Appendix

MatStat Potentiostat Control Software Manual and Maintenance Guide

A.1 Preamble

This appendix describes the installation, usage, and internal workings of the MatStat potentiostat control software written in Matlab. The software is designed to control analog Princeton Applied Research (PAR) potentiostats using National Instruments (NI) data acquisition cards, although it could be expanded to work with any analog potentiostat and any data acquisition cards that can communicate with Matlab's data acquisition package. MatStat enables both analog and digital waveforms as well as high precision and high rate data acquisition on both current and voltage channels. MatStat is currently capable of making measurements in both potential control and current control modes, as well as open circuit voltage measurements, although the specific use is obviously coupled with the analog potentiostat being used. Furthermore, because the source code for the software is available and extensible, new modules and waveforms can be readily added. The maintenance section describes in some detail the inner workings of the software and should enable future users to both correct any remaining bugs and expand the capabilities of the software. Not only does MatStat enable the acquisition of high quality electrochemical data, it is also significantly more flexible than most of the currently available electrochemical instrumentation software.

A.2 Installation Instructions

A.2.1 Software Modes

Interpreted mode. Software is run from the Matlab command line. For the Caltech Matlab license, this requires network access, which may not always be possible for instruments on carts. MatStat typically runs a bit faster in interpreted mode, and errors or bugs are much easier to diagnose because the error message is printed to the Matlab command window.

Compiled mode. Software is run from the compiled binary, without starting it from the Matlab command line. The main benefit of this mode is that it can be used when Matlab cannot be started (e.g., when there is no network access) and even when there is not a complete installation of Matlab on the computer. However, it tends to run somewhat more slowly than interpreted mode. For the purposes of installation, setting up the software to run in compiled mode on a machine with Matlab will be referred to as *Full Compiled Mode*, and setting up the software on a machine without Matlab will be referred to as *Standalone Compiled Mode*.

A.2.2 System Requirements

A.2.2.1 General Requirements

- Analog potentiostat. MatStat has been tested with only Princeton Applied Research (PAR) analog potentiostats, but should in theory be compatible with any analog potentiostat having an external input and current and voltage monitor outputs.

- Data Acquisition (DAQ) Card. MatStat currently requires a National Instruments (NI) DAQ card, although it could be expanded to support other cards. In particular, the software has been configured and tested with NI PCI-6221 DAQ cards, which have a particular maximum data acquisition rate. This rate is currently hard-coded into the software, so the use of a different model card may require modification of the software for best performance.
- Window XP, SP3. MatStat has been exclusively tested with Windows XP. It may function on other Windows platforms, but no promise of compatibility with other versions of Windows is made.
- Microsoft Office Excel 2007. For saving directly to Excel format, Excel 2007 is recommended. MatStat has been tested extensively with Excel 2007, but may also work with Excel 2003. No promise of compatibility with later versions is made.

A.2.2.2 Stand Alone Compiled Mode

- Matlab Compiler Runtime (MCR), version 7.10
- Microsoft Visual C++ Redistribution (included in MCR)
- MatStat compiled module
- MatStat source code

A.2.2.3 Full Compiled Mode

- Full Matlab Installation, version R2009a
- MatStat compiled module
- MatStat source code

A.2.2.4 Interpreted Mode

- Full Matlab Installation, version 2009a
- MatStat source code

A.2.3 Software Installation

Before undertaking any of these installations, be sure that the NI DAQ card is properly installed on the system. Follow the manufacturer's instructions for this step.

A.2.3.1 Stand Alone Compiled Mode

- Extract the archive *compiled.zip* to anywhere convenient and download the Matlab Compiler Runtime, *MCRInstaller.exe*.
- Double click *MCRIntsaller.exe* to begin installation of the Matlab Compiler Runtime. Follow the instructions, accepting any defaults. The installer should automatically install VCREDIST_X86 if necessary. If prompted about installing the .NET framework, you can ignore the warning. The .NET framework is not needed for MatStat.
- After MCR installation, the following directory should contain the MATLAB runtime:

  C:\Program Files\MATLAB\MATLAB Compiler Runtime\v710

  Change into this directory and create a new folder named *work*. This is the folder from which MatStat expects to operate.
- Copy the following files and folders into *work* from the extracted *compiled.zip* directory:

  - MatStat.m

- MatStat.exe
- MatStat1.ico
- folder: MatStat

- Change into the *MatStat* directory and create a copy of the configuration file that is appropriate for the potentiostat configuration being used (for example, *config_PAR_173_175.mat*). Change the name of this copy to *config.mat.* NOTE: Windows XP does not show file extensions by default. If the file shows up as *config_PAR_173_175*, then the file extension is hidden, and the name should only be changed to *config*.
- To create a shortcut on the Desktop with a MatStat icon, right click on the desktop and select New➔Shortcut. Next, click "Browse..." and navigate to:

  C:\Program Files\MATLAB\MATLAB Compiler Runtime\v710\work

  Select *MatStat.exe*. Finish the wizard and the shortcut will be present on the desktop. To change the icon on the shortcut, right click on it and select Properties. On the "Shortcut" tab, click on "Change Icon..." and ignore the warning that MatStat.exe does not contain any icons. Select "Browse..." and navigate again to:

  C:\Program Files\MATLAB\MATLAB Compiler Runtime\v710\work

  Select *MatStat1.ico*, and click "OK." Click "Apply" and close the MatStat Properties window.
- MatStat can now be started by double clicking on either the shortcut or MatStat.exe.

A.2.3.2 Full Compiled Mode

- Extract the archive *compiled.zip* to anywhere convenient.
- Make sure that the *work* directory exists, or create it if it does not:

  C:\Program Files\MATLAB\R2009a\work

- Copy the following files and folders into the *work* directory:
  - MatStat.m
  - MatStat.exe
  - MatStat1.ico
  - folder: MatStat
- Change into the *MatStat* directory and create a copy of the configuration file that is appropriate for the potentiostat configuration being used (for example, *config_PAR_173_175.mat*). Change the name of this copy to *config.mat*. NOTE: Windows XP does not show file extensions by default. If the file shows up as *config_PAR_173_175*, then the file extension is hidden, and the name should only be changed to *config*.
- To create a shortcut on the Desktop with a MatStat icon, right click on the desktop and select New➔Shortcut. Next, click "Browse..." and navigate to:

  C:\Program Files\MATLAB\R2009a\work

  Select *MatStat.exe*. Finish the wizard and the shortcut will be present on the desktop. To change the icon on the shortcut, right click on it and select Properties. On the "Shortcut" tab, click on "Change Icon..." and ignore the warning that MatStat.exe does not contain any icons. Select "Browse..." and

navigate again to:

C:\Program Files\MATLAB\R2009a\work

Select *MatStat1.ico*, and click "OK." Click "Apply" and close the MatStat Properties window.

- MatStat can now be started by double clicking on either the shortcut or MatStat.exe.

A.2.3.3 Interpreted Mode

- Make sure that Matlab 2009a is installed on the system.
- Make sure that the *work* directory exists, or create it if it does not:

  C:\Program Files\MATLAB\R2009a\work
- Download the latest source code of MatStat and copy the following files and folders into the *work* directory:
  - MatStat.m
  - folder: MatStat
- Change into the *MatStat* directory and create a copy of the configuration file that is appropriate for the potentiostat configuration being used (for example, *config_PAR_173_175.mat*). Change the name of this copy to *config.mat*. NOTE: Windows XP does not show file extensions by default. If the file shows up as *config_PAR_173_175*, then the file extension is hidden, and the name should only be changed to *config*.
- Start Matlab

- Choose File➔Set Path from the pull-down menu. Check that the following folders are at the top of the list:

  C:\Program Files\MATLAB\R2009a\work\MatStat

  C:\Program Files\MATLAB\R2009a\work

  If they are not at the top of the list, move them up so that they are. If they are not present on the list, choose "Add Folder..." and navigate to the appropriate folders, adding each of the two shown above. Finally, chose "Save" and close the Set Path dialog box.
- Now MatStat can be started from the command line by typing "MatStat" and pressing enter.

