Contents

1 Introduction

SCO `LabTools` is developed using the Python 3 language. It is a comprehensive toolkit for SCO research, which provides programs to plot graphs from common measurement methods: namely, SQUID magnetometry, Raman spectroscopy and reflectometry.

The intention of this software package is to be as user-friendly as practically possible. Unfortunately, no time allowed for introduction of a guided user interface (GUI) so the code will have to be run from the terminal; to compensate, this manual will be comprehensive. Please refer to this manual as your first port of call for help.

2 Installation

2.1 Installing Python 3 (Anaconda 3)

SCO `LabTools` requires that the Python 3 have access to both scientific & numeric modules and elements of the Python imaging library (PIL): the Anaconda 3 distribution of Python comes as standard with all of these required modules.

Download Anaconda 3 from `https://www.continuum.io/downloads`

This will probably install Anaconda 3 to the `C://Users/`*<user>*`/Anaconda3/` directory.

2.2 Installing additional dependencies

Hopefully, Anaconda should contain all the dependencies (*i.e.* packages, modules) we need. Just to be sure, we can explicitly install requirements using pip (pip should be installed with Python). We can do this using the requirements.txt file placed in the SCOLabTools project folder by running the following at the terminal line:

```
>> pip install -r requirements.txt
```

3 Running

3.1 Running programs

Codes must be run from the command line in the terminal. For Windows users this means using *Command Prompt*.

Point to the interpreter. Load up the terminal and do the following:

1. Locate the interpreter.

   We must tell the terminal *where* our interpreter is[1]. The interpreter will be reading the code and translating it from human readable Python code to a list of binary instructions which the computer can understand.

   - We will be using Python 3 to interpret, so we need to include the path of the Python distribution we want to use, which will be located in the Anaconda 3 directory (probably `C://Users/<user>/Anaconda3`, where *<user>* is the name of the user who installed Anaconda.) Try looking through the directory structure for the named '`Anaconda3`' if you're unsure which user has installed Anaconda 3.
   - Inside the `Anaconda3` directory there should be a file named '`python.exe`'. This is our interpreter.

2. Copy interpreter file path.

   On Windows, to get the path to the `python.exe` file, hold shift and right-click. Select 'Copy as path' from the drop-down menu: this will save the path to your clipboard.

3. Now go back to your terminal window, right click and select 'Paste' from the drop-down menu.

   Note that the keyboard shortcut '`Ctrl + v`' does not work.

In your terminal window you should now have[2]:

```
>> C://<path to Anaconda>/Anaconda3/python.exe
```

Point to the program. Now we can locate the program we want to run.

1. Locate program code file in directory structure. This will have the extension '`.py`' or '`.pyc`'.

[1] We can instead include Python 3 to our PATH variable—meaning we can call our Anaconda 3 distribution using the simple command `python`—but this is more complicated than it needs to be so I'll leave it out. Google it if you want to do this but be careful you don't accidentally default to Python 2 because the code won't run.

[2] The ">>" signifies that we're talking about the command line input. You do not need to include this when you write into the terminal. Think of it like a bullet point in a list.

2. Copy code file path. Windows: `Ctrl` + right-click, 'Copy as path'.

3. Paste code file path into terminal next to interpreter path.

In your terminal window you should now have:

```
>> C://<path to Anaconda>/Anaconda3/python.exe     C://<path to program>/<program name>.py
```

At this point, one must include compulsory command line arguments after the program or optional command line arguments after flags. SCO `LabTools` does not use any compulsory arguments but here is the general structure:

```
>> C://<path to Anaconda>/Anaconda3/python.exe     C://<path to program>/<program name>.py
     <compulsory args> <optional flags>
```

More detail is provided in the next section.

3.2 Using flags

Flags are optional command line arguments which follow the calling of a code. They are called using a hyphen '-' or double-hyphen '--' proceeded by an identifying word or letter, the *<flag id>*. The argument of the flag, the *<flag arg>*, is then given after a blank space.

