normalization, fitting a model (triple-exponential) function enables to extrapolate relaxation curves towards zero time, thus providing a better fulfillment of the limiting conditions underlying the integral (12) in Ref. [2]. The data extrapolation is included in `logmoments()` by default, that is, prior to integration, a magnetization curve is extended by N model points at the beginning of the curve.

The (triple-exponential) fit is implemented through the Powell's optimization by means of the `lmfit` package (see page 8). For numerical integration, Simpson's rule integration is used.

The `logmoments()` function returns three central lower-order LM's. The 1st and 2nd LM's are reported in the form of the geometric mean (GM) relaxation time and the geometric standard deviation (GSD). These quantities, being defined on the linear time scale, appear to be more convenient when comparing to other relaxation presentations. The accuracy of the third moment strongly depends on the signal-to-noise ratio of the data and, as such, is often of no use. Hence, no skewness parameter is reported at present.

2.3. The logmoments() function

The `logmoments()` function is defined in `logarithmic_moment_analysis.py`. The latter module also defines two auxiliary functions, `decay()` and `recovery()`, which calculate model exponential functions used for data extrapolation. Because of the need of data normalization, `logmoments()` depends on the module `lmfit_exponential.py`, which implements a triple-exponential fit to the relaxation data (see below).

`logmoments``(`x, y, y_fit=[], extrapol=*True*, plotdata=*False*, verbose=*True*, **fit_kws`)`.

Parameters: **x**: *array-like*

Input xdata, supposed to be logarithmically equispaced time points

y: *array-like*

Input ydata, a signal intensity measured at respective time points

y_fit: *array-like, optional*

A pointer to the best-fit model function sought inside `logmoments()`

extrapol: *boolean, optional*

Defines whether to execute data extrapolation; if *True*, *N* original data points are extended by *N* points at the shorter-time side of the relaxation function

plotdata: *boolean, optional*

Defines whether to plot relaxation functions and their best-fit curves in course of the logarithmic moment calulation

verbose: *boolean, optional*

Defines whether to output the calculated quantities to the console

****fit_kws**: *dictionary, optional*

Options to pass to the minimizer being used. If `lmfit_exponential.minimize()`, the dictionary is to include three options: 'order' , 'reusable', and 'verbose' (see below).

Returns:

A list of three central lower-order logarithmic moments. By default, the list is saved in a text file `logmoments_output.txt` in a working directory. An implicit return is possible via the optional parameter **y_fit** which points to the best-fit model of the relaxation function (useful for visual control of data normalization from within a calling function).

3. Stretched Exponential Fit

The stretched exponential function (1), also called the Kohlrausch function, is a long-used, compact model for phenomenological description of non-exponential relaxation, also in NMR relaxometry.

$$\varphi(t) \propto \exp[-(t/\tau_K)^{\beta}] \qquad (1)$$

A stretching exponent $\beta \in [0, 1]$ measures the deviation from a mono-exponential relaxation, with $\beta = 1$ corresponding to the usual exponential function. The first moment of a normalized (1)

$$<\tau>=\int_0^{\infty} \exp[-(t/\tau_K)^{\beta}]dt = \frac{\tau_K}{\beta}\Gamma(\frac{1}{\beta})$$

where Γ is the gamma function, represents the arithmetic mean relaxation time.

The module `lmfit_kohlrausch.py` contains four functions: `initialize_params()`, `perturb_params()`, `residual()`, and `minimize()`. The function `initialize_params()` sets up initial values for three fitting parameters – time, stretching exponent, and offset, by solving a linear matrix equation for relaxation data. Thus initialized parameters are randomized within few percents of their values in `perturb_params()` and passed to `minimize()` which calls a same-name function from the `lmfit` package to minimize the residual between the stretched exponential model and the experimental relaxation function. The optimization method used is Powell's. As an option, the best-fit parameters are cached and retrieved as initial values in the next fit in a series.

