

User and Developer Guide

Alexander Bell

November, 2025

Overview

This software has been developed to automatically conduct a range of potentiostatic and galvanostatic experiments. It extends the typical capability of open-source control software by enabling the setup of multiple experiments using comma-separated variable inputs, allowing flexibility in parameters (e.g., using “OCP” for potentials, “STEP” for scan rates, and “None” for current and potential limits), and the execution of more intricate experimental protocols. Key features include:

- Expanded experiment suite: This program supports cyclic voltammetry (CV), galvanostatic charge/discharge (GCD), linear sweep voltammetry (LSV), chronoamperometry (CA), chronopotentiometry (CP), self-discharge (SD) and rate-testing experiments.
- Open circuit potential (OCP) equilibration: If selected/required, pre-experiment equilibration is handled automatically, ensuring the cell reaches a stable condition before measurements begin. The OCP value can also be used directly as an input parameter.
- Cutoff and equilibration conditions: Potential and current cutoffs, as well as equilibration conditions, can be configured to automatically stop or advance experiments upon reaching specified thresholds. These features minimise the risk of cell damage and enhance experiment efficiency.

- Dynamic mid-experiment adjustments: The program enables on-the-fly updates to the number of half-cycles performed during GCD experiments. This feature allows users to extend their view into cycling lifetime without requiring the cell to stop cycling and setting up a further experiment.
- Summary file creation: A detailed summary file is constructed as experiments proceed, logging progress and storing relevant metadata not included in the raw data files.
- The graphical user interface (GUI) includes:
 - Hover-over tooltips for intuitive user guidance.
 - Progress bars and information displays to indicate experiment status.
 - Comprehensive plotting options for real-time and comparative data visualisation.
 - Optional experiment notes added to the summary file for better data contextualisation.
- Potential/current profile previews: Certain experiments offer the ability to preview the potential/current profile across queued experiments, allowing users to visualise the cell's expected behaviour and verify that the experimental setup aligns with their intentions.
- Dynamic experiment timing and progress updates: Certain experiments dynamically calculate and update the remaining experiment time, ensuring the progress bar and estimated time remaining reflect real-time conditions, including variations caused by OCP equilibration duration and potential/current thresholds being reached.
- Error handling and compatibility: A parameter check button ensures input compatibility, whilst robust error-handling mechanisms improve reliability and prevent disruptions during experiments.

- Modular code structure: The modular design of the program results in standardised code across different experiments, ensuring high readability and simplifying the addition of new experimental procedures with minimal modification to the existing code.
- Designed with adaptability in mind: May be modified to interface with other low-cost, “home-built” potentiostat devices by adjusting the hardware control functions as needed.

1 The Core Program and its Functions

1.1 Setting up

After connecting the USB potentiostat device to the PC, click “Connect” within the Hardware tab in the GUI to toggle the device to the connected state. By default, the calibration information from the device will automatically populate the calibration fields, but this can be loaded from the device or set manually.

Once the device is connected and calibrated, the experiments found in the adjacent GUI tabs can be run.

1.2 class States

The States class functions as a simple state machine, defining a named list of states to represent different operational modes or behaviours in the code. A global variable, `state`, is used to hold the current state by setting it to one of the predefined values in the States class (e.g., `States.Measuring`). This global state is then referenced within `periodic_update()` to determine and execute the appropriate behaviour for each operational mode. The states cover key phases of the program, such as measurement modes and idle states.

1.3 `periodic_update()`

When the program is launched, a program timer is set up to call this function at regular intervals. The function handles reading and displaying live cell data and calls the `update()` function for the various experiments. Its behaviour is dictated by the global state, which determines the current operational mode. Together with the `States` class, this function forms the backbone of the program, orchestrating the main control loops and managing operations outside of the execution stacks.

1.4 `read_potential_current()`

This function retrieves the most recent potential and current values from the device's analog-to-digital converter (ADC). Once the ADC conversion is complete, the raw potential data is processed and converted to volts (V), with compensation for the calibration offset from the Hardware tab. Similarly, the raw current data is converted to milliamperes (mA), accounting for the selected current range and applying shunt calibration adjustments from the Hardware tab. These values in V and mA are stored as the global parameters `potential` and `current`, respectively.

1.5 `set_output()`

This function sends the desired current or potential to the digital-to-analog converter (DAC) to be applied to the cell. It is used to control the cell in either a potentiostatic mode (by sending a potential to set across the device) or a galvanostatic mode (by sending a current to apply to the device).

1.6 Software options menu

A menu containing values for universal parameters, together with more advanced experiment-specific parameters, can be found in the Hardware tab by clicking the Software options menu button. As described within the menu, changes

to these default values are only applied to the current session, so users should take care between sessions if the need to frequently update them arises. Should the user wish to change them permanently within the code, the default values for these parameters are read from the `global_software_settings` dictionary at the top of the `.py` file.

1.7 Hardware changes

The base USB potentiostat model has a sampling period of 90 ms using an MCP3550-type ADC. If the user wishes to increase the time resolution of the data, this can be achieved by replacing this ADC with the MCP3553 for a sampling period of ~17 ms.

Note: If this hardware upgrade is performed, ensure that the global variable in the code, `adcread_interval`, is updated to reflect the new sampling period. This adjustment is crucial for data to be sampled at the increased rate.

