The Journal of Open Source Software

Gramm: grammar of graphics plotting in Matlab

Pierre Morel[1]

1 German Primate Center, Göttingen, Germany

DOI: 10.21105/joss.00568

Software

- Review
- Repository
- Archive

Submitted: 31 January 2018
Published: 06 March 2018

Summary

Gramm is a data visualization toolbox for Matlab (The MathWorks Inc., Natick, USA) that allows to produce publication-quality plots from grouped data easily and flexibly. Matlab can be used for complex data analysis using a high-level interface: it supports mixed-type tabular data via tables, provides statistical functions that accept these tables as arguments, and allows users to adopt a split-apply-combine approach (Wickham 2011) with `rowfun()`. However, the standard plotting functionality in Matlab is mostly low-level, allowing to create axes in figure windows and draw geometric primitives (lines, points, patches) or simple statistical visualizations (histograms, boxplots) from numerical array data. Producing complex plots from grouped data thus requires iterating over the various groups in order to make successive statistical computations and low-level draw calls, all the while handling axis and color generation in order to visually separate data by groups. The corresponding code is often long, not easily reusable, and makes exploring alternative plot designs tedious (Example code Fig. 1A).

Inspired by ggplot2 (Wickham 2009), the R implementation of "grammar of graphics" principles (Wilkinson 1999), gramm improves Matlab's plotting functionality, allowing to generate complex figures using high-level object-oriented code (Example code Figure 1B). Gramm has been used in several publications in the field of neuroscience, from human psychophysics (Morel, Ulbrich, and Gail 2017), to electrophysiology (Morel et al. 2016; Ferrea et al. 2017), human functional imaging (Wan et al. 2017) and animal training (Berger et al. 2017).

Gramm use and features

The generation of a figure with gramm includes the following steps, which typically take few lines of code (Example gramm code Fig. 1B). In a first line, the user creates an instance of a gramm object by providing the constructor with the variables necessary for the plot: x and y values as well as grouping variables. Each grouping variable can simply be associated with a given plot feature using named arguments: color hue and lightness, sub-axis rows and columns, marker and line type and size. In the example Fig 1B, the two grouping variables are associated with sub-axis columns and color hues. After this initialization, the user can add graphical layers to the figure by making calls to corresponding methods of the newly created object. `geom_` methods are used to add individual layers that represent the x/y data directly, such as points, lines, bars, rasters, etc. `stat_` methods are used to add statistical visualization layers, some of which reflecting built-in Matlab statistics functions, others extending Matlab's capabilities: histograms, linear or non-linear fits, density estimates, statistical summaries, spline smoothing, violin plots, ellipse fits, etc. Most geom_ and stat_ methods can take optional arguments for customization of the layer. At this stage, the user can also call optional `set_` methods in order to fine tune the figure design (colormaps, legends, ordering, titles, etc.). The actual plotting is made when the user finally calls the `draw()` method.