A.2.4 Potentiostat Setup

In the following section, the correct BNC cable connections for each supported potentiostat configuration will be described. In all cases, the NI PCI-6221 DAQ is connected to an NI BNC-2110 connector breakout block. Connections designated as AO or AI refer to the corresponding analog output or analog input on the BNC breakout block. Furthermore, all of the analog inputs should be set to floating source, not grounded source.

A.2.4.1 PAR 173 Potentiostat

- AI 0 ➔ PAR 173 Electrometer Monitor
- AI 1 ➔ PAR 179 I Out or PAR 176 Output (depending on configuration)
- AO 0 ➔ PAR 173 Ext. Sig. Inputs (right input)

A.2.4.2 PAR 173 Potentiostat with PAR 175 Programmer

- AI 0 ➔ PAR 173 Electrometer Monitor
- AI 1 ➔ PAR 179 I Out or PAR 176 Output (depending on configuration)
- AO 0 ➔ PAR 173 Ext. Sig. Inputs (right input)
- AO 1 ➔ **BOTH** PAR 175 Frame Reset and PAR 175 Ext. Trig. In. (These should be shorted together at the PAR 175 to enable this connection.)
- PAR 175 Signal Output ➔ PAR 173 Ext. Sig. Inputs (left input)

A.2.4.3 PAR 362 Potentiostat

- AI 0 ➔ PAR 362 Potential Monitor (need a BNC to banana plug converter, make sure that the ground plug goes into the black terminal)
- AI 1 ➔ PAR 362 Current Monitor (need a BNC to banana plug converter, make sure that the ground plug goes into the black terminal)
- AO 0 ➔ PAR 362 Ext. In

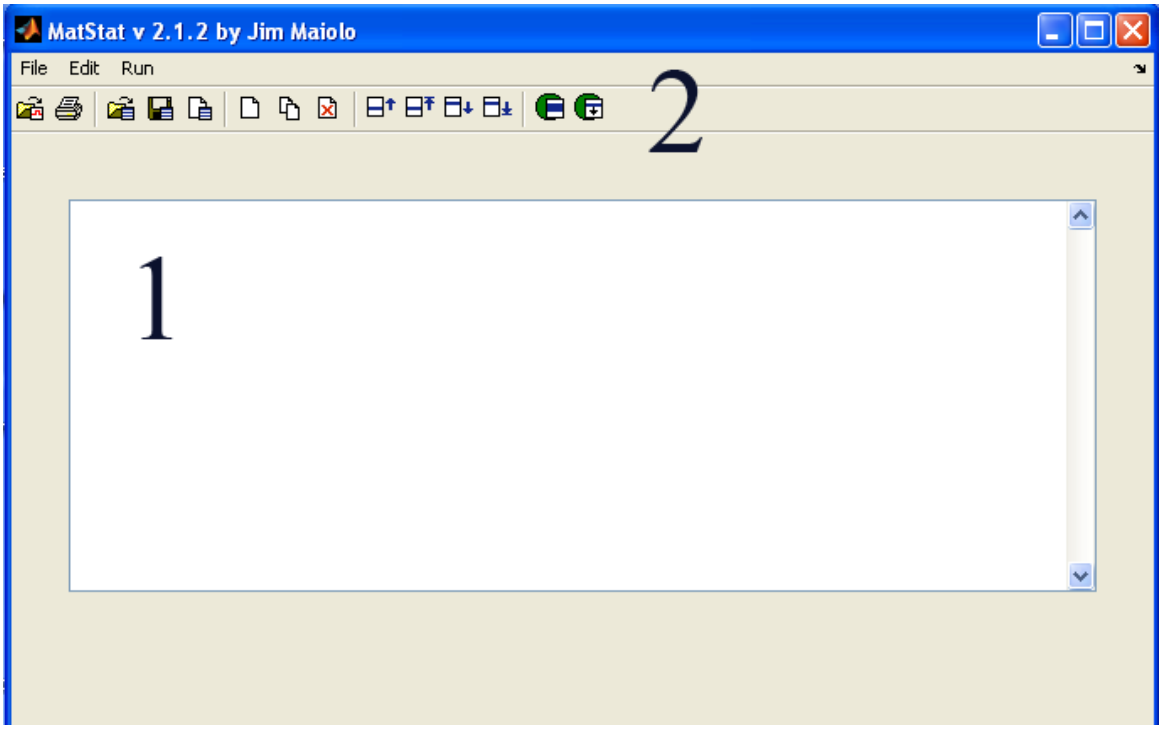
A.3 Using MatStat

A.3.1 Starting MatStat

For either Stand Alone Compiled or Full Compiled Mode, locate either MatStat.exe or a shortcut to MatStat.exe and double click on it. For Interpreted Mode, start Matlab, type "MatStat" at the command prompt (omit the quotes), and press enter.

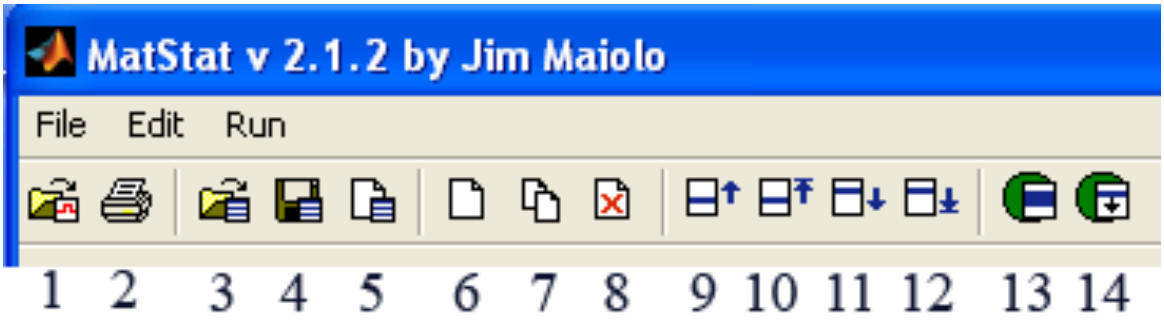
A.3.2 Setup Environment

A.3.2.1 Overview



1. Experiment List. Holds a list of all the programmed experiments along with pertinent data about each.
2. Toolbar and Drop-down Menu. Define the possible operations in the Setup Environment.

A.3.2.2 Toolbar Buttons



1. Open Graph. Opens a previously saved MatStat file, loading the relevant experimental parameters, and switches to the Run Environment.
2. Print - Prints the visible screen.
3. Open Setup. Opens a previously saved MatStat setup with all experiments.
4. Save Setup. Save this setup.
5. New Setup. Clear all the current experiments (you will be prompted to save).
6. New Experiment. Insert a new experiment below the one(s) currently selected.
7. Copy Experiment. Duplicates the currently selected experiment(s).
8. Delete Experiment. Deletes all of the currently selected experiments.
9. Move Up. Moves the selected experiment(s) up one in the order.
10. Move to Top. Moves the selected experiments(s) to the top of the list.
11. Move Down. Moves the selected experiment(s) down one in the order.
12. Move to Bottom. Moves the selected experiments(s) to the bottom of the list.
13. Run Selected. Runs all selected experiments in order.
14. Run From Selected to End. Runs all experiments starting from the first selected one to the end of the list.

