

IRIS for Users & Developers

Dre Mahaarachchi

July 30, 2018

Contents

0.1	Introduction	2
0.2	Background	2
1	A User's Guide to IRIS	3
1.1	Starting the Application	3
1.2	Configuring Inputs	3
1.2.1	Requirements for Input Files	3
1.2.2	Selecting Inputs	4
1.2.3	Data Configurations	5
1.2.4	Generate Analysis	5
1.3	Navigating the Plots	6
1.3.1	Using the Widgets	6
1.3.2	Plot Types	7
1.4	Downloading Outputs	10
2	A Dev's Guide to IRIS	11
2.1	Introduction	11
2.2	iris.py	11
2.2.1	Widget Creation and Event Handling	11
2.2.2	Plotting	13
2.3	trials.py	14
2.3.1	Trial Object	14
2.3.2	Additional Functions	14
2.4	utils.py	15
2.4.1	Utilities	15
2.5	Creating an Executable	16
2.5.1	Installing Pyinstaller	16
2.5.2	Installing Python 3.5	16
2.5.3	Running Pyinstaller	17

0.1 Introduction

IRIS, Infrared Informatics Software, is an application intended to provide quick, easy viewing of raw current-voltage data of tested infrared photodetectors. IRIS combines data processing and graphical display into a small windowed Python application. IRIS is modulated into three sections: input, display, and output. This guide will cover how to navigate the application most effectively. Additionally, the guide will go over how to expand IRIS's source code for future features.

0.2 Background

IRIS for JPL's Infrared Photonics Group (389G). 389G focuses their research and efforts on developing and improving focal plane array devices, namely infrared photodetectors. These detectors are typically microns in size. In a nutshell, the quality of an infrared photodetector is highly dependent on the amount of **dark current** (more specifically, the lack thereof) released by the device, a current quantity that exists as noise for the detected signal. Dark current is measured in the absence of light, as it is noise that is present regardless of how much light is detected. The amount of dark current released by an infrared photodetector is mostly a function of detector size, operating voltage, and operating temperature. Still, imposing the same parameters on two identical detectors may produce slightly different dark current outputs. Thus, various convolutions are performed on the harvested IV data to gain insight as to why these deviations occur. IRIS packages these data processing algorithms into the back end of a small and simple user interface, complete with a graphing widget capable of zoom and scrolling.

Chapter 1

A User's Guide to IRIS

1.1 Starting the Application

IRIS is packaged in three forms: for Windows, for Mac, for development. The former two versions are packaged as executable files that were built with a special installer for Python. **Double clicking the executable will open the application for use.** The development version is the main Python file for the application, *iris.py*. This version must be run from the command line in its directory:

```
$ python iris.py
```

Keep in mind that to run the development version on the command line, your system must have Python 3 installed as well as all of the packages the source code uses (which will be listed in A Dev's Guide to IRIS).

1.2 Configuring Inputs

1.2.1 Requirements for Input Files

There are two input file types that IRIS takes in, IV data files and the corresponding pin size file for the measurement run. IRIS was designed to parse the formatting of the .txt and .xslm files produced by the LabView interface with the spectrometer. Below are the guidelines for the two files types:

IV Data Files

- Must follow this naming format: (date)_[device types]_Ta(low temp)_Tb(high temp).I.txt
- Tab separated between items
- First column is titled 'Vs'
- Have matching 'Vs' columns between each data file

