

Supporting Information

Open Source Software for the Real-Time Control, Processing and Visualization of High-Volume Electrochemical Data

Samuel D. Curtis^{a,f}, Kyle L. Ploense^a, Martin Kurnik^a, Gabriel Ortega^{a,b}, Claudio Parolo^{a,b}, Tod E. Kippin^{c,d,e}, Kevin W. Plaxco^{a,b} and Netzahualcóyotl Arroyo-Currás^{f,*}

Affiliations

- ^a Center for Bioengineering, University of California Santa Barbara, Santa Barbara, CA 93106, USA.
- ^b Department of Chemistry and Biochemistry, University of California Santa Barbara, Santa Barbara, CA 93106, USA.
- ^c Department of Psychological and Brain Sciences, University of California Santa Barbara, Santa Barbara, CA 93106, USA.
- ^d Neuroscience Research Institute, University of California Santa Barbara, Santa Barbara, CA 93106, USA.
- ^e Department of Molecular Cellular and Developmental Biology, University of California Santa Barbara, Santa Barbara, CA 93106, USA.
- ^f Department of Pharmacology and Molecular Sciences, Johns Hopkins School of Medicine, Baltimore, MD 21205, USA.

Table of Contents

| | |
|---|-----------|
| Figure S1. The program user interface: parameter inputs..... | 5 |
| Figure S2. The real-time processing interface..... | 6 |
| Python Installation Instructions | 7 |
| GitHub: Cloning the Program to Your Desktop..... | 8 |
| Other Free Software for Python Editing (Optional) | 8 |
| Mac Bug When Running Tkinter-Based Python Programs | 8 |
| Standard Operating Procedure for SACMES.py | 9 |
| <i>Brief Description of the Script Structure</i> | <i>9</i> |
| Figure S3. Code snippet of the Global Variables used in SACMES. | 9 |
| Figure S4. Code snippet of the Global Functions used in SACMES..... | 10 |
| <i>Input File Formatting and Data Extraction</i> | <i>10</i> |
| Input File Format | 10 |
| Figure S5. Example of file generated by multiplexer potentiostat. | 11 |
| Figure S6. Example of file generated by multichannel potentiostat. | 11 |
| Default Input File Nomenclature | 11 |
| Customizing Data Extraction from the Input File..... | 12 |
| <i>ElectrochemicalAnimation</i> | <i>12</i> |
| Table S1. The ElectrochemicalAnimation function..... | 13 |
| Data Processing Module (generator) | 14 |
| Figure S7. Code snippet showing extraction of potentials and currents from input file..... | 15 |
| Figure S8. Code snippet showing Savitzky-Golay smoothing function. | 15 |
| Figure S9. Code snippet showing polynomial regression analysis of smoothed currents. ... | 15 |
| Figure S10. Code snippet showing peak current and area calculations. | 16 |
| Figure S11. Code snippet showing the return of analyzed data to the animate function | 16 |

| | |
|---|-----------|
| Animation Module (func) | 16 |
| Table S2. The func module..... | 17 |
| Figure S12. Code snippet showing the pass of framedata to the animate function | 17 |
| <i>Graphical User Interface</i> | 18 |
| MainWindow | 18 |
| Figure S13. The SACMES Main Window | 18 |
| Initial Input Frame (Start Page) | 19 |
| Figure S14. The SACMES Input Frame..... | 19 |
| Figure S15. Code snippet showing formula for y-axis range calculation | 22 |
| Figure S16. The SACMES Initialize button | 23 |
| Real Time Data Manipulation Frame | 23 |
| Figure S17. The SACMES Real Time Data Manipulation Frame | 23 |
| Figure S18. The SACMES drift correction box | 24 |
| Figure S19. The SACMES box for real-time analysis manipulation | 24 |
| Visualization Frame | 25 |
| Figure S20. The SACMES Visualization Frame..... | 25 |
| <i>Data Export</i> | 26 |
| Activating Data Export | 26 |
| Figure S21. Code snippet showing the TextFileExport() class | 26 |
| Export File Nomenclature | 26 |
| Format of exported file | 27 |
| Standard Operating Procedure for SACMES_CV.py | 28 |
| <i>Brief Description of the Script Structure</i> | 28 |
| <i>Input File Formatting and Data Extraction</i> | 28 |
| Input File Format | 28 |
| Default Input File Nomenclature | 28 |

| | |
|---|-----------|
| <i>Graphical User Interface</i> | 29 |
| MainWindow | 29 |
| Initial Input Frame (Start Page) | 29 |
| Figure S22. The SACMES_CV Input Frame | 30 |
| Setup Frame | 30 |
| Figure S23. The SACMES_CV Setup Frame | 30 |
| Visualization Frame | 31 |
| Figure S24. The SACMES_CV Visualization Frame | 31 |
| Settings Toolbar | 31 |
| <i>Data Analysis</i> | 32 |
| Data Processing Module (generator) | 32 |
| Figure S25. Code snippet showing function to adjust voltage range considered | 32 |
| Figure S26. Code snippet showing calculation of the peak potential by generator function | 32 |
| Figure S27. Code snippet showing creation of linear baselines | 33 |
| Figure S28. Code snippet showing calculation of slope for solution phase analysis | 33 |
| <i>Data Export</i> | 34 |
| Format of exported file | 34 |
| Altering the Export Path | 34 |

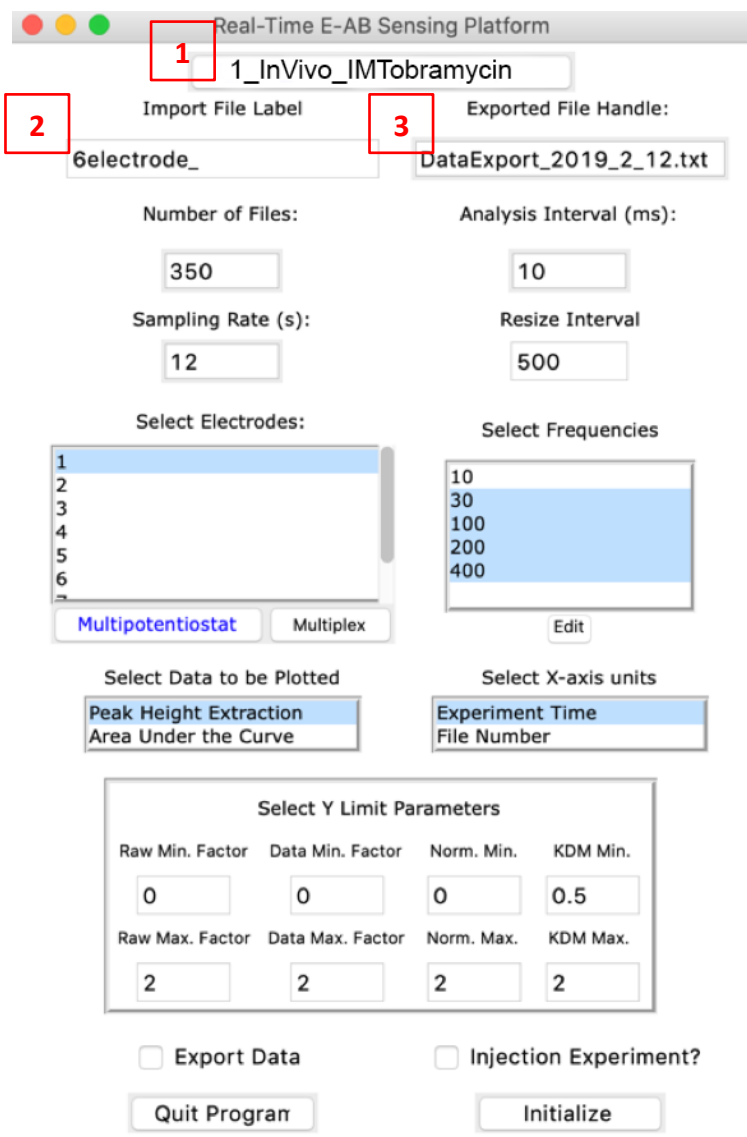


Figure S1. The program user interface: parameter inputs

Left: Here we show a screenshot of the main control panel, where: **1**) the path to the data folder is selected (here '1_InVivo_IMTobramycin'); **2**) the file handle is defined (here '6electrode_') and **3**) a handle to be assigned to the exported data file is defined. Following these boxes we included controls for: number of files, electrode number, square-wave frequency and other parameters as defined in the SOP.

Right: Apple macOS operating system window showing the data folder with the data files and full file name used. Note the match between the "Import File Label" in the left panel and the name of the files in this screenshot.

