TRIOS Software and JSON Export: Working With Matplotlib.Pyplot in Python

Keywords: TRIOS, JSON, Python, Matplotlib

TB107

ABSTRACT

This technical brief discusses how to use the matplotlib.pyplot library in Python® to visualize data. It complements TA Instruments™ TRIOS™ Software by enabling consistent visualization layouts and the ability to plot data from various instruments, not just thermal/rheological data. This note covers the basic information required for plotting data and is not intended to replace the extensive documentation for both Python and the matplotlib.pyplot library.

Note: The exported JSON files (DSC and MDSC), and the final Python scripts (Jupyter® format) used in this technical brief can be downloaded (Script 1, Script 2, and Script 3) here.

INTRODUCTION

The TRIOS Software instrument control and data analysis package offers numerous features for both routine and in-depth analysis of scientific data. However, you may sometimes need to generate uniform plot formats, overlay data from multiple sources, or apply advanced functions not available within TRIOS.

TRIOS Software has always supported data export in various formats, including plain text (ASCII), Comma Separated Values (CSV), and Excel™ workbooks. TRIOS Software version 5.8 introduced the ability to export data to a JavaScript™ Object Notation (JSON) file. The import of this JSON file into Python, and the creation and interaction with the dataframe, were discussed in previous technical briefs TB105 and TB106. This note will explore how to generate plots to visualize that data.

MATPLOTLIB LIBRARY AND PYPLOT

Matplotlib is a comprehensive Python library for visualizing numerical data. The Pyplot functions allow Matplotlib and Python to work similarly to MATLAB.

To work with Matplotlib, it must first be installed and imported into the script. For installation, refer to the Matplotlib guides [1]. To import it into the script, you use the following command:

```
1. import matplotlib.pyplot as plt
```

This allows you to call functions using the abbreviation plt. For example, to show the plot, you simply use:

```
1. plt.show()
```

PLOTTING EXAMPLE 1: HEAT – COOL – REHEAT DATA OF POLYETHYLENE TEREPHTHALATE (PET)

If running a heat-cool-reheat measurement, you may wish to plot all three cycles on the same axes. In this example, a sample of quenched cooled PET was run through the following method:

1. Equilibrate 20.00 °C
2. Isothermal 5.0 min
3. Ramp 10.00 °C/min to 290.00 °C
4. Isothermal 5.0 min
5. Ramp 10.00 °C/min to 20.00 °C
6. Isothermal 5.0 min
7. Ramp 10.00 °C/min to 290.00 °C

This data was then exported into a JSON file for import into Python. The first section of the Python script sets up the required libraries, imports the JSON file, and sets the dataframes for each step:

```
1. from tadatakit.classes import Experiment
2. import matplotlib.pyplot as plt
3. experiment = Experiment.from_json("DSC - PET.json")
4.
5. step_name, pet_df = experiment.get_dataframes_by_step
   ("processed")
6.
7. for idx, step_name in enumerate(step_name):
       print(f"{idx}\t{step_name}")
```

This script creates seven dataframes:

- pet_df[0]: Equilibrate 20.00 °C
- pet_df[1]: Isothermal 5.0 min
- pet_df[2]: Ramp 10.00 °C/min to 290.00 °C
- pet_df[3]: Isothermal 5.0 min
- pet_df[4]: Ramp 10.00 °C/min to 20.00 °C
- pet_df[5]: Isothermal 5.0 min
- pet_df[6]: Ramp 10.00 °C/min to 290.00 °C

The next code block will set up the plot. You have control over the plot's appearance, including size, axes, signals, labels, and legend:

```
fig, ax1 = plt.subplots(figsize=(10, 6))

ax1.set(xlabel='Temperature (°C)', ylabel='Heat Flow (W/g)')

ax1.plot(pet_df[2]['Temperature / °C'], pet_df[2]['Heat Flow (Normalized) / W/g'], label='First Heat', color='green')

ax1.plot(pet_df[4]['Temperature / °C'], pet_df[4]['Heat Flow (Normalized) / W/g'], label='Cool', color='blue')

ax1.plot(pet_df[6]['Temperature / °C'], pet_df[6]['Heat Flow (Normalized) / W/g'], label='Second Heat', color='red')

plt.legend()

plt.show()
```

This script defines a plot with a size of 10 × 6 inches, labels for the axes, and plots three dataframes on the same axes. The plt.legend() command adds the legend, and plt.show() displays the plot as shown in Figure 1.