A.3.2.3 Drop-down Menu Contents

- File
  - New Setup. Clear all the current experiments (you will be prompted to save).
  - Open Setup. Opens a previously saved MatStat setup with all experiments.
  - Save Setup. Save this setup under the current file (you will be prompted for a filename if this setup has not been saved before).
  - Save Setup As... Opens a dialog box to choose a save location and filename.
  - Open Graph. Opens a previously saved MatStat file, loading the relevant experimental parameters, and switches to the Run Environment.
  - Print. Prints the visible screen.
  - Quit. Exit MatStat.
- Edit
  - New Experiment. Insert a new experiment below the one(s) currently selected.
  - Duplicate Experiments. Duplicates the currently selected experiment(s).
  - Delete Experiments. Deletes all of the currently selected experiments.
  - Select All. Selects all experiments in the list.
  - Move Up. Moves the selected experiment(s) up one in the order.
  - Move to Top. Moves the selected experiments(s) to the top of the list.
  - Move Down. Moves the selected experiment(s) down one in the order.

  - Move to Bottom. Moves the selected experiments(s) to the bottom of the list.

- Run
  - Run Selected. Runs all selected experiments in order.
  - Run From Selected to End. Runs all experiments starting from the first selected one to the end of the list.

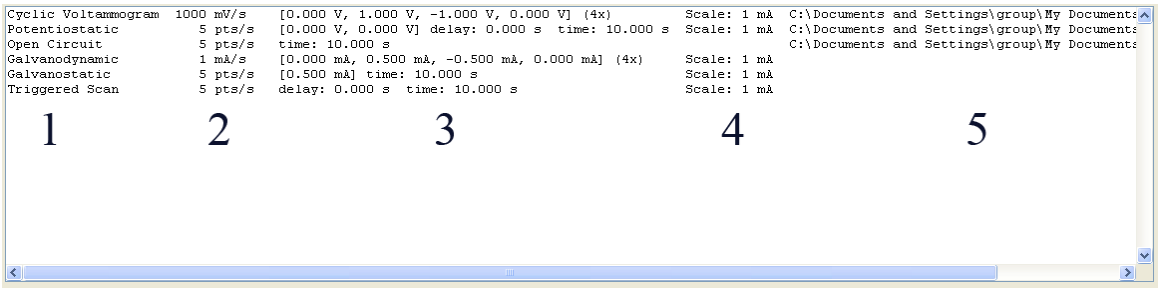
A.3.2.4 Shortcut Keys

- Ctrl+A. Select All
- Ctrl+B. Move to Bottom
- Ctrl+D. Move Down
- Ctrl+E. New Experiment
- Ctrl+F. Run From Selected to End
- Ctrl+G. Open Graph
- Ctrl+K. Duplicate Experiments
- Ctrl+N. New Setup
- Ctrl+O. Open Setup
- Ctrl+P. Print Screen
- Ctrl+Q. Quit
- Ctrl+R. Run Selected
- Ctrl+S. Save Setup
- Ctrl+T. Move to Top
- Ctrl+U. Move Up

- Ctrl+W. Delete Experiments

A.3.2.5 The Experiment List

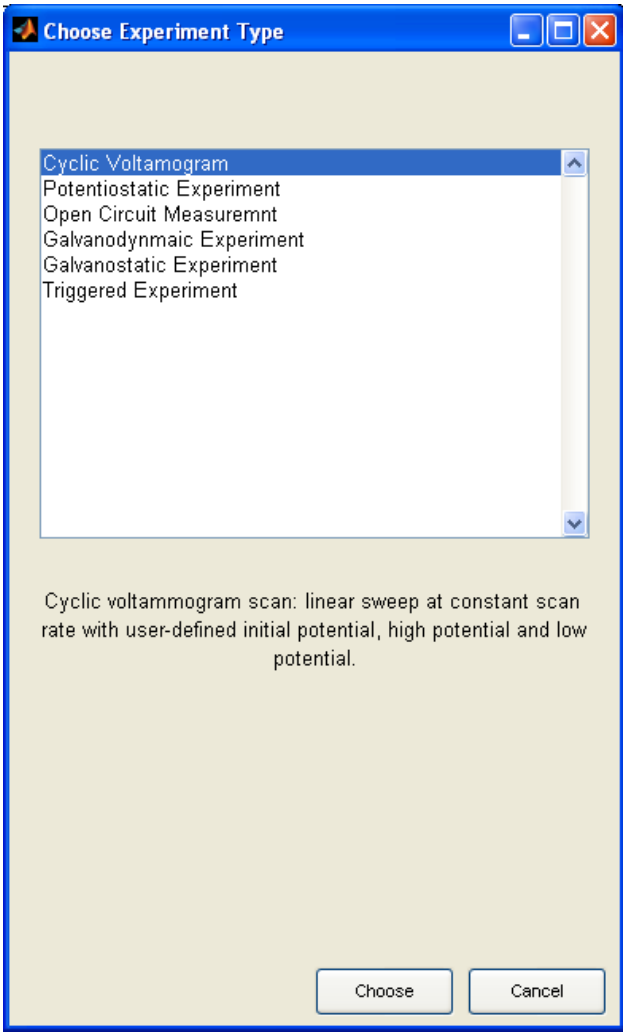
When populated with experiments, the experiment list should look something like the image below. When using the experiment list, double clicking on a given experiment will pull up the parameters window and allow the experiment to be modified. Furthermore, pressing delete will remove any selected experiments from the list.



1. Experiment type
2. Scan rate for sweep experiments or data collection rate for other.
3. Experiment-type specific details
4. Current scale (where applicable)
5. Save file path and name (if any)

A.3.2.6 Adding a New Experiment

After choosing to insert a new experiment in any of the ways described above, the user will first be presented with a window for choosing which type of experiment to insert. Note that the contents of this window will depend on the particular analog instruments being used (e.g., the PAR 362 cannot perform triggered experiments). For this reason, it is extremely important that the configuration file match the instruments being used (See Section A.2.3).



For each experiment, a short description is given. After selecting "Choose," the user will be directed to the configuration window specific to that experiment. See Section A.3.3 for details specific to configuring each type of experiment as well as information on configuring the potentiostats to correctly run each type of experiment.

A.3.2.7 Initiating a Run

Once the experiments have been configured, they can be run by either selecting all of the desired experiments and selecting Run Selected, or by selecting an experiment and choosing Run From Selected to End. When running multiple experiments, it is desirable

to select a save file for each one before running. However, if some experiments in the sequence do not have specified save files, MatStat will pause after each experiment with no specified save file and wait for user input. A warning is issued before the run begins in this case.

NOTE: Use extreme caution when setting up a list of multiple experiments to run in sequence. Since the instruments being used are analog in nature, all of the experiments must have compatible settings on the potentiostat's physical controls. This means, for example, that current controlled experiments cannot be mixed with voltage controlled experiments, and that the current scale must be identical for all of the experiments conducted.

A.3.3 Experiment Types

It should be noted here that one of the strengths of MatStat is its expandability, so it is hoped that more experiments will become available as people use the software and discover new functionality that would be beneficial. In that case, supplemental documentation for the new experiments should be provided. Before presenting specific details about each type of experiment, some general information about MatStat and the experimental parameters will be presented. Also, the names of the fields in the various parameter windows will be given in *italics* throughout this section.