In general:

```
>> <interpreter>.exe <program>.py -<flag1 id> <flag1 arg> -<flag2 id> <flag2 arg> ···
```

Examples of running using flags follow:

(1)
```
>> D://Interpreters/Anaconda3/python.exe C://Python/Projects/io_program.py --input
     C://Documents/Path/to/input/dir/ --output F://Documents/Path/to/output/dir/
```

This is a standard input/output (IO) example. Here, the user specifies their interpreter to be in their chosen interpreters depository on their `D` drive, '`D://Interpreters/`, where they use Anaconda 3's Python interpreter to run the program '`io_program.py`'. The flags `--input` and `--output` intuitively specify the paths to the relevant directories.

(2)
```
>> C://Users/User1/Anaconda3/python.exe C://MyPythonCodes/python_code.py -p
     C://Documents/Path/to/input/file.in --diagonalize True
```

This example calls the Anaconda 3 Python interpreter 'python.exe' from the default location and asks it to interpret the program code 'python_code.py' from this user's code depository directory, 'C://MyPythonCodes/' using the optional flags, '-p' and '--diagonalize'. Evidently, '-p' expects a file path; '--diagonalize' expects a Boolean (or logical) specification.

4 Creating External Files

Many of programs require external files to provide information in order to run effectively. The most common are as follows:

File	Description	Compulsory?	Notes
`times.dat`	Image creation times file	No	Required for intuitive temperature.
`curve.dat`	Curve profile file	No	Required for intuitive temperature.
`roi.dat`	Region of Interest (ROI) file	No	
`config.yaml`	Configuration file	No	

Table 1: Summary of common external files used by codes in SCO `LabTools`.

4.1 Creating `.dat` and `.yaml` files by hand

4.2 Image creation times file, `times.dat`

The `times.dat` file must be created using the `time.py` program.

4.3 Curve profile file, `curve.dat`

The `curve.dat` file may be produced by hand; it does not require the use of additional software to create the file.

4.3.1 General format

For a curve profile of n elements, the `curve.dat` file has the following general formulation:

$$\begin{array}{lll} \texttt{rate}_1 & \texttt{limit}_1 & \texttt{time}_1 \\ \texttt{rate}_2 & \texttt{limit}_2 & \texttt{time}_2 \\ \vdots & \vdots & \vdots \\ \texttt{rate}_n & \texttt{limit}_n & \texttt{time}_n \end{array}$$

Notice here that every element on each row is separated by a *tab*.

Rates are in degrees Celsius per minute, $^{\circ}\mathrm{C\,min^{-1}}$; limits are in degrees Celsius, $^{\circ}\mathrm{C}$; and wait times are in minutes, min. This is almost analogous to the process of inputting a temperature curve profile into the Linkam temperature

stage, with the exception of the time. If you have a time t in seconds and need to convert to minutes, use the conversion formula

$$t_{\text{minutes}} = 0.0167\, t_{\text{seconds}}. \tag{1}$$

4.3.2 Examples

(1)

```
50.00  25.0   1.00
10.00  70.0   5.00
10.00  130.0  5.00
10.00  160.0  5.00
10.00  130.0  5.00
10.00  70.0   5.00
10.00  25.0   5.00
10.00  70.0   5.00
10.00  130.0  5.00
10.00  160.0  5.00
10.00  130.0  5.00
10.00  70.0   5.00
10.00  25.0   5.00
10.00  70.0   5.00
10.00  130.0  5.00
10.00  160.0  5.00
10.00  130.0  5.00
10.00  70.0   5.00
10.00  25.0   60.00
```

This example is a temperature curve at $10\,^{\circ}\mathrm{C}\,\mathrm{min}^{-1}$ with a wait time of 5 min between every section of the curve: it produces the curve shown in figure 1.