1.8 Compatibility with MYSTAT

If the user wishes to update the software to be compatible with the MY-STAT potentiostat, the functions and global parameters relevant to current range switching and shunt calibration at the top of the code can be straightforwardly amended to reflect the extra 200 mA range option. This can be done by comparing this code with the controller provided with the MYSTAT (<https://doi.org/10.1016/j.ohx.2020.e00163>).

2 Experiment Functions

To run an experiment, the button must first be pressed to validate the parameters provided in the GUI by calling the functions within Section 2.1.

After the parameters have been successfully validated, experiments may be started by pressing the button, which executes the functions

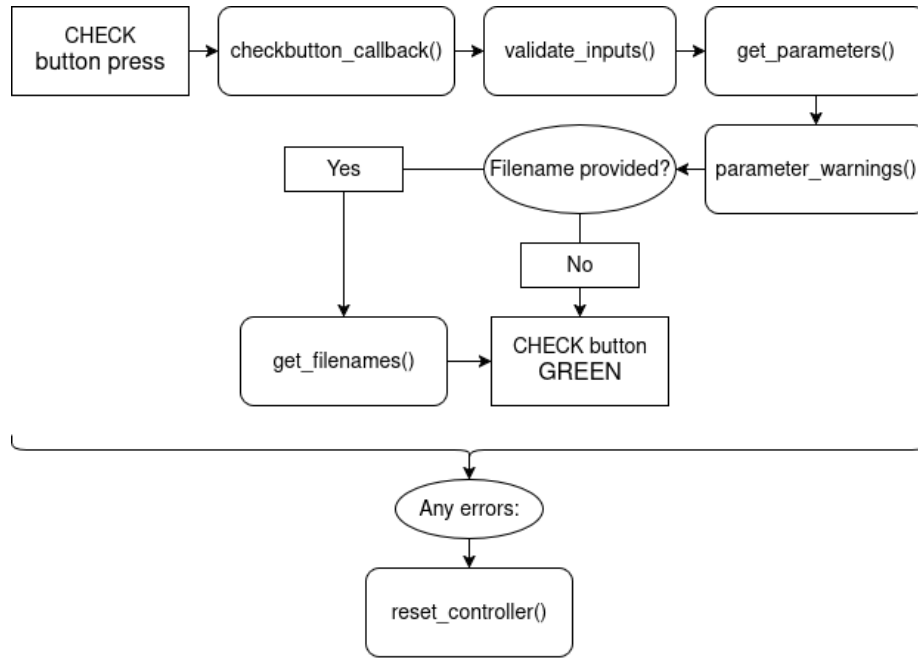



Figure 1: Flowchart of the execution stack handling the validation of input parameters.

within Section 2.2.

The following logic and functions are common to all experiments. Experiment-specific features are described in Section 3.



2.1 Parameter functions


The following functions comprise an execution stack for reading the GUI experiment parameter inputs and ensuring they are appropriate for the experiments being performed. After execution, the user is either allowed to proceed with their experiments or is prevented from doing so until compatible parameters have been provided. A simplified flowchart for this execution stack is illustrated in Figure 1.

Upon clicking the  button, `checkbutton_callback()`, the calling function for all parameter functions, is executed.

checkbutton_callback()

Handles the execution of the parameter functions listed below. Each function is performed in turn, providing the user with feedback *via* dialogue boxes if any parameters are invalid or incompatible.

Upon successful execution of the parameter functions, the  button turns green, , indicating that the experiments may now be started. This is handled within the program by changing a global Boolean check-state to True.

If any parameter inputs are changed, the button returns to its original state, , the global check-state is reset to False, and the updated parameters must pass a successful check before the experiments may be started.

validate_inputs()

Ensures all parameter inputs are of the correct format, are of the correct length if multiple experiments are being performed, and that inputs are provided where expected.

get_parameters()

Constructs a global dictionary containing all of the experiment parameters.


parameter_warnings()

Provides users with warnings *via* dialogue boxes if any entered parameter inputs are unused (checkboxes unchecked), or if any parameters may present compatibility issues depending on equilibrated OCP values.

get_filenames()

Constructs filenames and filepaths for the experiments along with a summary file (see Section 2.3.3). The base filename/filepath is extracted from the GUI and appended with a suffix including the experiment type and relevant experiment parameters.

If a filename but no filepath is given, the user is prompted whether the current directory (the directory the program file is located in) should be used to store the files. If no filepath or filename is given, `get_filenames()` and `validate_filenames()` are not ran, and the check-state relies only on the experiment input parameters.

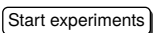
Note: A valid filename/filepath must be given for experiments to run, so  must be pressed again once a filename/filepath has been provided.

validate_filenames()

Validates the constructed filepaths and by attempting to open each file.

2.2 Main loop and core functions

The following functions comprise an execution stack for initialising, running, and stopping the experiments. A simplified flowchart for this execution stack is illustrated in Figure 2.

Upon clicking the  button (provided that the global check-state of the given parameters is True), `intialise()` is called to initialise the experiments to be ran.

Once the experiments have been initialised, `start()` begins the first experiment/segment and sets the machine state, `state`, to `States.Measuring`, so that `periodic_update()` continuously calls `update()`.