`lmfit_kohlrausch.`**`minimize`**`(`x, y, options={'reusable': *False*, 'verbose': *True*}`)`.

Parameters: **x**: *array-like*

Input xdata, supposed to be logarithmically equispaced time points

y: *array-like*

Input ydata, a signal intensity measured at respective time points

options: *dictionary, optional*

◦ 'reusable': caches the best-fit parameters to retrieve at the next fit in a series;

◦ 'verbose': reports the best-fit parameters and several goodness-of-fit statistics.

Returns:

`MinimizerResult` object containing the best-fit parameters and several goodness-of-fit statistics (see `lmfit` documentation for details).

4. Mono-, Double-, and Triple-Exponential Fit

The module `lmfit_exponential.py` implements fitting of an exponential functions of up to third order to data. It contains the same set of functions as `lmfit_kohlrausch.py`, namely, `initialize_params()`, `perturb_params()`, `residual()`, and `minimize()`. The latter function has only one addition input parameter, which is the order of the exponential fit. The function `initialize_params()` sets up initial values for time constants and amplitudes of the exponential components through solving a linear matrix equation for relaxation data, and initializes the offset parameter by a minimum value in the data set. Thus initialized parameters are randomized within few percents of their values in `perturb_params()` and passed to `minimize()` which calls a same-name function from the `lmfit` package to minimize the residual between the (multi-) exponential model and the given relaxation function. The optimization method used is Powell's. As an option, the best-fit parameters can be cached and retrieved as initial values in the next fit in a series.

`lmfit_exponential.`**`minimize`**`(`x, y, options={ 'order': 3, 'reusable': *False*, 'verbose': *True*}`).`

Parameters: **x**: *array-like*

Input xdata, supposed to be logarithmically equispaced time points

y: *array-like*

Input ydata, a signal intensity measured at respective time points

options: *dictionary, optional*

◦ 'order': the order of the exponential function being used (1, 2, or 3);

◦ 'reusable': caches the best-fit parameters to retrieve at the next fit in a series;

◦ 'verbose': reports the best-fit parameters and several goodness-of-fit statistics.

Returns:

`MinimizerResult` object containing the best-fit parameters and several goodness-of-fit statistics (see `lmfit` documentation for details).

5. Drivers

5.1. Working with explicit data sets

Two modules, `profiles.py` and `evaluate_sdf.py`, have been provided to serve as user interface to parameterization routines. I call them drivers. The module `profiles.py` is intended to work with relaxation functions stored in a tab separated text file, as a series of XY columns of *same length*, without headers. Fig. 1 shows an example of such a file containing two relaxation functions.

Relaxation function #1		*Relaxation function #2*	
1E-4	6848.4	1E-4	7368
1.6337E-4	7216.3	1.6628E-4	7542.4
2.669E-4	7485.8	2.7648E-4	8105.2
4.3605E-4	7632.1	4.5973E-4	7897.5
7.1238E-4	7285.3	7.6443E-4	8597.6
0.00116	7758.2	0.00127	9469.7
0.0019	8468.2	0.00211	10100
0.00311	9960.2	0.00351	
1.8347	110030	2.6102	154590
2.9973	120320	4.3402	170260
4.8968	125200	7.2168	180940
8	127520	12	184040

Fig. 1. *An input text file with two saturation-recovery functions subject to parameterization. Note the logarithmic spacing of xdata (mandatory if the logarithmic moment analysis is considered).*

The file `profiles.py` has numerous parameters that can be chosen near the beginning of the program code. These are:

Parameters: **datadir**: *text*

Data directory, *e.g.* "C:/Documents and Settings/NMR/"

datafile: *text*

Text (ASCII) file containing relaxation data, *e.g.* "serrano_adipose.txt"

method: *text*

Parameterization method:

◦ '*stretched*': the stretched exponential fit;

◦ '*monoexp*': mono-exponential fit;

◦ '*biexp*': double-exponential fit;