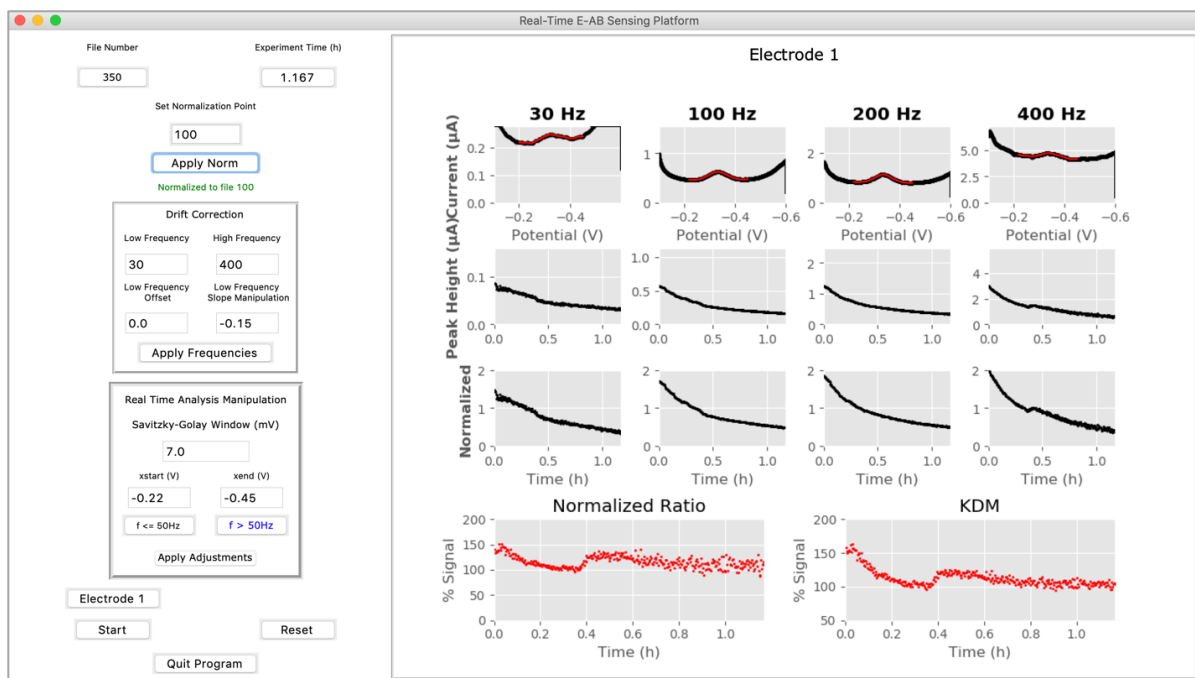


Figure S2. The real-time processing interface

When the experiment is initialized, this window accepts real-time parameter inputs for signal processing, such as normalization point and drift correction. For more details read SOP below.

Python Installation Instructions

SACMES is based on the Python programming language and employs three publicly available, pre-developed python libraries:

- (1) SciPy;
- (2) Matplotlib; and
- (3) NumPy

These libraries are usually included within Anaconda's Python distribution, an open-source scientific library for Python and R, found here: <https://www.anaconda.com/distribution/>. If this link stops working in the future, please search for newer anaconda distributions on the web. Anaconda is available for all major operating systems (Linux, macOS, Windows) and will require roughly 650MB of disk space. While macOS comes with python installed, it is still necessary to download all of the extra scientific packages and thus we still recommend the Anaconda installation. SACMES was built on Python v.3.

When installing anaconda on Windows-based computers, we recommend checking the box "Add Anaconda to my PATH environment variable". **Note:** In some instances, the Anaconda installer will omit the installation of SciPy and Matplotlib. If this occurs, you may install them by opening the app called "Anaconda Prompt" and entering the following commands:

For Matplotlib write:

- 1) `conda install -c conda-forge matplotlib`
then hit "Enter"

If prompted to say yes or no to perform these updates, write Y and hit "Enter."

For SciPy write:

- 1) `conda install -c anaconda scipy`
then hit "Enter"

If prompted to say yes or no to perform these updates, write Y and hit "Enter."

GitHub: Cloning the Program to Your Desktop

We have set up an institutional account with GitHub –an online repository for software development– where we have uploaded all versions of SACMES (sacmes.py) and where we will upload any future updates to the program. GitHub is available in browser and desktop formats, which can be accessed via the links: <https://github.com> and <https://desktop.github.com/>. Our repository is called “SACMES” and can be accessed free of charge (free GitHub account needed) via the link: <https://github.com/netzlab/SACMES.git>

Other Free Software for Python Editing (Optional)

Python code can be written on any text editor. However, if you need to customize SACMES for a particular application we recommend using the free editors Atom or Sublime. These editors are able to recognize Python functions in files saved with the .py extension and offer useful text coloring features. To download any of them go to:

- 1) <https://atom.io/>
- 2) <https://www.sublimetext.com/3>

Mac Bug When Running Tkinter-Based Python Programs

At the time of this submission, the apple OS was updated to version Mojave 10.14.6. This update causes a problem with Tkinter, Python’s base GUI, in new Macbook Pro computers. These laptops may crash when running our code or any Python-based code that uses Tkinter. This issue is a problem caused by the apple OS update (and its developers) and should be fixed in future versions of the operating system (most likely after the migration to Catalina in 2019). There was no workaround at the time of this submission. SACMES users with 2018 MacBook Pro laptops or newer will need to run the code on an older mac or a windows PC.

Standard Operating Procedure for SACMES.py

The following documentation includes a description of the different sections contained within the SACMES script and provides instructions on how to use and edit the code. This documentation is not exhaustive, however, so we encourage the community to reach out to the authors with questions or comments. We will maintain a GitHub folder (<https://github.com/netzlab/SACMES.git>) where any updates to the code will be uploaded.

Brief Description of the Script Structure

The script starts with a series of commands to import all the Python libraries that are needed for the proper operation of the program. This section does not need to be edited unless user-customization requires the use of different libraries. Next, you will see the list of Global Variables (variables that are shared by many functions throughout the program) (**Figure S3**).

```
#####  
### Global Variables ###  
#####  
  
#####  
### Polynomial Regression Parameters ###  
#####  
sg_window = 5      ### Savitzky-Golay window (in mV range), must be odd number (increase signal:noise)  
sg_degree = 1      ### Savitzky-Golay polynomial degree  
polyfit_deg = 20   ### degree of polynomial fit  
  
cutoff_frequency = 50  ### frequency that separates 'low' and 'high'  
                    ### frequencies for regression analysis and  
                    ### smoothing manipulation  
  
#####  
### Checkpoint Parameters ###  
#####  
key = 0            ### SkeletonKey  
search_lim = 15    ### Search limit (sec)  
PoisonPill = False  ### Stop Animation variable  
FoundFilePath = False  
ExistVar = False  
AlreadyInitiated = False  ### indicates if the user has already initiated analysis  
HighAlreadyReset = False  
LowAlreadyReset = False  
analysis_complete = False  
  
delimiter = 1      ### default delimiter is a space; 2 = tab  
column_index = -2  
  
#####  
### Low frequency baseline manipulation Parameters ###  
#####  
LowFrequencyOffset = 0  ### Vertical offset of normalized data for  
                        ### user specified 'Low Frequency'  
LowFrequencySlope = 0   ### Slope manipulation of norm data for user  
                        ### specified 'Low Frequency'
```

Figure S3. Code snippet of the Global Variables used in SACMES.

And then a list of Global Functions such as, for example, the ones specifying the filename of the input data files (**Figure S4**).

```
#####  
### Global Functions ###  
#####  
  
#####  
### Retrieve the file name ###  
#####  
def _retrieve_file(file, electrode, frequency):  
  
    if e_var == 'single':  
        filename = '%s%dHz.txt' % (handle_variable, frequency)  
        filename2 = '%s%dHz_%d.txt' % (handle_variable, frequency, file)  
  
    elif e_var == 'multiple':  
        filename = 'E%s %s%sHz %d.txt' % (electrode, handle_variable, frequency, file)  
        filename2 = 'E%s_%s%sHz_%d.txt' % (electrode, handle_variable, frequency, file)  
  
    return filename, filename2
```

Figure S4. Code snippet of the Global Functions used in SACMES.

Below the list of Global Functions you will find all the modules that make the Graphical User Interface (GUI) and the bulk of the program. These will be discussed in detail next. We have invested significant time in fully documenting each section of the code and, as a result, you will see different sections clearly denoted by headers and line breaks like the ones seen in the snippets above. For example, you will see pound signs denoting headers (**Figure S4**).

