ImpedanceFitter Documentation

Release 1.0.0

Julius Zimmermann, Leonard Thiele

Apr 02, 2020

CONTENTS:

CHAPTER

ONE

MAIN MODULE

1.1 How it works

The script will cycle through all .TXT and .xlsx files in a selected directory and fit those files to a selected model. You can exclude certain files by using the excludeEnding flag.

The first possible step is to compensate the electrode polarization of the experiment by fitting the data to a cole-cole-model. After this fit it will fit the data to a selected model, this can either be the Double Shell model or the Single Shell model. If there is no electrode polarization to compensate for, you can also skip this. See *impedancefitter.main.Fitter.main()*

After the fit to one of the models has finished, the calculated values get written into the 'outfile.yaml' in the data-directory.

The following parameters need to be provided in yaml-files:

1.2 Units

Usually, SI units are used. However, some parameters have a very small numerical value. In this case, different units are used. These parameters are:

Parameter	Name in Script	Unit
c_0	c0	pF
c_f	cf	pF
C	C	pF
L	L	nH
τ	tau	ps

1.3 constants.yaml (or alternatively as input dict)

Parameter	Name in Script	Description
c_0	c0	air capacitance
c_f	cf	stray capacitance
p	p	volume fraction of cells
ε_cp	ecp	permittivity of cytoplasm
d_m	dm	membrane thickness
R_c	Rc	outer cell radius
only double shell model		
R_n	Rn	outer Radius of the nucleus
d_n	dn	thickness of the nuclear envelope
ε_np	enp	permittivity of nucleoplasm

Exemplary file:

```
c0 : 3.9e-13    # air capacitance
cf : 2.42532194241202e-13   # stray capacitance

Rc : 5.8e-6  # unit(m). cell radius in buffer, this should be changed according to␣
↪the chosed cell line
dm : 7.e-9    # thickness of cell membrane m
ecp : 60     # permittivity for cytoplasm
p : 0.075

Rn : 0.8 * constants['Rc']  # radius of nucleus
dn : 40.e-9   # thickness of nulear membrane
enp : 120    # Permittivity for nucleoplasm
```

1.4 cole_cole_input.yaml

Parameter	Name in Script	Description	Physical Boundaries
k	k	constant phase element parameter	has to be non 0, otherwise the function wil throw NAN or quit(1/k in the formula)
α	alpha	constant phase element exponent	$0 < \alpha < 1$
ε_l	epsi_l	low frequency permitivity	$el \geq 1$
ε_h	eh	high frequency permitivity	$eh \geq 1$
τ	tau	relaxation time	$\tau > 0$
a	a	exponent in formula for ε^*	$0 < a < 1$
σ	conductivity	low frequency conductivity	$\sigma > 0$

Exemplary file:

```
k: {value: 1.5e-7,  min: 1.e-7, max: 1.e-2, vary: True}
epsi_l: {value: 1000, min: 200, max: 3000, vary: True}
tau: {value: 5.e-7, min: 1.e-8, max: 1.e-3, vary: True}
a: {value: 0.9, min: 0.8, max: 1.0, vary: True}
alpha :   {value: 0.3, min: 0., max: .5, vary: True}
conductivity : {value: 0.05, min: 0.05, max: 0.2, vary: True}
eh :  {value: 78., min: 60., max: 85., vary: True}
```

With the *vary* flag, one can choose whether a variable should be included in the fitting procedure or fixed.

1.5 single_shell_input.yaml

Parameter	Name in Script	Description	Physical Boundaries
σ_{m}	km	conductivity of the membrane	$\sigma_{\mathrm{m}} > 0$
ε_{m}	em	pemitivity of the membrane	$\varepsilon_{\mathrm{m}} \geq 1$
σ_{cp}	kcp	conductivity of the cytoplasm	$\sigma_{\mathrm{cp}} > 0$
σ_{med}	kmed	conductivity of the supernatant($\varepsilon_{\mathrm{med}} = \varepsilon_{\mathrm{h}}$ from cole cole fit)	$\sigma_{\mathrm{med}} > 0$

Exemplary file:

```
km: {value: 1.e-8, min: 1.e-12, max: 1e-2, vary: True}
em: {value: 11, min: 1., max: 50., vary: True}
kcp: {value: 0.4, min: 0.1, max: 2., vary: True}
kmed: {value: 0.15, min: 0.0001, max: 1.0, vary: True}
```

With the *vary* flag, one can choose whether a variable should be included in the fitting procedure or fixed.