Pin Size File

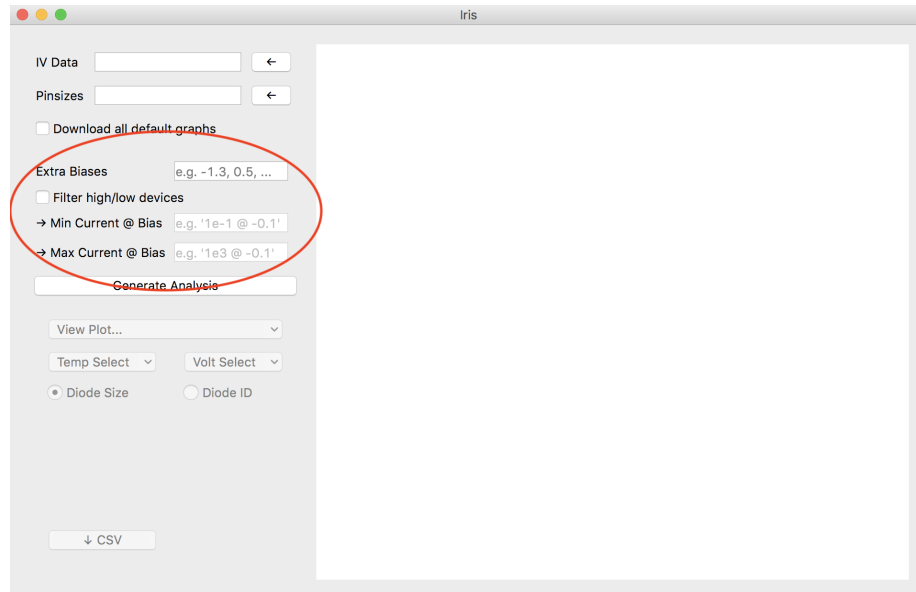
- If in .xslm, column AI must list pin numbers under 'PIN #' and column AJ must list pin sizes under 'SIZE'.
- If in .txt, first column must list the pin number and the second column must list the associated pin size.
- Set pin size to 0 for devices that are not tested/to be ignored.

1.2.2 Selecting Inputs

The screenshot shows the 'Iris' software window. On the left is a control panel with the following elements: 'IV Data' and 'Pinsizes' text boxes, each with a left-pointing arrow button; a checkbox for 'Download all default graphs'; an 'Extra Biases' text box with the example '-1.3, 0.5, ...'; a checkbox for 'Filter high/low devices'; two bias current settings: '→ Min Current @ Bias' with example 'e.g. '1e-1 @ -0.1'' and '→ Max Current @ Bias' with example 'e.g. '1e3 @ -0.1''; a 'Generate Analysis' button; a 'View Plot...' dropdown; 'Temp Select' and 'Volt Select' dropdowns; radio buttons for 'Diode Size' (selected) and 'Diode ID'; and a '↓ CSV' button at the bottom. The main area on the right is a large empty plot window.

To begin inputting your IV data, click the left arrow button next to the IV data textbox. This will prompt a file selection dialog for the selection of one or multiple IV data files. The file dialog only allows the selection of .txt files. Selecting the pin size file is the exact same process, except you can only select one file. Additionally, you can select the types of file (either .xslm or .txt) you want to view with the "Files of type" widget at the bottom.

1.2.3 Data Configurations



The screenshot shows the 'Iris' application window. On the left is a configuration panel with the following elements: 'IV Data' and 'Pinsizes' text boxes with left-pointing arrows; a 'Download all default graphs' checkbox; an 'Extra Biases' text box containing 'e.g. -1.3, 0.5, ...'; a 'Filter high/low devices' checkbox; two text boxes for current thresholds: '→ Min Current @ Bias' (with example 'e.g. '1e-1 @ -0.1''') and '→ Max Current @ Bias' (with example 'e.g. '1e3 @ -0.1'''); a 'Generate Analysis' button; a 'View Plot...' dropdown; 'Temp Select' and 'Volt Select' dropdowns; 'Diode Size' (selected) and 'Diode ID' radio buttons; and a '↓ CSV' button. On the right is a large empty plot area. A red circle is drawn around the 'Extra Biases', 'Filter high/low devices', and the two current threshold fields.

This section of widgets is dedicated to adding extra views and filtering the data.

Extra Biases

The 'Extra Biases' textbox takes in a comma separated list of voltages that may be of interest in the graphical viewing. Given that 6 of the 10 plot types depend on the voltage selection for the viewing of the data, adding more voltages adds more viewing options. By default, the voltage options will always include -0.1 V and +0.1V. This field can be left blank.

Filters

If you select the "Filter high/low devices" checkbox, you must select current densities to assign as low and high thresholds at specific biases. The selections should be in the form: (Current Density) @ (Bias). The '@' sign is necessary. By default, all devices that emit current densities below 10^{-11} A cm⁻² at -0.1 V or above 10^3 A cm⁻² are filtered and grayed out from selection for viewing. Additionally, any device that outputs 0 A at any bias is also filtered and grayed out.

