Manual EINSfit class for Python 3, version 1.0.0

Dominik Zeller

November 2019

Contents

Disclaimer **1**

Abbreviations **1**

1 Introduction **2**
1.1 Gaussian approximation = GA model . . . 2
1.2 PK model . . . 2
1.3 Yi model . . . 2
1.4 Do model . . . 3

2 Installation **4**
2.1 Requirements . . . 4
2.2 Step to step install instructions . . . 4
2.2.1 Installation with pip (recommended): . . . 4
2.2.2 Installation without pip: . . . 4
2.2.3 Without installation . . . 5

3 Config file **6**

4 How to use **8**

5 Help Doc **11**

Disclaimer

This documentation is NEITHER thought as an introduction to neutron scattering NOR does it explain the theory behind the used models. It is ONLY thought as a help to understand and use the EINSfit class and its output data files! Basic knowledge of elastic incoherent neutron scattering (EINS) is explicitly needed and assumed. If you are interested in the reasoning behind the EINSfit class, you are invited to read my Ph.D. thesis (or `http://www.theses.fr/s146914`) [1].

Abbreviations

EINS - elastic incoherent neutron scattering
Q - neutron momentum transfer
T - temperature
$S_{\text{inc}}(Q, \omega)$ - dynamic incoherent structure factor
EISF(Q) - elastic incoherent structure factor, defined as $S_{\text{inc}}(Q, \omega = 0)$, so that in theory: EISF(Q=0)=1
EISF(Q=0) - EISF at zero momentum transfer, also called offset in text
EI(Q) - (experimental) elastic intensity, defined like in [1, 2], $\sim$ EISF(Q)
GA - Gaussian Approximation
MSD or $\langle r^2 \rangle$ - mean square displacement, defined as a time independent/static MSD
STD - standard deviation of the MSD
MSD3 or STD3 - value of the MSD or STD, defined with the pre-factor 3 in the GA

1 Introduction

The EINSfit class helps to fit EINS data with different models at the same time. The main model is the Gaussian Approximation (GA), which is only valid in the low momentum transfer range Q. The results of the GA model can be used to set the EISF(Q=0) parameter for the other models. In the following EISF(Q=0) will also be called 'offset'. Using the offset obtained by the GA helps to fit the other available models more consistently since they have more degrees of freedom and can use a larger momentum transfer range Q.
The other models are:

- Peters and Kneller (PK model) [3], first publication of model [4]
- Yi et al. (Yi model) [5], first publication of model [6]
- Doster et al. (Do model) [7]

The used fitting functions are defined in the following. Additional information can also be found in Zeller et al. [2] and in my Ph.D. thesis [1] (`http://www.theses.fr/s146914`).

1.1 Gaussian approximation = GA model

The GA is applied via a linear fit of the natural logarithm of the intensities ln(EISF) vs the momentum transfer Q squared:

$$\mathrm{EISF(Q)} = \mathrm{EISF(Q=0)} * \exp\left\{-Q^2 \frac{\langle R^2 \rangle_{\mathrm{GA}}}{3}\right\} \tag{1}$$

$$\ln(\mathrm{EISF}) = \ln(\mathrm{EISF(Q=0)}) - Q^2 \frac{\langle R^2 \rangle_{\mathrm{GA}}}{3} \tag{2}$$

$$\Rightarrow y = \mathrm{slope} * x + \mathrm{intercept} \tag{3}$$

$$\text{with the substitution: } x = Q^2,\ y = \ln(\mathrm{EISF}),\ \mathrm{slope} = -\frac{\langle R^2 \rangle_{\mathrm{GA}}}{3},\ \mathrm{intercept} = \ln(\mathrm{EISF(Q=0)}) \tag{4}$$

Therefore, the MSD3 and offset is:

$$\mathrm{MSD3_{GA}} = -3 * \ \mathrm{slope} \tag{5}$$

$$\mathrm{offset} = \mathrm{EISF(Q=0)} = \exp(\mathrm{intercept}) \tag{6}$$

1.2 PK model

In the program the PK model is defined as in the original paper [3]:

$$\mathrm{EISF(Q)} = \mathrm{EISF(Q=0)} * \left(1 + \frac{(\sigma_{\mathrm{PK}} * Q)^2}{\beta}\right)^{-\beta} \tag{7}$$

The MSD3 and STD3 is calculated according to:

$$\mathrm{MSD3_{PK}} = 3 * \langle R^2 \rangle_{\mathrm{PK}} = 3 * \sigma^2_{\mathrm{PK}} \tag{8}$$

$$\mathrm{STD3_{PK}} = 3 * \frac{\langle R^2 \rangle_{\mathrm{PK}}}{\sqrt{\beta}} = \frac{\mathrm{MSD3_{PK}}}{\sqrt{\beta}} \tag{9}$$