Note:

If you run without electrode polarization, you must include a line for emed as here:

```
emed: {value: 78, min: 70, max: 80, vary: True}
```

1.6 double_shell_input.yaml

Parameter	Name in Script	Description	Physical Boundaries
σ_{m}	km	conductivity of the membrane	$\sigma_{\mathrm{m}} > 0$
ε_{m}	em	pemitivity of the membrane	$\varepsilon_{\mathrm{m}} \geq 1$
σ_{cp}	kcp	conductivity of the cytoplasm	$\sigma_{\mathrm{cp}} > 0$
$\varepsilon_{\mathrm{ne}}$	ene	permitivity of the nulear envelope	$\varepsilon_{\mathrm{ne}} \geq 1$
σ_{ne}	kne	conductivity of the nuclear envelope	$\sigma_{\mathrm{ne}} > 0$
σ_{np}	knp	conductivity of the nucleoplasm	$\sigma_{\mathrm{np}} > 0$
σ_{med}	kmed	conductivity of the supernatant($\varepsilon_{\mathrm{med}} = \varepsilon_{\mathrm{h}}$ from cole cole fit)	$\sigma_{\mathrm{med}} > 0$

Exemplary file:

```
km: {value: 300.e-6, min: 1.e-8, max: 5.e-4, vary: True}
em: {value: 11., min: 1., max: 20., vary: True}
kcp: {value: 0.4, min: 0.1, max: 2., vary: True}
ene: {value: 50, min: 1., max: 50., vary: True}
kne: {value: 2.e-3, min: 1.e-8, max: 1.e-1, vary: True}
knp: {value: .8, min: 0.1, max: 1., vary: True}
kmed: {value: 0.05, min: 0.04, max: 0.3, vary: True}
```

With the *vary* flag, one can choose whether a variable should be included in the fitting procedure or fixed.

Note:

If you run without electrode polarization, you must include a line for emed as here:

```
emed: {value: 78, min: 70, max: 80, vary: True}
```

1.7 possible values

In Ermolina, I., Polevaya, Y., & Feldman, Y. (2000). Analysis of dielectric spectra of eukaryotic cells by computer modeling. European Biophysics Journal, 29(2), 141–145. https://doi.org/10.1007/s002490050259, there have been reported upper/lower limits for certain parameters. They could act as a first guess for the bounds of the optimization method.

Parameter	lower limit	upper limit
ε_m	1.4	16.8
σ_m	8e-8	5.6e-5
ε_cp	60	77
σ_cp	0.033	1.1
ε_ne	6.8	100
σ_ne	8.3e-5	7e-3
ε_np	32	300
σ_np	0.25	2.2
R	3.5e-6	10.5e-6
R_n	2.95e-6	8.85e-6
d	3.5e-9	10.5e-9
d_n	2e-8	6e-8

class impedancefitter.main.**Fitter**(*directory=None*, *parameters=None*, *write_output=True*, *compare_CPE=False*, *fileList=None*, ***kwargs*)

The main fitting object class. The chosen fitting routine is applied to all files in the chosen directory.

directory: {string, optional, path to data directory} provide directory if you run code from directory different to data directory

parameters: {dict, optional, needed parameters} provide parameters if you do not want to use a yaml file (for instance in parallel UQ runs).

write_output: {bool, optional} decide if you want to dump output to file

compare_CPE: {bool, optional} decide if you want to compare to case without CPE

model: {string, 'SingleShell' OR 'DoubleShell'} currently, you can choose between SingleShell and DoubleShell.

solvername [{string, name of solver}] choose among global solvers from lmfit: basinhopping, differential_evolution, ...; local solvers: levenberg-marquardt, nelder-mead, least-squares. See also lmfit documentation.

inputformat: {string, 'TXT' OR 'XLSX'} currently only TXT and Excel files with a certain formatting (impedance in two columns, first is real part, second imaginary part, is accepted).