Upon completion or interruption of the experiment, `stop()` is called to either call back to `start()` if there are experiments remaining, or, if the experiment is

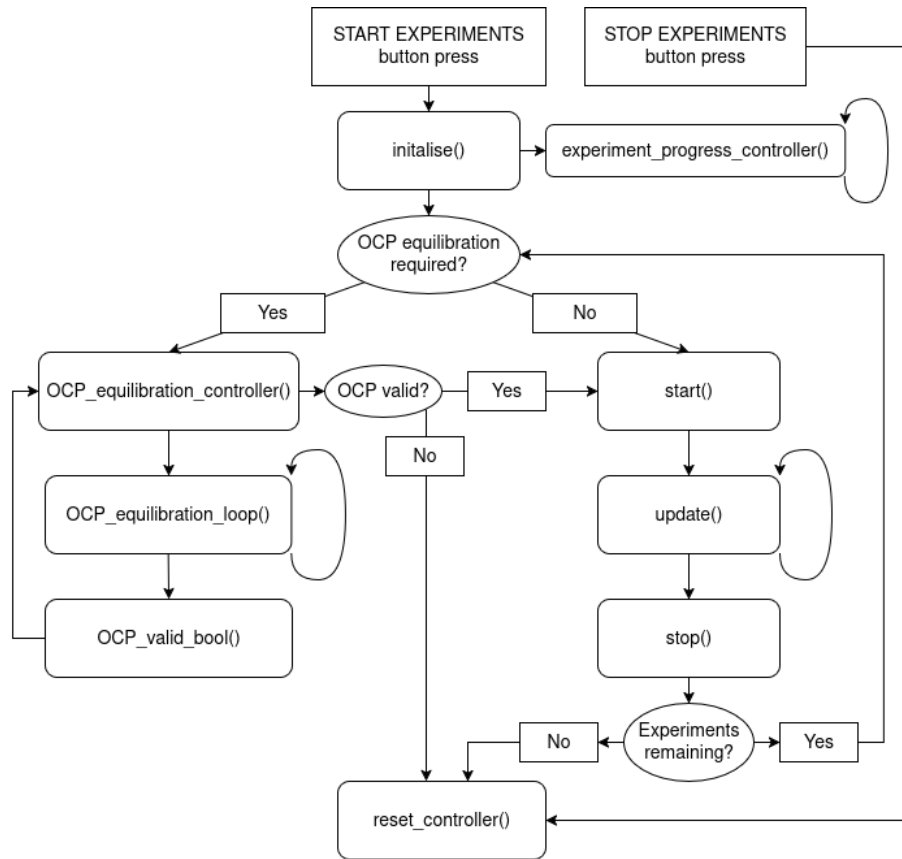


Figure 2: Flowchart of the execution stack handling the initialisation, running, and stopping of experiments.

either interrupted using the `Stop experiments` button or if all experiments have completed, to switch the cell off, set the machine state to `States.Stationary_Graph`, and reset global the experiment parameters.

In some cases, the user may wish to equilibrate the open circuit potential (OCP) to use as an input parameter, or to ensure the cell is relaxed before proceeding with experiments at the next potential window. If so, `initialise()` and `stop()` pass to the OCP equilibration functions (described in Section 2.4) before `start()` is called, and the experiments proceed as usual.

2.2.1 `initialise()`

Initialises the experiment data dictionary, freezes the GUI input fields by calling `freeze_inputs()`, and writes all experiment information to the summary file. If OCP equilibration is required the program passes control to `OCP_equilibration_controller()`, otherwise, it calls `start()` after a fixed time delay defined in the input parameters. This delay is reflected within the GUI by the "Current program state" box under "Experiment information" and a yellow border around the progress bar.

2.2.2 `start()`

Begins the experiment by opening the experiment data file, writing the experiment/segment information to the summary file, setting up the plot area, updating GUI experiment information, setting the progress bar border to green, and initialising the experiment data buffers. These data buffers add the measured potential and current values until the number of samples to average is reached, and the average data point is calculated.

The cell is then prepared for measurement, and an initial potential or current is applied to the cell. Following this, state is set to `States.Measuring`, and `update()` is continuously called *via* `periodic_update()`.

2.2.3 `update()`

This function handles the core loop of the experiment. Experiment-specific detail is described in Section 3, but the function proceeds in the typical fashion:

- The elapsed time for the experiment is stored.
- If the experiment or segment has completed, `stop()` is called.
- For some experiments, the elapsed time is used to determine the potential or current at which to set the cell, and this command is then sent to the DAC.
- The potential between the working and reference electrode and the current between the working and counter electrode is measured.
- If an average data point has been calculated, the time, potential, and current are written to the experiment data file, alongside experiment-specific data.
- This averaged data is then stored in the data dictionary for plotting.
- `update_plot()` is called to display the data (see Section 2.3.1).
- For some experiments, break conditions (*e.g.*, potential/current limits or equilibration parameters) in the experiment/segment are evaluated based on the recent data, and `stop()` is called if these conditions are met.

This sequence repeats for each call of `periodic_update()` whilst `state` remains set to `States.Measuring`.

2.2.4 `stop()`

Handles the completion of an experiment, either setting up the next experiment/segment or resetting the program for new experiments to be ran. If the experiment has not been interrupted, the program proceeds as follows:

- If there are more experiments/segments: The experiment/segment index is incremented, the GUI is updated, and either the completed experiment information is written to the summary file and the next experiment/segment begins by calling `start()`.
- If the experiments have completed: The completed experiment information is written to the summary file, the GUI is updated and accompanied by a completed progress bar, and `reset_experiment_controller()` is called.