1.3 Yi model

In the program the Yi model is defined as in the original paper [5]:

$$\mathrm{EISF(Q)} = \mathrm{EISF(Q=0)} * \exp\left\{-Q^2 \frac{\langle R^2 \rangle_{\mathrm{Yi}}}{6}\right\} \left(1 + \frac{Q^4}{72}\sigma^2_{\mathrm{Yi}}\right), \tag{10}$$

The MSD3 and STD3 is calculated according to:

$$\mathrm{MSD3_{Yi}} = \frac{\langle R^2 \rangle_{\mathrm{Yi}}}{2} \tag{11}$$

$$\mathrm{STD3_{Yi}} = \frac{\sigma_{\mathrm{Yi}}}{2} \tag{12}$$

1.4 Do model

In the program the Do model is defined as in the original paper [7]:

$$\begin{aligned}\mathrm{EISF(Q)} &= \mathrm{EISF(Q=0)} * \exp\left\{-Q^2 \left\langle R^2 \right\rangle_{\mathrm{Do},G}\right\} \\ &\quad \times \left(1 - 2p_{12}\left(1 - \operatorname{sinc}\left(Qd\right)\right)\right), \end{aligned} \tag{13}$$

$$p_{12} = p_1 * p_2 \tag{14}$$

$$p_1 + p_2 = 1 \tag{15}$$

The total MSD is defined as :

$$\begin{aligned}\left\langle R^2 \right\rangle_{\mathrm{Do,tot}} &= -\left(\frac{d\ln\left[\mathrm{EISF(Q)}\right]}{d(Q^2)}\right)_{Q=0} \\ &= \left\langle R^2 \right\rangle_{\mathrm{Do},G} + p_{12}d^2. \end{aligned} \tag{16}$$

The MSD3, population p1 and p2 are calculated according to:

$$\mathrm{MSD3_{Do}} = 3 * \left\langle R^2 \right\rangle_{\mathrm{Do,tot}} \tag{17}$$

$$p_1 = 0.5 + \sqrt{0.25 - p_{12}} \tag{18}$$

$$p_2 = 1 - p_1 \tag{19}$$

2 Installation

2.1 Requirements

Requirements for functionality of EINSfit class:

- Python version >= 3.6
- Packages needed:
  - numpy
  - lmfit (from channel conda-forge)
  - matplotlib
  - cycler (normally included in matplotlib)
  - ipython or jupyter (not needed but for interactive usage)

Conda install latest versions:

- ```
  conda create --channel conda-forge --name p3_EINSfit
  python=3 numpy matplotlib lmfit ipython
  ```

Conda install tested version:

- ```
  conda create --channel conda-forge --name p3_EINSfit
  python=3.7 numpy=1.16 matplotlib=3.1 lmfit=0.9.13
  ipython=7.5
  ```

2.2 Step to step install instructions

1. download git repository at https://github.com/DominikZ/EINSfit/archive/master.zip
2. unpack zip file
3. install package with or without pip OR save EINSfit/definitions.py

2.2.1 Installation with pip (recommended):

Install:

- open terminal (lnx) or conda shell (win)
- activate conda environment via:

  lnx: `source activate p3_EINSfit`

  win: `activate p3_EINSfit`
- navigate to un-packed EINSfit package and type:

```
python setup.py sdist
pip install .dist/EINSfit-***VERSION***.tar.gz
```

To uninstall use:

```
pip uninstall EINSfit
```

2.2.2 Installation without pip:

Install:

- open terminal (lnx) or conda shell (win)
- activate conda environment via:

  lnx: `source activate p3_EINSfit`

  win: `activate p3_EINSfit`
- navigate to un-packed EINSfit package and type:

```
python setup.py build
python setup.py install
```

2.2.3 Without installation

Save "EINSfit/definitions.py" in wished directory and import class EINSfit to your python3 script via:

```
# add directory where definitions.py is saved to path variable
import sys
sys.path.append('DIRECTORY') #replace DIRECTORY by the path of your chosen directory

from definitions import EINSfit
```

3 Config file

The config file defines how the fitting of the 4 different models is performed. The config file has different 5 categories:

- Global: defines general fitting parameters, e.g. no weighting of fit or use the offset of the GA model to fix the offset of the other models
- GA: define maximal Q value used for fit
- PK: define maximal Q value used for fit; define start, min and max value of β and σ_{PK}
- Yi: define maximal Q value used for fit; define start, min and max value of $\langle R^2 \rangle_{\mathrm{Yi}}$ and σ_{Yi}
- Do: define if model is used; if d is fixed and its value; used Q range is equal to PK model

The values can be changed by the user at any time in two different ways:

1. `set_config_dic(input_dic)` : loads parameters which are defined in an nested dictionary `input_dic`
2. `load_config_dic(FILENAME)` : loads parameters which are defined in a config file (see Listing 1)

The nested dictionary has to have as first key the category and as second key the option, e.g.:

```
#create dictionary which sets the Q value for GA model to 2.0 and for the PK model to 4.0
input_dic={'GA' : {'Q_max' :  2.0}, 'PK' : {'Q_max' :  4.0}}
```

The complete default config is shown in Listing 1 and each parameter is explained in the following.