LogLevel: {string, optional, 'DEBUG' OR 'INFO' OR 'WARNING'} choose level for logger. Case DEBUG: the script will output plots after each fit, case INFO: the script will output results from each fit to the console.

excludeEnding: {string, optional} for ending that should be ignored (if there are files with the same ending as the chosen inputformat)

minimumFrequency: {float, optional} if you want to use another frequency than the minimum frequency in the dataset.

maximumFrequency: {float, optional} if you want to use another frequency than the maximum frequency in the dataset.

solver_kwargs: {dict, optional} Contains infos for the solver. find possible kwargs in the lmfit documentation.

data_sets: {int, optional} Use only a certain number of data sets instead of all in directory.

current_threshold: {float, optional} use only for data from E4980AL LCR meter to check current

omega: list Contains frequencies.

Z: list Contains corresponding impedances.

protocol: None or string Choose 'Iterative' for repeated fits with changing parameter sets, customized approach. If not specified, there is always just one fit for each data set.

emcee_main(*electrode_polarization=True*, *lnsigma=None*, *inductivity=False*, *high_loss=False*)
Main function that iterates through all data sets provided.

electrode_polarization: True or False Switch on whether to account for electrode polarization or not. Currently, only a CPE correction is possible.

lnsigma: dict Add lnsigma parameter with custom initial value and bounds to method.

fit_emcee_models(*filename*)
fit the data to the cole_cole_model first (compensation of the electrode polarization) and then to the defined model.

fit_to_RC(*omega*, *Z*)
Fit data to full RC model without known c0.

fit_to_cole_cole(*omega*, *Z*)
Fit data to cole-cole model.

If the `protocol` is specified to be 'Iterative', the data is fitted to the Cole-Cole-Model twice. Then, in the second fit, the parameters 'conductivity' and 'eh' are being fixed.

The initial parameters and boundaries for the variables are read from the .yaml file in the directory of the python script. see also: *impedancefitter.cole_cole.cole_cole_model()*

fit_to_cole_cole_r(*omega*, *Z*)
Fit data to cole-cole model.

The initial parameters and boundaries for the variables are read from the .yaml file in the directory of the python script. see also: *impedancefitter.cole_cole.cole_cole_model()*

fit_to_double_shell (*omega*, *Z*, *k_fit*, *alpha_fit*)
k_fit and alpha_fit are the determined values in the cole-cole-fit

If `protocol` is equal to *Iterative*, the following procedure is applied:

1. 1st Fit: the parameters k and α are fixed(coming from cole-cole-fit), and the data is fitted against the double-shell-model. To be determined in this fit: $\sigma_{\mathrm{sup}}/k_{\mathrm{med}}$.
2. 2nd Fit: the parameters k, α and σ_{sup} are fixed and the data is fitted again. To be determined in this fit: $\sigma_{\mathrm{m}}, \varepsilon_{\mathrm{m}}, \sigma_{\mathrm{cp}}$.
3. 3rd Fit: the parameters k, α, σ_{sup}, σ_{m} and ε_{m} are fixed and the data is fitted again. To be determined in this fit: σ_{cp}.
4. last Fit: the parameters k, α, $\sigma_{\mathrm{sup}}, \sigma_{\mathrm{ne}}, \sigma_{\mathrm{np}}, \sigma_{\mathrm{m}}, \varepsilon_{\mathrm{m}}$ are fixed. To be determined in this fit: $\varepsilon_{\mathrm{ne}}, \sigma_{\mathrm{ne}}, \varepsilon_{\mathrm{np}}, \sigma_{\mathrm{np}}$.