If the experiment has been interrupted the cell is switched off, state is set to `States.Stationary_Graph`, interruption information is written to the summary file, the GUI is updated, and `reset_experiment_controller()` is called.



Note: For galvanostatic experiments, a current setpoint of 0 is sent to the DAC upon completion for safety, as switching to manual control within the Hardware tab will apply the last DAC command.




2.2.5 `reset_experiment_controller()`

This function controls the resetting of global variables to allow future experiments to be ran cleanly, without the global variables assigned during previous experiments or `checkbox_callback()` calls causing unintended behaviour.

In all scenarios in which this function is called, a try/except block attempts to close the data output and summary files to ensure no files are left open before executing the scenario-specific code.

This function is called in the following 4 scenarios:

- A parameter input is changed: If the parameters have been checked, i.e.,  is displayed in the GUI, the checkbox style is reset to , all global parameters governing the check-status and experiment progress are reset, and the progress bar is reset.

- A `checkbutton_callback()` fails: The checkbutton style is set to , all global parameters governing the check-status and experiment progress are reset, and the progress bar is reset.
- All experiments have completed: Global parameter and data dictionaries are cleared, all global parameters governing experiment progress and check-status are reset, any experiment progress timers are stopped, the checkbutton style is reset to , the progress bar is reset, and `freeze_inputs()` is called to unfreeze the GUI input fields and checkboxes.
- Experiments are interrupted: Global parameter and data dictionaries are cleared, all global parameters governing experiment progress and check-status are reset, any experiment progress timers are stopped, the checkbutton style is reset to , the progress bar is frozen and set to solid red to reflect the interruption, and `freeze_inputs()` is called to unfreeze the GUI input fields and checkboxes.

2.2.6 `freeze_inputs()`

Controls freezing and unfreezing of all parameter input fields and checkboxes in the GUI. When experiments begin, the input fields and checkboxes are frozen to save the initial setup of the experiments whilst they are running. After experiments have finished, whether by interrupting or running to completion, the input fields and checkboxes are unfrozen to allow new experiments to be entered.

2.3 GUI and additional functions

2.3.1 `update_plot()`

This function is called within each call of `update()`, with different visualisation options for each experiment type (see Section 3 for a detailed explanation). For all experiments, the user is presented with the option to display the data

from the current experiment/cycle/segment alone, or accompanied by data from previous experiments/cycles/segments for comparison.

The plot area is first initialised by clearing the legend and the plotting frame to which the data is added. Relevant data is then extracted for plotting, depending on which plotting options have been selected in the “Plot options” box within the experiment tab.

2.3.2 `update_progress_bar()`

Updates the progress bar to reflect the time elapsed or current cycle/segment as a fraction of the total experiment time or total cycles/segments.

2.3.3 `write_summary_file()`

A detailed explanation of the experiments being performed is constructed within a summary file by calling this function and writing blocks of text in the following scenarios:

- When experiments begin, this function is called within `initialise()`. The summary file is opened and populated with all the experiment parameter inputs from the GUI, i.e., the experiment parameters for all the experiments the user has queued to run. It also writes the filepath locations for all experiment output data files, and any experiment notes written to the optional experiment notes box in the GUI.
- As each sub-experiment begins this function is called within `start()`. The experiment parameters are written, along with the start time of the experiment.
- If potential/current limits or equilibration conditions are met, this function is called within `update()` and the relevant data is written.
- Upon completion of an experiment this function is called within `stop()`. The finish time of the completed experiment is written along with any

other significant information about the experiment, e.g., whether any current or potential limits were reached.

- Once all experiments are complete, this function is again called within `stop()` and “MEASUREMENTS COMPLETE” is written to the final line of the file. If for any reason the experiments are interrupted, the nature of the interruption and time it occurred are written to the file, and the final line instead reads “MEASUREMENTS INCOMPLETE”.

If the OCP equilibration process is used, this function is also called within `OCP_eqilibration_controller()` at the start and end of OCP equilibration to write information about the process.

2.3.4 Tool tips

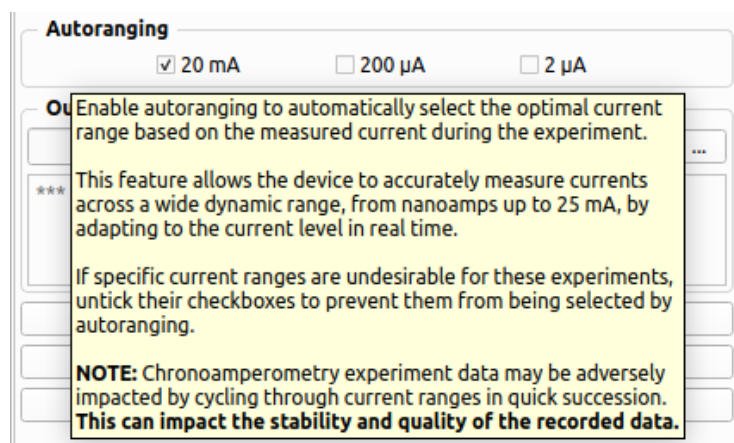


Figure 3: A tooltip which displays for the autoranging box on the chronoamperometry experiment tab.