Global:

- GA_intercept_max: float, maximal intercept value of linear fit of GA
- GA_refit_if_interceptGreater0_OR_slopeGreater0: bool, if True, the intercept of the linear GA is fixed to 0 when intercept is > 0 or slope is greater 0 (=negative MSD)
- fix_to_offset: bool, if True, fix the EISF(Q=0) of all models to the value of the GA
- no_weighting_in_fit: bool, if True, fits are not weighted by errors
- offset_start_min_max: list, Fitting variable boundaries for offset [starting value, min value, max value], only taken into account if fix_to_offset = False
- print_report: bool, if True give a report example for each fit during the fitting process (helps to verbose)

GA:

- Q_max: float, maximal Q value of GA fit

PK:

- Q_max: float, maximal Q value of PK fit (same is taken for Do model, if used)
- beta_start-min-max: list, Fitting variable boundaries for beta [starting value, min value, max value]
- sigma_start-min-max: list, Fitting variable boundaries for sigma [starting value, min value, max value]

Yi:

- Q_max: float, maximal Q value of Yi fit
- msd_start-min-max: list, Fitting variable boundaries for msd [starting value, min value, max value], msd here from original paper = factor 6 instead of 3
- sigma_start-min-max: list, Fitting variable boundaries for sigma [starting value, min value, max value]

Do:

- use_fit_doster: bool, if True, Doster fit is also calculated
- doster_d_fixed: bool, if True, fix the d variable
- doster_d_val: float, value of fixed d variable

```
[Global]
GA_intercept_max=0.4
GA_refit_if_interceptGreater0_OR_slopeGreater0=False
fix_to_offset=True
no_weighting_in_fit=False
offset_start_min_max=[0.9, 0.1, 1.4918246976412703]
print_report=False
[GA]
Q_max=2.0
[PK]
Q_max=4.5
beta_start-min-max=[0.5, 0.01, 100.0]
sigma_start-min-max=[1.0, 0.01, 10.0]
[Yi]
Q_max=4.5
msd_start-min-max=[0.2, 1e-07, 5.0]
sigma_start-min-max=[0.6, 1e-07, 5.0]
[Do]
doster_d_fixed=False
doster_d_val=1.5
use_fit_doster=False
```

Listing 1: Default config file

4 How to use

The EINSfit class is designed to save each data set (consisting of multiple temperature values T and momentum transfer values Q) in a separate object. Therefore, you have to create a new object for each measured sample, e.g.:

```
from EINSfit import EINSfit
sample_1 = EINSfit('Sample_1_file_Elascan')
sample_2 = EINSfit('Sample_2_file_Elascan')
```

Each data set can be manipulated independently. If you want to use the standard config, the fitting process can be started directly via

```
sample_1.run_fit()
```

and saves the results in record number 0. If wished, another fit can be conducted on the same object by running the same command again. Of course, the fitting options (config dictionary) `config_dic` should be changed before to obtain different results. Each fitting result will be saved in a new record number and can be compared or saved later. If no record number, `record_nb`, is given during a function call, the results of the latest record will be shown. The config dictionary can be changed by defining an input dictionary `input_dic` and set it via `set_config_dic(input_dic)` or loading an existing config file via `read_config_file('config_file.ini')`. The `input_dic` has to be a nested dictionary. Only values with the same key as the default config dictionary will be replaced, e.g.:

```
#changes maximal considered Q value for GA model to 2.0
sample_1.set_config_dic({'GA' : {'Q_max' :  2.0},})
```

For more details about loading a config file or all the available options for the config, see section 3.