See also: *impedancefitter.double_shell.double_shell_model()*

fit_to_rc (*omega*, *Z*)
Fit data to RC model.

fit_to_single_shell (*omega*, *Z*, *k_fit*, *alpha_fit*)
if `protocol` is *Iterative*, the conductivity of the medium is determined in the first run and then fixed. See also: *impedancefitter.single_shell.single_shell_model()*

fit_to_suspension_model (*omega*, *Z*)
fit data to pure suspension model to compare to cole-cole based correction. See also: *impedancefitter.cole_cole.suspension_model()*

main (*protocol=None*, *electrode_polarization_fit=False*, *electrode_polarization=False*, *inductivity=False*, *high_loss=False*)
Main function that iterates through all data sets provided.

protocol: None or string Choose 'Iterative' for repeated fits with changing parameter sets, customized approach. If not specified, there is always just one fit for each data set.

electrode_polarization_fit: True or False Switch on whether to account for electrode polarization or not. Currently, only a CPE correction is possible. If True, a Cole-Cole model is used for fitting.

electrode_polarization: True or False Switch on whether to account for electrode polarization or not. Currently, only a CPE correction is possible.

prepare_txt ()
Function for txt files that are currently supported. The section for "TRACE: A" is used, also the number of skiprows_txt needs to be aligned, if there is a deviating TXT-format.

process_data_from_file (*filename*)
fit the data to the cole_cole_model first (compensation of the electrode polarization) and then to the defined model.

process_fitting_results (*filename*)
function writes output into yaml file to prepare statistical analysis of the result

readin_Data_from_file (*filename*, *max_rows*)
Data from txt files get reads in, returns array with omega and complex-valued impedance Z

select_and_solve (*solvername*, *residual*, *params*, *args*)
selects the needed solver and minimizes the given residual

impedancefitter.main.**logger = <Logger impedancefitter-logger (WARNING)>**

We use an own logger for the impedancefitter module.

CHAPTER

TWO

UTILITITES

impedancefitter.utils.**add_stray_capacitance**(*omega*, *Zdut*, *cf*)

add stray capacitance to impedance. The assumption made is that the stray capacitance is in parallel to the DUT. Hence, the total impedance is

$$Z_{\mathrm{total}} = \left(\frac{1}{Z_{\mathrm{DUT}}} + \frac{1}{Z_{\mathrm{stray}}} \right)^{-1}$$

Note: The stray capacitance is always given in pF for numerical reasons! It is checked if the stray capacitance is greater than 0 with an absolute tolerance of 10^{-5} pF.

omega: double or ndarray of double frequency array

Zdut: complex or array of complex impedance array, data of DUT

cf: double stray capacitance to be accounted for

impedancefitter.utils.**available_file_format**()

Todo: Update documentation here.

impedancefitter.utils.**available_models**()

return list of available models

impedancefitter.utils.**compare_to_data**(*omega*, *Z*, *Z_fit*, *subplot=None*, *title=''*, *show=True*, *save=False*)

plots the relative difference of the fitted function to the data

omega: double or ndarray of double frequency array

Z: complex or array of complex impedance array, experimental data or data to compare to

Z_fit: complex or array of complex impedance array, fit result

subplot: optional decide whether it is a new figure or a subplot. Default is None (yields new figure). Otherwise it can be an integer to denote the subfigure.

title: str, optional title of plot. Default is an empty string.

show: bool, optional show figure (default is True). Only has an effect when *subplot* is None.

save: bool, optional save figure to pdf (default is False). Name of figure starts with *title*.

impedancefitter.utils.**get_labels**()
return the labels for every parameter in LaTex code.

labels: dict dictionary with parameter names as keys and LaTex code as values.

impedancefitter.utils.**model_information**(*model*)

model: string name of model, see available models from *available_models()*.

description: string Description what is behind the model, where it can be found in literature and where it is implemented.

Todo: Update documentation here.