For each of the experiment parameter input fields and checkboxes in the GUI, hover-over tool tips are available to assist the user in setting up their experiments.

For parameter inputs, the tool tips include information on accepted inputs, and any requirements on length based on the length of other inputs. For checkboxes, the tool tip provides context on their behaviour if checked or unchecked.

More complex tool tips are included to better describe features such as autoranging, as depicted in Figure 3.

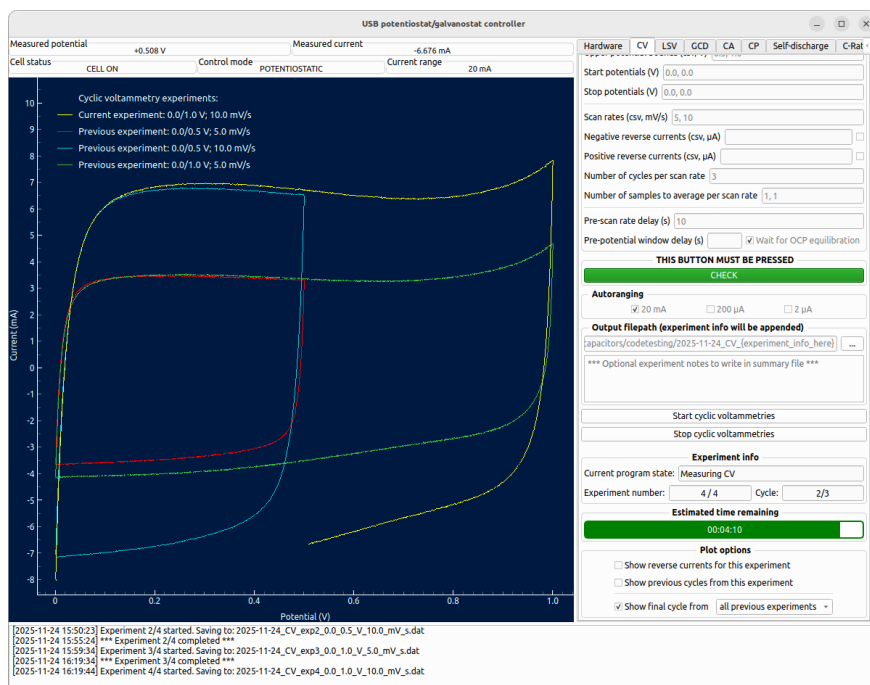


Figure 4: The GUI during a cyclic voltammetry experiment.

2.4 OCP equilibration functions

The following functions are used in the cases where a user may wish to equilibrate the open circuit potential (OCP) to use as an input parameter, or to ensure the cell is relaxed before proceeding with the next experiment.

The equilibrated OCP is defined by the potential when the change in the potential over a particular timescale falls below some threshold. The threshold and timescale are defined in the 'Software options menu' and stored in the

global scope as `OCP_eq_threshold` and `OCP_eq_timescale`, respectively.

Once the OCP has been equilibrated, the experiments proceed as described in Section 2.2.

2.4.1 `OCP_eqilibration_controller()`

This function controls the OCP equilibration process, called by `initialise()` before the OCP has equilibrated, and afterwards by `OCP_eqilibration_loop()` if the equilibration is successful, or if the equilibration reaches the OCP timeout threshold defined in the global scope. The behaviour of this function in these three scenarios is as follows:

- Before equilibration: A deque data container for the potential history is initialised, alongside time and potential data buffers with the number of samples to average defined in the 'Software options menu'. The plot area is initialised, and the OCP equilibration is reflected in the GUI and accompanied by a solid yellow border around the progress bar. The cell is switched off, the start time of the equilibration is stored, and state is set to `States.Measuring_OCP_eq`. The function handling the equilibration, `OCP_eqilibration_loop()`, is then continuously called by `periodic_update()`.
- Successfully equilibrated: Upon successful equilibration, the OCP value and finish time is stored, equilibration information is written to the summary file, the GUI input parameters are updated to reflect the OCP value where 'OCP' was used, and the following experiment begins via `start()` following OCP validity checks within `OCP_valid_bool()`, if required.
- Equilibration timeout reached: If the elapsed time of the equilibration process surpasses the timeout threshold defined in the 'Software options menu', state is set to `States.Stationary_Graph`, equilibration timeout information is written to the summary file, and the experiments are interrupted and reset *via* `reset_experiment_controller()`.

2.4.2 OCP_valid_bool()

For experiments which accept 'OCP' as an input parameter and which may become invalid with other experiment parameters, this function is called after OCP equilibration to determine the validity. The experiments continue if the OCP is valid with the other experiment parameters. If the OCP is invalid with the experiment parameters, however, the experiments will stop in the same fashion as if they were manually interrupted, with the invalid nature of the equilibrated OCP described in the summary file.

2.4.3 OCP_eqilibration_loop()

The main loop of the OCP equilibration, called *via* `periodic_update()` continuously whilst state is set to `States.Equilibrating_OCP`.

With each call, the most recent cell potential and elapsed time is measured. Once the data buffers have calculated a new average, the averaged potential and elapsed time are appended to the potential history deque.