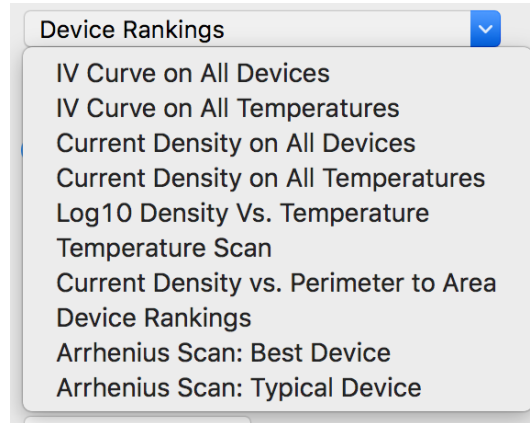
1.2.4 Generate Analysis

Once inputs have been selected, click the "Generate Analysis" button. After about one second, a plot of your inputted IV data will appear on the right.

1.3 Navigating the Plots

1.3.1 Using the Widgets

There are four controls to update the graphics widgets: Plot Type Select, Temperature Select, Voltage Select, and Device Selection.



Plot Type Selection is a drop down menu of ten plot types. The button label will always display the name of the currently displayed plot. Selecting another plot type will update the graph to that plot type. Upon the change, the Temperature Select and Voltage Select buttons will default to the lowest temperature and -0.1V, respectively.

Temperature Select is a drop down display of the temperatures that were parsed from the IV data file names. The button label will always display the temperature for the current plot unless temperature is not a variable parameter for the current plot type.

Voltage Select is a drop down display of voltages that encompass -0.1 V, 0.1 V, and any additionally biases you may have inputted. The button label will always display the voltage for the current plot unless voltage is not a variable parameter for the current plot type.

Device Selection is composed of two selection methods, "Diode Size" and "Diode ID". Only one option can be selected at one time. When "Diode Size" is selected, any selected check boxes will trigger the display the data of all unfiltered devices of that size on the current plot. When "Diode ID" is selected, any selected check boxes will trigger the display of the corresponding device's data on the current plot. Note that all filtered devices will have a "(X)" next to its label and be disabled from selection.

1.3.2 Plot Types

This section will outline how to interpret the plots from a application-specific perspective.

IV Curve on All Devices

- X-axis: Voltage (V)
- Y-axis: Observed Dark Current (A)
- Semi-Log scale
- Volt Select disabled

This plot type displays the IV data for each selected device at the selected temperature.

IV Curve on All Temperatures

- X-axis: Voltage
- Y-axis: Observed Dark Current (A)
- Semi-Log scale
- Temp Select disabled
- Intended for viewing single device

This plot type displays IV data as single curve for each temperature in the data set. If multiple devices are selected, their respective IV curves are averaged for each temperature in the data set.

Current Density on All Temperatures

- X-axis: Voltage (V)
- Y-axis: Dark Current Density (A cm^{-2})
- Semi-Log scale
- Volt Select disabled

This plot type displays the current density for each selected device at the selected temperature.

Current Density on All Temperatures

- X-axis: Voltage
- Y-axis: Dark Current Density (A cm^{-2})
- Semi-Log scale
- Temp Select disabled
- Intended for viewing single device

This plot type displays dark current density as single curve for each temperature in the data set. If multiple devices are selected, their respective dark current density curves are averaged for each temperature in the data set.

Log10 Density Vs. Temperature

- X-axis: Temperature (K)
- Y-axis: $\text{Log}_{10}(\text{Dark Current Density})$ (A cm^{-2})
- Semi-Log scale
- Temp Select disabled

This plot type displays the log 10 current density of each device against the temperatures in the data set.

Temperature Scan

- X-axis: $1000/\text{Temperature}$ (K^{-1})
- Y-axis: $\ln(\text{Dark Current Density} / \text{Temperature}^3)$ ($\text{A cm}^{-2} \text{K}^{-3}$)
- Temp Select disabled

This plot type displays the temperature scan of each selected device against the inverse of each temperature in the data set multiplied by 1000.

Current Density vs. Perimeter-to-Area

- X-axis: Device Perimeter/Area (cm^{-1})
- Y-axis: Dark Current Density (A cm^{-2})
- Semi-Log scale

This plot type displays the dark current density at the selected temperature and voltage vs. the selected devices' perimeter-area ratios.

Device Rankings

- X-axis: Device ID
- Y-axis: Dark Current Density (A cm^{-2})
- Semi-Log scale

This plot type will display the selected devices' dark current density at the selected temperature and voltage as an ascending scatter plot, where each point corresponds to the ID it aligns with on the X-axis. Lower points are considered better devices.