---

impedancefitter.utils.**parameter_names**(*model*, *ep_cpe=False*, *ind=False*, *loss=False*, *stray=False*)
Get the right order of parameters for a certain model. Is needed to pass the parameters properly to each model function call.

model: string name of the model

ep_cpe: bool, optional switch on electrode polarization correction by CPE

ind: bool, optional switch on correction of inductive effects by LR model

loss: bool, optional switch on correction of inductive effects by LCR model for high loss materials

stray: bool, optional switch on stray capacitance

impedancefitter.utils.**plot_dielectric_properties**(*omega*, *Z*, *c0*, *Z_comp=None*)

omega: double or ndarray of double frequency array

Z: complex or array of complex impedance array

c0: double unit capacitance of device

Z_comp: optional complex-valued impedance array. Might be used to compare the properties of two data sets.

impedancefitter.utils.**plot_results**(*omega*, *Z*, *Z_fit*, *title*, *show=True*, *save=False*)
Plot the *result* and compare it to data *Z*. Generates 4 subplots showing the real and imaginary parts over the frequency; a Nyquist plot of real and negative imaginary part and the relative differences of real and imaginary part as well as absolute value of impedance.

omega: double or ndarray of double frequency array

Z: complex or array of complex impedance array, experimental data or data to compare to

Z_fit: complex or array of complex impedance array, fit result

title: str title of plot

show: bool, optional show figure (default is True)

save: bool, optional save figure to pdf (default is False). Name of figure starts with *title*.

impedancefitter.utils.**return_diel_properties**(*omega*, *Z*, *c0*)
return relative permittivity and conductivity from impedance spectrum Use that the impedance is

$$Z = (j\omega\varepsilon^*)^{-1},$$

where ε^* is the complex permittivity (see for instance the paper with DOI 10.1063/1.1722949 for further explanation).

The relative permittivity is the real part of ε^* divided by the vacuum permittivity and the conductivity is the imaginary part times the frequency.

omega: double or ndarray of double frequency array

Z: complex or array of complex impedance array

c0: double unit capacitance of device

eps_r: double relative permittivity

conductivity: double conductivity in S/m

impedancefitter.utils.**set_parameters**(*modelName*, *parameterdict=None*, *ep_cpe=False*, *ind=False*, *loss=False*, *stray=False*)

modelName: string name of the model. must be one of those listed in `available_models()`

parameterdict: optional a dictionary containing parameters for model with *min*, *max*, *vary* info for LMFIT.

ep_cpe: bool, optional switch on electrode polarization correction by CPE

ind: bool, optional switch on correction of inductive effects by LR model

loss: bool, optional switch on correction of inductive effects by LCR model for high loss materials

stray: bool, optional switch on stray capacitance

params: Parameters LMFIT *Parameters*.

CHAPTER

THREE

INPUT ROUTINES

impedancefitter.readin.**readin_Data_from_collection**(*filepath*, *fileformat*, *minimumFrequency=None*, *maximumFrequency=None*)

read in data collection from Excel or CSV file that is structured like: frequency, real part of impedance, imaginary part of impedance there may be many different sets of impedance data, i.e. there may be more columns with the real and the imaginary part. Then, the frequencies column must not be repeated.

filepath: string Provide the full filepath

fileformat: string Provide fileformat. Possibilities are 'XLSX' and 'CSV'.

minimumFrequency: optional Provide a minimum frequency. All values below this frequency will be ignored.

maximumFrequency: optional Provide a maximum frequency. All values above this frequencies will be ignored.

omega: array of doubles frequency array

zarray: array of complex contains collection of impedance spectra.

impedancefitter.readin.**readin_Data_from_csv_E4980AL**(*filepath*, *minimumFrequency=None*, *maximumFrequency=None*, *current_threshold=None*)

read in data that is structured like: frequency, real part of impedance, imaginary part of impedance, voltage, current

---

Note: There is always only one data set in a file.