◦ '*logmoments*': logarithmic moment analysis.

plotfit_toggle: *Boolean*

Plot best-fit curves with original data (default: *True*)

reusable_toggle: *Boolean*

Cash and retrieve best-fit parameters as initial for the next fit (default: *False*)

extrapol_toggle: *Boolean*

Define whether to extrapolate relaxation functions toward zero time; relevant to the ‘logmoments’ method (default: *True*)

verbose_toggle: *Boolean*

Report best-fit parameters and several statistics from the minimizer being used.

rcParams*: dictionary*

These control the appearance of two subplots: one for original relaxation functions and one for relaxation parameters. They are less frequently modified.

Output:

The `profiles.py` outputs the sought-after relaxation parameters in three complementary ways. One way is a console output; the other is plotting the parameter(s) against experiment’s index (a.k.a. relaxation profiles); and the third way is writing them to a file in the data directory (the filename is `profiles_output.txt`).

5.2. Working with Stelar's data files

The module `evaluate_sdf.py` works on the relaxation functions after first evaluating them from a Stelar's data file (a sdf-file). The file is a series of zones, each zone representing one relaxation experiment, with a list of experimental parameters followed by blocks of data. The number of blocks equals the number of signals (FID's or spin echoes) acquired at a particular relaxation field strength, while the block size equals the number of points in individual signals. A data block format allows either two or three values per line. In the former case, the values represent complex FID points (Re/Im pairs). **The current version of `evaluate_sdf.py` works only with the sdf-files that have two values per line**.

The file `evaluate_sdf.py` has numerous parameters that can be chosen near the beginning of the program code. These are:

Parameters: **datadir**: *text*

Data directory, *e.g.* "C:/Documents and Settings/NMR/"

datafile: *text*

Text (ASCII) file containing relaxation data, *e.g.* "serrano_adipose.txt"

zone_from, **zone_to**: *int*

Zones of interest, defines the number of profiles points

fid_from, **fid_to**: *int*

FID integration limits, in points; the integration range determines the signal-to-noise ratio of evaluated relaxation curves – make effort to find the optimum!

method: *text*

Parameterization method:

◦ '*stretched*': the stretched exponential fit;

◦ '*monoexp*': mono-exponential fit;

◦ '*logmoments*': logarithmic moment analysis.

plotfit_toggle: *Boolean*

Plot best-fit curves with original data (default: *True*)

reusable_toggle: *Boolean*

Cash and retrieve best-fit parameters as initial for the next fit (default: *False*)

extrapol_toggle: *Boolean*

Define whether to extrapolate relaxation functions toward zero time; relevant to the 'logmoments' method (default: *True*)

verbose_toggle: *Boolean*

Report best-fit parameters and several statistics from the minimizer being used.