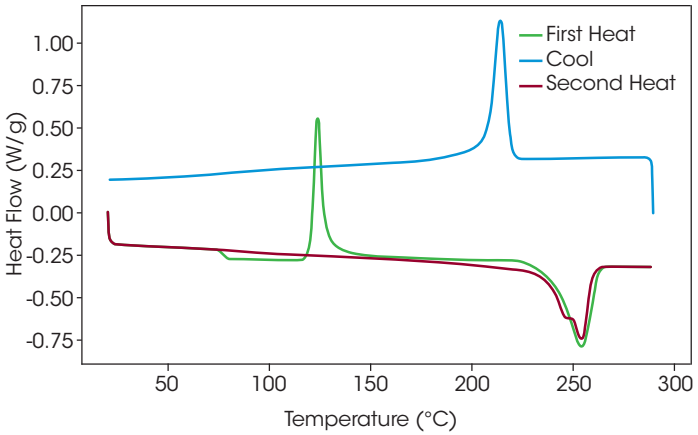

Figure 1. Plot generated from the above script

PLOTTING EXAMPLE 2: HEAT-ISO MODULATED DSC OF POLYETHYLENE TEREPHTHALATE (PET)

In this example, separate signals on the same axes are plotted. The measurement method used was:

1. Data Off
2. Equilibrate 20.00 °C
3. Modulate Temperature ±0.318 °C for 60.0 s
4. Isothermal 10.0 min
5. Data On
6. Sample Interval 2.0 s/pt
7. Ramp 2.00 °C/min to 290.00 °C

As with Example 1, the data was exported into a JSON file for import into Python. Again, the first section of the script sets up the required libraries, imports the JSON file, and sets the dataframes for each step:

```
experiment = Experiment.from_json("mdsc - pet.json")

step_name, petmdsc_df = experiment.get_dataframes_by_step ("processed")

for idx, step_name in enumerate(step_name):
    print(f"{idx}\t{step_name}")
```

This script creates seven dataframes, but only the final dataframe (petmdsc_df[6]) contains numerical information. The plot code is like example 1, but calls different columns for each of the required signals:

```
fig, ax1 = plt.subplots(figsize=(10, 6))

ax1.set(xlabel='Temperature (°C)', ylabel='Heat Flow (W/g)')

ax1.plot(petmdsc_df[6]['Temperature / °C'], petmdsc_df[6]['Total Heat Flow (Normalized) / W/g'], label='Total Heat Flow', color='green')

ax1.plot(petmdsc_df[6]['Temperature / °C'], petmdsc_df[6]['Reversing Heat Flow (Normalized) / W/g'], label='Reversing Heat Flow', color='blue')

ax1.plot(petmdsc_df[6]['Temperature / °C'], petmdsc_df[6]['Non-Reversing Heat Flow (Normalized) / W/g'], label='Non-Reversing Heat Flow', color='red')

plt.legend()

plt.show()
```

As before, this script defines a plot with a size of 10 × 6 inches and labels for the axes. However, in this case, three different signals from the same dataframe are plotted on the same axis. The labels for each signal are noted in the legend, this is shown with the command "plt.legend()" and as before "plt.show()" displays the plot. This is shown in Figure 2.