Note that all text preceded by a pound sign, #, is ignored by the Python compiler and will not be executed. We now proceed to describe the main functions and classes in SACMES.

Input File Formatting and Data Extraction

Input File Format

The Input File is the raw data created by an external analytical device that contains, in the case of SVW and other voltammetric techniques, the experimentally-applied potentials and corresponding measured currents. For example, see right panel of **Figure S1**.

SACMES can analyze multielectrode data contained within a single file ('Multichannel' setting; default) or in separate files ('Multiplexed' setting, one file per electrode). The default format for data files in a SWV experiment – independent of whether there are individual files for each electrode or a single file for all electrodes – should be as follows:

- Column 1: Potentials
- Column 2: Differential SWV Currents

Column 3: Forward SWV Currents

Column 4: Reverse SWV Currents

Repeat Columns 2-4 for further electrodes

- Example of a file in the ‘*Multiplexed*’ setting containing **data for a single electrode**, in this case ‘E1’; with headers (**Figure S5**).

```
May 19, 2017 02:01:08
Square Wave Voltammetry
File: c:\users\crossover\desktop\my mac desktop\surface biophysics\E1_ProtData\
Data Source: Experiment
Instrument Model: CHI650C

Potential/V, Diff(i/A), For(i/A), Rev(i/A)

-0.001, 3.765e-8, 1.485e-8, -2.280e-8      Applied Voltage
-0.002, 3.831e-8, 1.637e-8, -2.194e-8
-0.003, 3.847e-8, 1.727e-8, -2.121e-8      Currents - Electrode 1
-0.004, 3.860e-8, 1.775e-8, -2.085e-8
-0.005, 3.851e-8, 1.816e-8, -2.034e-8
```

Figure S5. Example of file generated by multiplexer potentiostat.

- Example of a file in the ‘*Multichannel*’ format containing **data for 3 electrodes** from a single continuous experiment; without headers (**Figure S6**).

```
-0.101, 8.359e-8, 1.094e-8, -7.265e-8, 4.028e-8, 8.560e-9, -3.172e-8, 5.640e-8, 8.532e-9, -4.787e-8
-0.102, 1.002e-7, 3.861e-8, -6.154e-8, 5.001e-8, 2.372e-8, -2.629e-8, 7.426e-8, 3.316e-8, -4.110e-8
-0.103, 1.008e-7, 4.304e-8, -5.776e-8, 5.072e-8, 2.599e-8, -2.473e-8, 7.544e-8, 3.683e-8, -3.860e-8
-0.104, 1.007e-7, 4.516e-8, -5.553e-8, 5.163e-8, 2.782e-8, -2.381e-8, 7.568e-8, 3.790e-8, -3.778e-8
-0.105, 1.002e-7, 4.667e-8, -5.355e-8, 5.101e-8, 2.787e-8, -2.315e-8, 7.562e-8, 3.868e-8, -3.693e-8
-0.106, 9.855e-8, 4.663e-8, -5.192e-8, 5.050e-8, 2.782e-8, -2.268e-8, 7.695e-8, 3.988e-8, -3.707e-8
-0.107, 9.853e-8, 4.718e-8, -5.134e-8, 5.150e-8, 2.938e-8, -2.212e-8, 7.504e-8, 3.997e-8, -3.506e-8
-0.108, 9.793e-8, 4.816e-8, -4.976e-8, 5.083e-8, 2.884e-8, -2.199e-8, 7.653e-8, 4.066e-8, -3.587e-8
-0.109, 9.906e-8, 4.905e-8, -5.001e-8, 4.997e-8, 2.851e-8, -2.145e-8, 7.577e-8, 4.037e-8, -3.540e-8
-0.110, 9.826e-8, 4.957e-8, -4.869e-8, 5.086e-8, 2.942e-8, -2.143e-8, 7.544e-8, 4.102e-8, -3.442e-8
-0.111, 9.815e-8, 4.970e-8, -4.845e-8, 5.021e-8, 2.922e-8, -2.099e-8, 7.639e-8, 4.122e-8, -3.518e-8

Applied Voltage      Currents - E1      Currents - E2      Currents - E3
```

Figure S6. Example of file generated by multichannel potentiostat.

Default Input File Nomenclature

If the data for each electrode are within a single file (‘*Multichannel*’)

- The file name can begin with any ASCII string – here written as ‘*ExampleHandle_*’ – followed by the frequency of the experiment, an underscore, the file number, and the extension ‘.txt’

- E.g. ExampleHandle_10Hz_1.txt → ExampleHandle_10Hz_2.txt → ExampleHandle_10Hz_3.txt → etc.
- Successive data files must have the same file handle with the file number increasing sequentially

If the data for each electrode are separated into individual files ('Multiplexed')

- The file name must then start with the electrode number and an underscore:
E5_ExampleHandle_10Hz_1.txt

Customizing Data Extraction from the Input File

By using the 'Settings' menu (located at the very top of SACMES where the Windows or Mac Finder Navigation Bars are typically found) and selecting 'Adjust Current Extraction', the user can change the file column that is read by SACMES (default is differential currents; column 2). If multiple electrodes are contained within a single file ('Multichannel' setting) then the columns with the data will be spaced 3 columns apart (e.g. 2, 5, 8, 11), assuming that the columns with the forward and reverse currents (independent of the order) will follow consecutive to the column with the differential currents.

Note: If a different nomenclature is to be used, the script must be edited to reflect such new nomenclature. Otherwise the script will not find the files to be analyzed. To do this, the section Global Functions (**Figure S4**), function '_retrieve_file' must be edited from the current definition of the file name: 'E%s_%s%sHz_%d.txt' to a string matching the new nomenclature.

ElectrochemicalAnimation

The majority of the code for file processing and visualization in SACMES is contained within the function titled ElectrochemicalAnimation. This function is a Python class (a code template for creating objects) that controls the progression of all data processing and visualization. Specifically, ElectrochemicalAnimation contains two main modules: (1) a "generator" that analyzes and stores

data; and (2) an “animator” that subsequently visualizes the data. If data recorded on multiple electrodes is being analyzed, SACMES will call a separate, independent ‘instance’ of the ElectrochemicalAnimation class for each electrode. All instances run in parallel, however, making SACMES highly efficient.

When the user presses the ‘Start’ button on the Real Time Manipulation Frame (**Figure S17**), SACMES initializes data processing and visualization by creating an instance of the ElectrochemicalAnimation class for each electrode. Specifically, the ‘Start’ button passes the parameters chosen by the user in the StartPage user interface – including the figure to be animated and the electrodes to be analyzed – to the ElectrochemicalAnimation class. The arguments (noted between parenthesis) are presented in **Table S1**.

Table S1. The ElectrochemicalAnimation function.

ElectrochemicalAnimation (self, fig, ax, electrode, generator = None, func = None, resize_interval = None, fargs = None):

| | |
|--------------------|---|
| Parameters: | <p>fig : matplotlib.figure.Figure object</p> <p>The figure object that is used to draw, resize, and visualize processed data.</p> <p>ax : matplotlib.axes.Axes object</p> <p>The axes that contain all of the “artists” within a given figure such as Axis, Tick, Line2D, and Polygon objects</p> <p>electrode : integer</p> <p>Electrode corresponding to the given figure. Indicates which electrode is being analyzed (e.g. E3, E5, E7). Most importantly, the electrode number indicates how the data-generator reads the data from the input txt file (See Default Input File Nomenclature p.11 for more information).</p> <p>generator : generator class instance, optional</p> <p>Iterable data analysis function. Generates data to pass to func to be animated. Iterates through each file – processing the data within the file and returning this processed data to func to be visualized.</p> <p>func : callable, optional</p> <p>Visualization function. In SACMES, func is the Animation function that obtains and animates data received from generator.</p> <p>resize_interval : integer, optional</p> <p>Integer indicating after how many files the will resize. The user can choose this value within the resize-interval widget (tk.Entrybox) in the StartPage frame. If None is selected it is defaulted to 200.</p> <p>fargs : dictionary</p> |
|--------------------|---|

extra user-given arguments to pass to **func**

The ElectrochemicalAnimation class **requires two main modules:** (1) the data processing module (*generator*) and (2) the animation module (*func*).

Data Processing Module (generator)

The Data Processing Module performs three main functions. First, it reads and processes the data file – performing regression analyses, smoothing algorithms, and finally either peak height extraction or area under the curve (whichever is chosen by the user on the StartPage frame). Second, the generator saves this data into global lists for processes such as data export, ratiometric analyses, data normalization, and real-time data manipulation. Lastly, the module ‘returns’ the analyzed data to the animation module, *func*, to be visualized.