A.3.3.1 Data Collection and Averaging

Because the DAQ card can collect data at an extremely high rate, it is always configured to collect data near its maximal rate. Since the user-requested data rate is usually significantly smaller than this rate, MatStat performs software averaging to

produce the desired number of data points. For example, if the DAQ card can collect data at 100,000 points per second and only 10 points per second are requested, then 10,000 points from the DAQ will be averaged for every point recorded for the user. In each of the configuration windows, the number of points being averaged is always shown, but is set indirectly by the user-entered parameters and is not directly accessible from the user interface. It should be noted that most commercial digitization software does largely the same kind of averaging before reporting data to the user because DAQ cards are capable of such high rates of data acquisition. In the event that a user attempts to enter configuration parameters that would necessitate the DAQ to collect data faster than its maximum rate, the number of points averaged will read as "0-1" indicating that the DAQ cannot keep up with the requested rate. When trying to run an experiment configured in this way, an error will be reported and the user will not be allowed to continue with the experiment until the error is corrected.

A.3.3.2 The Current Scale Setting

For all experiment types except the open circuit voltage measurement, current data will be collected. It is very important in this case to set the current scale of the potentiostat at a value such that the current will not exceed the set limit during the measurement. Furthermore, it is critical that the *Current Scale* setting in the experiment description match the setting on the potentiostat. This setting is used to convert the signal from the potentiostat to a real current, so the reported current values will be off by orders of magnitude if the software current scale is not set to agree with the potentiostat current scale.

A.3.3.3 The Save File Setting

When setting up an experiment, a save file can be specified. The data will be automatically saved to this file when the experiment is completed, overwriting any files with the same path and filename. When using this setting, be sure to change the filename when running multiple experiments from the same configuration.

A.3.3.4 A Note About Final Potentials

Since this is an analog instrument, it cannot be set to return to open circuit following a measurement. Therefore, in each experiment type below, the potential that the DAQ returns to is listed in the description. To return to open circuit, it is necessary to throw the switch on the potentiostat manually.

A.3.3.5 Cyclic Voltammogram

For this experiment, the waveform is generated in the software and sent to the potentiostat through the DAQ card. Therefore, the potentiostat should be in Control E mode and the external input from the DAQ card should be enabled. The scan will begin at *Initial Potential (V)* and proceed in the direction indicated by *Initial Scan Direction*. Upon reaching either the *Upper Scan Limit (V)* or the *Lower Scan Limit (V)*, the scan will reverse directions and proceed to the other scan limit. From that limit it will reverse directions once more and proceed back to the initial potential. Note that scans can also be set up in which the initial potential is equal to one of the limiting potentials. In this case the scan will proceed from the initial potential to the limiting potential and back to the initial potential. Note that the initial sweep direction must be set appropriately when the initial potential is equal to one of the scan endpoints. The programmed scan will be

repeated a number of times equal to *Number of Scans*. The *Scan Rate (mV/s)* sets the sweep speed of the waveform. The *Precision (V/point)* indicates the frequency with which to collect data along the potential axis. When the experiment is finished, the potential will remain at the initial potential.

A.3.3.6 Potentiostatic Experiment

As with the cyclic voltammogram, the potential is set from the DAQ card so the potentiostat should be in Control E mode and the external input from the DAQ card should be enabled. The experiment will hold at *Initial Potential (V)* for a time equal to *Initial Delay (s)*, and will subsequently step to *Run Potential (V)*, holding at that potential for a time equal to *Run Time (s)*. When this time has expired, the potential will return to the initial potential. The rate of data acquisition is set by *Rate (points/s)*.

A.3.3.7 Open Circuit Measurement

In this case, only measurement is performed. When using a PAR 173, the operating mode should be set to Direct Meas. Only and the potential will only be measured. For the PAR 362, which does not have this mode, the potentiostat should be set to Control I and the DAQ card will cause a current of 0 A to be applied, resulting in a simulated open circuit condition. Voltage data only will be collected for a time equal to *Run Time (s)* and data will be collected at a rate of *Rate (points/s)*. The applied potential will remain at zero from the DAQ card no matter what potentiostat is used.

A.3.3.8 Galvanodynamic Experiment

This experiment is a controlled current sweep experiment, very similar to the cyclic voltammogram. The external input from the DAQ card should be enabled, and the

potentiostat should be set to Control I mode. The scan will begin by stepping immediately to *Initial Current (mA)* and proceed in the direction indicated by *Initial Scan Direction*. Upon reaching either the *Upper Scan Limit (mA)* or the *Lower Scan Limit (mA)*, the scan will reverse directions and proceed to the other scan limit. From that limit it will reverse directions once more and proceed back to the initial current. Note that scans can also be set up in which the initial current is equal to one of the limiting currents. In this case the scan will proceed from the initial current to the limiting current and back to the initial current. Note that the initial sweep direction must be set appropriately when the initial current is equal to one of the scan endpoints. The programmed scan will be repeated a number of times equal to *Number of Scans*. The *Scan Rate (mA/s)* sets the sweep speed of the waveform. The *Precision (mA/point)* indicates the frequency with which to collect data along the current axis. Note that it is extremely important with this technique that the current scale in the software match the scale on the potentiostat as the current applied by the potentiostat depends on the current scale setting. Also note that galvanodynamic experiments always return the applied current to zero after the scan, independent of the initial current. This is for safety.

A.3.3.9 Galvanostatic Experiment

This experiment applies a constant current to the cell, so the external input from the DAQ card should be enabled, and the potentiostat should be set to Control I mode. Unlike the potentiostatic experiment, there is no initial delay period enabled for galvanostatic experiments. The scan will immediately jump to *Applied Current (mA)* and hold at that current for a time equal to *Run Time (s)*. The data will be collected at a rate

of *Rate (points/s)*. Note that it is extremely important with this technique that the current scale in the software match the scale on the potentiostat as the current applied by the potentiostat depends on the current scale setting. Also note that galvanostatic experiments always return the applied current to zero after the scan. This is for safety.

A.3.3.10 Triggered Experiment

Triggered experiments are only available with a combination PAR 173 potentiostat and PAR 175 programmer. This allows the user to program the waveform on the programmer, but then the programmer is activated by MatStat and the data is collected by MatStat. For this method, the external input on the PAR 173 coming from the PAR 175 should be enabled. The potentiostat can then be operated in either Control E or Control I mode, depending on the intentions of the user. Any desired waveform can be set on the programmer, and the programmer should then be set to Initial, with the Ext. Trig. button depressed. When running the triggered experiment, the program will wait for *Initial Delay (s)*, collecting data at whatever initial potential the programmer is set to apply. The software will subsequently trigger the programmer and allow it to run for *Run Time (s)*. When the time has elapsed, MatStat will trigger the programmer to return to its initial potential. During the scan, data is collected at of *Rate (points/s)*.