Arrhenius Scan: Best Device

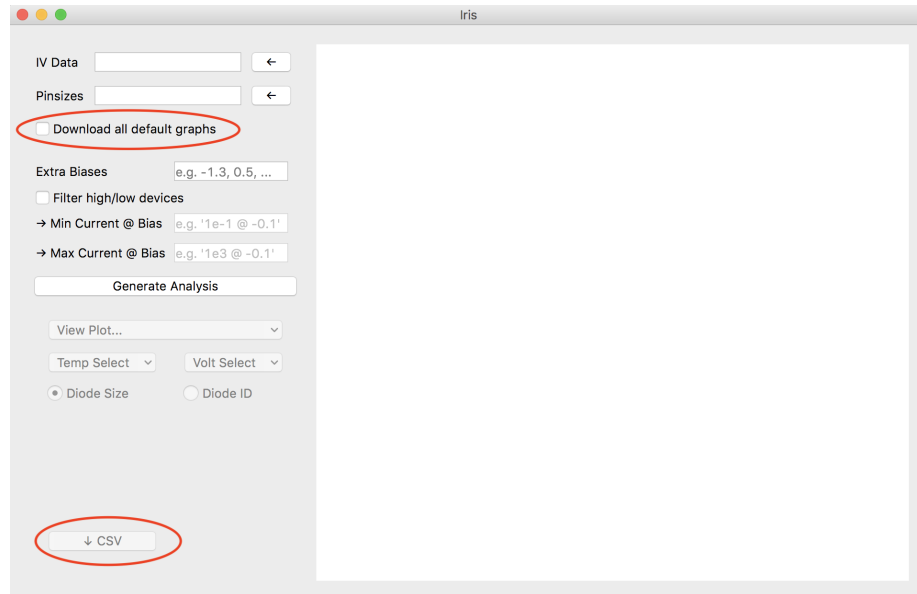
- X-axis: $1000/\text{Temperature}$ (K^{-1})
- Y-axis: $\ln(\text{Dark Current Density} / \text{Temperature}^3)$ ($\text{A cm}^{-2} \text{ K}^{-3}$)

This plot type depicts 3 items. The plotted points marked with green stars represent the "best" device's temperature scan, where the "best" device is the device with the lowest dark current density at the selected temperature and voltage among the selected devices. The red line represents the generation recombination current slope (a linear regression of the device temperature scan's top points) and the blue line represents the diffusion slope (a linear regression of the device temperature scan's bottom points). These slopes have their corresponding lambda values marked in the legend.

Arrhenius Scan: Typical Device

This plot is identical to Arrhenius Scan: Best Device, except that the device with the median current density is displayed among the selected devices.

1.4 Downloading Outputs



All plotted data in IRIS is downloadable as a tab separated .txt file. Like the input files, these can be opened in Excel. Each plot type's downloadable file is formatted slightly differently.

Notice the two circled section on the image above. When clicked, the "Download all default graphs" check box will open a file dialog to allow selection of the location to save the files to name the folder with the plot data. The default downloads will be further partitioned by plot type. "Default" signifies that the download plot data will only be of plots with all devices (or one device where intended) selected. Every data file will be named automatically where the format is as follows: (plottype)-(temperature)K-(voltage)V.txt.

To download plots with custom device selections (or to download individual plots without downloading all), click the CSV button at the bottom of the UI when viewing the intended plot. Like for "Download all default plots", this will open a file dialog to select the save location and name of the file.

Do note that downloading all default will create about 3 MB of data with 7 temperatures and 2 voltages in the data set. Selecting more voltages or having data sets with more temperatures will increase this quantity quite a bit.

Chapter 2

A Dev's Guide to IRIS

2.1 Introduction

IRIS is implemented with 3 Python files, `iris.py`, `trials.py`, and `utils.py`. `iris.py` is the main file for the program, so the program can be run from the command line by calling `'python iris.py'`.

Each file imports only the function and data structures utilized in the program from the package. Please refrain from importing entire packages.

2.2 `iris.py`

`iris.py` contains about 80% of all the code that runs IRIS. `iris.py` handles the window construction, widget creation and layout management, event handling, download creation, and plotting. Almost all functions in `iris.py` are created as methods of the `MainWindow` object. Below `self` refers to the `MainWindow` object for each instance of an IRIS program.