rcParams: *dictionary*

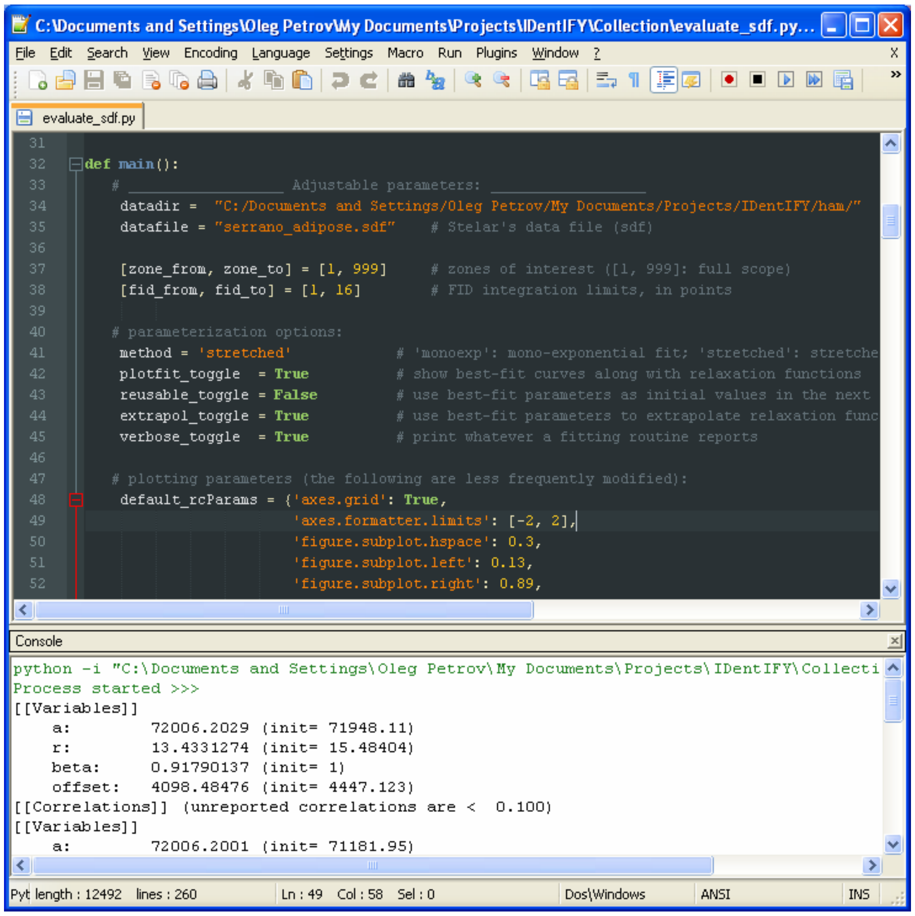
These control the appearance of two subplots: one for original relaxation functions and one for relaxation parameters. They are less frequently modified.

Output:

The `evaluate_sdf.py` outputs the sought-after relaxation parameters in three complementary ways. One way is a console output; the other way is plotting the parameter(s) against experiment's index (a.k.a. relaxation profiles); and the third way is writing them to a file in the data directory (the filename is `profiles_output.txt`).

5.3. Use of the drivers

Either of the drivers, `profiles.py` or `evaluate_sdf.py`, is run after loading a Python IDE or any text editor with a built-in ability to execute console commands. Then one opens the .py module of choice, adjust necessary parameters, and run the program by means of the IDE being used. For example, Fig. 2 shows the `evaluate_sdf.py` module opened in the Notepad++ editor.


C:\Documents and Settings\Oleg Petrov\My Documents\Projects\IDentIFY\Collection\evaluate_sdf.py...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
evaluate_sdf.py
Console

```
31
32  def main():
33      #  ____________________ Adjustable parameters: ____________________
34      datadir =  "C:/Documents and Settings/Oleg Petrov/My Documents/Projects/IDentIFY/ham/"
35      datafile = "serrano_adipose.sdf"    # Stelar's data file (sdf)
36
37      [zone_from, zone_to] = [1, 999]     # zones of interest ([1, 999]: full scope)
38      [fid_from, fid_to] = [1, 16]        # FID integration limits, in points
39
40      # parameterization options:
41      method = 'stretched'               # 'monoexp': mono-exponential fit; 'stretched': stretche
42      plotfit_toggle  = True             # show best-fit curves along with relaxation functions
43      reusable_toggle = False            # use best-fit parameters as initial values in the next
44      extrapol_toggle = True             # use best-fit parameters to extrapolate relaxation func
45      verbose_toggle  = True             # print whatever a fitting routine reports
46
47      # plotting parameters (the following are less frequently modified):
48      default_rcParams = {'axes.grid': True,
49                          'axes.formatter.limits': [-2, 2],
50                          'figure.subplot.hspace': 0.3,
51                          'figure.subplot.left': 0.13,
52                          'figure.subplot.right': 0.89,
```

```
python -i "C:\Documents and Settings\Oleg Petrov\My Documents\Projects\IDentIFY\Collecti
Process started >>>
[[Variables]]
    a:         72006.2029 (init= 71948.11)
    r:         13.4331274 (init= 15.48404)
    beta:      0.91790137 (init= 1)
    offset:    4098.48476 (init= 4447.123)
[[Correlations]] (unreported correlations are <  0.100)
[[Variables]]
    a:         72006.2001 (init= 71181.95)
```

```
Pyt length : 12492   lines : 260   Ln : 49   Col : 58   Sel : 0   Dos\Windows   ANSI   INS
```