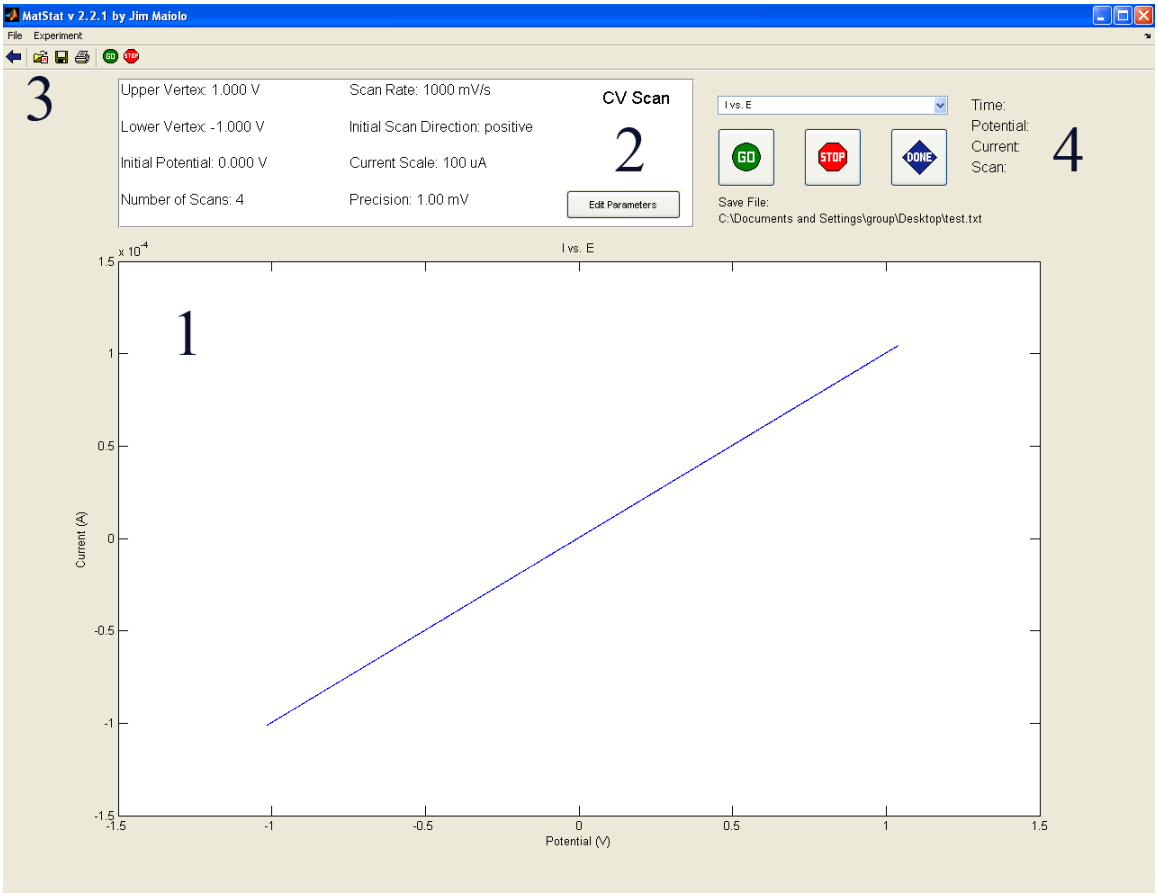
A.3.4 Run Environment

A.3.4.1 Overview

After the user chooses to start an experiment, MatStat will switch to the Run Environment. The experiment will immediately be started, and the data will be shown in in the graph window. When the experiment is finished, it will be saved if a save file was

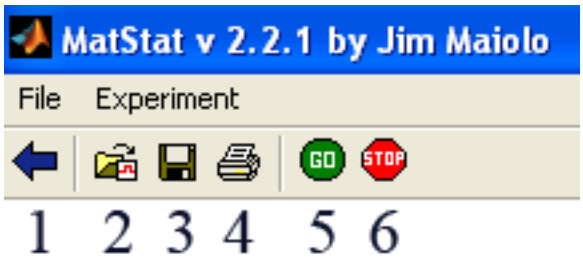
specified previously. Otherwise, the user will be prompted for a save file. The user may choose to cancel saving at this point. Once the experiment has been completed, the behavior will depend on how the experiment was started. If there are more experiments in the series to be conducted automatically, then they will be started immediately. If the experiment is the last in the series, or if only one experiment was started, then the Run Environment will remain active, and the user can continue to interact with it. This behavior is particularly important if the user has not specified a save file in advance or cancels saving up on completion of the experiment. If there are more experiments in the series to be run, then the data will be lost if not saved.

See the figure on the next page for the parts of the Run Environment.



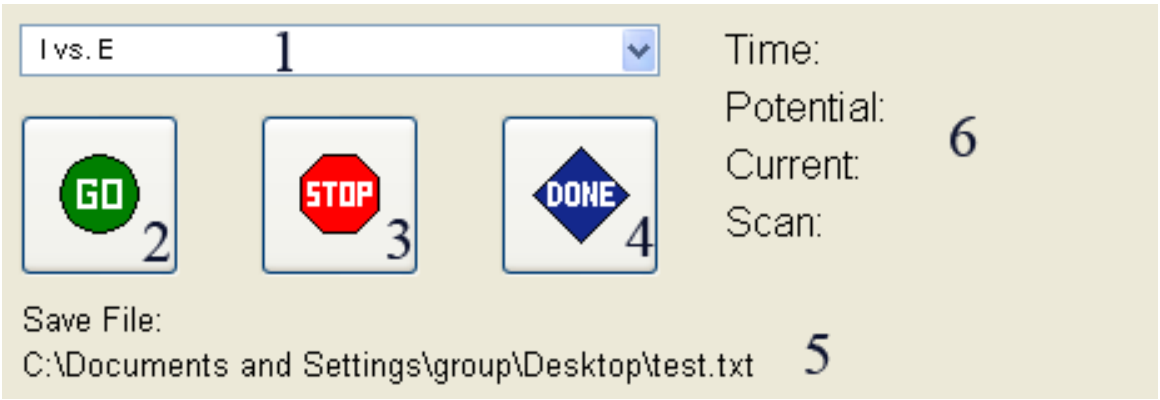
1. Graph area, showing data collected. Zooming is enabled on the plot after data collection has finished by dragging. Right click and select "Reset to Original View" to zoom all the way out. Zooming from the Experiment➔Zoom... menu is always available.
2. Parameter display area and button to edit parameters. The type of experiment is always shown here.
3. Toolbar.
4. Interaction and scan information area.

A.3.4.2 Toolbar Buttons



1. Back button. Return to the manager.
2. Open graph. Opens another data set.
3. Save. Saves the currently active data set.
4. Print. Print the current screen.
5. Go. Start running the experiment.
6. Stop. Stop running the current experiment.

A.3.4.3 Interaction and Scan Information Area



1. Plot type chooser. Display the same data in different ways.
2. Go button. Starts the experiment.
3. Stop button. Stops the experiment.
4. Done button. Return to the setup environment.
5. Save file. Experiments conducted will be saved to this file.

6. Information. Numbers will appear here when an experiment is in progress. Not all experiments will have the same set of information in this area.

A.3.4.4 Drop-down Menu Contents

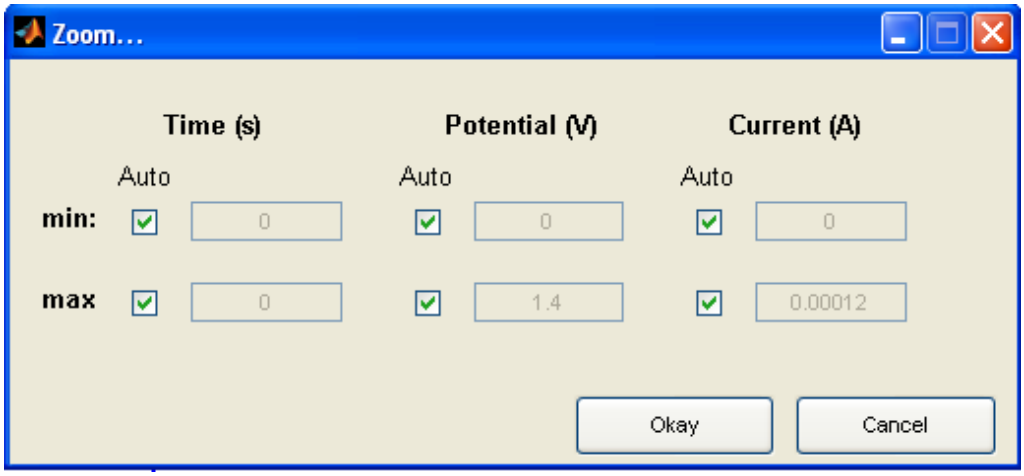
- File
  - Open Graph. Opens another data set.
  - Save Data. Saves the currently active data set.
  - Print Screen. Print the current screen.
  - Close. Return to the setup environment.
  - Quit. Leave MatStat entirely.
- Experiment
  - Zoom... Opens the zoom window.
  - Start Measurement. Begin running experiment.
  - Stop Measurement. Stop running experiment.
  - Edit Parameters. Change experiment parameters.