2.2.1 Widget Creation and Event Handling

```
def initInputUI(self)
```

Creates and connects widgets for data input and data configurations. All `iris.py` actions are handled within `MainWindow` object.

```
def initPlotUI(self)
```

Creates and connect widgets for plot navigation and selection.

```
def openInputDialog(self, textfield, mode)
```

Opens the file dialog for selecting and setting the IV data files and pin size file.

```
def processAllData(self)
```

Creates the DataFrame and Trial objects that hold the program data by parsing the IV data files and pin size file.

```
def generate(self)
```

Configures plot control widgets for the selected data set and initializes the creation of the data structures that will store the parsed data. Begins process of downloading all defaults if option has been selected.

```
def downloadAllAsCSV(self, path)
```

If a path has been selected for saving downloads, process the parsed data and format into .txt files for download.

```
def getDownloadPath(self)
```

Open a file dialog for user to select save location and name for default downloads folder.

```
def initSelectButtonMenus(self)
```

Add the temperatures and voltages for the generated data set into Temperature Select and Voltage Select buttons, respectively.

```
def initSelectionLists(self)
```

Create and populate the Device Selection list widgets.

```
def updateFilterCheckBox(self)
```

Enables and disables low filter and high filter text boxes depending on if the "Filter high/low devices" check box is selected.

```
def updatePlotBy(self)
```

Configure Device Selection lists according to which device selection criteria is selected ("Diode Size" or "Diode ID").

```
def downloadCSV(self)
```

Download the current plot data as a .txt file.

```
def getSelectedSizes(self)
```

Return a list of devices according to which sizes are selected in the "Diode Size" list.

```
def getSelectedDevices(self)
```

Return a list of devices according to which IDs are selected in the "Diode ID" list.

```
def changePlot(self, plottype, temp, volt)
```

Update the graphics widget according to the plot type (plottype), temperature (temp), and voltage (volt) parameters.

2.2.2 Plotting

Every plot type has an associate "plot" and "get" function. Each is configured uniquely for the desired plot type.

```
def plot[plottype](self, temp, volt)
```

Depending on the plot type, the parameters may require both temperature and voltage, one of the two only, or neither. Each plot function selects the devices for plotting based on which devices are selected in the Device Selection menu. Then, the plot is titled, labeled, set to semi log or linear. A legend is created for each unique plot. Finally, the function will iterate through `plotitems`, which is returned by the `get[plot type]` function.

```
def get[plot type](self, temp, volt, selected)
```

Returns a specifically formatted iterable (usually a dictionary) mapping device IDs to the associated plot type data for the selected plot type. Only devices in selected will be included in the `plotitems` object.

```
def show[plot type]Settings(self)
```

Configures the plot control widgets to reflect the parameter selection options for the current plot type. Will map the buttons in Temperature Select and Voltage Select to update to plots of the current type, if applicable.

2.3 trials.py

trials.py contains the handling of the Trial object, the main data structure that stores and processes the parsed IV data.

2.3.1 Trial Object

```
def __create_current_and_filter(self, filename, trial)
```

Creates the DataFrame to hold the IV data passed through for the given temperature and filters out all devices that have zero size in the pin size file.

```
def __create_jd(self)
```

Creates the DataFrame to hold the dark current density data for the given temperature and its IV data.

```
def __create_tempscan(self)
```

Creates the DataFrame to hold the temperature scan data at this temperature for every voltage in the data set.

```
def __flag_errors(self, maxjd, minjd)
```

Append the name of any device that has erroneous current or current density data to `self.flags`.

```
def __nomeas_flagger(self, column)
```

Flag any device that has a 0 anywhere in its current data.

```
def __threshold_flagger(self, column, maxjd, minjd)
```

Flag any device that exceeds the lower or upper thresholds (`minjd`, `maxjd`) in current density.

```
def __parse_filename(self, filename)
```

Extract temperature information from the name of the file.

2.3.2 Additional Functions

```
def get_tempscans(trials, BIAS, PINSIZES, selected, v)
```

Return a dictionary mapping device ID to the associated temperature scan at the given voltage `v`. This is a function outside of the Trial object since this needs to handle every trial in a data set to create a temperature scan across all temperatures. A Trial object only contains data for one temperature.

2.4 utils.py

utils.py contains a few extra utility functions to ease the processing and display of data in iris.py.