To make sure the same fitting parameters are used for each data set you can save the config dictionary, `config_dic`, either in a variable or a file and load its content into the other samples.

```
#local variable for config dic (can also be directly passed to function)
config_dic=sample_1.get_config_dic()
sample_2.set_config_dic(config_dic)
#or save config
sample_1.save_config_file(save_path='./') #saves config_dic in path './' with the name "
    config_file_SAMPLENAME.ini"
sample_2.load_config_file('./config_file'+sample_1.name+'.ini') #load created config file from
    sample_1
```

After the fitting process the result can be plotted via

```
#plot results of last fit
sample_1.plot_results()
```

This will plot the results of the last fit. In order to plot an result from a previous fit, the record number (integer number) can be passed via the optional keyword `record_nb`. To compare two different config settings of the same sample, the function `print_diff_in_config()` can be used and to compare the config settings of two different samples use the function `print_diff_between_two_dics()`.

```
#print difference between config of record_nb1 and record_nb2. As default the first (=0) and last
    (=-1) are taken.
sample_1.print_diff_in_config(record_nb1=0,record_nb2=-1)
#print difference between two config dictionaries of different samples:
EINSfit.print_diff_between_two_dics(sample_1.get_config_dic(),sample_2.get_config_dic())
```

In order to make it easier to compare the obtained fitting results a results dictionary is created after each fit. It can be accessed with the command `get_nice_results_dic()`.

```
#get results dictionary of last fit (record_nb=-1) and save it to variable results_dic
results_dic=sample_1.get_nice_results_dic(record_nb=-1)
```

This dictionary saves all important information in the following keys.

- `'EISF_T'` : all temperature values
- `'EISF_Q'` : all used Q values
- `'EISF_data'` : all used data for fits (2D array) [T,Q]
- `'EISF_data_err'` : error of all used data (2D array) [T,Q]
- `'EISF_data_log'` : logarithm of all used data (2D array) [T,Q]
- `'EISF_data_err_log'` : error of the logarithm of all used data (2D array) [T,Q]
- `'raw_data'` : contains a nested dictionary with the keys above which contain the raw data
- `'Q_range'` : contains a nested dictionary with the Q ranges used for each model
  - `'GA','PK','Yi','Do'` : key to choose model

        * 'Q_fit' : all Q values used for chosen model
        * 'Q_min' : minimal Q value used for chosen model
        * 'Q_max' : maximal Q value used for chosen model

- 'MSD3' : contains a nested dictionary with the MSD3 values and errors of each model
    - 'GA','PK','Yi','Do' : key to choose model
        * 'vals' : MSD3 values for each temperature (same order as 'EISF_T')
        * 'errors' : MSD3 errors for each temperature (same order as 'EISF_T')
- 'STD3' : contains a nested dictionary with the STD3 values and errors of PK and Yi model (see 'MSD3' key)
- 'redchi' : all reduced χ values for each temperature (same order as 'EISF_T')
- 'EISF(Q=0)' : all offset values for each temperature (same order as 'EISF_T')
- 'name' : name of sample

An example to compare the results with this dictionary is show in Listing 2, plot 1 and 2.
To save all the results of one record number, the function `sample_1.save_all(record_nb=-1)` can be used.

Examples for how to use the EINSfit class are shown in Listing 2. In addition, there are three examples which show more advanced methods:

A.Ex1: shows how to choose a subset of data by defining a `dic_data_to_use`

A.Ex2: shows how to load a saved data set (only data and config, not fitting results!)