If measurements are recorded on different sensors/electrodes, SACMES analyzes data from each using a separate instance of the *generator* function. Because SACMES has a modular architecture, it can be easily customized to support the analysis of different electroanalytical techniques. This is done by creating a custom-made generator class that would read and process different data files (e.g., from chronoamperometry or linear sweep voltammetry) or use different data smoothing, regression analysis approaches. Whenever this is done, the new generator class must be passed as an argument to ElectrochemicalAnimation.

By default SACMES is configured to analyze square-wave voltammograms. The *generator* therefore always begins by processing the lowest square-wave frequency dataset found in a newly-generated file (remember that SACMES is continuously probing for the existence of new files in a user-defined folder). To do this, the *generator* retrieves the text file, finds the data corresponding to the low square-wave frequency measurement and then returns the data to the animation class to be visualized. It then iterates through the remaining frequencies, from low to high. Once it has analyzed each frequency it moves on to the next file and returns to the first (lowest) frequency.

The default data processing steps are illustrated in **Figures S7-11**.

```
#####
### Retrieve data from the File ###
#####
potentials, currents, data_dict = self.read_raw_data(myfile, self.electrode)
print(myfile)

cut_value = 0
for value in potentials:
    if value == 0:
        cut_value += 1

if cut_value > 0:
    potentials = potentials[:-cut_value]
    currents = currents[:-cut_value]

#####
### Adjust the potentials depending on user-input parameters ###
#####
adjusted_potentials = [value for value in potentials if xend <= value <= xstart]
```

Figure S7. Code snippet showing extraction of potentials and currents from input file.

```
#####
### Savitzky-Golay Smoothing ###
#####
min_potential = min(potentials) # find the min potential
sg_limit = sg_window/1000 # mV --> V

# shift all values positive
sg_potentials = [x - min_potential for x in potentials]

# find how many points fit within the sg potential window
# this will be how many points are included in the rolling average
sg_range = len([x for x in sg_potentials if x <= sg_limit])

# Apply the smoothing function and create a dictionary pairing
# each potential with its corresponding current
smooth_currents = savgol_filter(currents, sg_range, sg_degree)
data_dict = dict(zip(potentials, smooth_currents))
```

Figure S8. Code snippet showing Savitzky-Golay smoothing function.

```
#####
### Polynomial fit ###
#####
polynomial_coeffs = np.polyfit(adjusted_potentials, adjusted_currents, polyfit_deg)

#####
### Polynomial Regression ###
#####
eval_regress = np.polyval(polynomial_coeffs, adjusted_potentials).tolist()
regression_dict = dict(zip(eval_regress, adjusted_potentials)) # dictionary with current: potential
```

Figure S9. Code snippet showing polynomial regression analysis of smoothed currents.

```
#####
### Extract the local minima and maxima of the polynomial regression ###
#####
fit_half = round(len(eval_regress)/2)

min1 = min(eval_regress[:fit_half])
min2 = min(eval_regress[fit_half:])
max1 = max(eval_regress[:fit_half])
max2 = max(eval_regress[fit_half:])

#####
### If the user selected Peak Height Extraction, analyze PHE ###
#####

if SelectedOptions == 'Peak Height Extraction':
    #####
```

```

### Peak Height Extraction ###
#####
if PHE_method == 'Abs':
    Peak_Height = max(max1,max2)-min(min1,min2)
elif PHE_method == 'Linear':
    Peak_Height = max([(y - y0) for y,y0 in zip(adjusted_currents,linear_approx)])

data = Peak_Height

#####
### If the user selected AUC extraction, analyze AUC ###
#####

elif SelectedOptions == 'Area Under the Curve':
#####
### Integrate Area Under the ###
### Curve using a Riemann Sum ###
#####
AUC_index = 1
AUC = 0

AUC_potentials = [abs(potential) for potential in adjusted_potentials]

#-- Find the minimum value and normalize it to 0 --#
AUC_min = min(adjusted_currents)
AUC_currents = [Y - AUC_min for Y in adjusted_currents]

#-- Midpoint Riemann Sum --#
while AUC_index <= len(AUC_currents) - 1:
    AUC_height = (AUC_currents[AUC_index] + AUC_currents[AUC_index - 1])/2
    AUC_width = AUC_potentials[AUC_index] - AUC_potentials[AUC_index - 1]
    AUC += (AUC_height * AUC_width)
    AUC_index += 1

data = AUC

#####
### Save the data into global lists ###
#####

data_list[self.num][self.count][self.index] = data
data_normalization.Normalize(self.file, data, self.num, self.count, self.index)

```

Figure S10. Code snippet showing peak current and area calculations.

```

#####
### Return data to the animate function as 'framedata' ###
#####

return potentials, adjusted_potentials, smooth_currents, adjusted_currents, eval_regress

```

Figure S11. Code snippet showing the return of analyzed data to the animate function

Animation Module (func)

The default animation module receives from the generator an iterable list of objects, *framedata*, to be animated. SACMES comes with a default animation module which can be customized as needed to support different visualization needs/applications. The arguments for Animation Module are presented in **Table S2**.

Table S2. The func module.

```
def __func (self, framedata, *args):
```

Parameters:

framedata: list

The processed data – contained within a list – that is generated by the generator class.

***args:** tuple, optional

Additional arguments to be passed to each iteration of the `__func` function by the `ElectrochemicalAnimation` class.

In the current version of SACMES, the Animation Module receives the argument *framedata* from the default *generator*. The *framedata* passed from *generator* to Animation Module is then ‘set’ as the animated data that will be visualized (**Figure S12**).

```
def _animate(self, framedata):  
    if key > 0:  
        while True:  
            potentials, adjusted_potentials, smooth_currents, adjusted_currents, regression, regression_potentials,  
            linear_fit = framedata
```

Figure S12. Code snippet showing the pass of framedata to the animate function

Graphical User Interface

The SACMES GUI consists of four principle frames (windows for the user to interact with the program): 1) a **Main Window** that acts as a container in which all other frames are embedded; 2) a **Start Page** frame for the initial user inputs; 3) a **Visualization Frame** that contains a matplotlib canvas (the canvas where each graph will be drawn and data will be visualized) for each electrode; and 4) a **Real Time Manipulation Frame** that contains widgets such as buttons and entry boxes enabling the real-time control of the data. See **Figs. S1, S2** for screenshots of all 4 frames.

MainWindow

The MainWindow (**Figure S13**) contains all program frames and widgets: (1) the Initial Input Frame (Start Page), (2) the Visualization Frame, and (3) the Real Time Manipulation Frame. The MainWindow has one row and two columns. The Initial Input Frame and Real Time Manipulation Frame lie in column 1 (overlaid on each other) while the Visualization Frame lies in column 2. The MainWindow acts as a container and only serves as a space holder for the other frames – it does not perform any functions or have any widgets.

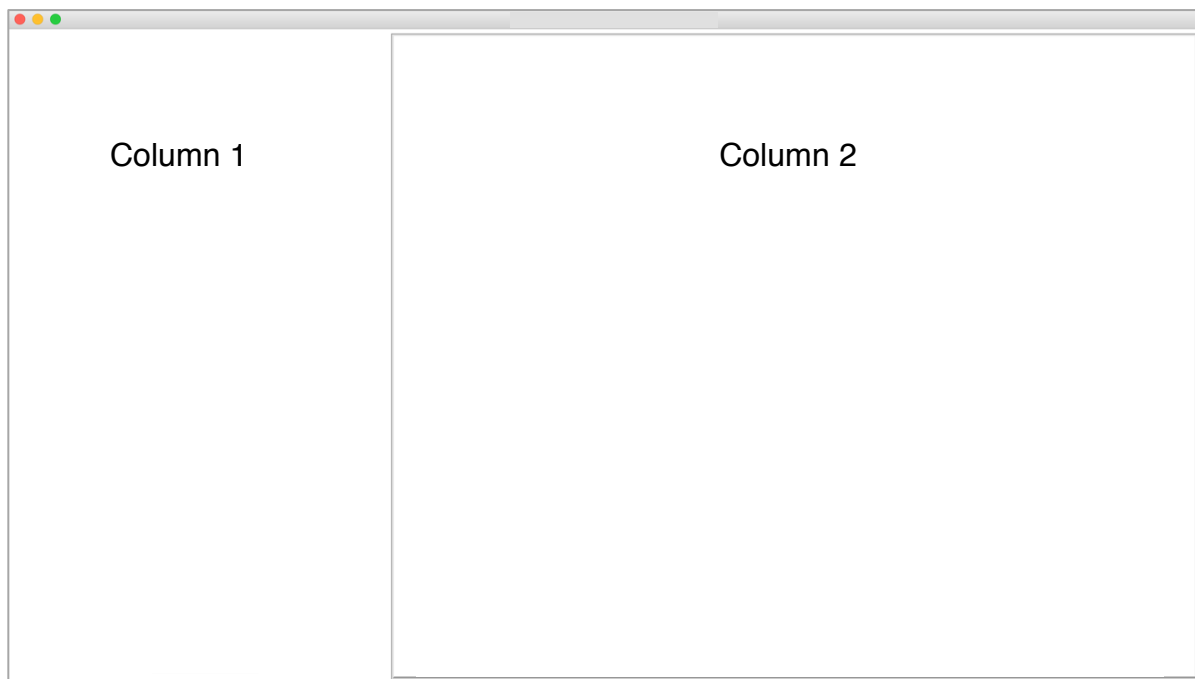


Figure S13. The SACMES Main Window

The Main Window that acts as a container in which all other frames are embedded.

