# eisfit Documentation

**Name**

**Jun 13, 2018**

# Contents

# CHAPTER 1

# Examples

# CHAPTER 2

# Fitting

`eisfit.fitting.`**`buildCircuit`**(*circuit*, *parameters*, *frequencies*)

transforms a circuit, parameters, and frequencies into a string that can be evaluated

**Parameters**

**circuit** [str]

**parameters** [list of floats]

**frequencies** [list of floats]

**Returns**

**eval_string** [str] Python expression for calculating the resulting fit

`eisfit.fitting.`**`circuit_fit`**(*frequencies*, *impedances*, *circuit*, *initial_guess*, *algorithm='leastsq'*, *bounds=None*)

Main function for fitting an equivalent circuit to data

**Parameters**

**frequencies** [numpy array] Frequencies

**impedances** [numpy array of dtype 'complex128'] Impedances

**circuit** [string] string defining the equivalent circuit to be fit

**initial_guess** [list of floats] initial guesses for the fit parameters

**algorithm: string** Name of algorithm to pass to scipy.optimize.minimize or to instantiate scipy.optimize.leastsq

**Returns**

**p_values** [list of floats] best fit parameters for specified equivalent circuit

**p_errors** [list of floats] error estimates for fit parameters

### Notes

Need to do a better job of handling errors in fitting. Currently, an error of -1 is returned.

`eisfit.fitting.`**`computeCircuit`**(*circuit*, *parameters*, *frequencies*)
evaluates a circuit string for a given set of parameters and frequencies

**Parameters**

**circuit** [string]

**parameters** [list of floats]

**frequencies** [list of floats]

**Returns**

**array of floats**

`eisfit.fitting.`**`residuals`**(*param*, *Z*, *f*, *circuit*)
Calculates the residuals between a given circuit/parameters (fit) and *Z/f* (data). Minimized by scipy.leastsq()

**Parameters**

**param** [array of floats] parameters for evaluating the circuit

**Z** [array of complex numbers] impedance data being fit

**f** [array of floats] frequencies to evaluate

**circuit** [str] string defining the circuit

**Returns**

**residual** [ndarray] returns array of size 2*len(f) with both real and imaginary residuals

`eisfit.fitting.`**`rmse`**(*a*, *b*)
A function which calculates the root mean squared error between two vectors.

### Notes

$$RMSE = \sqrt{\frac{1}{n}(a-b)^2}$$

`eisfit.fitting.`**`valid`**(*circuit*, *param*)
checks validity of parameters

**Parameters**

**circuit** [string] string defining the circuit

**param** [list] list of parameter values

**Returns**

**valid** [boolean]

### Notes

All parameters are considered valid if they are greater than zero – except for E2 (the exponent of CPE) which also must be less than one.

# CHAPTER 3

# Circuits

**class** eisfit.circuits.**BaseCircuit**(*initial_guess=None*, *name=None*, *algorithm='leastsq'*, *bounds=None*)

Base class for equivalent circuit models

### Methods

| | |
|---|---|
| *fit*(frequencies, impedance) | Fit the circuit model |
| *plot*([f_data, Z_data, CI]) | a convenience method for plotting Nyquist plots |
| *predict*(frequencies) | Predict impedance using a fit equivalent circuit model |

**fit**(*frequencies*, *impedance*)

Fit the circuit model

**Parameters**

**frequencies: numpy array** Frequencies

**impedance: numpy array of dtype 'complex128'** Impedance values to fit

**Returns**

**self: returns an instance of self**

**plot**(*f_data=None*, *Z_data=None*, *CI=True*)

a convenience method for plotting Nyquist plots

**Parameters**

**f_data: np.array of type float** Frequencies of input data (for Bode plots)

**Z_data: np.array of type complex** Impedance data to plot

**CI: boolean** Include bootstrapped confidence intervals in plot

**Returns**

**ax: matplotlib.axes** axes of the created nyquist plot

**predict** (*frequencies*)
Predict impedance using a fit equivalent circuit model

**Parameters**

**frequencies: numpy array** Frequencies

**Returns**

**impedance: numpy array of dtype 'complex128'** Predicted impedance

**class** eisfit.circuits.**DefineCircuit** (*circuit=None*, ***kwargs*)

### Methods

| | |
|---|---|
| fit(frequencies, impedance) | Fit the circuit model |
| plot([f_data, Z_data, CI]) | a convenience method for plotting Nyquist plots |
| predict(frequencies) | Predict impedance using a fit equivalent circuit model |

**class** eisfit.circuits.**FlexiCircuit** (*max_elements=None*, *generations=2*, *popsize=30*, *initial_guess=None*)

### Methods

| | |
|---|---|
| *fit*(frequencies, impedances) | Fit the circuit model |
| plot([f_data, Z_data, CI]) | a convenience method for plotting Nyquist plots |
| predict(frequencies) | Predict impedance using a fit equivalent circuit model |

**fit** (*frequencies*, *impedances*)
Fit the circuit model

**Parameters**

**frequencies: numpy array** Frequencies

**impedance: numpy array of dtype 'complex128'** Impedance values to fit

**Returns**

**self: returns an instance of self**

**class** eisfit.circuits.**Randles** (*CPE=False*, ***kwargs*)
A Randles circuit model class

### Methods

| | |
|---|---|
| fit(frequencies, impedance) | Fit the circuit model |
| plot([f_data, Z_data, CI]) | a convenience method for plotting Nyquist plots |