A.3.4.5 Shortcut Keys

- Ctrl+E. Edit Parameters
- Ctrl+G. Start Measurement
- Ctrl+H. Stop Measurement
- Ctrl+K. Return to Setup Environment
- Ctrl+O. Open Graph
- Ctrl+P. Print Screen
- Ctrl+Q. Quit MatStat

- Ctrl+S. Save Data
- Ctrl+Z. Zoom...

A.3.4.6 Zoom Window



The zoom window allows zooming on all three variables available in the data. In order to set the zoom manually, the *Auto* box must be unchecked for the corresponding value. When zoom parameters have been manually set, they will persist if the plot type is changed. However, if the plot is zoomed using the mouse, the manually set parameters will be overwritten.

A.3.4.7 Run Environment Interaction

While there is an experiment actively running, the interaction options are somewhat limited. The user can specify manual zoom parameters with the zoom window, but cannot zoom using the mouse. The plot type can be changed (e.g., I vs. E or E vs. T), but the data cannot be saved. Choosing Stop will stop the experiment from running. If the user chooses to close the Run Environment or to quit MatStat, a prompt is displayed confirming that the experiment should be stopped in order to close or quit. Similarly,

upon attempting to open a different data set, the user will be asked if they wish to terminate the current run.

When no experiment is currently running (as when the Run Environment is reached by loading another data set or after an experiment has been completed), there are several options available to the user. The user can examine the data by changing the plot type and by changing the zoom either using the mouse or through the zoom window. The user can also edit the experimental parameters. If the Run Environment was initiated from an experiment in the Setup Environment, then any changes made to the experimental parameters in the Run Environment will be reflected in the experiment in the Setup environment. When there is not experiment running, the user can also start a new experiment at any time, either with the original parameters or after modifying the parameters. The user can also choose to save the current data set at any time when there is no experiment running, or as many times as desired, although no modification of the data is possible from within MatStat. Finally, the user can choose to open a previously collected MatStat data set.

A.4 Maintenance and Development Guide

This section describes the inner workings of MatStat and gives some general tips for debugging and further development. In all cases, when making a new change to the software, a new version number should be selected and the files from the previous version preserved so that the changes can be rolled back in the event of unexpected consequences. As of the writing of this manual, the latest version is 2.2.1. Any significant changes after this version should be accompanied by documentation where possible.

A.4.1 Overview

MatStat was designed to be as modular as possible so that new types of experiments could be easily plugged into the existing framework. In an object-oriented language, this could be easily accomplished using inheritance. However, the initial version of Matlab used to develop MatStat did not support inheritance or object-oriented programming (although more recent versions now do). The result, unfortunately, has been significant repetition of code, which is somewhat problematic from a maintenance point of view. Specifically, each type of experiment needs three files: **__GuiSetup**, **__Params**, and **__Open**, where __ should be filled in with an abbreviation for the experiment type. In these files, many of the inner functions are expected to have some differences, but they are largely similar. The end result is that changes to the operation of the program typically need to be propagated through all six of the currently available experiment types. I have usually found it beneficial to fully develop the behavior in one module before translating to the correct behavior for other modules.

In addition to the three files need for each module, I have also maintained generic modules to be used as templates for developing new types of experiments. These are **XXGuiSetup**, **XXParams**, and **XXOpen**, and they should also be updated whenever a change is made so that new experiments can always be added easily. For each experiment, the **__GuiSetup** module is the main workhorse. This module loads all of the graphical user interface (GUI) elements for the Run Environment and controls the acquisition, saving, and plotting of data. All of the main user interactions with the Run Environment are encapsulated in these files as callback functions. The **__Params** file manages the GUI for changing parameter values, while the **__Open** file contains the logic necessary to load data of the specified type. It is necessary to have these functions in a separate file (unlike the save function, which is included in **__GuiSetup** as an inner function) because they must be available to functions outside of the **__GuiSetup** file.

In addition to the three main files for each experiment, there are a number of other modules needed. The most significant is the **MainGuiSetup** file which includes all of the code for the GUI and behavior of the Setup Environment. I will typically refer to this entity as the manager in this document as well as in the program comments. Apart from this file, there is the main **MatStat** file which serves only to set up the figure window and call the manager. There are also a number of smaller modules that will be discussed below.

Matlab functions are typically found in their own file with the same name. I will give the names of these functions in bold, e.g., **MatStat** or **newExp**. In addition, Matlab supports inner functions defined within the body of other functions. I will give the names

of these functions in italics, e.g., *openFile* or *hScanButtonCallback*. Variables will typically be underlined as in Config.

Matlab does not support explicit typing, so I have typically included some type information in my variable names, particularly in variables that are part of structures. Variables with a leading "h" are typically handles to either functions or GUI elements (or regular functions whose handles will be passed as callback functions). Variables with a leading "i" are integers, those with a leading "d" are doubles, and those with a leading "s" are strings. Typically arrays and structures have no leading character indicating type. I will also refer repeatedly to "global variables," by which I mean variables defined at the top level in a main function as opposed to variables defined in an inner function.

A.4.2 Typical Execution Path

I always find that the first step toward understanding a program is to see its typical execution path. I will try to give a brief description of the execution of MatStat in this section. When it is started, **MatStat** first creates a new figure with a toolbar and maximizes it (using the third party function **maximize**). It subsequently attempts to load a configuration file, which currently only contains information about which potentiostat setup is being used. See Section A.4.5 for more information on how to create a valid configuration file. **MatStat** then passes a handle to the figure, a handle to the toolbar, and and a string indicating the potentiostat type to **MainGuiSetup**. The manager, after setting itself up, will then return two handles: one to its main panel so that **MatStat** can make it visible, and one to its kill function so that **MatStat** can ask it to quit when a user tries to close the figure. When making any changes to the way that MatStat quits, errors

may cause the program never to exit. In this case, uncomment the first line in **MatStat**:*hCloseRequestCallback* and the program can exit.

Once **MainGuiSetup** has completed its setup process and **MatStat** has registered the required handles, control of the program has been successfully passed of to **MainGuiSetup**. With control of the execution, **MainGuiSetup** allows users to add, change, and delete experiments as well as to save and load experimental setups. These functions will be described in more detail in Section A.4.4.1. The next major change in execution occurs when an experiment is run. At this point, the manager needs to hand off control to one of the **__GuiSetup** modules. In order to do this, it passes pertinent data to **runDispatch** which then calls the correct **__GuiSetup** module. That module then performs its various setup operations and passes back a handle to its main display panel as well as handles to its kill and run functions. With these handles, the manager can make the Run Environment visible and actually start the run automatically. When the user has finished with the Run Environment, the **__GuiSetup** will signal **MainGuiSetup**, which will then invoke the kill function handle to regain control.

Finally, MatStat can be closed under a variety of circumstances. When the user clicks the close button on the figure (the "X" in the corner), **MatStat** receives the signal and uses its handle to the **MainGuiSetup**:*hKill* function. In **MainGuiSetup**:*hKill*, the manager can check to see if the run window is currently active, and if so it can use the handle it received to **__GuiSetup**:*hKillThis* during setup of the Run Environment to tell the Run environment to close itself. Once the Run Environment has been closed, or if *hKill* is invoked when **MainGuiSetup** has control, the function will simply clean up and

can then exit by deleting the main figure (the handle of which was passed in from **MatStat** during the setup process).