Initial Input Frame (Start Page)

When the user initializes the program, the Initial Input Frame is the first frame created (**Figure S14**).

The screenshot shows the SACMES Initial Input Frame. The window title is "SACMES". At the top, there is a "Select File Path" button. Below it, there are two text boxes: "Import File Label" (empty) and "Exported File Handle:" containing "DataExport_2019_4_12.txt".

Next are four input fields: "Number of Files:" (50), "Analysis Interval (ms):" (10), "Sampling Rate (s):" (20), and "Resize Interval:" (200).

Below these are two lists: "Select Electrodes:" (a list box containing 1, 2, 3, 4, 5, 6) with "Multichannel" and "Multiplex" buttons below it; and "Select Frequencies:" (a list box containing 30, 80, 240) with an "Edit" button below it.

Further down are "Select Data to be Plotted" (a list box containing "Peak Height Extraction" and "Area Under the Curve") and "Select X-axis units" (a list box containing "Experiment Time" and "File Number").

A "Select Y Limit Parameters" section contains a table of input fields:

| Raw Min. Factor | Data Min. Factor | Norm. Min. | KDM Min. |
|-----------------|------------------|------------|----------|
| 0.5 | 0.5 | 0 | 0.5 |
| Raw Max. Factor | Data Max. Factor | Norm. Max. | KDM Max. |
| 2 | 2 | 2 | 2 |

At the bottom, there are two checkboxes: "Export Data" and "Injection Experiment?". At the very bottom are "Quit Program" and "Initialize" buttons.

Figure S14. The SACMES Input Frame

The Input Frame contains widgets that allows adjusting the parameters of the experiment to the user's needs.

The Input Frame can be edited to:

Set the file path for the data that is being analyzed

- To set the file path, click the button widget labeled 'Select File Path'. This prompts a file selection dialog where the user can choose the file from any location on a computer. This is the folder that contains all of the data files to be analyzed.

Set the File Name for Data Export

- If File Export has been activated, the user can type the name of the exported data file. This handle can contain any name that excludes backslashes ('\') and is followed by the extension '.txt'

Set the Number of Files to be Analyzed

- The user can input the number of files to be analyzed.
- If the input contains more files than exist, the program will simply stop analysis once it processes the last file and will wait idle until a new file appears.

Set the Sampling Rate

- The Sampling Rate (in seconds) is indicative of the time in between measurements.
- The Sampling Rate affects multiple variables:
 1. It is used to display the Experiment Time (a tk.Label widget at the top right of the Real Time Manipulation Frame)
 2. If 'Experiment Time' is selected in the 'Select X-Axis' Listbox in the Initial Input Frame then the Sampling Rate will be used to calculate the X axis units.

The experiment time is calculated in hours as $(File\#) \times (SamplingRate) / 3600$

Select Electrodes to be Analyzed

- The user has the ability to select precisely which electrodes will be analyzed.
- The Listbox under 'Select Electrodes' is a tk.Listbox widget that is set on the 'multiple' selection method – meaning that the user can select multiple electrodes to be analyzed simultaneously.
- If the user selects, for example, electrodes '1', '3', and '5' it will analyze only those electrodes.

Electrode Settings

- The format of the input file (data from the external device that is to be analyzed) can vary between potentiostat manufacturers or depending on user preferences.

- By selecting either “Multiplex” or “Multichannel” the user can indicate if their data across all electrodes is exported as a single .txt file for all electrodes (as is the case with most multipotentiostats) or if each electrode is exported into its own .txt file (as with most multiplexed potentiostats).

Select Frequencies to be Analyzed

- The ‘Select Frequencies’ Listbox is a tk.Listbox widget that contains a list of available frequencies for the user to choose from. Pressing the ‘Edit’ button beneath the Listbox allows the user to add or delete frequencies from this list. If the user selects ‘10’, ‘20’, and ‘100’, for instance, each electrode will be analyzed at those frequencies in order from lowest to highest. If the selected frequency is not available in the original files, a pop-up window called “Searching for files...” will display, in red color, the missing frequency in the data set.
- Each time a new frequency is selected the FrequencyCurSelect() function within the InputFrame class is activated - reading which frequencies have been selected, converting them to integers, ordering them from low to high, calculating the maxima and minima, and inputting them into a global list.

Edit Frequencies

- By selecting the ‘Edit’ button the user may manipulate all the frequencies in the ‘Select Frequencies’ Listbox. The button will switch the frame to editing mode in which the user will find a tk.Entry box with ‘Add’, ‘Delete’, ‘Clear’, and ‘Return’ buttons beneath.
- The Entry box is where the user can input any frequencies to add or delete from the ‘Select Frequencies’ Listbox. Multiple frequencies can be added at once if they are separated by a single space. By pressing the ‘Add’ or ‘Delete’ buttons the specified frequencies will be added or deleted from the Listbox, respectively. The ‘Clear’ button clears the entire list to be repopulated by using the ‘Add’ button. The ‘Return’ button returns the user to the ‘Select Frequencies’ Listbox.
- After every edition, the Listbox is reorganized to show frequencies in an ascending order.

Select the Data Analysis Method

- *Peak Height Extraction*
 - The Peak Current is calculated as the difference between the absolute maximum and minimum currents of the polynomial regression.
- *Area Under the Curve*
 - The Area Under the Curve is calculated as a Riemann Sum with the number of rectangles equaling the number of current points within the polynomial fit of the smoothed voltammogram.

Selecting X-axis units

- The user has the ability to select the x-axis units from the available ‘Experiment Time’ and ‘File Number’ options.
- *Experiment Time*
 - If the user selects Experiment Time as the x-axis units then the program will use the sample rate to calculate the time it takes to read each point.
- *File Number*
 - If the user selects File Number the x-axis units will be file number.

Select Y Limit Parameters

- To change the y axis limits, one may increase or decrease the values presented in the “Select Y Limit Parameters” entry box. The code is presented in **Figure S15**.

```
#- PHE/AUC Data -#  
ax[1,subplot_count].set_ylim(data-abs(min_data*data),data+abs(max_data*data))
```

Figure S15. Code snippet showing formula for y-axis range calculation

Here the values ‘min_data’ and ‘max_data’ correspond to the values in the second column of the entry box, titled ‘Data Min. Factor’ and ‘Data Max Factor’. The program first calculates the max and min value of the data set – here being the peak height or AUC – and then increases and decreases the axis limits by a factor of that value. If both min_data and max_data are set to 0 then the axes will be exactly at the peak height. If the peak height is 1 and both values are set to 2 then the boundaries will be (1-

$(2*1), 1+(2*1)$), or $(-1,3)$. If the min factor is set to 10 and the max factor is set to 5 then the limits will be $(1-(10*1), 1+(5*1))$, or $(-9,6)$.

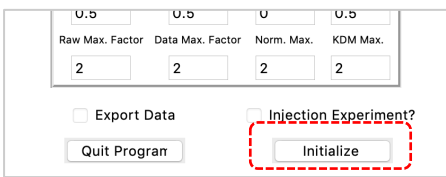


Figure S16. The SACMES Initialize button

When pressed, the initialize button transfers the input parameters from the initial frame to the Analysis module.

Activating Injection Experiment

- A tk.Checkbox labeled ‘Injection Experiment’ is located at the bottom right hand of the InputFrame. Here the user can indicate whether he/she is performing an injection experiment.

- If an injection experiment is being pursued, in which a drug or analyte is introduced at specific times into the system, the user can track the change in measurement response by visually differentiating pre- and post-injection data using different colors.