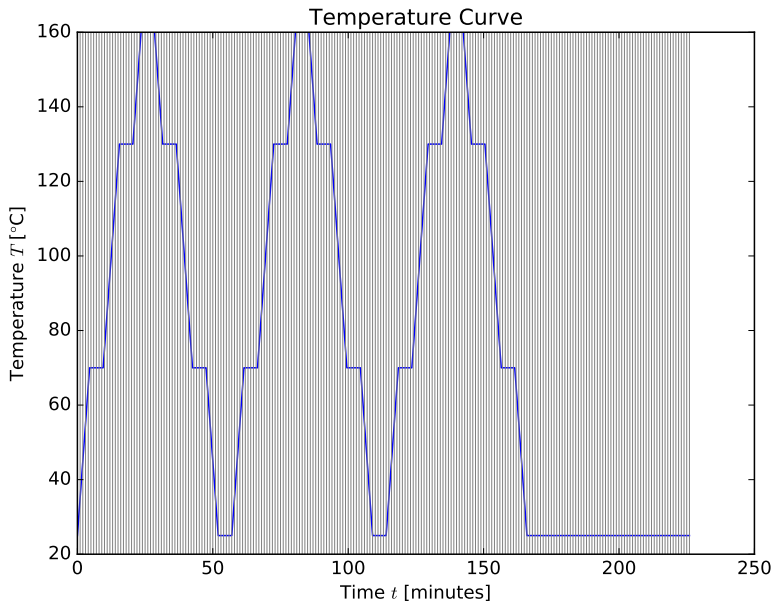

Figure 1: Temperature curve for the profile in example (1).

4.4 Region of Interest (ROI) file, `roi.dat`

The `roi.dat` file specifies the region of interest within the image. It may be produced by hand.

4.4.1 General format

```
x_0   y_0   window_width
```

Here, x_0 and y_0 specify the initial x and y co-ordinates of the ROI, window_width specifies the window width, such that the ROI will have the top-left co-ordinate (x_0, y_0) and the bottom-right ($x_0 +$ window_width, $y_0 +$ window_width).

4.5 Configuration files, `config.yaml`

For most of the codes, configuration files are depreciated. Some still use them, however.

It is important to note that *Boolean operators are entirely lower case* in configuration files due to the conventions of the YAML. Thus, *TRUE* is designated 'true' and *FALSE* is designated 'false'.

4.5.1 Config files for `plot.py`

Example config file

```
title:  Rate-dependency of Hysteresis
xlabel:  Ramping rate $ \dotT $ [K/min]
ylabel:  Thermal hysteresis width $ \Delta T $ [K]

format:  o
grid:  false

outname:  graph_rate_Dtemp
outexts:  [.png, .pdf, .svg]

fitline:  true
a:  1.5
b:  36.125
linerange:  [-1, 12]

errorprovided:  true
biglabels:  true
```

Line fitting, *fitline*

Uses the equation

$$y = a\,x + b. \tag{2}$$

Exponential fitting, *fitexp*

Uses the equation

$$y = a\, e^{b\, x}. \tag{3}$$

Additional notes

- If `errorprovided: true`, the input `data.dat` file must be of the following general form:

$$\begin{array}{ccc} x_1 & y_1 & \Delta y_1 \\ x_2 & y_2 & \Delta y_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & \Delta y_n \end{array}$$

- For general advice on using YAML files, see the guide provided by *Robot Has No Heart* here:

  https://rhnh.net/2011/01/31/yaml-tutorial/[3].

- For general advice on appropriate `matplotlib` graphing `formats`, see the `matplotlib` documentation.

[3] Date accessed: 2017-02-28.

5 Reflectivity, `reflectivity.py`

This section refers specifically to the program located at *<root>*/SCOLabTools/reflectivity.py. The program can be called using the following general form:

```
>> <path to Anaconda>/Anaconda3/python.exe      <root>/SCOLabTools/reflectivity.py
     <flags>
```

The code will run the following processes:

1. Obtain input directory. (required)

   One has the choice to provide an input directory at the command line or during running. The flags are provided in table 2. For more information on using flags, see section 3.2.

Flag	Description	Required?
-i or --input	Input directory path	No
-o or --output	Output directory path	No

Table 2: Flags for optional command line arguments in `reflectivity.py`.

