
plopy Documentation

Finnventor

Apr 11, 2021

Contents

1	Features	3
2	Installation	5
3	Usage	7
3.1	Parsing	7
3.2	Scripting	8

A program for making data plotting with matplotlib easily customizable in a GUI.

CHAPTER 1

Features

- Intelligent data file parsing
 - Automatically skips over headers (or any line that cannot be converted to `float` or `datetime`)
 - Supports many date formats in input automatically using `dateutil.parser`
 - Works with comma-separated and/or whitespace-separated input files
- Graph configuration: line style, color, points, legend
- Dialogs to create multiple axes and assign lines to them
- Supports all output formats supported by `matplotlib (.png, .jpg, .pdf ...)`

CHAPTER 2

Installation

After installing [Python 3](#), run the command `pip install plop`

You can also [try it online](#) on repl.it, but that loads slowly and is not recommended for normal use.

To open the GUI, run the command `python -m plopy`, or also preselect files by appending their paths to the command (`python -m plopy data.csv log.txt ...`).

`plopy` can also be controlled from a script. Use `plopy.add_file` and `plopy.add_array`, then call `plopy.start()`

If you're already accustomed to `matplotlib`, or want to convert pre-existing programs, you can import `plopy.fig` and `plopy.ax` to plot on, and call `plopy.start()` when done. This method also works in combination with the previous one.

See *Scripting* for more examples.

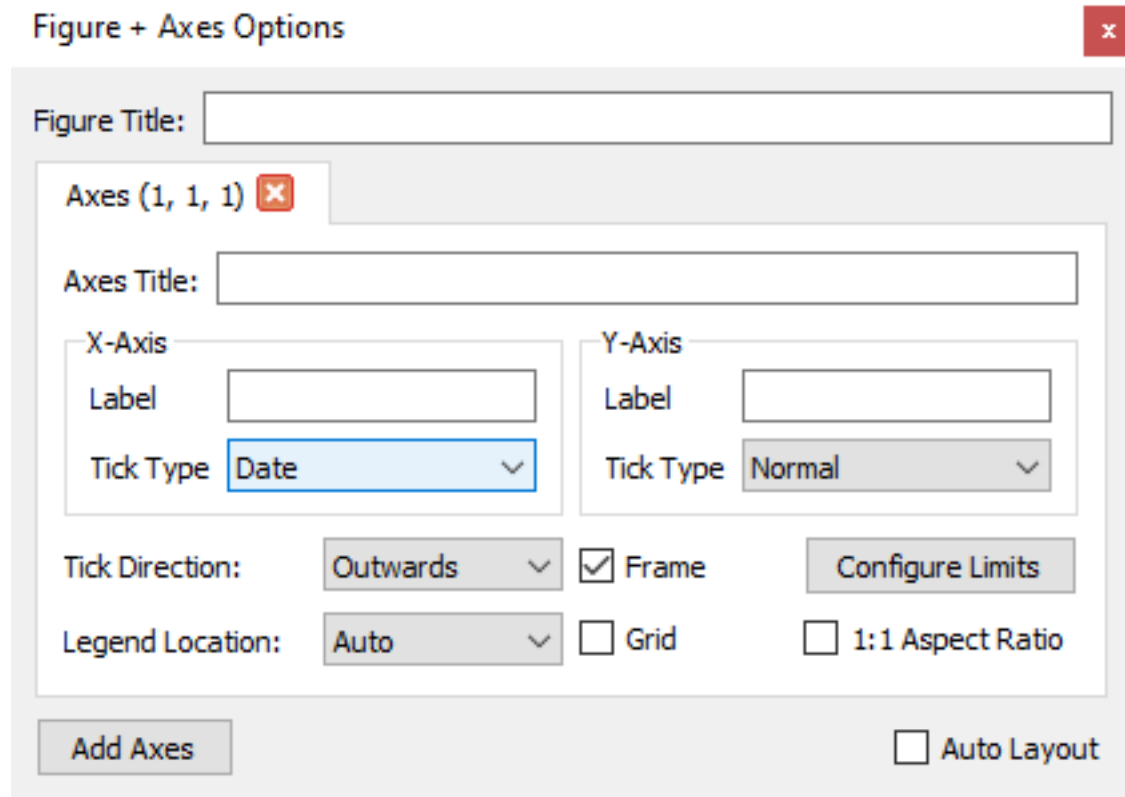
3.1 Parsing

`plopy` loops over the lines in a file, splitting lines into columns by spaces or commas. If every column can be converted to a value, the values are added to an array. Then, if the array has at least one row and the number of columns does not change from row to row, the file is plotted.

This process will work on files where columns are delimited by spaces, commas, or a combination of both, and will automatically skip headers or other lines that cannot be parsed.

3.1.1 Dates

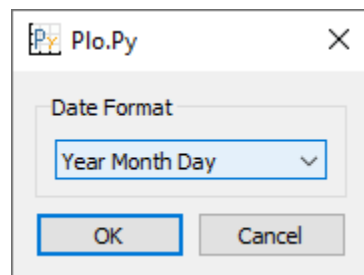
`plopy` will attempt to parse dates automatically with `dateutil.parser` if parsing as float is not successful. This will parse ISO-formatted dates with any delimiter between them, except spaces or commas (since that will result in parts of the date being treated as separate columns). If there are spaces in the date, make sure the file is in `.csv` format. Dates are then converted into `matplotlib`'s representation of dates as the number of days since the year 0. To display this as a proper date, set the tick type to `Date` in `Edit > Figure + Axes Options`. Note that this won't work if negative numbers are included in the column that is set to be displayed as dates.



The dialog box is titled "Figure + Axes Options" and has a red close button in the top right corner. It contains the following fields and controls:

- Figure Title:** A text input field.
- Axes (1, 1, 1)** with a red close button.
- Axes Title:** A text input field.
- X-Axis:**
 - Label:** A text input field.
 - Tick Type:** A dropdown menu currently showing "Date".
- Y-Axis:**
 - Label:** A text input field.
 - Tick Type:** A dropdown menu currently showing "Normal".
- Tick Direction:** A dropdown menu showing "Outwards".
- Frame:** A checked checkbox.
- Configure Limits:** A button.
- Legend Location:** A dropdown menu showing "Auto".
- Grid:** An unchecked checkbox.
- 1:1 Aspect Ratio:** An unchecked checkbox.
- Add Axes:** A button at the bottom left.
- Auto Layout:** An unchecked checkbox at the bottom right.

If the file's dates aren't in ISO 8601 format, you can choose a different format in `File > Choose Input Date Format`.



The dialog box is titled "PloPy" and has a close button in the top right corner. It contains the following fields and controls:

- Date Format:** A dropdown menu currently showing "Year Month Day".
- OK:** A button.
- Cancel:** A button.

When working with very large files that take a significant time to load, consider using the `Load Input File with Fast Date Processing...` option, which uses `datetime.datetime.strptime`. It requires manually specifying the format, such as `%Y-%m-%dT%H:%M:%S`, but can run significantly faster (31 vs 142 seconds on a 86 MB file). A full list of format codes can be found [here](#).

3.2 Scripting

```
>>> from plop import add_file, add_array, start
>>> add_array([(0, 0), (1, 2), (2, -2), (3, 2), (4, -2), (5, 0)], "zigzag")
True
>>> add_file("test-data.txt")
True
>>> add_file("nonexistent.csv")
[PloPy]: 'nonexistent.csv' is not a file.
```

(continues on next page)

(continued from previous page)

```
False  
>>> start()
```