Activating Text File Export

- On the bottom left hand corner of the Input Frame there is a checkbox labeled ‘File Export’ that will activate file export. See Text File Export for more information.

Real Time Data Manipulation Frame

The Real Time Data Manipulation Frame (**Figure S17**) is created by the StartProgram function within the Initial Input Frame (StartPage) when the ‘Initialize’ button is pressed (indicated with red dashes in **Figure S16**). This frame is viewed alongside the VisualizationFrame and contains widgets for the manipulation of parameters for real-time data processing:

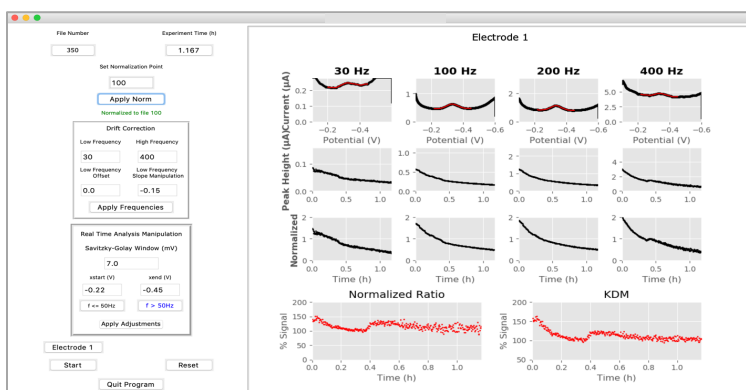


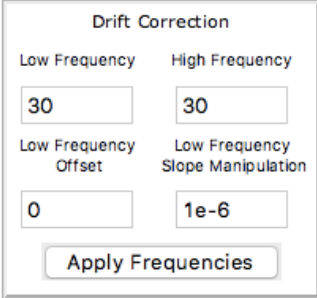
Figure S17. The SACMES Real Time Data Manipulation Frame

This frame allows the user to change processing parameters in real time like the points used for rolling averaging, the voltage range used for data processing or the normalization point.

The widget used to initialize data processing is a button located on the bottom left of the frame labeled 'Start' (see **Figure S17** above) that calls upon the ElectrochemicalAnimation class.

Drift Correction

- In order to correct for baseline drift, the program offers both ratiometric and differential methods that can be controlled in real time.
- Two frequencies, a high and low, may be chosen for the ratiometric and differential analysis. If a frequency that does not exist is entered, a red warning label will appear indicating which frequency does not exist.
- The Low Frequency Slope Manipulation entry allows for the user to manually adjust the slope of the low frequency output (by adding to the dataset a line with the specified slope) in real-time. The Low Frequency Offset is a vertical offset that will raise or lower the y-intercept by the value entered.
- In order to apply these changes, the user must press the 'Apply Frequencies' button (**Figure S18**).

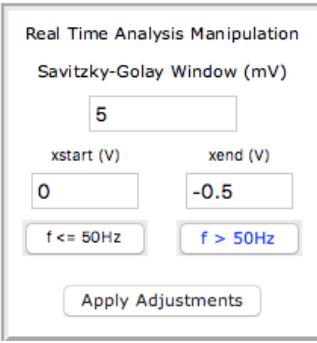


The screenshot shows a dialog box titled "Drift Correction". It has four input fields arranged in a 2x2 grid. The top-left field is labeled "Low Frequency" and contains the value "30". The top-right field is labeled "High Frequency" and also contains "30". The bottom-left field is labeled "Low Frequency Offset" and contains "0". The bottom-right field is labeled "Low Frequency Slope Manipulation" and contains "1e-6". Below these fields is a button labeled "Apply Frequencies".

Figure S18. The SACMES drift correction box

Real-Time Analysis Manipulation

- The program offers the ability to adjust the parameters of the Savitzky-Golay smoothing function, the polynomial regression, the peak height extraction, and the area under the curve (**Figure S19**).
- The Savitzky-Golay smoothing function (essentially a rolling average when a one-degree polynomial correction is selected) can be manipulated by choosing the amount of points averaged by altering the mV window. In other words, the function will calculate the average of all points contained within the mV window selected (this window must be an odd number). Increasing the window range will increase the amount of points that will be averaged and thus the extent of smoothing. **Caution:** the



The screenshot shows a dialog box titled "Real Time Analysis Manipulation". It has a section for "Savitzky-Golay Window (mV)" with an input field containing "5". Below this are two input fields for "xstart (V)" (0) and "xend (V)" (-0.5). There are two buttons: "f <= 50Hz" and "f > 50Hz". At the bottom is a button labeled "Apply Adjustments".

Figure S19. The SACMES box for real-time analysis manipulation

use of large Savitzky-Golay windows can over-smooth data, resulting in deformed voltammograms. Any use of smoothing functions for the analysis of published data should be reported, clearly indicating the parameters used.

- The parameters ‘xstart’ and ‘xend’ allow the user to control the x-axis points considered for the polynomial regression and peak height extraction/area under the curve. They allow the user to manually remove end-points that may present poorer signal-to-noise levels.
- The above-described parameters can be adjusted independently for high and low frequencies (**Figure S19**). The split point for the application of these high/low properties is, by default, 50 Hz, but could be customized by the user in the program by changing the ‘cutoff_frequency’ value in the global variables section (see p.9).

Visualization Frame

The Visualization Frame contains all the matplotlib canvases and artists (e.g., Ticks, Axes, etc.) for data visualization. There is one Visualization Frame for each electrode included in the analysis. These frames are overlaid over one another allowing the user to switch between electrodes using the interactive “Electrode” buttons found in the Real-Time Manipulation Frame:

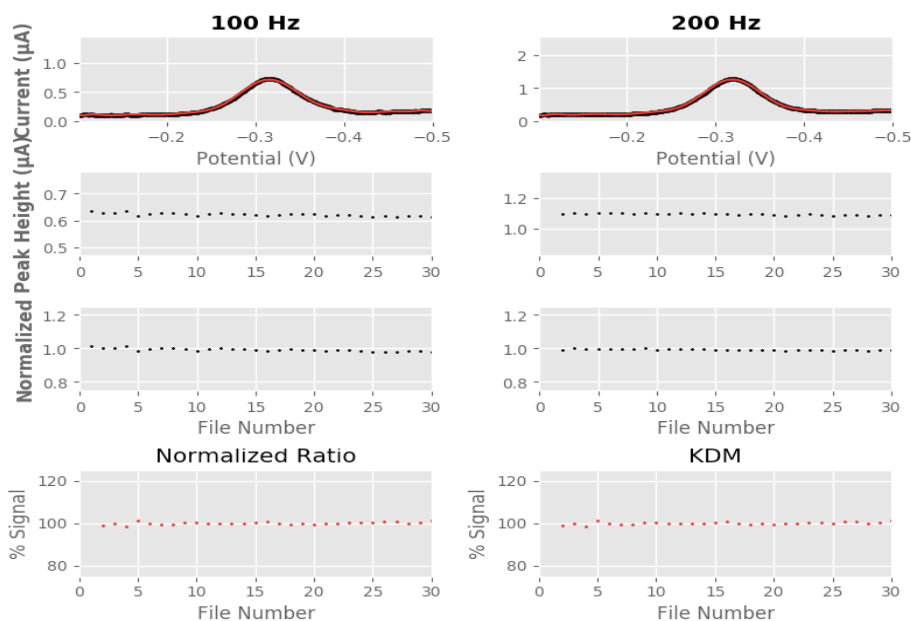


Figure S20. The SACMES Visualization Frame

This frame is where all graphs are presented to the user.

The Visualization Frame is organized as follows:

- Row 1: Overlay of raw voltammograms (black) and polynomial fit of smoothed data (red)
 - Row 2: Raw Peak Currents/Areas Under the Curve
 - Row 3: Normalized Peak Heights/Areas Under the Curve
 - Row 4, Column 1 (left): Ratio of Normalized Frequencies (High/low)
 - Row 4, Column 2 (right): Kinetic Differential Measurement
- The number of columns in each Visualization Frame is determined by the number of frequencies being analyzed, with one column per frequency.
 - The 4th row is always 2 columns while Rows 1-3 will have as many columns as there are frequencies.

Data Export

Activating Data Export

- To activate Data Export, select “Export Data” at the bottom left of the Initial Input Frame (Start Page). This will activate the module ‘TextFileExport’. To personalize the data exported, the class TextFileExport() must be edited (**Figure S21**).

```
#####  
#####  
### Functions for Real Time Text File Export ###  
#####  
#####  
#####  
### Real-Time Text File Export ###  
#####  
class TextFileExport():
```

Figure S21. Code snippet showing the TextFileExport() class

This class covers several lines of code which we are not shown above for the sake of brevity.