2.4.1 Utilities

```
def process_sizes_txt(filename)
```

Parse a pin size file with .txt extension. Return a dictionary with device IDs as keys and associated sizes of the device as the values.

```
def process_sizes_xlsm(filename)
```

Parse a pin size file with .xlsm extension. Return a dictionary with device IDs as keys and associated sizes of the device as the values.

```
def verify_domain(*args)
```

Given IV DataFrames as args, return a boolean signifying that the "Vs" columns in the IV data are identical and a Series object representation of the "Vs" column.

```
def trim_nan(*args)
```

Filter all NaN values off of the "Vs" columns in the IV DataFrames in args.

```
def format_Vs(*args)
```

Filter all other erroneous characters and convert types to numeric in the "Vs" columns of the IV DataFrames in args.

```
def colorhash(label)
```

Given a device label (formatted as I(int)), return two chars to denote which symbol and color to display the plot data of that device.

```
def get_arrhenius_lambdas(gr_coeffs, diff_coeffs)
```

Given the line data of the generation recombination line and diffusion line, determine the lambda values of each.

```
def get_flags(trials)
```

Return a list of devices that have been flagged in any of the trials passed in.

```
def format_voltage(trials)
```

Given a voltage v, return a string representation of that voltage.

2.5 Creating an Executable

Any time there is an update or patch to the IRIS source code, new executables must be generated to match the update.

However, Python is an interpreted language, and thus by default, cannot be compiled into an executable without external help. Pyinstaller is a package that will remedy this issue.

2.5.1 Installing Pyinstaller

It is important to note that the default version of Pyinstaller (which can be installed with pip) cannot correctly compile IRIS. This is perhaps due to a library dependency conflict within the PyQt5 library. A developer version of Pyinstaller exists to circumvent this issue. To install this version, type into the command line:

```
$ pip install https://github.com/pyinstaller/pyinstaller/archive/develop.tar.gz
```

This will uninstall Pyinstaller, if it exists, and reinstall it as the developer copy.

2.5.2 Installing Python 3.5

IRIS is written for compatibility with Python 3.5.5. However, Pyinstaller only successfully creates executables for pre-3.6 versions of Python. Anaconda offers an easy way to activate a Python environment with the preferred version. Assuming Anaconda is installed, do the following:

For Mac

1. Create the Python 3.5 environment.

```
$ conda create --name py35 python=3.5
```
2. Activate the created environment.

```
$ source activate py35
```
3. Install all the packages that IRIS uses that aren't already part of Anaconda.

```
$ pip install PyQt5  
$ pip install pyqtgraph
```
4. From here, follow the steps in Running Pyinstaller.

For Windows

1. Open an Anaconda Command Prompt.
2. From here, follow the For Mac installation steps.

2.5.3 Running Pyinstaller

Once, Pyinstaller (dev version) has been correctly installed we can generate the executable files for IRIS.

IMPORTANT: Pyinstaller is compatible with all operating systems. However, compiling IRIS on a Windows system will only allow that version to be opened on Windows. The same goes for Mac. Thus, to have both Mac and Windows versions, these steps must be followed twice, once each for the two operating systems.

1. From your working directory run on the command line:
`$ pyinstaller iris.py`
2. After about 20 seconds, the command prompt will output the following message:
`RecursionError: maximum recursion depth exceeded`
3. To circumvent the recursion limit, open `iris.spec`, which was created by the previous command.
At the top of the file under `"# -*- mode: python -*-"`, insert these two lines and save:
`import sys`
`sys.setrecursionlimit(5000)`
4. Run Pyinstaller once more but this time on `iris.spec`
`$ pyinstaller iris.spec`
5. After about one minute, this will have created and populated two directories labeled `dist` and `build`. We only care about `dist`. Inside `dist` is another directory labeled `iris`. Inside the `iris` is the IRIS executable (labeled `iris`) and every library and dependency that IRIS needs to be opened as an executable. For Windows, change the name of the executable `iris` to `iris.exe`. For Mac, change to `iris.app`. The `iris` directory can now be moved to different locations. `iris.exe` and `iris.app` must remain in this directory. To open these, click their icons or create a shortcut for remote access.
6. **(OPTIONAL)** If the executables close immediately upon launch or output an error about dependencies on the command line, delete `iris.spec`, `dist`, and `build`. This is because PyQt5 may be incorrectly installed on your system. To fix this, run the following commands:
`$ pip uninstall PyQt5`
`$ pip install PyQt5`
This will reconfigure PyQt5 for a fresh start.
7. **(OPTIONAL)** Follow steps 1 through 5.