---

filepath: string Provide the full filepath

minimumFrequency: optional Provide a minimum frequency. All values below this frequency will be ignored.

maximumFrequency: optional Provide a maximum frequency. All values above this frequencies will be ignored.

current_threshold: optional Provides a current that the device had to pass through the sample. This threshold has to be met with 1% accuracy.

omega: array of doubles frequency array

zarray: array of complex contains collection of impedance spectra.

impedancefitter.readin.**readin_Data_from_file**(*filepath*, *skiprows_txt*, *skiprows_trace*, *trace_b*, *minimumFrequency=None*, *maximumFrequency=None*)

Data from txt files get reads in, returns array with omega and complex-valued impedance Z. The TXT files may contain two traces; only one of them is read in.

filepath: string Provide the full filepath

skiprows_txt: int header rows inside the *.txt file

skiprows_trace: int lines between traces blocks

trace_b: string Flag for beginning of second trace in data.

fileformat: string Provide fileformat. Possibilities are 'XLSX' and 'CSV'.

minimumFrequency: optional Provide a minimum frequency. All values below this frequency will be ignored.

maximumFrequency: optional Provide a maximum frequency. All values above this frequencies will be ignored.

omega: array of doubles frequency array

Z: array of complex contains impedance spectrum

CHAPTER

FOUR

CIRCUIT ELEMENTS

impedancefitter.elements.**Z_C**(*omega*, *C*)
capacitor impedance

omega: double or array of double list of frequencies

C: double capacitance of capacitor

impedancefitter.elements.**Z_CPE**(*omega*, *k*, *alpha*)
CPE impedance

$$Z_{\mathrm{CPE}} = k^{-1} j\omega^{-\alpha}$$

omega: double or array of double list of frequencies

k: double CPE factor

alpha: double CPE phase

impedancefitter.elements.**Z_in**(*omega*, *L*, *R*)
Lead inductance of wires connecting DUT. Described for instance in 10.1002/cmr.b.21318

omega: double or array of double list of frequencies

L: double inductance of coil

R: double resistance of resistor

impedancefitter.elements.**Z_loss**(*omega*, *L*, *C*, *R*)
impedance for high loss materials, where LCR are in parallel. Described for instance in 10.1002/cmr.b.21318.

omega: double or array of double list of frequencies

L: double inductance of coil

C: double capacitance of capacitor

R: double resistance of resistor

impedancefitter.elements.**Z_sus**(*omega*, *es*, *kdc*, *c0*)
impedance of suspension as used in paper with DOI: 10.1063/1.4737121

omega: double or array of double list of frequencies

es: complex complex valued permittivity, check e.g. :func:e_sus

kdc: double conductivity

c0: double unit capacitance

impedancefitter.elements.**Z_w**(*omega*, *Aw*)

Warburg element

$$Z_{\mathrm{W}} = A_{\mathrm{W}} \frac{1-j}{\sqrt{\omega}}$$

omega: double or array of double list of frequencies

A_w: double Warburg coefficient

impedancefitter.elements.**e_sus**(*omega*, *eh*, *el*, *tau*, *a*)

Complex permitivity after Cole-Cole. See their paper with DOI: 10.1063/1.1750906 Difference: the exponent $1-\alpha$ is here named *a*.

omega: double or array of double list of frequencies

eh: double value for ε_∞

el: double value for ε_0

tau: double value for τ_0

a: double value for $1-\alpha$

CHAPTER

FIVE

COLE-COLE MODEL

impedancefitter.cole_cole.**cole_cole_model**(*omega*, *c0*, *cf*, *el*, *tau*, *a*, *kdc*, *eh*, *k=None*, *alpha=None*, *L=None*, *C=None*, *R=None*)

function holding the cole_cole_model equations, returning the calculated impedance Equations for calculations:

$$Z_{\mathrm{ep}} = k^{-1} j\omega^{-\alpha}$$

$$\varepsilon_{\mathrm{s}} = \varepsilon_{\mathrm{h}} + \frac{\varepsilon_{\mathrm{l}} - \varepsilon_{\mathrm{h}}}{1 + (j\omega\tau)^{a}}$$