   Notice that the flags are not required to run the code. If the input directory is excluded from the command line arguments the program will simply ask the user to provide an input directory via the terminal.

```
>> C://Users/<user>/Anaconda3/python.exe C://Users/<user>/.../SCOLabTools/reflectivity.py
     Running 'reflectivity.py'.
     Please provide an input path:
```

2. Reads and processes optional configuration files from input directory.

   (a) Reads ROI file, *<input dir>*/output/roi.dat.

   (b) Reads temperature curve profile file, *<input dir>*/output/curve.dat.

   (c) Reads file creation times file[4], *<input dir>*/output/times.dat.

   (d) Reads config file, *<input dir>*/config.yaml. Note this is depreciated in the most recent versions of this code but can still provide useful input.

[4] File creation times file **must** be produced using the *original* images, not copies. Please see section 8.1 for more information on running the `times.py` code to produce a `times.dat` file.

3. Produces a function describing the temperature curve (uses `curve.py`.)

4. Reads and processes images from input directory.

   (a) Reads each image, indexed i.

   (b) Crops i^{th} image to ROI.

   (c) Compresses cropped i^{th} image.

   (d) Extracts color (RGB) data for each pixel in cropped i^{th} image and calculates mean. Also calculates grayscale (k) intensity.

5. Plots and outputs graphs.

   Graphs will be saved to output directory under the names '`graph.`*<ext>*', where *<ext>* $\in$ {.png, .pdf, .svg} by default.

   Graph axes are intuitively plotted: if temperature data is available then intensity *vs.* temperature will be plotted, else intensity *vs.* image number (can be related to time) will be plotted.

   Hysteresis loops will be automatically split into individual loops with different colours for each loop.

5.1 Example of output

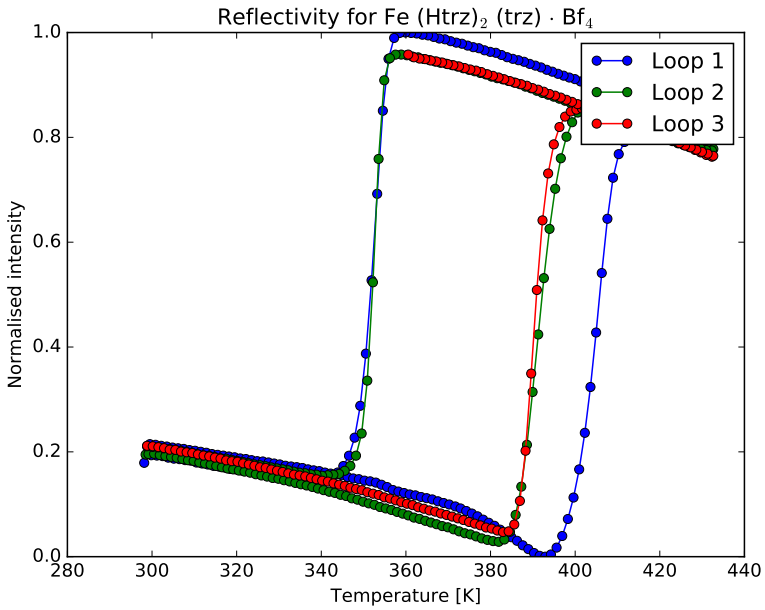

Figure 2: SCO transition curve using reflectometry. Plotted using default output option of `reflectivity.py`.

6 Raman, `raman.py`

This section refers specifically to the program located at *<root>*/SCOLabTools/raman.py. The program can be called using the following general form:

```
>> <path to Anaconda>/Anaconda3/python.exe     <root>/SCOLabTools/raman.py     <flags>
```

The code will run the following processes:

1. Obtain input directory. (required)

   One has the choice to provide an input directory at the command line or during running. The flags are provided in table 3. For more information on using flags, see section 3.2.

Flag	Description	Required?
-i or --input	Input directory path	No
-o or --output	Output directory path	No

Table 3: Flags for optional command line arguments in `raman.py`.

   Notice that the flags are not required to run the code. If the input directory is excluded from the command line arguments the program will simply ask the user to provide an input directory via the terminal.