A.Ex3: shows how to load user data defined in numpy arrays

Additional information can be found in the internal help of the EINSfit class.

```
from EINSfit import EINSfit  #import EINSfit class from EINfit.py file

############################
# Minimal examples for 1 or 2 data sets
#####

#####
# Ex1: 1 data set loaded from elascan file called EISF_sample1_q.dat and EISF_sample1_t.dat
my_sample=EINSfit('EISF_sample1')
my_sample.save_config_file()  #creates default config file
#my_sample.read_config_file() #reads the created config file above
my_sample.run_fit()           # runs the sample with the parameters given by config file
my_sample.plot_results()      #show plots of results
my_sample.save_all()          #save results
#####

#####
# Ex2: 2 data sets with config file = config-all.ini
my_samples=[]
results_dic=[]
sample_names=['EISF_sample1','EISF_sample2']
for sample in sample_names:
    my_samples.append(EINSfit(sample))
    my_samples[-1].read_config_file('config-all.ini')
    my_samples[-1].run_fit()
    my_samples[-1].save_all()
#save ordered results in list of dictionaries
for sample in my_samples:
    results_dic.append(sample.get_nice_results_dic())
#####
############################

############################
# examples to compare results of 2 data sets (from above created dictionary)
#####

import matplotlib.pyplot as plt
color_l=['blue','red']

#####
# plot 1, MSD or STD of different models
plt.figure()
model='GA'  # GA, PK, Yi or Do
para='MSD3' # MSD3 or STD3
for i,dic in enumerate(results_dic):
```

```
    plt.errorbar(dic['EISF_T'],dic[para][model]['vals'],dic[para][model]['errors'],label=dic['name'], color=color_l[i])
plt.title(model)
plt.legend()
#####

#####
# plot 2, compare fits at same/similar temperature
from EINSfit import take_closest_value #gives you index of value, which is closest to the desired value
import numpy as np

plt.figure()
t_wanted=360 # desired temperature value
model='PK'   # GA, PK, Yi or Do
q=np.linspace(0,5,100) # x axis
for i,sample in enumerate(my_samples):
    t_used,t_idx=take_closest_value(sample.used_T,t_wanted)
    plt.errorbar(sample.used_q,sample.used_data[t_idx,:],sample.used_data_err[t_idx,:],label=sample.name,color=color_l[i])
    plt.plot(q,sample.give_fit_value(q,t=t_idx,model=model),label='T=%iK'%t_used,color=color_l[i])
plt.title(model)
plt.legend()
#####
############################

############################
# Additional examples
#####

#personal save direcory
my_save_path='all-data'

#####
# A.Ex1: Change input data set:

# exampe for dic_data_to_use with no changes, values have to be floats
dic_data_to_use_example={'T_start': None, 'T_end': None, 'Q_min': None, 'Q_max': None, 'delete_specific_T-values_list': None, 'delete_specific_Q-values_list': None}
# example for dic_data_to_use with all temperature values, but without Q values smaller than 0.48A-1 and the maximal Q value allowed is 4.5A-1,
#                                 and Q value 3.1252 A-1 is deleted
dic_data_to_use_example2={'Q_min': 0.48, 'Q_max': 4.5, 'delete_specific_Q-values_list': [3.1252,],}

my_sample=EINSfit('EISF_sample1',name='sample1',data_type='elascan',dic_data_to_use=dic_data_to_use_example,save_dir_path=my_save_path)
#####

#####
# A.Ex2: load saved data set (only raw/used data and if wanted config file, but NOT results!)
my_sample=EINSfit(datafile=my_save_path,name='loaded sample1',data_type='save')
my_sample.read_config_file() #loads saved config file if wanted
#####

#####
# A.Ex3: data set created by user
data_example={'raw_data': np.array([[1,1],[2,2],[3,3]]),'raw_data_err': np.array([[1,1],[2,2],[3,3]]),'raw_T':np.array([100,200,300]),'raw_q':np.array([0.5,1.5])}
my_sample=EINSfit(datafile=data_example,name='created data',data_type='numpy_dic')
#####
############################
```

Listing 2: Example tasks (can be found in folder tasks/)