$$Z_{\mathrm{s}} = \frac{1}{j\varepsilon_{\mathrm{s}}\omega c_0 + \frac{\sigma_{\mathrm{dc}} c_0}{\varepsilon_0} + j\omega c_{\mathrm{f}}}$$

$$Z_{\mathrm{fit}} = Z_{\mathrm{s}} + Z_{\mathrm{ep}}$$

find more details in formula 6 and 7 of https://ieeexplore.ieee.org/document/6191683

impedancefitter.cole_cole.**cole_cole_residual**(*params*, *omega*, *data*)

compute difference between data and model. Attention: *c0* and *cf* in terms of pF, *tau* in ps

impedancefitter.cole_cole.**get_cole_cole_impedance**(*omega*, *params*)

Provide the angular frequency as well as the resulting parameters from the fitting procedure. The dictionary *params* is processed.

impedancefitter.cole_cole.**plot_cole_cole**(*omega*, *Z*, *result*, *filename*)

plot results of cole-cole model and compare fit to data.

impedancefitter.cole_cole.**suspension_model**(*omega*, *c0*, *cf*, *el*, *tau*, *a*, *kdc*, *eh*)

Simple suspension model using `impedancefitter.utils.Z_sus()`

impedancefitter.cole_cole.**suspension_residual**(*params*, *omega*, *data*)

use the plain suspension model and calculate the residual (the difference between data and fitted values) Attention: *c0* and *cf* in terms of pF, *tau* in ps

CHAPTER

SIX

SINGLE SHELL MODEL

impedancefitter.single_shell.**get_single_shell_impedance**(*omega*, *params*)
Provide the angular frequency as well as the result from the fitting procedure. The dictionary *params* is processed.

impedancefitter.single_shell.**plot_single_shell**(*omega*, *Z*, *result*, *filename*)
plot the real part, imaginary part vs frequency and real vs. imaginary part

impedancefitter.single_shell.**single_shell_model**(*omega*, *em*, *km*, *kcp*, *ecp*, *kmed*, *emed*, *p*, *c0*, *cf*, *dm*, *Rc*)
Equations for the single-shell-model(ν_1 is calculated like in the double-shell-model):

$$\varepsilon^*_{\mathrm{cell}} = \varepsilon^*_{\mathrm{m}} * \frac{(2*(1-\nu_1)+(1+2*\nu_1)*E_1}{((2+\nu_1)+(1--\nu_1)*E_1}$$

$$E_1 = \frac{\varepsilon^*_{\mathrm{cp}}}{\varepsilon^*_{\mathrm{m}}}$$

$$\varepsilon^*_{\mathrm{sus}} = \varepsilon^*_{\mathrm{med}} * \frac{2*(1-p)+(1+2*p)*E_0}{(2+p)+(1-p)*E_0}$$

$$E_0 = \frac{\varepsilon^*_{\mathrm{cell}}}{\varepsilon^*_{\mathrm{med}}}$$

impedancefitter.single_shell.**single_shell_residual**(*params*, *omega*, *data*)
calculates the residual for the single-shell model, using the single_shell_model.

CHAPTER

SEVEN

DOUBLE SHELL MODEL

impedancefitter.double_shell.**double_shell_model**(*omega*, *km*, *em*, *kcp*, *ecp*, *ene*, *kne*, *knp*, *enp*, *kmed*, *emed*, *p*, *c0*, *cf*, *dm*, *Rc*, *dn*, *Rn*)