They can be found ~below line 40,000 of the SACMES script.

Export File Nomenclature

- In the tk.Entry box labeled ‘Exported File Handle’ on the top right of the Input Frame the user can enter any ASCII string without backslashes followed by the extension ‘.txt’ to be the handle of the exported file.

Format of exported file

- The user can choose to export the analyzed data in a .txt file delimited by a single white space. The file will contain the following data for an experiment with two electrodes at two frequencies:

Column 1: File Number

Column 2: Peak Height/AUC Electrode 1, Freq1

Column 3: Peak Height/AUC Electrode 2, Freq1

Column 4: Peak Height/AUC Electrode 1, Freq2

Column 5: Peak Height/AUC Electrode 2, Freq2

Column 6: Normalized Peak Height/AUC Electrode 1, Freq1

Column 7: Normalized Peak Height/AUC Electrode 2, Freq1

Column 8: Normalized Peak Height/AUC Electrode 1, Freq2

Column 9: Normalized Peak Height/AUC Electrode 2, Freq2

Column 10: Average Normalized Peak Height/AUC Freq1

Column 11: Average Normalized Peak Height/AUC Freq2

Column 12: Standard Deviation Normalized Peak Height/AUC Freq1

Column 13: Standard Deviation Normalized Peak Height/AUC Freq2

Column 14: Normalized Ratio Electrode 1

Column 15: Normalized Ratio Electrode 2

Column 16: Average Normalized Ratio

Column 17: Standard Deviation Normalized Ratio

Column 18: Kinetic Differential Measurement Electrode 1

Column 19: Kinetic Differential Measurement Electrode 2

Column 20: Average Kinetic Differential Measurement

Column 21: Standard Deviation Kinetic Differential Measurement

- If the user alters the normalization point in real-time, the text file will be rewritten to match the parameters given by the user.

Standard Operating Procedure for SACMES_CV.py

Brief Description of the Script Structure

SACMES_CV.py follows the same script structure as SACMES_SWV.py: we first import all module libraries, then global functions, create the user interface, invoke *ElectrochemicalAnimation*, and finally export data via *TextFileExport*. However, SACMES_CV does not use any data processing parameters that can be changed in real time. The purpose of the program is to calculate peak currents or surface areas from voltammograms, and report currents at specific, fixed voltages.

Input File Formatting and Data Extraction

Input File Format

The Input File is the raw data created by an external analytical device. Just like in SACMES_SWV, SACMES_CV can analyze multielectrode data contained within a single file ('Multichannel' setting; default) or in separate files ('Multiplexed' setting, one file per electrode). The default format for data files in a SWV experiment – independent of whether there are individual files for each electrode or a single file for all electrodes – should be as follows:

Column 1: Potentials

Column 2: Currents, Electrode 1

Column 3: Currents, Electrode 2

Column 4: Currents, Electrode 3

And so forth..

Default Input File Nomenclature

If the data for each electrode are within a single file ('Multichannel')

- The file name can begin with any ASCII string; for example, 'ExampleHandle_', followed by an underscore, the file number, and the extension '.txt'. e.g. "ExampleHandle_1.txt" → "ExampleHandle_2.txt" → "ExampleHandle_3.txt" → etc.

- Successive data files must have the same file handle with increasing file number as illustrated above.

If the data for each electrode are separated into individual files ('Multiplexed')

- The file name must then start with the electrode number and an underscore: "E5_ExampleHandle_1.txt"

Graphical User Interface

The SACMES_CV GUI consists of four principle frames (windows for the user to interact with the program): 1) a **Main Window** that acts as a container in which all other frames are embedded; 2) a **Start Page** frame for initial user inputs; 3) a **Setup Frame** to select the forward and reverse segments and adjust data analysis; and (4) a **Visualization Frame** that contains a matplotlib canvas (the canvas where each graph will be drawn and data will be visualized) for each electrode

MainWindow

The MainWindow contains all program frames and widgets: (1) the Initial Input Frame (Start Page), (2) the Setup Frame; and (3) the Visualization Frame. Like in SACMES_SWV.py, the MainWindow (**Figure S13**) acts as a container and only serves as a space holder for the other frames – it does not perform any functions or have any widgets. Refer to p. 18 of this document.

Initial Input Frame (Start Page)

When the user runs SACMES_CV.py from the command line or terminal, the *Input Frame* is the first frame created and the first window to appear (**Figure S22**). In this window the user can specify the file path where the data is located, an import file label and an export file label. Other entry widgets include the number of files to be processed, the analysis interval at which SACMES will read each file (fastest is 1 ms), the scan rate used in the experiments, and a resize interval to adjust the size of the data graphs based on number of files. These parameters are very similar to those seen in SACMES_SWV.py as described above. For reference please go to page 19.

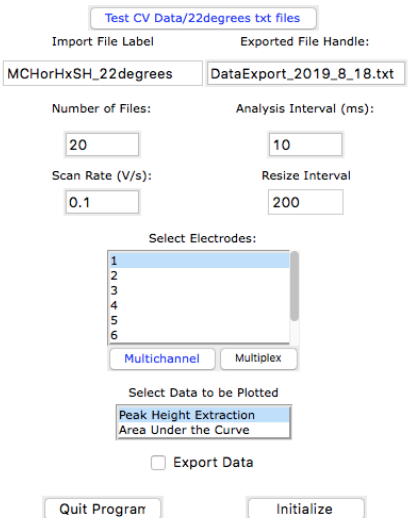


Figure S22. The SACMES_CV Input Frame

The Input Frame contains widgets for parameter entries needed to analyze the data.

Setup Frame

The Setup Frame (**Figure S23**) has four principle functions: (1) selection of forward and reverse segments; (2) adjustment of the voltage range used for data analysis; (3) selection of specific voltages to be analyzed during the course of data analysis; and (4) adjustment of the Y Limit Parameters for creation of the axes within the Visualization Frame (see p. 21). These adjustment values follow the same convention as that for SWV.

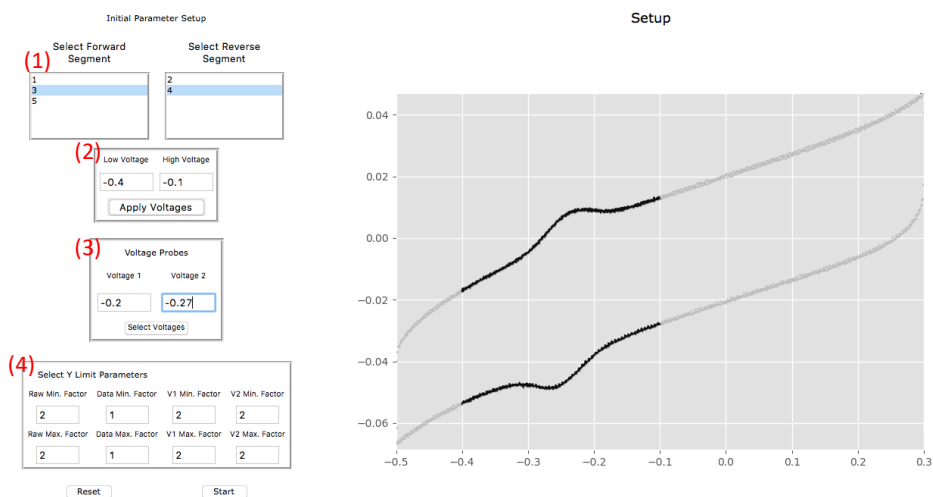


Figure S23. The SACMES_CV Setup Frame

The Setup Frame contains widgets where segment numbers, voltage range and voltage probes are set.

Visualization Frame

The Visualization Frame (**Figure S24**) contains widgets to switch between electrodes as well as the matplotlib canvases that contain animated data. The color green represents data for the forward segment while the color red represents the reverse segment. Row one contains (1) the Cyclic Voltammogram, containing both the forward and reverse segments; and (2) the analyzed data (either PHE or AUC). Row 2 contains (3) the peak potential for the forward segment and (4) the peak potential for the reverse segment. Row 3 contains (5) the currents for voltage probe 1 – here shown as -0.18V – and (6) the currents for voltage probe 2 – shown as -0.27V.