Control can also be passed from **MainGuiSetup** to one of the **__GuiSetup** modules by opening previously saved data. In this case, the same procedure as above is invoked, but the experiment is not caused to run once it has been opened. Furthermore, it is possible to open data from the Run Environment. In this case, **__GuiSetup** uses the handle it received during setup to **MainGuiSetup**:*openFile*, allowing it to destroy itself if another file has been successfully opened. **MainGuiSetup** can then make the new data visible and thus remain in proper communication with the Run Environment at all times.

A.4.3 Important Communication Structures

Descriptions of the most important data structures used for communication among the various modules follow.

A.4.3.1 Config

The Config structure is used to pass both configurational information and key function handles to the **__GuiSetup** modules during setup. The following fields are present:

- hMainFigure. A handle to the main figure. Used to set the parent of the panel created in the Run Environment.
- hToolbar. A handle to the toolbar. Used to add Run Environment toolbar buttons.
- hSwitchBack. A handle to **MainGuiSetup**:*switchBack*.
- hFileOpen. A handle to **MainGuiSetup**:*openFile*.

- sPstat. The potentiostat configuration string.
- hParamUpdate. A handle to **MainGuiSetup**:*updateExp*.
- paramPos. An integer specifying the position of the current experiment in the list from **MainGuiSetup**. A value of -1 is used to indicate an experiment not on the list (i.e., one that was opened from a previous data set).
- hSetSaveDir. A handle to **MainGuiSetup**:*setSaveDir*.
- hGetSaveDir. A handle to **MainGuiSetup**:*getSaveDir*.
- hSetSaveType. A handle to **MainGuiSetup**:*setSaveType*.
- hGetSaveType. A handle to **MainGuiSetup**:*getSaveType*.

A.4.3.2 saveInfo

This structure just contains the information needed to keep track of the last save path and type. This is passed to smaller modules such as the **__Params** modules that need to save and load files, but not run experiments. The fields are:

- hSetSaveDir. A handle to **MainGuiSetup**:*setSaveDir*.
- hGetSaveDir. A handle to **MainGuiSetup**:*getSaveDir*.
- hSetSaveType. A handle to **MainGuiSetup**:*setSaveType*.
- hGetSaveType. A handle to **MainGuiSetup**:*getSaveType*.

A.4.3.3 Scan (or params)

The fields in this structure vary by the type of experiment being used. The fields for each experiment type are given in the file structs.txt in the MatStat source folder. I will only show the fields from the structure used with **IVGuiSetup** here, but the fields tend to be similar with the other experiment types. (However, it should be noted that, for

the Constant I experiments, the parameters are actually stored internally as the potentials that will need to be applied from the DAQ card, not as the currents specified by the user.)

- Type - a string that should be 'IV Scan' for this structure. In general, the strings for the currently available set of experiments are:
  - IV Scan. Cyclic Voltammogram.
  - Pot Scan. Potentiostatic Scan.
  - Trig Scan. Triggered Scan.
  - Voc Scan. Open Circuit Measurement.
  - GalDyn Scan. Galvanodynamic Experiment.
  - GalStat Scan. Galvanostatic Experiment.
- dUpperLimit. The upper scan limit in V.
- dLowerLimit. The lower scan limit in V.
- dInitial. The initial potential in V.
- iScans. The number of scans.
- ScanRate. The scan rate in mV/s.
- Step. The precision of the scan in V/point.
- sDirection. A direction string that should be either "+" or "-."
- dCurrentScale. The current scale setting stored as an integer. To accomplish this, the scale is stored as (1 A)/scale. Thus, at a current scale of 1 A, dCurrentScale = 1, and at a current scale of 1 μA, dCurrentScale = 10^6.
- SaveFile. The full path to use when saving this experiment automatically. The empty string is used when no value has been selected.

- SaveType - The extension of the file type to use (.txt, .mat, or .xls) when saving the experiment automatically. The empty string is used when no value has been selected.

A.4.4 Major Module Descriptions

A.4.4.1 MainGuiSetup

The setup process of **MainGuiSetup** starts with the definition of a number of useful global variables, as well as two structures that are used to communicate with other modules: Config and saveInfo. All of the GUI components that reside directly in the main panel are defined at the top level since they can be made invisible with the main panel when switching to the Run Environment. All of the toolbar and menu items of the manager are declared but not defined at the top level so that they can be readily removed and reinstated as the program switches between the Setup Environment and the Run Environment. The function *menuSetup* is then responsible for configuring the menus and the toolbars. Finally, **MainGuiSetup** attempts to open the default set of parameters, which simply stores the last used set of parameters so that the program does not start empty.

In addition to the communication structures described above, there are several important global variables in **MainGuiSetup**. The handle hKillOther stores the kill function for the currently active Run Environment. The experiments and last_saved variables are both horizontal cell arrays, with each cell consisting of a Scan structure. This is how the currently configured experiments are stored, with experiments being the current list and last_saved being the most recently saved list, which is used to determine

whether the list needs to be saved. In addition saveDir and saveType hold the most recently used directory for saving or opening as well as the last file type so that these can be used when saving or loading new files.

Once the setup process is complete, **MainGuiSetup** is in interactive mode. The process of switching between **MainGuiSetup** and the various **__GuiSetup** modules is described in Section A.4.2. There is a significant amount of additional communication between these modules, however. In particular, the following functions are passed as handles to all of the **__GuiSetup** modules: *switchBack*, *openFile*, *updateExp*, *setSaveDir*, *getSaveDir*, *setSaveType*, and *getSaveType*. The function *switchBack* allows the Run Environment to signal that it has finished. In **MainGuiSetup** this function first tells the Run Environement to quit, checking for success by the return value, and then it makes itself visible again. The function *openFile* allows either **MainGuiSetup** or one of the **__GuiSetup** functions to open another data file, with the **__GuiSetup** closing itself upon successfully opening another data set. The function *updateExp* allows the Run Environment to communicate changes to experiments that were initiated in the Setup Environment, so that the changes are registered there as well. Finally, the function *setSaveDir*, *getSaveDir*, *setSaveType*, and *getSaveType* are present so that every module that calls up a browse window to save or load can both check the last path and file type used (in order to open the browse window in a similar state to the last time it was closed) and report back the new path and file type chosen.

Most of the other behavior of **MainGuiSetup** is fairly straightforward. In particular, the offer to save the setup is always made whenever the program will be closed

or the current setup will be wiped. Whether or not to offer saving is determined by comparing the current setup with the last saved setup. As with the other modules that will be described, memory management of global variables and GUI elements is accomplished by clearing those variables during the process of exiting the **MainGuiSetup** module.

A.4.4.2 The __GuiSetup Family of Modules

These modules actually run all the experiments. Unfortunately, they share a significant amount of code between them, but they have small differences to account for the different experiments being performed. After being called from **MainGuiSetup**, **__GuiSetup** first initializes several global variables and then configures most of the GUI elements. The parameter display is created by the function *updateDisplay* so that it can be remade when the parameters are changed. Next, **__GuiSetup** calls *aoSetup* which attempts to configure the analog output. If it fails due to incorrect parameters, **__GuiSetup** will set all of its return values to -1 to flag that an error has occurred and **MainGuiSetup** will behave accordingly. Note that *aoSetup* is called every time the parameters are changed so that the software is always using the most up-to-date information when initiating a scan. Finally, **__GuiSetup** attempts to load any data that was passed in (if it was called to open previous data). A value of 0 for outData indicates that no data was given (since otherwise outData should be an array, not a scalar).

Descriptions of the global variables are provided in the code comments, so I will only describe a few here. Of particular importance are DAQdata, data, and Time. The data variable holds column data for the voltage and current at each point (two columns,

except for **VocGuiSetup** which only measures potential and therefore has data of only one column), while Time holds column data for the time at each point. Both of these variables are preallocated for the entire length of the run to avoid memory errors in the middle of a run. The values for both of these variables are determined by averaging the data pulled directly from the DAQ card, which is temporarily stored in DAQdata. In addition, the variable Running is used as a flag that a scan is running and certain user interface features should be disabled.