The Journal of Open Source Software

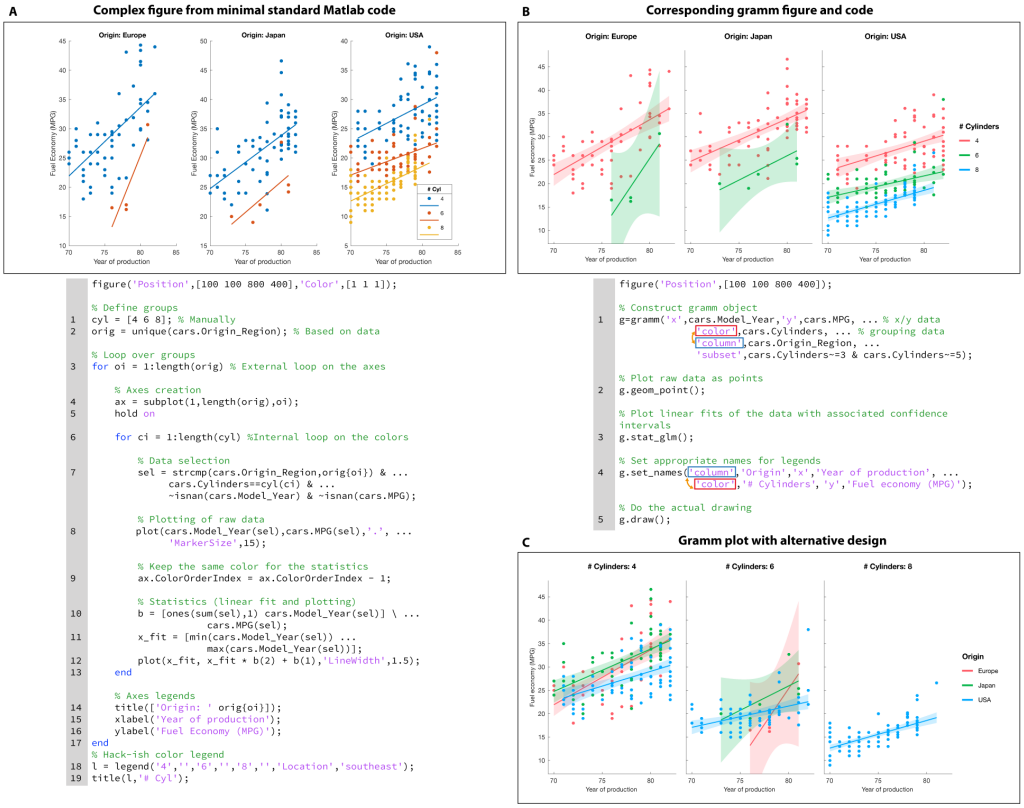

Figure 1: Comparison of default Matlab plotting and gramm plotting with corresponding code. A. Grouped data Matlab plot generated using minimal code. Note the absence of confidence interval representations, the placement and aspect of the color legend, the non-matching axes limits, as well as the improper fit of content to the figure window width (the black outline represents the figure window edges). B. Gramm plot and corresponding code. C. Gramm plot with alternative design resulting from swapping the arguments highlighted with red and blue boxes in panel B

Once the `draw()` call is issued and the figure plotted, some interactions with the gramm object and plot are still possible. The user can retrieve the results of the statistical computations from the gramm object as well as the handles of the graphic primitives making up the plot, which allows further customization of individual plot elements. Plots with different data or different groups can also be superimposed on the existing plot using the `update()` method. Coding in this case works the same as if a new gramm object was created.

Multiple gramm plots can be combined in the same figure window simply by creating a 2D array of gramm objects and calling the `draw` method on the whole array. The plots are then placed in the figure window according to their indices in the array.

Detailed documentation on the use of gramm is available through the built-in Matlab help navigator with `doc gramm`, but a quick overview of all methods and their arguments can be found in the "gramm cheat sheet.pdf" file. Typical examples that demonstrate the various ways of using gramm as well as some use tips can be found in "examples.m", the output of which is published as an html page in "html/examples.html".

Comparison with standard Matlab code

The code necessary to generate the example plot of Fig. 1B with gramm is more than three times shorter than the minimal standard Matlab code that would generate a conceptually similar plot (Fig. 1A). Moreover, the standard Matlab plot is missing several features that are built in gramm and would require adding more complex code, such as proper common color legends, matching scales across axes, shaded areas for confidence bounds or advanced dynamic content fitting to figure window size. Last, this simple example makes apparent the flexibility of gramm: exchanging which group is represented by sub-plots and which group by colors in order to produce the plot in Fig. 1C only requires argument name changes (red and blue boxes Fig. 1B), while standard Matlab code requires for-loops to be swapped, axis creation code to be modified, and the legend code to be completely reworked (Fig. 1A lines 3, 4, 6, 14, 15, 16, 18 and 19).