Once `OCP_eq_timescale` seconds have passed, data older than this timescale is removed from the deque, and the most recent potential data is compared with the oldest data in the deque.

If the difference between the most recent potential and the oldest potential (the potential from approximately `OCP_eq_timescale` seconds ago) is less than `OCP_eq_threshold`, the OCP is considered equilibrated and the controller function, `OCP_eqilibration_controller()`, is called. Otherwise, this main loop is called continuously until the potential equilibrates or the `OCP_eq_timeout` threshold is reached.

If the elapsed time since OCP equilibration began surpasses `OCP_eq_timeout`, the experiments are interrupted by calling `OCP_eqilibration_controller()`.

2.4.4 OCP_update_live_graph()

Similarly to `update_plot()` described in Section 2.3.1, this function plots the live OCP equilibration data within the plot frame. It also prints a relevant

equilibration data to the plot legend, including OCP equilibration parameters and the change in potential over the most recent equilibration timescale, giving indication to the progress of the equilibration.

3 The Experiments

Due to the differing requirements of the various experiments which may be performed, each experiment, whilst following the standardised code structure described in Section 2.2, contains behaviour specific to that experiment. This section describes the main differences, and any additional functionality available to that experiment.

3.1 Cyclic voltammetry

Cyclic voltammetry (CV) experiments, indicated by the “cv_” prefix within the code, are performed as a series of unique experiments for each scan rate at each potential window. An experiment is performed for each scan rate across the first potential window, followed by an experiment for each scan rate across the second potential window, and so forth, until all experiments are complete.

If any experiment uses ‘OCP’ as a lower or upper potential limit, or the user wishes to wait for OCP equilibration to ensure that the cell is in a stable state, the OCP equilibration is performed before the first experiment at each potential window.

Output data files are generated for each experiment, named according to the base filename provided in the GUI and appended with a suffix containing “CV” followed by the potential window and scan rate of the experiment. Each file contains data columns of cycle number, elapsed time (s), measured potential (V), and measured current (A).

3.1.1 Reverse current limits

A positive and negative reverse current limit can be provided for each scan rate. Each time a limit is reached, the time to the next lower/upper bound is calculated and the applied potential sweep is advanced to the same potential scanning in the opposite direction. The remaining time is then updated to reflect the advance in the potential sweep due to a reverse current limit being reached.

Note: To prevent undesirable re-triggering of the current limit, a reverse current lockout time delay, defined in the ‘Software options menu’, is used to ensure that sufficient time has passed before re-evaluating whether the current has surpassed the triggered current limit.

Warning: If a current limit is triggered during the 0th cycle (i.e., a cycle used to sweep the cell from the start potential to within the potential limits if the starting potential is outside of the potential limits of a given experiment), the experiments are interrupted.

3.1.2 `cv_calculate_experiment_time()`

The total and remaining experiment times are calculated within this function. It is first called during `checkbutton_callback()`, and subsequently called after each experiment to maintain accuracy in the remaining time.

If OCP equilibration is used, the duration for each equilibration is initially approximated as the `OCP_eq_timescale` set in the ‘Software options menu’, and then approximated as the average of all preceding equilibration stages for subsequent experiments.

Note: If the pre-experiment estimated OCP (the measured cell potential) or the equilibrated OCP is invalid with any future experiment parameters, experiments may proceed but the total and remaining experiment times are not calculated, and the progress bar will not display an estimated time remaining.

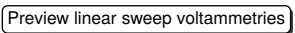
3.2 Linear sweep voltammetry

Linear sweep voltammetry (LSV) experiments, indicated by the “lsv_” prefix within the code, are performed as a series of unique experiments for each scan rate at each potential window. An experiment is performed for each scan rate across the first potential window, followed by an experiment for each scan rate across the second potential window, and so forth, until all experiments are complete.

If any experiment uses ‘OCP’ as a start or stop potential, or the user wishes to wait for OCP equilibration to ensure that the cell is in a stable state, the OCP equilibration is performed before the first experiment at each potential window.

Output data files are generated for each experiment, named according to the base filename provided in the GUI and appended with a suffix containing “LSV” followed by the potential window and scan rate of the experiment. Each file contains data columns of experiment stage, elapsed time (s), measured potential (V), and measured current (A). The experiment stage column contains strings indicating whether the experiment is in the initialising, holding, or linear sweep stage.

3.2.1 lsv_preview()

Upon clicking the  button, this function is called to display the potential profile applied to the cell across the experiments within the plot area. This offers the user the ability to visualise the potential profile constructed by the input parameters to ensure that the experiments are set up as intended.

3.2.2 Current limits

A positive and negative reverse current limit can be provided for each scan rate. If a limit is reached, the experiments proceed to the next queued

experiment, and the remaining time is updated to reflect the experiments being advanced.

3.2.3 `lsv_calculate_experiment_time()`

The total and remaining experiment times are calculated within this function. It is first called during `checkboxbutton_callback()`, and subsequently called after each experiment to maintain accuracy in the remaining time.

If OCP equilibration is used, the duration for each equilibration is initially approximated as the `OCP_eq_timescale` set in the 'Software options menu', and then approximated as the average of all preceding equilibration stages for subsequent experiments.

