chronoamperometry Documentation

Release 1

Clayton Rabideau

Mar 28, 2018

Getting Started

CHAPTER 1

Overview

1.1 Installation

pip install chronoamperometry

1.2 Purpose

Chronoamperometric measurement generates a such a large quantity of data that the usual high-level methods of data manipulation such as MS Excel function poorly because they are not optimized to handle millions of datapoints. Complex methods of analysis are also difficult or impossible to implement using Excel, Origin, etc.

It is also my hope that this tool-set will assist with the goal of using Microbial Fuel Cells to examine the underlying biology of organisms by enabling the comparison of current production by various mutants, etc.

Finally, this tool produces publication-quality graphs via the PlotNine package.

1.3 Tools

This repository includes code that will:

1. Directly digest the output excel file from PalmSens MultiTrace potentiostat software.
2. Arrange the data into a standard dataframe format
3. Assist with statistical analysis for estimation of noise estimation and calculations for t-tests.
4. Plot the time series of raw chronoamperometric measurement using the python port of ggplot2, plotnine
5. Plot statistical or other analysis using plotnine

To do:

1. Calculations for cohen's D

2. Implement integral calculation for chronoamperometric curves
3. Calculations for sensitivity index d'

CHAPTER 2

Example Usage

2.1 Overview

Generally just point the method you'd like to use to the excel file output by the MultiTrace software and apply a sub-method

2.2 Noise Esimation (Absolute median deviation from signal):

```
from chronoamperometry import statistics

# Point to location of Multi-Trace data
data = '<path to dataset>'

# Calculate the noise in the system
noise = statistics.ReplicateStatistics(data).calculate_median_absolute_deviation_from_
↪signal()

print (noise)
```

2.3 T-test on two independent measurements:

```
from chronoamperometry import statistics

# importing datasets
data1 = '<path to data for first variable>'
data2 = '<path to data for second variable>'

# loading data into t-test
t_test = statistics.ExperimentalStatistics(data1, data2).t_test()
```

```
print(t_test)
```

2.4 Plot P-values vs time:

```
from chronoamperometry import plotting

# import data
data1 = '<path to data for first variable>'
data2 = '<path to data for second variable>'

# plot data
plot = plotting.ExperimentPlot(data1, data2).plot_t_test()
```

2.5 Split DataFrame into two Subsets:

```
from chronoamperometry import utils

# Point to location of Multi-Trace data
data = '<path to dataset>'

# Create lists of channels to be placed in first and second subsets
variable1 = [1, 2, 3, 4, 5, 6]
variable2 = [7, 8, 9, 10, 11, 12]

# Split data
subset1_df, subset2_df = utils.SelectData(data, subset_1=variable1, subset_
↪2=variable2).split_dataframes()
```

CHAPTER 3

Utilities

3.1 Utilities

class chronoamperometry.utils.**DataFrameBuild**(*mt_excel_data*)

This class consumes excel file outputs from PalmSens Multitrace and converts them to a standardized pandas dataframe format

dataframe_from_mtxl()

Consumes excel files and produces unmelted dataframes where each column is a different channel, except the first, which is time.

melted_dataframe_from_mtxl()

Consumes excel files and produces melted dataframes, grammar of graphics style.

class chronoamperometry.utils.**SelectData**(*data*, *subset_1=None*, *subset_2=None*)

This class is useful for splitting data into two subsets (e.g.: if there are different variables on the same run, etc.) or for deleting a subset of the data. Simply pass in a list of channel numbers that should be

in the first or second subsets.

delete_subset()

This method will keep the first subset and delete the second subset of the input data.

split_dataframes()

This method will convert the raw output from the PalmSens Multitrace software for the MultiEmStat potentiostat or a processed dataframe to multiple dataframes, one for each experimental variable so that they can be compared.

Is useful for multiple variables on the same device

CHAPTER 4

Statistics

4.1 Statistics for Replicate Data

class chronoamperometry.statistics.**ReplicateStatistics**(*data*, *span=0.2*, *df_name='mads_df'*, *periodicity=None*, *cycles=None*, *stabilization_time=None*)

This class contains statistical tools for the analysis of replicate traces

anova_test_magnitude_of_current_variance()

Anova

Not Working Yet :return:

calculate_absolute_deviation_from_signal_per_channel()

Estimates noise by calculating distance of the noise of each trace from the 'signal' produced by the regression analysis

calculate_median_absolute_deviation_from_signal()

Estimates noise by calculating distance of median noise in the traces from the 'signals' produced by the regression analysis

construct_lowess_regression()

Creates a smoothed regression based on the Lowess algorithm.

4.2 Statistics for Experimental Validation

class chronoamperometry.statistics.**ExperimentalStatistics**(*data1*, *data2*, *span=0.2*, *significance_threshold=0.05*)

This class contains tools for the analysis of a single variable between two groups of replicate traces.

anova_test()
: returns an an analysis of variance comparing distribution of current magnitude between two experiments at each timepoint.

compare_absolute_deviation_from_signal_between_experiments()
: Allows for a comparison of noise between two experiments with a single variable

t_test()
: t-test on raw chronoamperometric data

CHAPTER 5

Plotting

5.1 Plotting for Replicate Data

5.2 Plotting for Experimental Validation

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index