***Fig. 2**. The `evaluate_sdf.py` module loaded into Notepad++. The lines 34-45 contain adjustable parameters.*

Behavior of the program is virtually independent on the parameterization method chosen. A difference is mostly in quantities that are output on the console and in titles of the parameter subplot axes. As a rule, convergence of the optimization procedure is slower for the logmoments method since it invokes a higher-order exponential fit. In their present version, the drivers call optimization functions *five times* for averaging, which is controlled by the variable `accus`. If it turns out to be too slow, set `accus`=1. Attention must be paid to the `reusable` option: although it can reduce scattering in results, it requires relaxation functions that (i) are all of same type (decay or recovery) and (ii) allow only moderate changes in parameters when going from previous to next one.

6. Examples

Here are several demos of the graphical output of the program `evaluate_sdf.py` generated from actual data sets acquired in fast field-cycling experiments on Stelar SpinMaster FFC2000 relaxometer. The sdf-files used can be found in the subfolder `~/Example Data/`. Adjustable parameter values have been reported too.

6.1. Composite water/cyclohexane sample

The sample is made of two nested coaxial NMR tubes of 5 and 10 mm diameters, filled with cyclohexane and $CuSO_4$-doped water, respectively. The temperature was kept at 273K to provide essentially different T_1 dispersions for the two components with up to a sixty-fold gap between their T_1 values. A cross-point for T_1 is observed at relaxation field ~1 MHz. Data file: **`composite.sdf`**.

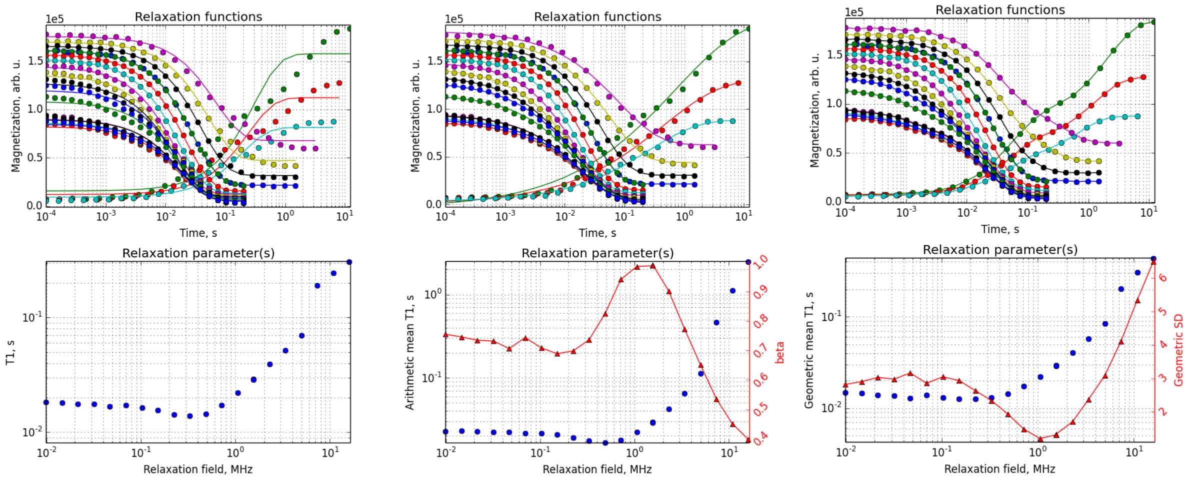


***Fig. 3**. Parameterization method used: '`monoexp`' (left); '`stretched`' (middle); '`logmoments`' (right). Other adjustable parameters: [`fid_from, fid_to`] = [1, 15]; `reusable_toggle` = False; `extrapol_toggle` = True. Note the difference in lower subplot's axes for a characteristic relaxation time (T_1, the arithmetic mean T_1, the geometric mean T_1) and the parameter for deviation from exponential (beta, the geometric standard deviation) depending on the parameterization method being used. Note a bad fit in cases of mono-exponential and stretched model in upper subplots.*

