pylustrator: code generation for reproducible figures for publication

Richard Gerum[1]

1 Department of Physics, University of Erlangen-Nürnberg, Germany

Background

In recent years, more and more researchers have called attention to a growing "reproducibility crisis" (Sayre & Riegelman, 2018). An important factor that contributes to problems in reproducing results from published studies is the unavailability of the raw data from the original experiment as well as the unavailability of the methods or the code used for the evaluation the raw data (Baker & Penny, 2016). One major step to overcome these shortcomings is the publication of all raw data as well as a documented version of the code used for evaluation (Baker, 2016). The ideal case would be that anyone interested can download the raw data and exactly reproduce the figures of the publication.

To address the issue of data availability, researchers are encouraged to provide their data in online repositories, e.g. dryad ("Dryad," n.d.). However, this data is useless unless the complete evaluation procedure in the terms of all evaluation and visualisation steps can be comprehended by other scientists. The best way to to so is to provide a complete well documented evaluation code, including all important steps from basic artifact corrections up to the final plot to be published. Open Source scripting languages like Python (Van Rossum & Drake Jr, 1995) or R (R Core Team, 2019) are ideal for such code as open source languages are accessible for everyone. Furthermore, interpreted languages do not need to be compiled, therefore have less obstacles for the user to run the code. The last part of the evaluation of the data is the visualisation, which is crucial to communicate the results (Tufte, 1893). This paper deals with the visualization step which consists of two parts: the generation of simple plots from data and composing meaningful figures from these plots.

The first part of generating the building blocks of figures, the plots, is already covered in various toolkits, e.g. Matplotlib (Hunter, 2007), Bokeh (Bokeh Development Team, 2019) or Seaborn (Waskom et al., 2017). But to generate reproducible figures from simple plots, no convenient toolkit is yet available. Matplotlib already offers figures composed of multiple subplots, but to generate a complete figure ready for publication a lot of code is needed to add all formatting, annotations and styling commands. Therefore, this approach is often not followed as it is impractical for real applications. Users often prefer graphical tools such as image manipulation software, e.g. GIMP (GIMP Development Team, 2019) or Inkscape (Team, 2019). These offer great flexibility, but cannot provide a reproducible way of generating figures and bear the danger of accidentally changing data points. Also important to note is that by using an image manipulation software, any small change to the evaluation requires to re-edit the figure in the image manipulation software. A process that slows down the creation of figures and is prone to errors.

Algorithm and Exemplary Results

`pylustrator` was developed to address this issue. A tool to fill the gap from single plots to complete figures, by a code generation algorithm, that turns user input into python code

for reproducible figure assembly (Fig.~**??**). Small changes to the evaluation or new data only require to run the code again to update the figure.


```
#% start: automatic generated code from pylustrator
plt.figure(1).set_size_inches(8.000000/2.54, 8.000000/2.54, forward=True)
plt.figure(1).axes[0].set_position([0.191879, 0.148168, 0.798133, 0.742010])
plt.figure(1).axes[0].set_xlabel("data x")
plt.figure(1).axes[0].set_ylabel("data y")
plt.figure(1).axes[1].set_position([0.375743, 0.603616, 0.339534, 0.248372])
plt.figure(1).axes[1].set_xlabel("data x")
plt.figure(1).axes[1].set_ylabel("data y")
plt.figure(1).axes[1].set_ylim(-40.0, 90.0)
#% end: automatic generated code from pylustrator
```
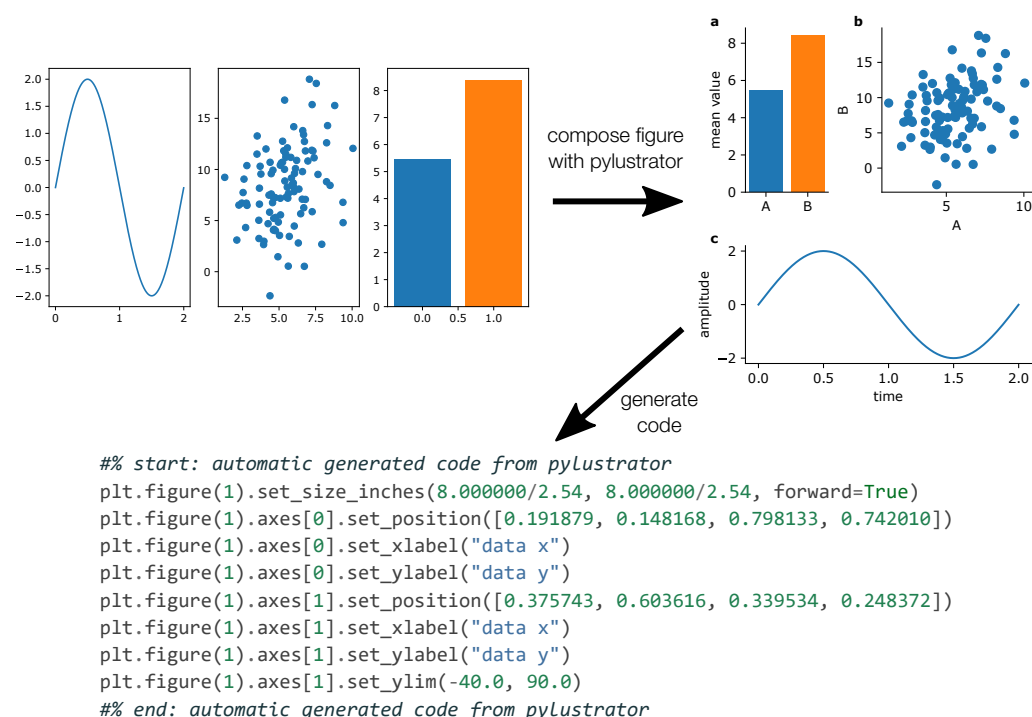
Figure 1: Example how code for composing a figure can be generated with pylustrator.