Most of the remaining behavior of **__GuiSetup** is self-explanatory apart from initiating, running, and stopping a scan. When the go button is pushed, any old data is first cleared from the analog output, and then the analog input and output devices are loaded and started. The function *hScanButtonCallback* then enters into its main loop, which gets new data from the DAQ card, averages it, then adds it to the current data set and replots the data. This loop has a built in 0.25 s pause in order to keep from overtaxing the system with updates. The data averaging is accomplished in the following way. During *aoSetup* the total number of points for the entire run is calculated and stored in TotalPoints, and the total time for the run is calculated and stored in InputTime. From this data, the time end points for each interval can be calculated and they are stored in the global variable tEndPoints. During averaging, data from DAQdata, which includes time information, is averaged according to these precalculated time intervals and stored in data.

A scan can be terminated either by user action or by reaching the end of the desired run. In order to obtain the same behavior in both cases, the function *hStopAiPushed* is

registered as the stop function on the analog input variable, and is therefore called when analog input has ceased either by user action or by reaching the end of the run. The function *hStopAiPushed* clears out the DAQ and averages the rest of the data before plotting the final data and calling the *mySave* function. The Running variable is not reset until all of these things have been accomplished so that a new scan cannot be initiated. Finally, the analog input is cleared by calling *reset_ai* to remove any remaining data and free up memory.

A.4.4.3 The __Params Family of Modules

This family of functions acts as the GUI for changing parameter values. They take as arguments a previous set of parameters (to populate the window with), a saveInfo structure for changing the global save and load behavior, and information about how to return the new parameters. This information is given as pos,which specifies the position in the experiment list (or -1 for an experiment not from a list), and returnParams, which is a function handle to send back the changed parameters. It is necessary to return the parameters in this way because the actual **__Params** function returns immediately after configuring the GUI, and the changed parameters need to be returned later, when the user is finished with the dialog box. Apart from this behavior, the workings of the **__Params** family of functions is fairly straightforward.

A.4.4.4 The __Open Family of Modules

This family of functions performs the correct procedures for opening data from a given experiment type. The **__Open** functions take in all the parameters necessary to call **__GuiSetup**, and they return all the parameters required by **MainGuiSetup** when

loading the Run Environment, as well as a parameter opened which is true if the file was read successfully.

A.4.4.5 IVCurve

This function generates the waveform for an IV curve given the pertinent input values. This is used by both **IVGuiSetup** and **GalDynGuiSetup** when configuring their analog output. The data is generated for a given step size that is tied directly to the analog output rate from the caller.

A.4.4.6 myDataPlot

This is the function used to do all of the data plotting. Its complexity has been increased dramatically by the necessity of dealing with five different types of plots as well as numerous possible autoscaling configurations. The function needs the data to plot as well as a handle to the axes to plot it on and the current setting of the window variable. In addition, it needs to know what type of plot is requested and also whether the experiment is running. This is important because mouse zooming is disabled for running experiments due to the constantly changing nature of the data. There is also a hack built into **myDataPlot** to account for the strange zooming behavior of Matlab.

A.4.4.7 updateParams

This function is used to dispatch a parameter update request to the appropriate **__Params** function. It also contains all of the default values used to initially populate the parameter setting dialog boxes for a new experiment. The arguments passed to this function are the same as those passed to the **__Params** functions since it must call those functions.

A.4.4.8 setWindow

This function provides a GUI for setting the window variable. The window variable consists of two rows of data. In the first row are the current values for min time and max time (both in s), min potential and max potential (both in V), and min current and max current (both in A). In the second row are booleans corresponding to whether each corresponding value is currently set to manual scaling or not (i.e., true for manual, false for auto). Thus, when the checkbox is checked on the GUI, the corresponding value is false in the window variable. As with other GUI functions **setWindow** also takes a function handle that allows it to return the updated window variable to the caller since **setWindow** returns immediately after setting up the GUI.

A.4.4.9 newExp

This function is an intermediary GUI that allows the user to choose the type of new experiment. It takes as parameters all of the values that are needed by **__Params** and therefore **updateParams**, since it must call **updateParams** after a selection has been made.

A.4.4.10 exp2str

This function takes the experiments cell array from **MainGuiSetup** and produces a correctly formatted string. It needs the entire array because it changes the spacing so that the columns line up for all of the experiments to be displayed. It is called every time there is a change to the experiment list in **MainGuiSetup**.

A.4.4.11 runDispatch

This is a simple function that starts the correct **__GuiSetup** based on the type of the Scan structure passed in.

A.4.4.12 typeToOpen

This is a simple function to translate between a given type string and an integer for the saving a loading of text files. This is necessary because for some reason Matlab will not allow text to be read from text files, only numbers. Thus, a number is written to the file indicating what type of file it is, so that when the file is opened later, the number can be read and translated back to a type.

A.4.4.13 putFileString

This function takes the last used file type and generates the list of choices for saving so that the last used file type is always the first choice.

A.4.5 Regenerating Configuration Files and Potentiostat Support

The current configuration file only contains a single field that specifies the type of potentiostat setup being used. If desired, future development could significantly expand the information in this file, perhaps to include information about the type of DAQ card being used and even local user preferences. Use the following sequence of commands at the Matlab command prompt to generate a configuration file containing the string PAR173/175 named config_PAR_173_175.mat.

```
>> config.sPstat = 'PAR173/175'
>> save('config_PAR_173_175','config')
```

At the time of this writing, the recognized potentiostat strings are PAR173, PAR173/175, and PAR362, with obvious meanings. When trying to support a new type of potentiostat, it is important to be aware of the following places in the code that make use of the potentiostat configuration. The **newExp** module requires knowledge of the potentiostat configuration so that triggered experiments can be made available only when a programmer is present. In **MainGuiSetup**, knowledge of the potentiostat configuration is required when loading a set of experiments in order to make sure that no unsupported experiments are in the list. Furthermore, in **GalStatGuiSetup** and **GalDynGuiSetup**, the potentiostat type is important because the potential polarity must be inverted on the PAR 173, but not on the PAR 362. Finally, in **TrigGuiSetup**, the configuration is checked on initiating a scan to make sure that the system can support it. When adding a new type of potentiostat, these are good starting points to make sure that MatStat behaves properly with the new hardware. There may be other issues, particularly polarity issues, however, that will need to be addressed as well.

A.4.6 Adding New Modules

When adding new modules to MatStat, the first step should be to modify the generic XX files: **XXGuiSetup**, **XXParams**, and **XXOpen** to suit the specific desired behavior. when making the changes to these functions, it is probably best to consult some of the similar modules to see how things are done in other, working, experiments. In addition to making these modified files, small modifications will be necessary to nearly all of the other small functions, most of which have some form of switch statement on the type of experiment being performed. Although these processes may be somewhat tedious, it is

expected that adding a completely new module should not take more than a few hours for someone who is already somewhat familiar with the inner workings of MatStat. This is one of the key advantages of the software—that it can be readily expanded to suit changing needs and imaginative ideas.

A.5 Final Notes

MatStat is certainly not the most elegant piece of software ever written, but it does work, and it does provide high quality data. More likely than not, with the object-oriented features of Matlab R2008a (most of the original development was done on R2007a), I could significantly improve the structure of the program, but that kind of sweeping overhaul would likely introduce more problems than it would fix. In all likelihood, few changes will be made to the software after I stop maintaining it, and so I hope that the current version is robust and stable enough to provide many years of happy data collection for many researchers. Enjoy!