6.2. Fat tissue from ham slices

Samples were cut from the representative fat regions of ham slices (Jamón Serrano, Spain) and measured at 283K in the relaxation field ranging from 20 MHz down to 10 kHz. Data file: **`serrano_adipose.sdf`**.

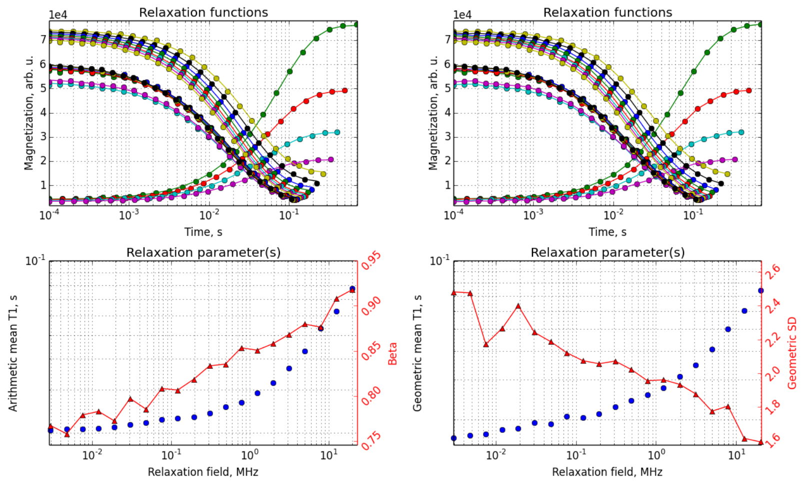


Fig. 4. *Parameterization method used: '`stretched`' (left); '`logmoments`' (right). Other adjustable parameters: [`fid_from, fid_to`] = [1, 16]; `reusable_toggle` = False; `extrapol_toggle` = True. The relaxation in fat tissue is essentially non-exponential (beta and GSD parameters are significantly different from 1). The deviation from exponential is the more, the lower the relaxation field.*

6.3. Muscle tissue from ham slices

Samples were cut from the representative muscle regions of ham slices. T_1 relaxation was measured first ranging the relaxation field from 20 MHz down to ~10 kHz (full scan) and then within 4-0.4 MHz, the region where ^{1}H T_1 relaxation is subject to a quadrupolar enhancement due to ^{1}H– ^{14}N cross-relaxation under level-matching conditions (notice characteristic Q-dips at 2.75 and 2.07 MHz). Data files: **`land_SM.sdf`**, **`land_SM_dips.sdf`**