Using `pylustrator` in any given Python file that uses Matplotlib do plot data, simply requires the addition of only two lines of code:

```
import pylustrator
pylustrator.start()
```

The Matplotlib figure is then displayed in an interactive window (Fig. 2) when the `plt.show()` command is called. In this interactive window, `pylustrator` enables the user to:

- resize and position plots by mouse-dragging
- adjust the position of plots legends
- align elements easily by automatic "snapping"
- resize the complete figure in cm/inches
- add text and annotations, and change their style and color
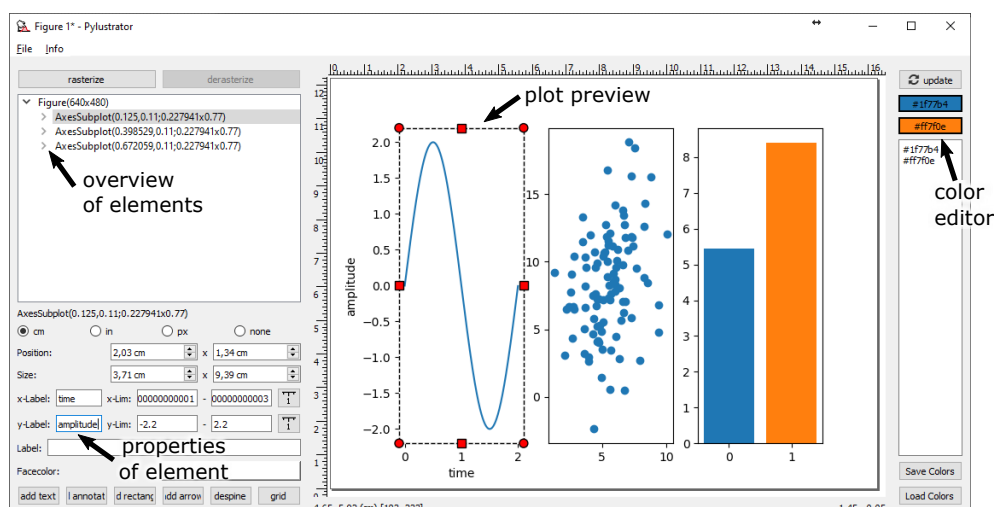- adjust plot ticks and tick labels

Figure 2: The interface of `pylustrator`. The user can view the elements of the plot, edit their properties, edit them in the plot preview and experiment with different color schemes.

`pylustrator` tracks all these changes to translate them into python code. Every change is split in four parts: the command object, the command text, the target object and the target command. The command object is the object instance (e.g. the Axes object) that has a method to call for this change and the command text is the methods name together with the parameters (e.g. ".annotate('New Annotation')"). The target object is the object that is affected by the command. In most cases this is the same as the command object, but in some cases when new child objects are created the target object is the child object. The target command is the methods name without the parameters.

Command objects are "serialized" by iteratively going up the parent-child tree from e.g. a text to the axis to the figure and generating a python command from this dependency (e.g. `plt.figure(1).axes[0].texts[0]`, the first text of the first axes of figure 1). When saving, `pylustrator` introspects its current stack to find the line of code from where it was called and inserts the automatically generated code directly before the command calling `pylustrator`.

When loading a file with automatically generated code, `pylustrator` splits all the automatically generated lines into reference objects and reference commands. New changes where both the reference object and the reference command match a previous change, the previous change is overwritten. This ensures that previously generated code can be loaded appropriately and saving the same figure multiple times generates the code only once.

It is important to note that the automatically generated code only relies on Matplotlib and does not need the `pylustrator` package anymore. Thus, the `pylustrator` import can later be removed to allow to share the code without an additional introduced dependency.

The documentation of `pylustrator` can be found on https://pylustrator.readthedocs.org.

Conclusion

This study introduces a novel method to create publishable figures from single plots based on an open source Python library called `pylustrator`. The figures can be arranged by drag and drop and the `pylustrator` library produces the according code. Thus, this library provides a valuable contribution to tackle the reproducibility crisis.

Gerum, (2020). pylustrator: code generation for reproducible figures for publication. *Journal of Open Source Software*, 5(45), 1989. https://doi.org/10.21105/joss.01989

Acknowledgements

References

Baker, M. (2016). Why scientists must share their research code. *Nature News*. doi:10.1038/nature.2016.20504

Baker, M., & Penny, D. (2016, May). Is there a reproducibility crisis? doi:10.1038/533452A

Bokeh Development Team. (2019). *Bokeh: Python library for interactive visualization*. Retrieved from https://bokeh.org/

Dryad. (n.d.). https://datadryad.org.

GIMP Development Team. (2019). *GIMP:GNU image manipulation program*. Retrieved from https://gimp.org/

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

R Core Team. (2019). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org

Sayre, F., & Riegelman, A. (2018). The reproducibility crisis and academic libraries. *College & Research Libraries*, *79*(1), 2. doi:10.5860/crl.79.1.2

Team, T. I. (2019). *Inkscape*. Retrieved from https://inkscape.org/

Tufte, E. (1893). *The visual display of quantitative information*. Cheshire, Connecticut: Graphics Press.

Van Rossum, G., & Drake Jr, F. L. (1995). *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., et al. (2017, September). Mwaskom/seaborn: V0.8.1 (september 2017). doi:10.5281/zenodo.883859