5 Help Doc

The documentation can be find in the git repository in the folder `doc/` and the example tasks shown in Listing 2 can be found in folder `tasks/`. Listing 3 shows the help of EINSfit class.

```
Help on class EINSfit in EINSfit:

EINSfit.EINSfit = class EINSfit(builtins.object)
 |  EINSfit.EINSfit(datafile, name=None, data_type='elascan', dic_data_to_use=None, save_dir_path=None)
 |
 |  Fits different EINS models (EISF vs Q) to one data set of one or multiple temperature scans.
 |  Data set has to be defined via a Path (string) datafile='your_elascan_baseName' or 'your_save_directory'.
 |  The datafile has to be
 |  - the prefix('your_elascan_BaseName') of the two elascan output files from LAMP (prefix+'_q.dat' and prefix+'_t.dat')
 |  or
 |  - the directory of your previously saved data set.
 |      --> This will ONLY load the raw input data + used input dictionary
 |      --> This will NOT load fit results or the configuration file (=config_dic), if wanted, load saved config file with read_config_file()
 |  or
 |  - data saved in a dictionary with entries 'raw_data','raw_data_err','raw_q','raw_T'
 |      --> data has to be a numpy array: numpy.ndarray
 |      --> 'raw_q' and 'raw_T' are 1D arrays
 |      --> 'raw_data' and 'raw_data_err' are 2D arrays, axis1=len(raw_T) and axis2=len(raw_q)
 |
 |  Parameters
 |  ----------
 |  datafile : string or dict, mandatory
 |      "data_type" == 'elascan'  : string = 'your_elascan_BaseName' (without '_q.dat' or '_t.dat')
 |      "data_type" == 'save'     : string = 'your_save_BaseDirectory'
 |      "data_type" == 'numpy_dic': dict   = {'raw_data': np.ndarray[Q,T],'raw_data_err': np.ndarray[Q,T],'raw_T': np.ndarray,'raw_q': np.ndarray}
 |
 |  name : string, optional if not "data_type" = 'save'
 |      Name you want to give your data set.
 |      ! Must be set if "data_type" = 'save'
 |
 |  data_type : 'elascan' or 'save' or 'numpy_dic', optional
 |      Defines your data input type.
 |      'elascan'   = load elascan output files from LAMP
 |      'save'      = load directory of your previously saved data set
 |      'numpy_dic' = load data dictionary which has to be defined in the input variable 'datafile'
 |
 |  dic_data_to_use : {'T_start': float, 'T_end': float, 'Q_min': float, 'Q_max': float, 'delete_specific_T-values_list': [], 'delete_specific_Q-values_list': [] } , optional
 |      Dictionary which defines the used data from the loaded data set.
 |      All values are optional, if set to None or not defined all values are used.
 |      'T_start' : first used temperature value (type: float)
 |      'T_end' : last used temperature value (type: float)
 |      'Q_min' : first used Q value (type: float)
 |      'Q_max' : last used Q value (type: float)
 |      'delete_specific_T-values_list' : list of T values which should be excluded, has to be the exact value! (type list)
 |      'delete_specific_Q-values_list' : list of Q values which should be excluded, has to be the exact value! (type list)
 |
 |
 |  save_dir_path : string, optional
 |      Defines where you want to save your data (Base directory).
 |
 |  Attributes
 |  ----------
 |  name : string, name of your data set
 |
 |  Readable Attributes (only a copy of the original variable is returned)
 |  --------------------
 |  config_dic : dict, dictionary of config for data fitting,
```

```
|      To change this dictionary, use set_config_dic() or read_config_file()
|      For a nice overview over this dictionary, use print_config()
|
|  raw_data_type : string, return the loaded data type ('elascan' or 'save')
|  raw_file_path : string, return loaded data set path
|
|  raw_T : numpy.ndarray, return raw temperature data
|  raw_q : numpy.ndarray, return raw Q data
|  raw_data : numpy.ndarray, return raw EISF data as 2D numpy.array with [T,Q]
|  raw_data_err : numpy.ndarray, return raw EISF data error as 2D numpy.array with [T,Q]
|
|  used_T : numpy.ndarray, return used temperature data
|  used_q : numpy.ndarray, return used Q data
|  used_data : numpy.ndarray, return used EISF data as 2D numpy.array with [T,Q]
|  used_data_err : numpy.ndarray, return used EISF data error as 2D numpy.array with [T,Q]
|  used_data_log : numpy.ndarray, return used log(EISF data) as 2D numpy.array with [T,Q]
|  used_data_err_log : numpy.ndarray, return used log(EISF data) error as 2D numpy.array with [T,Q]
|  used_Tmin : float, return first allowed T value in comparision to raw data
|  used_Tmax : float, return last  allowed T value in comparision to raw data
|  used_qmin : float, return first allowed Q value in comparision to raw data
|  used_qmax : float, return last  allowed Q value in comparision to raw data
|
|  Methods defined here:
|
|  __del__(self)
|      Remove created dictionary if it is empty.
|
|  __init__(self, datafile, name=None, data_type='elascan', dic_data_to_use=None, save_dir_path=None)
|      Initializes class object, for help see help(EINS_fit)
|
|  get_config_dic(self, record_nb=None)
|      Returns copy of config saved in config dictionary, either the current config or from a saved record.
|      Parameters:
|      record_nb : int, optional
|          Define from which record number you want to read the config (-1 = last record).
|          If None, current config is printed.
|
|  get_nice_results_dic(self, record_nb=-1, silent=False) -> dict
|      Returns nice dictionary with results for given record_nb.
|      Parameters
|      ----------
|      record_nb : int, optional
|          Define from which record number you want to have the results (-1 = last record).
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|
|  get_save_dir(self)
|
|  give_fit_value(self, x, t=0, model='GA', record_nb=-1, GA_lin=False)
|      Returns the y [=EISF(q)] value(s) to given x [=q] value(s) of requested model.
|      Parameters:
|      -----------
|      x : float / array (or list) of floats
|      t : int
|          Number of temperature set (0=first, len(self._used_T)=last)
|      model : 'GA' or 'PK' or 'Yi' or 'Do' or 'linAllQ', optional
|          Name of desired model.
|      record_nb : int, optional
|          Define from which record number you want to have the results. (-1 = last record).
|      GA_lin: bool, optional
|          If True, function gives values of linear fit defined via ln(EISF) vs Q**2, e.g. for such a plot:
|              ln(EISF(Q))=Q**2 * MSD + log(EISF(0))
|               --> ln(EISF(x)) = give_fit_value(x=x**2, GA_lin=True)
|          If False, definition as for other models:
|              EISF(Q)=exp(- Q**2 * MSD + EISF(0))
|               --> EISF(x) = give_fit_value(x=x, GA_lin=False)  (since internally x is squared)
|
|  load_lmfit_results_local(self) -> dict
```

```
|      Loads dictionary of lmfit results (pickle file) saved in default save path and returns the
    dictionary file.
|
|  load_nice_results_dic_local(self) -> dict
|      Loads dictionary of nice results (pickle or json file) saved in default save path and returns the
    dictionary file.
|
|  plot_results(self, record_nb=-1, save=False, close_all=False, save_path=None, silent=False,
    outputfile_type='png', outputfile_dpi=200)
|      Plots the results of the fitted data set.
|      Parameters:
|      -----------
|      record_nb : int, optional
|          Define from which record number you want to plot the results. (-1 = last record).
|      save : bool, optional
|          If True, saves the plots in the default save directory (can be changed with set_save_dir() )
|              or in path given in "save_path" parameter.
|      close_all : bool, optional