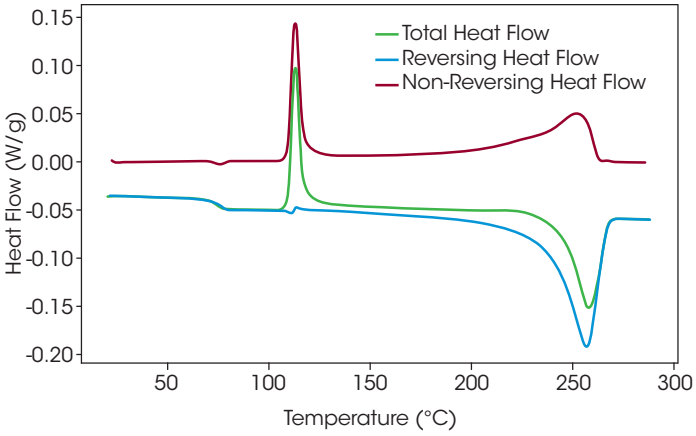

Figure 2. Modulated DSC results plotted from Example 2 script

PLOTTING EXAMPLE 3: HEAT-ISO MODULATED DSC OF POLYETHYLENE TEREPHTHALATE (PET) – SEPARATE AXES FOR EACH SIGNAL

With Modulated DSC™ data, it can be useful to view all signals on the same scale and on a maximum scale for each signal. This requires multiple y-axes but a common x-axis (temperature). The import of the JSON file and creation of the dataframe are the same as in Example 2. The visualization contains more detail and offsets the axes for visibility:

```
fig, ax1 = plt.subplots(figsize=(10, 6))

ax1.plot(petmdsc_df[6]['Temperature / °C'],
    petmdsc_df[6]['Heat Flow (Normalized) / W/g'],
    'red', label='Total Heat Flow (W/g)')
ax1.set_ylabel('Total Heat Flow (W/g)', color='red')
ax1.tick_params(axis='y', labelcolor='red')
ax1.spines['left']
ax1.yaxis.set_label_position('left')
ax1.yaxis.set_ticks_position('left')

ax2 = ax1.twinx()

ax2.plot(petmdsc_df[6]['Temperature / °C'],
    petmdsc_df[6]['Reversing Heat Flow (Normalized)
    / W/g'], 'blue', label='Reversing Heat Flow (W/g)')
ax2.set_ylabel('Reversing Heat Flow (W/g)',
    color='blue')
ax2.tick_params(axis='y', labelcolor='blue')
ax2.spines['left'].set_position(('outward', 60))
ax2.yaxis.set_label_position('left')
ax2.yaxis.set_ticks_position('left')

ax3 = ax1.twinx()

ax3.plot(petmdsc_df[6]['Temperature / °C'],
    petmdsc_df[6]['Non-Reversing Heat Flow (Normalized)
    / W/g'], 'green', label='Non-Reversing Heat
    Flow (W/g)')
ax3.set_ylabel('Non-Reversing Heat Flow (W/g)',
    color='green'
ax3.tick_params(axis='y', labelcolor='green')
ax3.spines['right']
ax3.yaxis.set_label_position('right')
ax3.yaxis.set_ticks_position('right')
plt.show()
```

This script defines multiple y-axes with a common x-axis, using color to differentiate the signals as shown in Figure 3.

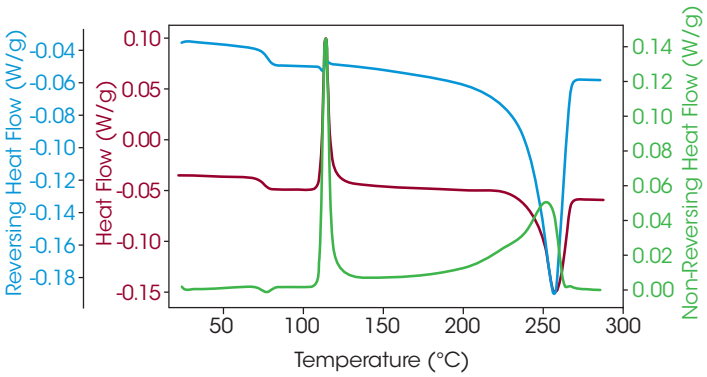

Figure 3. Modulated DSC data plotted with separate revering and non-reversing axes using Example 3 script

CONCLUSIONS

This technical brief looks at the options for plotting DSC data using the Python library Matplotlib.PyPlot. This use of Python allows consistent plotting for wide ranges of data which can be important when presenting data in publications.

REFERENCES

1. Matplotlib — Visualization with Python

ACKNOWLEDGMENTS

This technical brief was written by Philip Davies, Principal Applications Specialist, TA Instruments.

For more information or to request a product quote, please visit www.tainstruments.com to locate your local sales office information.