3.3 Galvanostatic charge-discharge cycling

Galvanostatic charge-discharge (GCD) experiments, indicated by the "gcd_" prefix within the code, are performed as a series of unique experiments for charge/discharge current pair at each potential window. An experiment is performed for each charge/discharge current pair across the first potential window, followed by an experiment for each charge/discharge current pair across the second potential window, and so forth, until all experiments are complete.

If any experiment uses 'OCP' as a lower or upper potential limit, or the user wishes to wait for OCP equilibration to ensure that the cell is in a stable state, the OCP equilibration is performed before the first experiment at each potential window.

Output data files are generated for each experiment, named according to the base filename provided in the GUI and appended with a suffix containing "GCD" followed by the potential window and charge/discharge current of the experiment. Each file contains data columns of cycle number, half-cycle number, elapsed time (s), measured potential (V), and measured current (A).

A further data file is generated for each experiment, named similarly but with “capacities” appended to the suffix, containing data columns of cycle number, charge capacity (Ah), and discharge capacity (Ah).

3.3.1 Update half cycles mid-experiment

When performing long-term GCD cycling to investigate the evolution of the potential profile and degradation in cell performance, it can be advantageous to extend the experiment to capture behaviour over a greater number of cycles than initially set in the experiment parameters. When GCD experiments are running, an button becomes available in the GCD tab to update the number of half-cycles performed per experiment, which applies to the current experiment and all subsequent experiments queued.

3.4 Chronoamperometry

Chronoamperometry (CA) experiments, indicated by the “ca_” prefix within the code, are performed as a single experiment with a series of unique segments for each sequence potential. Within each segment, the set potential can be stepped or ramped to the sequence potential and held for the defined hold time for that segment. Upon completion of the segment, `stop()` is called to increment the segment number, before passing to `start()` to begin the next segment. This is repeated until all segments are complete.

If any segments require OCP equilibration (i.e., if any segment potential is defined as ‘OCP’ in the input parameters, or the user wishes to equilibrate the cell to achieve a stable state before the experiment begins), the equilibration is performed once at the beginning of the experiment. Any parameters defined as ‘OCP’ use this equilibrated value throughout the experiment.

A single output data file is generated for this experiment, named according to the base filename provided in the GUI and appended with a suffix containing

“CA” followed by the potential sequence. It contains data columns of segment number, total elapsed time (s), segment elapsed time (s), measured potential (V), and measured current (A).

3.4.1 Alternating parameters

Alternating parameters may be set when the user wishes to apply different equilibration tolerances and timescales to potentials within the potential sequence by clicking the `Use Alternating Parameters` button. The original parameter boxes are renamed to reflect the A-sequence parameters, and a dropdown area displays the B-sequence parameters. This may be useful to users who, for instance, want different equilibration conditions for the A-sequence and B-sequence, or wish to hold the A-sequence potentials for a fixed time but advance the experiment whilst measuring B-sequence potentials if the B-sequence equilibration condition is met.

3.4.2 `ca_preview()`

Upon clicking the `Preview chronoamperometry experiment` button, this function is called to display the potential profile applied to the cell across the experiment within the plot area. This offers the user the ability to visualise the potential profile constructed by the input parameters to ensure that the experiment is set up as intended – this is particularly useful if alternating parameters are in use.

3.4.3 `ca_curr_eq_bool()`

If equilibration parameters are set for the segment, this function is called within each call of `periodic_update()` to determine whether the current has equilibrated according to those parameters. The most recent measured current is appended to a current history deque, and current history older than the equilibration timescale is removed from the deque. The current equilibration is evaluated by comparing the absolute difference between the first and last components of the deque with the equilibration tolerance.

If the current has not equilibrated, the experiment continues at this segment through the `periodic_update()` function . If the current has equilibrated, equilibration information is written to the summary file and the experiment advances to the next segment *via* `stop()`.

Note: The equilibration is only evaluated for the measured current after the set potential has reached the sequence potential. This is for the entire segment if the potential is stepped to the sequence potential, or after the potential ramp has completed if using a non-‘STEP’ numeric ramp rate for the segment.

3.4.4 Current limits

If the user wishes to limit the measured current flowing within the cell for safety, they may define lower and/or upper current limits which, if surpassed, interrupt the experiment.

3.5 Chronopotentiometry

Chronopotentiometry (CP) experiments, indicated by the “cp_” prefix within the code, are performed as a single experiment with a series of unique segments for each sequence current. Within each segment, the set current can be stepped or ramped to the sequence current and held for the defined hold time for that segment. Upon completion of the segment, `stop()` is called to increment the segment number, before passing to `start()` to begin the next segment. This is repeated until all segments are complete.

If the experiment requires OCP equilibration (i.e., if either potential limit is set to ‘OCP’, or the user wishes to equilibrate the cell to achieve a stable state before the experiment begins), the equilibration is performed once at the beginning of the experiment. Any parameters defined as ‘OCP’ use this equilibrated value throughout the experiment.

A single output data file is generated for this experiment, named according

to the base filename provided in the GUI and appended with a suffix containing “CP” followed by the current sequence. It contains data columns of segment number, total elapsed time (s), segment elapsed time (s), measured potential (V), and measured current (A).