Equations for the double-shell-model:

$$\varepsilon^*_{\mathrm{mix}} = \varepsilon^*_{\mathrm{sup}} \frac{(2\varepsilon^*_{\mathrm{sup}} + \varepsilon^*_{\mathrm{c}}) - 2p(\varepsilon^*_{\mathrm{sup}} - \varepsilon^*_{\mathrm{c}})}{(2\varepsilon^*_{\mathrm{sup}} + \varepsilon^*_{\mathrm{c}}) + p(\varepsilon^*_{\mathrm{sup}} - \varepsilon^*_{\mathrm{c}})}$$

$$\varepsilon^*_{\mathrm{c}} = \varepsilon^*_{\mathrm{m}} \frac{2(1-\nu_1) + (1+2\nu_1)E_1}{(2+\nu_1) + (1-\nu_1)E_1}$$

$$E_1 = \frac{\varepsilon^*_{\mathrm{cp}}}{\varepsilon^*_{\mathrm{m}}} \frac{2(1-\nu_2) + (1+2\nu_2)E_2}{(2+\nu_2) + (1-\nu_2)E_2}$$

$$E_2 = \frac{\varepsilon^*_{\mathrm{ne}}}{\varepsilon^*_{\mathrm{cp}}} \frac{2(1-\nu_3) + (1+2\nu_3)E_3}{(2+\nu_3) + (1-\nu_3)E_3}$$

$$E_3 = \frac{\varepsilon^*_{\mathrm{np}}}{\varepsilon^*_{\mathrm{ne}}}$$

with $R \hat{=}$ outer cell Radius; $R_{\mathrm{n}} \hat{=}$ outer Radius of the nucleus; $d \hat{=}$ thickness of the membrane

$$\nu_1 = \left(1 - \frac{d}{R}\right)^3$$

$$\nu_2 = \left(\frac{R_{\mathrm{n}}}{R-d}\right)^3$$

$$\nu_3 = \left(1 - \frac{d_{\mathrm{n}}}{R_{\mathrm{n}}}\right)^3$$

impedancefitter.double_shell.**double_shell_residual**(*params*, *omega*, *data*)

data is Z

impedancefitter.double_shell.**get_double_shell_impedance**(*omega*, *params*)

Provide the angular frequency as well as the result from the fitting procedure. The dictionary *params* is processed.

impedancefitter.double_shell.**plot_double_shell**(*omega*, *Z*, *result*, *filename*)

plot the real and imaginary part of the impedance vs. the frequency and real vs. imaginary part

CHAPTER

EIGHT

POST PROCESSING

class impedancefitter.postprocess.**PostProcess**(*model*, *electrode_polarization=True*, *yamlfile=None*)

This class provides the possibility, to analyse the statistics of the fitted data. The fitting results are read in from the outfile.

model: {string, DoubleShell OR SingleShell} define, which model has been used

electrode_polarization: {bool, default True} set to false if not used

yamlfile: {string, default None} define path to yamlfile or use current working directory

best_model_bic(*parameter*, *distributions*)

Test, which distribution models your data best based on the Bayesian information criterion (see https://openturns.github.io/openturns/master/user_manual/_generated/openturns.FittingTest_BestModelBIC.html#openturns.FittingTest_BestModelBIC). suitable for small samples

Parameters **distributions** – list of strings like: `['Normal', 'Uniform']`

best_model_chisquared(*parameter*, *distributions*)

Test, which distribution models your data best based on the chisquared test (see https://openturns.github.io/openturns/master/user_manual/_generated/openturns.FittingTest_BestModelChiSquared.html).

Parameters **distributions** – list of strings like: `['Normal', 'Uniform']`

best_model_kolmogorov(*parameter*, *distributions*)

Test, which distribution models your data best based on the kolmogorov test (see https://openturns.github.io/openturns/master/user_manual/_generated/openturns.FittingTest_BestModelKolmogorov.html#openturns.FittingTest_BestModelKolmogorov). suitable for small samples

Parameters **distributions** – list of strings like: `['Normal', 'Uniform']`

fit_to_histogram_distribution(*parameter*)

Generate histogram from results.

Parameters **parameter** – string, parameter that is to be fitted.

fit_to_normal_distribution(*parameter*)

Fit results for to normal distribution.

Parameters **parameter** – string, parameter that is to be fitted.

plot_histograms()

Plot histograms for all determined parameters. fails if values are too close to each other

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

R

S

Z