|          If True, closes all plotted figures after execution. Suggested if parameter "save" = True.
|      save_path : string, optional
|          Directory where plotted figures are saved. If None, the default save directory (can be changed
    with set_save_dir() ) is used.
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|      outputfile_type : string, optional
|          Define the type of your saved output, e.g. '.png', '.jpg', '.pdf'
|      outputfile_dpi : int, optional
|          Define the dpi (dots per inch) of your saved output, e.g. 200, 300, 600
|
|  print_config(self, record_nb=None)
|      Prints config saved in config dictionary, either the current config or from a saved record.
|      Parameters:
|      record_nb : int, optional
|          Define from which record number you want to read the config (-1 = last record).
|          If None, current config is printed.
|
|  print_diff_in_config(self, record_nb1=0, record_nb2=-1, all=False, record_nb_ref=0)
|      Prints the difference between the config of two records.
|      For differences between two different config dictionaries of different samples, use
    print_diff_between_two_dics()
|
|      Parameters:
|      -----------
|      record_nb1 : int, optional
|          Record number of first config to compare. ["0" = first config, "-1" = last config]
|      record_nb2 : int, optional
|          Record number of second config to compare. ["0" = first config, "-1" = last config]
|      all : bool, optional
|          Get differences of configs of all records. First record is the reference config.
|      record_nb_ref : int, optional
|          Record number of reference config -> all available configs are compared to this config. ["0" =
    first config, "-1" = last config]
|
|  print_nb_of_records(self)
|      Prints the number of records saved.
|
|  read_config_file(self, filename=None)
|      Reads the config from given file and overwrite config dictionary with new Values.
|      Parameters:
|      -----------
|      filename : string
|          Config file location.
|
|  run_fit(self)
|      Fits the data set.
|      Fits are done with the config defined in self.config_dic dictionary.
|      self.config_dic can be set via read_config_file() or set_config_dic().
|      The results and configurations are saved in a new record. To get the number of available records use
    : print_nb_of_records.
```

```
|
|  save_all(self, record_nb=-1, save_path=None, plot=True, silent=True)
|      Saves data set, config, results and if wanted also figures.
|      Parameters:
|      -----------
|      record_nb : int, optional
|          Define from which record number you want to save the results. (-1 = last record).
|      save_path : string, optional
|          Base directory where results are saved. If None, the default save directory (can be changed with
     set_save_dir() ) is used.
|      plot : bool, optional
|          If False, figures ar not plotted and are not saved.
|      silent : bool, optional
|          If False, all output is printed to the terminal.
|
|  save_config_file(self, record_nb=None, save_path=None, silent=False)
|      Saves the config dictionary to a file.
|      Parameters:
|      -----------
|      record_nb : int, optional
|          Define from which record number you want to have the config (-1 = last record).
|      save_path : string, optional
|          Directory where config file is saved (save_path / 'config_file_SAMPLENAME.ini'). If None, the
    default save directory (can be changed with set_save_dir() ) is used.
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|
|  save_input(self, save_path=None, silent=False)
|      Saves the raw data and if used data is different, the dictionary of the used data
|      Parameters:
|      -----------
|      save_path : string, optional
|          Directory where text files are saved. If None, the default save directory (can be changed with
    set_save_dir() ) is used.
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|
|  save_lmfit_results(self, record_nb=-1, save_path=None, silent=False)
|      Save dictionary of lmfit results of given record number as pickle file.
|
|      Parameters:
|      -----------
|      record_nb : int, optional
|          Define from which record number you want to plot the results. (-1 = last record).
|      save_path : string, optional
|          Directory where the pickle file is saved. If None, the default save directory (can be changed
    with set_save_dir() ) is used.
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|
|  save_nice_results_dic(self, record_nb=-1, file_type='json', save_path=None, silent=False)
|      Save dictionary of nice results of given record number as .pickle or .json file.
|
|      Parameters:
|      -----------
|      record_nb : int, optional
|          Define from which record number you want to plot the results. (-1 = last record).
|      file_type : 'pickle' or 'json', optional
|          Define the file type of the saved dictionary file.
|      save_path : string, optional
|          Directory where the pickle file is saved. If None, the default save directory (can be changed
    with set_save_dir() ) is used.
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|
|  save_results(self, record_nb=-1, save_path=None, silent=False)
|      Saves the results of the fitted data set to two text files (prefix+'.txt' and prefix+'-vals.txt').
    prefix=name_data_set + model_type
|      Parameters:
```

```
|      -----------
|      record_nb : int, optional
|          Define from which record number you want to plot the results. (-1 = last record).
|      save_path : string, optional
|          Directory where text files are saved. If None, the default save directory (can be changed with
    set_save_dir() ) is used.
|      silent : bool, optional
|          If True, no output is printed to the terminal.
|
|  set_config_dic(self, dic)
|      Set one or more values to config dictionary via a nested dictionary.
|
|  set_save_dir(self, save_dir_path)
|      Sets a new Base directory where data is saved as default.
|      Parameters
|      ----------
|      save_dir_path : string
|          Defines where you want to save your data (Base directory).
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  load_lmfit_results(loadfile) -> dict
|      Loads dictionary of lmfit results (pickle file) and returns the dictionary file.
|
|      Parameters:
|      -----------
|      loadfile : string
|          Filename of lmfit results dictionary with or without supported suffix.
|
|  load_nice_results_dic(loadfile) -> dict
|      Loads dictionary of nice results (pickle or json file) and returns the dictionary file.
|
|      Parameters:
|      -----------
|      loadfile : string
|          Filename of nice results dictionary with or without supported suffix.
|
|  print_diff_between_two_dics(d1, d2, as_string=False)
|      Prints the difference between two dictionaries d1 and d2; d1 and d2 can be interchanged.
|      Only works/tested with config_dic and fitting_dic
|
|      Parameters:
|      -----------
|      d1 : dict, first dictionary
|      d2 : dict, second dictionary
|      as_string : bool, optional
|          If True, function returns string, else the result is printed to stdout (normally terminal).
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  config_dic
|
|  name
|
|  raw_T
|
|  raw_data
|
|  raw_data_err
|
|  raw_data_type
```

```
|
|  raw_file_path
|
|  raw_q
|
|  used_T
|
|  used_Tmax
|
|  used_Tmin
|
|  used_data
|
|  used_data_err
|
|  used_data_err_log
|
|  used_data_log
|
|  used_q
|
|  used_qmax
|
|  used_qmin
```