Continued on next page

Table 4 – continued from previous page

| | |
|---|---|
| `predict`(frequencies) | Predict impedance using a fit equivalent circuit model |

`eisfit.circuit_elements.`**`A`**$(p, f)$
defines a semi-infinite Warburg element

`eisfit.circuit_elements.`**`C`**$(p, f)$
defines a capacitor

$$Z = \frac{1}{C \times j 2 \pi f}$$

`eisfit.circuit_elements.`**`E`**$(p, f)$
defines a constant phase element

### Notes

$$Z = \frac{1}{Q \times (j 2 \pi f)^\alpha}$$

where $Q$ = p[0] and $\alpha$ = p[1].

`eisfit.circuit_elements.`**`G`**$(p, f)$
defines a Gerischer Element

### Notes

$$Z = \frac{1}{Y \times \sqrt{K + j 2 \pi f}}$$

`eisfit.circuit_elements.`**`R`**$(p, f)$
defines a resistor

### Notes

$$Z = R$$

`eisfit.circuit_elements.`**`W`**$(p, f)$
defines a blocked boundary Finite-length Warburg Element

### Notes

$$Z = \frac{R}{\sqrt{ T \times j 2 \pi f}} \coth{\sqrt{T \times j 2 \pi f}} noqa: E501$$

where $R$ = p[0] (Ohms) and $T$ = p[1] (sec) = $\frac{L^2}{D}$

`eisfit.circuit_elements.p`(*parallel*)
adds elements in parallel

### Notes

$$Z = \frac{1}{\frac{1}{Z_1} + \frac{1}{Z_2} + ... + \frac{1}{Z_n}}$$

`eisfit.circuit_elements.s`(*series*)
sums elements in series

### Notes

$$Z = Z_1 + Z_2 + ... + Z_n$$

CHAPTER 4

# Validation

EIS data fundamentally relies on the conditions of linearity,

## 4.1 Measurement Model

Testing your data with the measurement model is straightforward:

```
import matplotlib.pyplot as plt
import numpy as np
import sys
sys.path.append('../')

from eisfit import validation  # noqa E402

data = np.genfromtxt('./data/exampleData.csv', delimiter=',')
f = data[:, 0]
Z = data[:, 1] + 1j*data[:, 2]

mask = np.imag(Z) < 0

model_list, error_list = validation.measurementModel(f, Z, max_k=25)

fig = plt.figure()
plt.plot(Z.real, -Z.imag, 'o')
for model in model_list:
    Z_fit = model.predict(f)
    plt.plot(Z_fit.real, -Z_fit.imag)

fig2, ax2 = plt.subplots()
ax2.plot(range(1, len(error_list)+1), error_list)
ax2.set_yscale('log')
ax2.set_ylabel('Root Mean Squared Error')
ax2.set_xlabel('Number of RC elements')
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

eisfit.validation.**measurementModel**(*frequencies*, *impedances*, *algorithm='SLSQP'*, *max_k=7*, *R_guess=0.1*, *C_guess=10*)

Runs a measurement model test for validating impedance data

Iteratively add RC circuit elements until the error converges. If error does not converge, it indicates that the data doesn't meet standards for linearity.

### Notes

$$RMSE = R_0 + \sum_0^k R_i||C_i$$

**frequencies: np.ndarray** A list of frequencies to test

**impedances: np.ndarray of complex numbers** A list of values to match to

**max_k: int** The maximum number of RC elements to fit

**initial_guess: np.ndarray** Initial guesses for R and C elements

eisfit.validation.**rmse**(*a*, *b*)

A function which calculates the root mean squared error between two vectors.

### Notes

$$RMSE = \sqrt{\frac{1}{n}(a-b)^2}$$

# CHAPTER 5

# Preprocessing

Methods for preprocessing impedance data from instrument files

`eisfit.preprocessing.`**`readAutolab`**(*filename*)
function for reading the .csv file from Autolab

**Parameters**

**filename: string** Filename of .csv file to extract impedance data from

**Returns**

**frequencies** [np.ndarray] Array of frequencies

**impedance** [np.ndarray of complex numbers] Array of complex impedances

`eisfit.preprocessing.`**`readFile`**(*filename*, *type=None*)
A wrapper for reading in many common types of impedance files

**Parameters**

**filename: string** Filename to extract impedance data from

**type: string** Type of instrument file

**Returns**

**frequencies** [np.ndarray] Array of frequencies

**impedance** [np.ndarray of complex numbers] Array of complex impedances

`eisfit.preprocessing.`**`readGamry`**(*filename*)
function for reading the .DTA file from Gamry

**Parameters**

**filename: string** Filename of .DTA file to extract impedance data from

**Returns**

**frequencies** [np.ndarray] Array of frequencies

**impedance** [np.ndarray of complex numbers] Array of complex impedances `eisfit.preprocessing.`**`readParstat`**(*filename*)

function for reading the .txt file from Parstat

**Parameters**

**filename: string** Filename of .txt file to extract impedance data from

**Returns**

**frequencies** [np.ndarray] Array of frequencies

**impedance** [np.ndarray of complex numbers] Array of complex impedances

CHAPTER 6

# Genetic Modeling

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## e

# Index