2. Read data from input directory.
3. Normalise the data between 0 and 1 to minimum and maximum value of each spectrum.
4. Find local peaks and select only significant peaks with intensity $> \gamma\,\mathrm{SNR}$, where SNR is the signal-to-noise ratio.
5. Output peak values and locations to a data file in '`./output/peaks.dat`'.
6. Plot spectra onto graph of intensity *vs.* Raman shift. Circle significant peaks.

7 SQUID, `squid.py`

1. Obtain input directory. (required)

   One has the choice to provide an input directory at the command line or during running. The flags are provided in table 4. For more information on using flags, see section 3.2.

Flag	Description	Required?
`-i` or `--input`	Input directory path	No
`-o` or `--output`	Output directory path	No
`-Mr` or `--relative_mass`	Relative molecular mass M_r [grams/mole]	Yes
`-B` or `--field`	External field B [oersted]	Yes
`-m` or `--mass`	Sample mass m [mass]	Yes

Table 4: Flags for optional command line arguments in `squid.py`.

   Notice that the flags are not required to run the code. If the input directory is excluded from the command line arguments the program will simply ask the user to provide an input directory via the terminal.

2. Read data from input directory. Note that `squid.py` requires a directory path and cannot yet handle a single file input.

3. Normalise the data between 0 and 1 to minimum and maximum value of each spectrum.

4. Extract long moment $\bar{\mu}$ and temperature T values.

5. Produce $\chi_M T$ value using the equation:

   $$\chi_M T = \frac{\bar{\mu} T M_r}{B m}$$

6. Separate hysteresis data into loops.

7. Plot data showing loops as different colours.

8. Plot arrows onto graph to show direction of travel (*i.e.* whether heating or cooling).

8 Miscellaneous Programs

8.1 Image Creation Times, `times.py`

This program retrieves image creation times from the original images. **It must be run on the original images and not a copy** in order to produce the times the photos were taken, otherwise it will provide the time that the files were copied.

It can be run using the following:

```
>> <path to Anaconda>/Anaconda3/python.exe      <root>/SCOLabTools/times.py      <flags>
```

The possible flags are provided in table 5.

Flag	Description	Required?
`-i` or `--input`	Input directory path	No
`-o` or `--output`	Output directory path	No

Table 5: Flags for optional command line arguments in `reflectivity.py`.

8.2 Temperature Curve, `curve.py`

This program produces plot of the curve profile provided in a `curve.dat` file. See section 4.3 for information on producing a `curve.dat` file.

1. Obtain input `curve.dat` file. (required)

   One has the choice to provide an input file at the command line or during running. The flags are provided in table 6. For more information on using flags, see section 3.2.

Flag	Description	Required?
`-i` or `--input`	Input curve file path	No

Table 6: Flags for optional command line arguments in `curve.py`.

   Notice that the flags are not required to run the code. If the input directory is excluded from the command line arguments the program will simply ask the user to provide an input directory via the terminal.

2. Read curve profile from input file, $\texttt{rate}_i$ $\texttt{limit}_i$ $\texttt{time}_i$.

3. Translate curve profile into a set of points, $\{T_j\}$.

4. Interpolate set of points into a function, $\{T_j\} \rightarrow T(t_i)$.

5. Produce and show plot to illustrate temperature curve.

6. Terminal line input will provide the user with the maximum allowable image number n_{max} and ask user to provide an image number n. The program will return the temperature $T(n)$ corresponding to the image of the given number. The user **must** provide an image number $n < n_{\text{max}}$ otherwise the program will crash.

Valid input:

>> '`####`' (int) – Any whole integer number.

>> '`help`' (str) – Will provide help on valid input.

>> '`exit`' (str) – Will safely exit program.

8.3 Plot, `plot.py`

Simple code to plot a graph from a data file.

1. Obtain input `curve.dat` file. (required)

   One has the choice to provide an input file and config file at the command line or during running. The flags are provided in table 7. For more information on using flags, see section 3.2.

Flag	Description	Required?
`-i` or `--input`	Input curve file path	No
`-c` or `--config`	Config file path	No

Table 7: Flags for optional command line arguments in `plot.py`.