Comparison with other packages

To my knowledge, gramm is the only implementation of this kind and breadth for Matlab, but two implementations of grammar of graphics' principles are available for other popular scientific languages: ggplot2 (Wickham 2009) for R, and Seaborn (Waskom et al. 2017) for python. Beyond obvious interface differences due to the different languages, a few specificities are noteworthy.

As a main conceptual difference between gramm and the two aforementioned implementations, gramm does not acccept data input within the layer creation methods. Thus, gramm provides a more rigid framework with the constructor and the `update()` method, as presented above. Compared to ggplot2, gramm can better handle repeated time series. These are very common in scientific fields where Matlab is widely used, such as neuroscience. With repeated time series, R/ggplot2 requires a memory-expensive "long" table (`n_repetitions * n_samples` rows) that contains not only x/y data but also the groups. On the other hand, gramm can handle 2D arrays or ragged-array (cell) input for the x/y data (`n_repetitions` rows; `n_samples` columns), and use less memory demanding arrays with `n_repetitions` rows for the groups. When the data is provided in this optimized way to gramm, some `geom_` and `stat_` functions adjust their behavior in order to account for the data struture. Seaborn has a basic support for repeated trajectory-like data, with the function `tsplot()`.

In terms of layer and data type support, gramm does not support geographical map representations (ggplot2), surface representations (ggplot2, seaborn), or clustermap data (seaborn). However, gramm supports 3D plots with line and point layers, can produce dedicated plots for repeated stochastic event time series (like spike raster plots that are frequently used in neuroscience), and is able to fit and represent confidence ellipses/ellipsoids for 2D and 3D point cloud data.

References

Berger, Michael, Antonino Calapai, Valeska Stephan, Michael Niessing, Leonore Burchardt, Alexander Gail, and Stefan Treue. 2017. "Standardized Automated Training of Rhesus Monkeys for Neuroscience Research in Their Housing Environment." *Journal of Neurophysiology* Advance online publication. https://doi.org/10.1152/jn.00614.2017.

Ferrea, Enrico, Lalitta Suriya-Arunroj, Dirk Hoehl, Uwe Thomas, and Alexander Gail. 2017. "Implantable Computer-Controlled Adaptive Multi-Electrode Positioning System

(Amep)." *Journal of Neurophysiology* Advance online publication. https://doi.org/10.1152/jn.00504.2017.

Morel, Pierre, Enrico Ferrea, Bahareh Taghizadeh-Sarshouri, Josep Marcel Cardona Audí, Roman Ruff, Klaus-Peter Hoffmann, Sören Lewis, et al. 2016. "Long-Term Decoding of Movement Force and Direction with a Wireless Myoelectric Implant." *Journal of Neural Engineering* 13 (1):016002. https://doi.org/10.1088/1741-2560/13/1/016002.

Morel, Pierre, Philipp Ulbrich, and Alexander Gail. 2017. "What Makes a Reach Movement Effortful? Physical Effort Discounting Supports Common Minimization Principles in Decision Making and Motor Control." *PLOS Biology* 15 (6). Public Library of Science:1–23. https://doi.org/10.1371/journal.pbio.2001323.

Wan, Nick, Allison S. Hancock, Todd K. Moon, and Ronald B. Gillam. 2017. "A Functional Near-Infrared Spectroscopic Investigation of Speech Production During Reading." *Human Brain Mapping* Advance online publication. https://doi.org/10.1002/hbm.23932.

Waskom, Michael, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, et al. 2017. "Mwaskom/Seaborn: V0.8.1 (September 2017)." https://doi.org/10.5281/zenodo.883859.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.

———. 2011. "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software, Articles* 40 (1):1–29. https://doi.org/10.18637/jss.v040.i01.

Wilkinson, Leland. 1999. *The Grammar of Graphics*. New York, NY, USA: Springer-Verlag New York, Inc. https://doi.org/10.1007/978-1-4757-3100-2.