3.5.1 Alternating parameters

Alternating parameters may be set when the user wishes to apply different equilibration tolerances and timescales to potentials within the current sequence by clicking the `Use Alternating Parameters` button. The original parameter boxes are renamed to reflect the A-sequence parameters, and a dropdown area displays the B-sequence parameters. This may be useful to users who, for instance, want different equilibration conditions for the A-sequence and B-sequence, or wish to hold the A-sequence currents for a fixed time but advance the experiment whilst measuring B-sequence currents if the B-sequence equilibration condition is met.

3.5.2 `cp_preview()`

Upon clicking the `Preview chronopotentiometry experiment` button, this function is called to display the current profile applied to the cell across the experiment within the plot area. This offers the user the ability to visualise the current profile constructed by the input parameters to ensure that the experiment is set up as intended – this is particularly useful if alternating parameters are in use.

3.5.3 `cp_pot_eq_bool()`

If equilibration parameters are set for the segment, this function is called within each call of `periodic_update()` to determine whether the potential has equilibrated according to those parameters. The most recent measured potential is appended to a potential history deque, and potential history older than the equilibration timescale is removed from the deque. The potential

equilibration is evaluated by comparing the absolute difference between the first and last components of the deque with the equilibration tolerance.

If the potential has not equilibrated, the experiment continues at this segment through the `periodic_update()` function. If the potential has equilibrated, equilibration information is written to the summary file and the experiment advances to the next segment *via* `stop()`.

Note: The equilibration is only evaluated for the measured potential after the set current has reached the sequence current. This is for the entire segment if the current is stepped to the sequence current, or after the current ramp has completed if using a non-‘STEP’ numeric ramp rate for the segment.

3.5.4 Potential limits

If the user wishes to limit the measured potential across the cell for safety, they may define lower and/or upper potential limits which, if surpassed, interrupt the experiment.

3.6 Self-discharge

Self-discharge (SD) experiments, indicated by the “sd_” prefix within the code, are performed as a single experiment with a series of unique segments for each charge potential. Each segment contains the initialisation stage, i.e., the ramp or step to the charge potential, the holding stage at the charge potential, followed by the self-discharge data collected for this charge potential after the cell is switched off. Upon completion of a segment, `stop()` is called to increment the segment number and initialise the following segment, bypassing `start()` and continuing the experiment *via* `update()`. This is repeated until all segments are complete.

If the experiment requires OCP equilibration (i.e., any segment uses ‘OCP’ as a charge potential or potential cutoff input, or the user wishes to equilibrate the cell to achieve a stable state before the experiment begins), the equilibration

is performed once at the beginning of the experiment. Any parameters defined as 'OCP' use this equilibrated value throughout the experiment.

A single output data file is generated for this experiment, named according to the base filename provided in the GUI and appended with a suffix containing "SD" followed by the charge potential sequence. It contains data columns of segment number, charging/self-discharging state, total elapsed time (s), segment elapsed time (s), measured potential (V), and measured current (A) data. The charging/self-discharging column contains strings indicating whether the experiment is in the charging or self-discharging stage.

3.6.1 Potential cutoff thresholds

If the user wishes to progress the experiment to the next charge potential once the measured potential has self-discharged to a specific potential, they may define a series of potential cutoffs with each corresponding to a charge potential.

3.6.2 `sd_pot_eq_bool()`

If equilibration parameters are set, this function is called within each call of `periodic_update()` to determine whether the self-discharge potential has equilibrated according to those parameters. The most recent measured potential is appended to a potential history deque, and potential history older than the equilibration timescale is removed from the deque. The potential equilibration is evaluated by comparing the absolute difference between the first and last components of the deque with the equilibration tolerance.

If the potential has not equilibrated, the self-discharge segment continues through the `periodic_update()` function. If the potential has equilibrated, equilibration information is written to the summary file and the experiment advances to the next charge potential *via* `stop()`.

Note: The equilibration is only evaluated for the measured potential during

self-discharge, i.e., once the cell is switched off after the holding stage at the charge potential has completed.

3.7 Rate-testing

Rate-testing experiments, indicated by the “rate_” prefix within the code, are performed as a series of unique experiments for each potential window. Each experiment consists of GCD cycling within the potential window at each of the C-rates defined in the parameter inputs.

If any experiments require OCP equilibration (i.e., any experiment uses ‘OCP’ as a lower or upper potential limit input, or the user wishes to equilibrate the cell to achieve a stable state before the experiment begins), the equilibration is performed before the experiment at each potential window begins.

Output data files are generated for each experiment, named according to the base filename provided in the GUI and appended with a suffix containing “RT” followed by the experiment number, potential window, and C-rates. Each file contains data columns of C-rate, cycle number, half-cycle number, elapsed time (s), C-rate elapsed time (s), measured potential (V), and measured current (A).

A further data file is generated for each experiment, named similarly but with “capacities” appended to the suffix, containing data columns of C-rate, cycle number, charge capacity (Ah), and discharge capacity (Ah).

3.7.1 Automated 1C calculation

The user may choose to automatically calculate the 1C value for each potential window instead of setting it within the input parameters. In this case, GCD cycling is performed before each experiment within the potential window using the 1C calculation current provided in the 1C calculation dropdown parameters box. The 1C value is then calculated as the charge integrated across

the final discharge half-cycle.

Note: The optional post-calculation OCP equilibration stage is for cell relaxation purposes only, and any parameters set to 'OCP' for a given experiment will use the equilibrated OCP value from the equilibration performed pre-calculation.