2. Read data file.

3. Read configuration file.

4. Apply formatting and trend-lines as outlined in the configuration file.

5. Output plot to screen and to files specified in configuration file.

9 Function Depositories

9.1 Common, `common.py`

This is the main function depository which is used by all of the main programs. It is often imported using the command

```
import common as com
```

and its functions are hence called using the general format

```
com.<function name>(*args, **cargs)
```

Notable functions. Some of the noteworthy functions in `common.py` are as follows:

- `draw_roi(img_arr, filename, output_path, roi)`

  Takes numpy array representing an image, `img_arr`, and draws lines representing the ROI outlined by `roi`. Outputs to path "`output_path/filename`".

  `roi` must be a list of the form `[x0, y0, x1, y1]`.

- `plot_hysteresis(plt, x, y, format='-', legend=True)`

  Plots a graph which automatically separates the hysteresis loops from one another. Optional legend.

  `plt`: The `matplotlib.pyplot` plot the user wishes to work with. For example, if you have called `import matplotlib.pyplot as funplotter` then you would pass `funplotter` into the `plt` place-holder.

  `x`, `y`: The x and y plot variables. Must be a 1D list or 1D numpy array.

  `format`: Specifies the graph format. See the matplotlib documentation for accepted plot formats. (Examples: `'o'` is data point dots, `'-'` is a line, `'o-'` is dots and a line.)

  `legend=True` Automatically produces a legend with the loop number specified for each loop, in the format 'Loop 1, Loop 2, $\cdots$, Loop n'. If you do not want a legend, set `legend=False` when calling the function.

- `normalize(X, min=None,min=None)`

Normalises `X` between 0 and 1 for limits of `min` and `max`. If `min` and `max` are unspecified, will use the minimum and maximum values found in `X`.

Uses the following normalisation equation:

$$\bar{X} = (X - X_{\text{min}})/(X_{\text{max}} - X_{\text{min}}).$$

Note that the function is written in American English by programming convention. Please be aware of this when calling the code.

- `io_from_args(args)`

  Takes arguments from the `argparse` package, provided by `args`, and separates into input and output paths, while proving user input1 validation. If the path is invalid, it will ask the user to provide another path. If the output path is not provided it will simply produce an output directory named '`./output/`' in the input directory.

- `if_not_exists_create_dir(dir)`

  Checks if a directory exists and creates it if not.

- `if_not_exists_create_file(file)`

  Checks if a file exists and creates it if not.

There are other noteworthy functions used. The function of these will be clear upon reading the code.

10 Glossary

Interpreter Similar in concept to a code compiler. In this case, the interpreter is Python 3 which will be an executable file with the name 'python.exe'. If Python was installed using Anaconda 3 using the default settings, the Python interpreter will probably be in the following location: C://Users/*<user>*/Anaconda3/python.exe.

Path A path is the address to any directory or file. For example, the following is a path to the file 'file_name.ext':

(*) C://Users/Documents/DirectoryName/file_name.ext.

Relative path A path relative to the root directory. "./" represents the current directory; "../" represents the previous directory. For example, if our current location is C://Users/Documents/ then we can find the file in the "path" example using the following line:

(*) ./DirectoryName/file_name.ext.

If our current location is C://Users/Documents/DirectoryName/SubDir then we can find the file in the "path" example using the following line:

(*) ../file_name.ext.

Absolute path An absolute path is the full path from the "home drive". In the absolute path example below, the home drive is "C://":

(*) C://Users/Documents/DirectoryName/file_name.ext.

Directory What Windows users may conventionally know as a 'folder': a directory is a location in which files are stored. Sometimes called a 'dir' for short. Directories can be recognised because they do not have a file extension '.ext'. For example, the following is a path to a directory named 'MyDir':

(*) C://Users/Documents/DirectoryName/MyDir/.

File A file is where information is stored. Files have extensions corresponding the *type* of the data they hold. Examples of files include image files (.jpg), text files (.txt), data files (.dat), music files (.mp3). For example, the following is a path to the file 'file_name.ext':

(*) C://Users/Documents/DirectoryName/file_name.ext.