Listing 3: EINSfit class help

References

[1] Dominik Zeller. "Investigations of new methods for studying molecular dynamics with elastic neutron scattering". PhD thesis. Université Grenoble Alpes, Mar. 13, 2019.

[2] D. Zeller et al. "Analysis of elastic incoherent neutron scattering data beyond the Gaussian approximation". *The Journal of Chemical Physics* 149.23 (Dec. 2018), p. 234908. DOI: 10.1063/1.5049938.

[3] J. Peters and G. R. Kneller. "Motional heterogeneity in human acetylcholinesterase revealed by a non-Gaussian model for elastic incoherent neutron scattering". *J Chem Phys* 139.16 (2013), p. 165102. ISSN: 1089-7690 (Electronic) 0021-9606 (Linking). DOI: 10.1063/1.4825199. URL: https://www.ncbi.nlm.nih.gov/pubmed/24182083.

[4] Gerald R. Kneller and Konrad Hinsen. "Quantitative model for the heterogeneity of atomic position fluctuations in proteins: A simulation study". *The Journal of Chemical Physics* 131.4 (2009), p. 045104. DOI: 10.1063/1.3170941. eprint: https://doi.org/10.1063/1.3170941. URL: https://doi.org/10.1063/1.3170941.

[5] Z. Yi et al. "Derivation of mean-square displacements for protein dynamics from elastic incoherent neutron scattering". *J Phys Chem B* 116.16 (2012), pp. 5028–36. ISSN: 1520-5207 (Electronic) 1520-5207 (Linking). DOI: 10.1021/jp2102868. URL: http://www.ncbi.nlm.nih.gov/pubmed/22471396.

[6] Torsten Becker and Jeremy C. Smith. "Energy resolution and dynamical heterogeneity effects on elastic incoherent neutron scattering from molecular systems". *Phys. Rev. E* 67 (2 Feb. 2003), p. 021904. DOI: 10.1103/PhysRevE.67.021904. URL: https://link.aps.org/doi/10.1103/PhysRevE.67.021904.

[7] W. Doster, S. Cusack, and W. Petry. "Dynamical transition of myoglobin revealed by inelastic neutron scattering". *Nature* 337.6209 (1989), pp. 754–6. ISSN: 0028-0836 (Print) 0028-0836 (Linking). DOI: 10.1038/337754a0. URL: http://www.ncbi.nlm.nih.gov/pubmed/2918910.