**Analysis Requirements - 1/18/2022**

**Direct questions towards Tyler Wang (**[**tylerwang@berkeley.edu**](mailto:tylerwang@berkeley.edu) **or Slack)**

More analysis requirements will likely be added in the future, but please start with this for now.

Note: eventually, most of the settings required by the frontend user will be automated and set to be the same, but for now, we should avoid hard coding this and maintain flexibility.

# Importing and Parsing Data

The relevant data from the FW will be saved in a *n* x *m* table in .csv format. The table will look something like the following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Frequency (Hz) | Ch 1 Impedance (Ohms) | Ch 2 Impedance (Ohms) | Ch 3 Impedance (Ohms) | … | Ch *n - 1* Impedance (Ohms) |
| 1 (Freq 1) | 10 | 20 | X | … | X |
| 2 (Freq 2) | 10 | 20 | X | … | X |
| 3 (Freq 3) | 20 | 30 | X | … | X |
| 4 (Freq 4) | 20 | 30 | X | … | X |
| … | … | … | … | … | … |
| *m* | X | X | X | X | X |

Where X represents an arbitrary number from the data. First, **import all of the data from the file into the backend for plotting and analysis.** The frequency and impedance values will all be floating point numbers, and we should be able to deal with a reasonable number of decimal points at least.

# Bare Bones Plotting

Using the frontend, **allow the user to group the channels into up to *n-1* groups as desired.** The names of these groups should be customizable from the frontend. This grouping as well as the names should then be sent to the backend during this analysis. For example, in Scenario A, we might not want to group any of the channels, and thus we will have *n-1* total groups. In a different Scenario B, we might want to combine Ch 1 and Ch 2 into one group and Ch 2 and Ch 5 into a second group.

**Also, using the frontend, allow the user to pass in the following information if desired:**

* **Title**
* **Function (a name and/or a formula should be sufficient; this will make more sense later)**

**For each group, average the impedances at each frequency for all channels within the group to create an ensemble mean impedance for the group**. For example, in Scenario B, the first four impedance values would be [15, 15, 25, 25] (all in units of ohms). Of course, if only one channel is in the group, skip this averaging step and take the impedances of the single channel.

**Similarly, for each group, take the standard deviation of the impedances at each frequency for all channels within the group as a sample impedance noise for the group.** Importantly, use the sample standard deviation formula for one degree of freedom rather than the population standard deviation formula. Explicitly,. So for Scenario B, the first four values would be [7.07, 7.07, 7.07, 7.07]. Again, skip this step if only one channel is in the group.

**On log-log axes, plot the mean impedance of each group vs the frequencies in Column 1.** Each group should be plotted in its own color. **Also, for each group, plot the impedances minus the standard deviation of the impedances, and plot the impedances minus the standard deviation of the impedances. Shade in the region between these lines with a lighter shade of the same color used to plot each of the mean impedance lines. Create a plot legend with the names of each group from the frontend.** **The *x*-axis should be labeled “Frequency (Hz),” and the *y*-axis should be labeled “Impedance (Ω)”.** If you cannot easily display the Omega symbol, feel free to instead write “Ohms” in the parenthesis. **If the title is specified from the frontend, label the plot with the title. Otherwise, simply use the filename as the title of the plot.** As an example of what one such plot might look like,

Chart, line chart

Description automatically generated

**This plot should then be saved as an image and sent back to the frontend for display to the user.**

# Impedance Change Plotting

**Using the frontend, allow for the further pairing of two groups.** For example, if I have already denoted 4 groups, then I might want to pair groups 1 and 2 and pair groups 3 and 4. It must be clear which group is the “baseline” and which group is the “data” (i.e. the two groups in the pair are not to be treated the same, and thus the user must be able to clearly select which group is which).

We will consider one such pair, although in principle the following analysis must be done for all pairs denoted by the user. For notational reasons, we will refer to the mean impedances generate by the analyses in “Bare Bones Plotting” as μ1 and μ2, and the standard deviations of the impedances as σ1 and σ2. **Using these arrays, generate two new arrays by the following formulas**

These values are respectively the change in impedance, and the noise in our comparison measurement.

**On a new log-log plot with the same axes, generate a similar looking plot to before. This time, plot vs the frequencies, vs the frequencies, and vs the frequencies. Again, shade in the regions between the plus and minus curves. Title the plot with something similar to the title supplied by the user, but clearly make it distinct that this is a “change in impedance” plot (just add a few keywords to the title or something).** Again, this plot should then be saved and sent back to the front end for display to the user.

# Curve Fitting

From the frontend, we have chosen a formula to use for curve fitting. The relevant formulas are as follows

* (simple capacitor)
* (series RC)
* (parallel RC across R)
* (parallel RC across C)
* (parallel RC transfer function for an input phasor)

Here, *Z* is the impedance, *f* is the frequency, *R* is the resistance, and *C* is the capacitance. **Implement the above formulas**. Please make it easy to add additional curve fitting formulas in the future, as we will likely need more in the future to accurately model our system.

**Now, for each group in “Bare Bones Plotting,” perform a nonlinear fit with uncertainty for the selected formula.** There is likely a method that is already implemented for this in whatever language you’re programming this in. For example, in Python, we could call the scipy library and use

params, cov = scipy.optimize.curve\_fit(function, fdata, Zdata, sigma=sigmaZ, absolute\_sigma=True)

If absolutely necessary, it’s also pretty easy to implement this algorithm (Levenberg-Marquardt in the literature). Please note that if you aren’t passing in the frequencies, mean impedances, and standard deviation of the impedances, you’re likely using the wrong method. The correct output should be an array of the *R* and *C* parameters generated from the fit, as well as an unnormalized covariance matrix (or equivalently some other representation of the variance and covariance in the *R* and *C* fit parameters).

**Pass the fit parameters *R* and *C* and their uncertainties back to the frontend for display to the user.** Please make sure that you are passing the standard deviations and not the variance (take the square root of the diagonal elements of the covariance matrix to obtain the standard deviations). The units of *R* are Ω (ohms), and the units of *C* are farads (F).

# Change in Impedance Calculation

From the “Curve Fitting” section, for each group, we have values of *R* and *C* (and potentially an additional variable named *L*, but we shall ignore that for now). Also, from the “Impedance Change Plotting” section, we have already denoted pairs of groups. **Perform the following calculations and output the results to the frontend.**