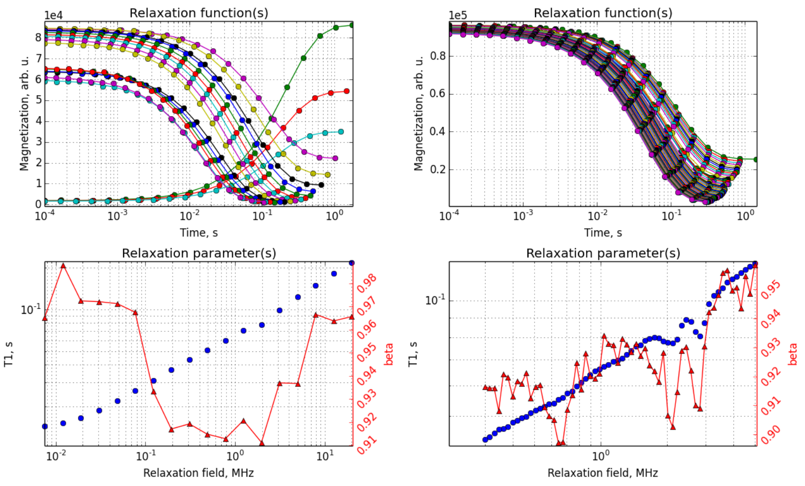


***Fig. 5**. FFC relaxometry on muscle tissue of ham. Left: full scan of relaxation field. Right: quadrupole enhancement range (Q-dips region). Parameterization method used: '`stretched`'. Unlike in fat, T_1 relaxation in muscle tissue is nearly mono-exponential (beta is close to 1). Deviation from exponential is maximum at the relaxation fields around 0.2-2 MHz. Local minima of the beta parameter are positively seen around 2.75, 2.07, and 0.67 MHz, the same field strengths where Q-dips in T_1 are observed.*

6.4. Muscle tissue from ham slices (continued)

The final example is muscle tissue but from different ham (Jamón Serrano, Spain). Once again, characteristic Q-dips due to quadrupolar enhancement of T_1 relaxation are well visible in all profiles. The Q-dips in the T_1 parameter are accompanied by the broadening of relaxation: both beta and GSD parameter have local extrema at the same relaxation field strength where the Q-dips are. Data file: **`serrano_SM_dips.sdf`**.

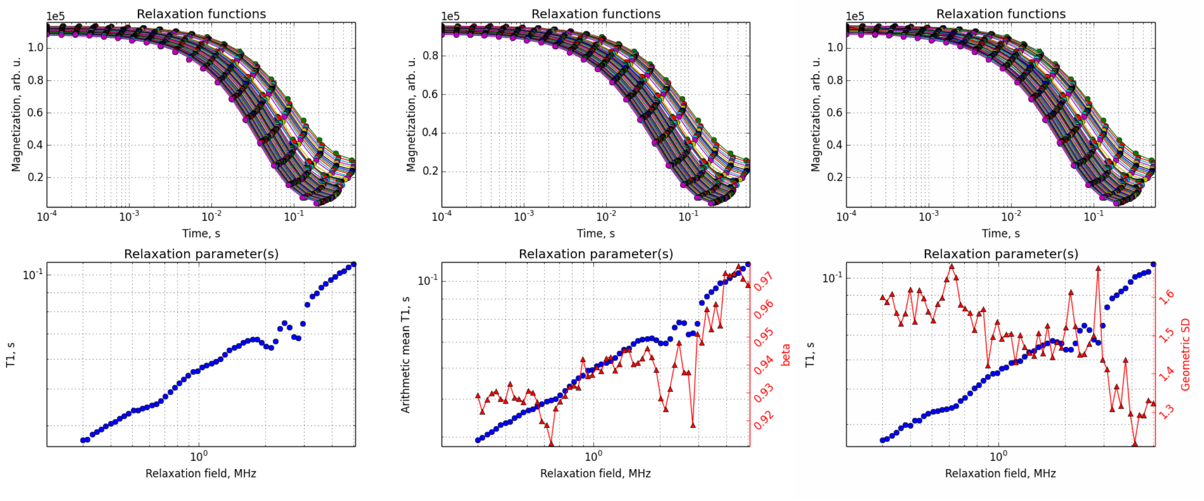


***Fig. 6**. Parameterization method used: '`monoexp`' (left); '`stretched`' (middle); '`logmoments`' (right). Other adjustable parameters: [`fid_from, fid_to`] = [1, 19]; `reusable_toggle` = True; `extrapol_toggle` = True.*

7. References

[1] R. Zorn, *J. Chem. Phys*. **116** (2002) 3204-3209

[2] O. V. Petrov, S. Stapf, *J. Magn. Reson*. **279** (2017) 29-38