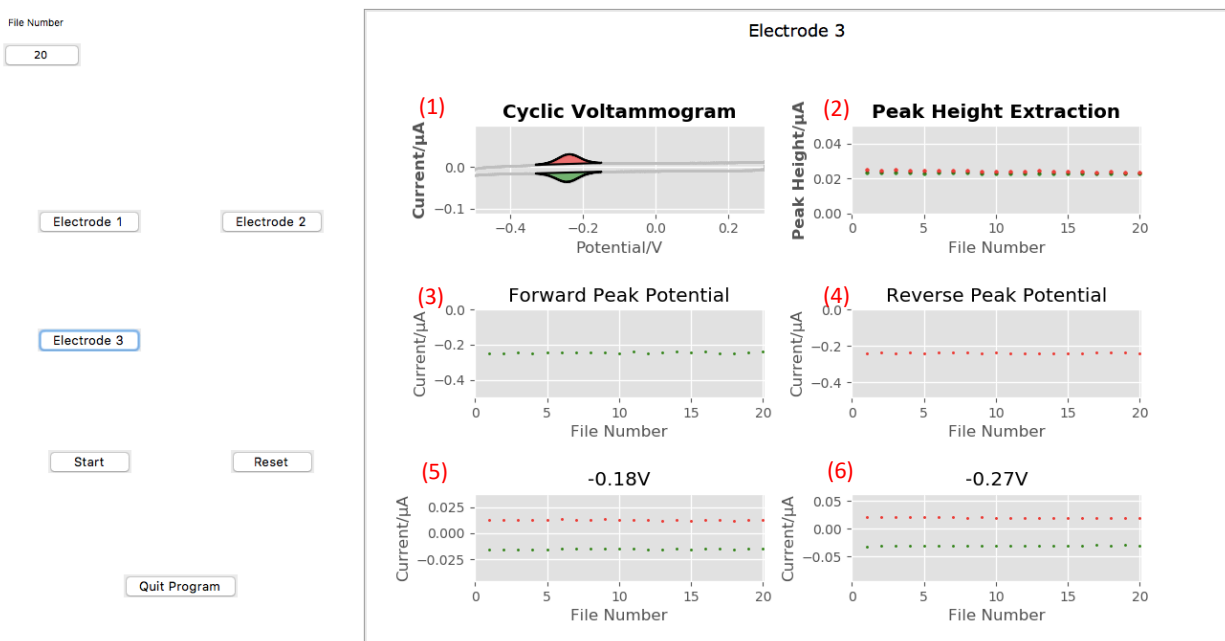


Figure S24. The SACMES_CV Visualization Frame

This frame contains all the data graphs produced by SACMES after data processing.

Settings Toolbar

SACMES_CV.py has a settings toolbar that at the time of this submission had two dropdown menus: (1) Mass Transport Settings and (2) Segment Visualization Settings. The Mass Transport drop down menu allows the user to switch between “solution phase” (CV experiments controlled by diffusion) and “surface bound”, which alter the way in which the baseline used for data processing is calculated (flat vs sloping baseline). Moreover, The Segment Visualization drop down menu allows the user to selectively choose which segments will be animated during data

analysis. This does not change any part of the data processing module, only what is animated in the animation module.

Data Analysis

Data Processing Module (generator)

SACMES_CV.py uses the same ElectrochemicalAnimation module as SACMES_SWV.py (see p.16) but with a new *generator* adapted for the analysis of cyclic voltammograms, illustrating the point that SCMES can be adapted for the processing of different datasets.

The Data Analysis Module for cyclic voltammetry first analyses the forward segment and finishes with the reverse segment. For each segment it performs four main functions. First, it adjusts the potentials and currents to the voltage range selected by the user in the Setup Frame (**Figure S25**). Second, it calculates the peak potential of each segment (**Figure S26**).

```
#####  
### Adjust the potentials and currents to ###  
### the voltage range chosen by the user ###  
#####  
self.adjusted_potentials = [potential for potential in self.segment_potentials if low_voltage <= potential <=  
high_voltage]  
self.adjusted_index_list = [self.potential_dict[potential] for potential in self.adjusted_potentials]  
self.adjusted_currents = [currents[index] for index in self.adjusted_index_list]  
adjusted_segment_data[count] = [self.adjusted_potentials, self.adjusted_currents]
```

Figure S25. Code snippet showing function to adjust voltage range considered

```
def extract_peak(self, count):  
    ### extract the potential of the peak current and then extract  
    ### the associated index adjusted to the user-chosen range  
    try:  
        #-- if the peak is positive (cathodic) --#  
        if self.sign_dictionary[count] == 'cathodic':  
            peak_potential = self.data_dict[max(self.adjusted_currents)][0]  
            peak_index = self.potential_dict[peak_potential] - self.adjusted_index_list[0]  
  
        #-- if the peak is negative (anodic) --#  
        elif self.sign_dictionary[count] == 'anodic':  
            peak_potential = self.data_dict[min(self.adjusted_currents)][0]  
            peak_index = self.potential_dict[peak_potential] - self.adjusted_index_list[0]  
        return peak_potential, peak_index  
  
    except:  
        print('\n%s_raw_generator: Error in extract_peak()\n' % self.spacer)
```

Figure S26. Code snippet showing calculation of the peak potential by generator function

Third, it creates a linear baseline – the only piece of data analysis that is dependent on the mass transport settings. If the mass transport settings are for **surface bound** species, the baseline

creation is identical to that of SWV – drawing a baseline between the local minima/maxima on either side of the peak potential (**Figure S27**).

```
#####
### Create a linear baseline ###
#####

!-- if it is a surface bound species --#
if mass_transport == 'surface':
    try:
        proto_baseline = np.linspace(vertex1,vertex2,len(baseline_currents))
        linear_baseline = []

        !-- make sure the baseline currents to not go out of the
        !-- boundaries of the current segment
        for index in range(len(proto_baseline)):
            if self.sign_dictionary[count] == 'cathodic':
                if proto_baseline[index] <= baseline_currents[index]:
                    linear_baseline.append(proto_baseline[index])
            if self.sign_dictionary[count] == 'anodic':
                if proto_baseline[index] >= baseline_currents[index]:
                    linear_baseline.append(proto_baseline[index])
                    baseline_indeces.append(index)
```

Figure S27. Code snippet showing creation of linear baselines

If the settings are for **solution phase** species, it locates the absolute maxima/minima of the adjusted currents (depending on the orientation of the segment) and then creates a baseline using the slope of the points around the extrema (**Figure S28**).

```
!-- if it is a solution phase species --#
elif mass_transport == 'solution':
    try:
        if self.sign_dictionary[count] == 'cathodic':
            vertex_current = min(vertex1,vertex2)
        elif self.sign_dictionary[count] == 'anodic':
            vertex_current = max(vertex1,vertex2)
        # calculate the slope of the data points surrounding the vertex #
        # (a ±5% range)
        slope = (slope_currents[-1]-slope_currents[0])/(slope_potentials[-1]-slope_potentials[0])
        #print('%s Vertex Current: ' % count,vertex_current)
        #print('%s Slope:' % (self.spacer,count),slope)

        # using this slope, create a linear baseline using the entire
        # range of potentials chosen by the user
        proto_baseline = []
        for index in range(len(baseline_currents)):
            proto_baseline.append(slope*(baseline_potentials[index]-vertex_potential)+vertex_current)
        # remove any currents that are above/below the currents of the segment
        linear_baseline = []
        baseline_indeces = []

        for index in range(len(proto_baseline)):

            if self.sign_dictionary[count] == 'cathodic':
                if proto_baseline[index] <= baseline_currents[index]:
                    linear_baseline.append(proto_baseline[index])
                    baseline_indeces.append(index)
            if self.sign_dictionary[count] == 'anodic':
                if proto_baseline[index] >= baseline_currents[index]:
                    linear_baseline.append(proto_baseline[index])
                    baseline_indeces.append(index)
```

Figure S28. Code snippet showing calculation of slope for solution phase analysis

Finally, it extracts the Peak Height or Area Under the Curve using the same methodology as that for SACMES_SWV.py.

Data Export

Data Export for SACMES_CV.py operates in the same fashion as SACMES_SWV.py but instead of a single exported file, it exports an independent file for each electrode.

Format of exported file

Column 1: File Number

Column 2: Forward Segment Peak Height/AUC

Column 3: Reverse Segment Peak Height/AUC

Column 4: Forward Segment Peak Potential

Column 5: Reverse Segment Peak Potential

Column 6: Forward Segment Currents for Voltage Probe 1

Column 7: Reverse Segment Currents for Voltage Probe 1

Column 8: Forward Segment Currents for Voltage Probe 2

Column 9: Reverse Segment Currents for Voltage Probe 2

Altering the Export Path

If you select “Data Export” on the Initial Input Frame you will be able to select a different file path for the exported folder by pressing the button ‘Alter Export Path’ on the Setup Frame. This will bring up a file dialog in which the user can browse the files on their computer and select a new file path.