Open Source Programming for Interpreted Language: Graphic Interface and Macro Bridging Interface in the EEGLAB software

Arnaud Delorme[1,2,3], Scott Makeig[1]
1. Swartz Center for computational Neuroscience
2. Université de Toulouse, UPS, Centre de Recherche Cerveau et Cognition, Toulouse, France,
3. CNRS, CerCo, Toulouse, France
arno@ucsd.edu

Abstract

Interpreted languages like Matlab or Python are popular in the Open Source community. Not-only do these software environments offer the possibility of developing comprehensive graphic interfaces but they also contain nearly unlimited scripting capabilities for automating procedures. We describe the GIMBI framework (Graphic Interface and Macro Bridging Interface) for easily bridging graphic interface functions with automated macro scripting in an interpreted language. We illustrate this method using the open source EEGLAB software we have developed, which is currently the most widely used public software for processing electrophysiological data.

1. Introduction

Interpreted languages are gaining momentum in the scientific community [1, 2]. One reason is that they allow increased flexibility compared to compiled languages at minimal cost in terms of speed of execution. Languages such as Matlab or Python that allow for designing graphic interfaces also offer unique capabilities for integrating graphic interface with macro scripting.

In most software, pure menu-based navigation must usually be accompanied by macro features to automate procedures. Such macro features are usually implemented by allowing users to use commands either directly on the command line entry of the interpreted language or through a dedicated mechanism implemented in the software. This interaction between graphical interface and command line scripting can, however, be optimized to allow users to fully benefit from using an interpreted language.

Here we describe an elegant way to integrate graphic interface and macro scripting which we termed GIMBI framework. In this framework, users typically invoke data processing functions by selecting menu items from a main window menu. These menus call hybrid functions that handle both a graphic interface and signal processing. When called without (or with minimal) arguments (as from the main graphic user interface), these re-entrant standalone functions query users to gather additional parameters from users. When called with all necessary parameters, these functions will not query users and may be used directly from the interpreted language command line or from scripts. These hybrid functions also return a line of script code that can be re-used to perform the same processing (including parameters entered via the query window into the function call). This functionality is described in detail in this article and greatly facilitates transition from menu-based use to basic scripting for users.

We believe that that use of the GIMBI framework that builds re-usable scripts as users navigate through menus, and the capacity to ease the transition from graphic interface use to use of a powerful scripting environment are some of the factors key to the wide acceptance of Matlab-based EEGLAB our open source platform. Below we provide more detail about how we have implemented the GIMBI framework in this environment.

2. GIMBI implementation in EEGLAB

EEGLAB [3] is an interactive menu- and scripting-based environment for processing electrophysiological data based under the interpreted

programming language Matlab (The Mathworks, Inc.). EEGLAB provides an interactive graphic user interface (GUI) allowing users to flexibly and interactively process their high-density electrophysiological data (from up to several hundred electrodes) or other time series data. It makes available many standard methods of electroencephalographic data analysis as well as tools for independent component analysis (ICA) [4] and time/frequency analysis developed by our group. EEGLAB helps researchers process their biophysical time series and share new techniques; a variety of user-developed methods (22 to date) have been implemented as plug-ins.

The architecture of the EEGLAB environment illustrates how the GIMBI framework is implemented. EEGLAB is a collection of about 400 stand-alone modular re-entrant functions. These functions may call each other to process data in

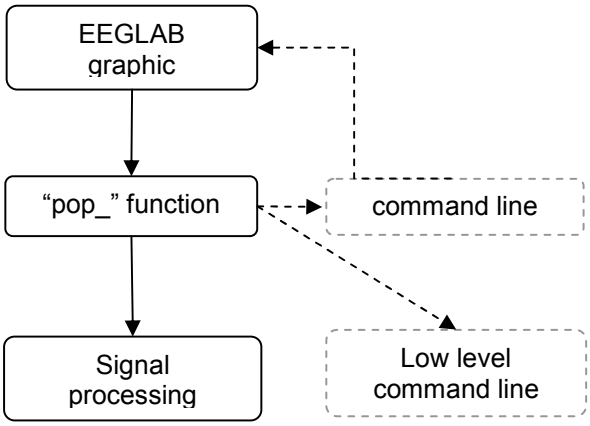


Figure 1: different function levels of the GIMBI framework implemented in EEGLAB. Dashed lines and boxes indicate text input/outputs.

various ways. However each function is stand-alone in the sense that it is programmed to perform a given task on given input data structures. Each function has its own specification and documentation and may be used separately by end users. Each function is also re-entrant which means that no global variable or global file definition is being modified by the function, so it does not alter the processing of other functions in any hidden or surprising way.

EEGLAB was designed for use by both novice and expert Matlab users. The GIMBI framework was implemented in EEGLAB to allow for a smooth transition from menu-based calling to macro command scripting. Novice users can interact exclusively with the EEGLAB graphic interface (GUI); intermediate users can also invoke EEGLAB functions directly from the Matlab command line. More advanced users can write their own Matlab scripts using modular EEGLAB functions and data structures. EEGLAB functions may be grouped into three layers according to different levels of usage:

1) *The main "eeglab()" graphic interface function and its almost 100 menu handlers:* EEGLAB users typically call data processing functions (*pop_functions,* below) by selecting menu items from the main EEGLAB window menu.

2) *Pop_functions.* Matlab functions with their own graphic interfaces. Called without or with minimal arguments, these functions display a query window to gather additional parameter choices from users. They then generally call one or more of the EEGLAB signal processing functions (below). When called with all essential parameters, *pop_functions* will not create a query window and may thus be used directly from the Matlab command line or in Matlab scripts for batch processing. Called from the EEGLAB menu, *pop_functions* also offer the advantage of returning a line of Matlab code that will perform the same processing as the menu call (including any parameters entered in the query window).

3) *Signal processing functions*. The experienced Matlab user can call low-level EEGLAB functions directly from the Matlab command line or from their own analysis scripts. These functions directly perform signal processing on EEG data for computing data spectrum, computing statistics, or any other time-series manipulation. Low-level signal processing functions do not require a specific input data structure or object. These could be standard Matlab function that take Matlab matrices as input, or custom functions tailored to process EEG data.

We will illustrate using this architecture to change the sampling rate of the EEG signal. The Matlab function *resample(),* for instance, (a native Matlab function) is used to change the sampling rate of the EEG data. A *pop_resample()* function would take the EEG dataset structure as input and pass its data on to the *resample()* function. In our implementation, all *pop_functions* take as input an object called EEG that contains all information for an EEG dataset (the data, sampling rate, channel labels and positions, etc…). Data in fields of in this structure/object would be used by *pop_resample()* to call low-level signal processing functions, e.g. *resample()*. The *pop_resample()* function would ensure conversion from the software

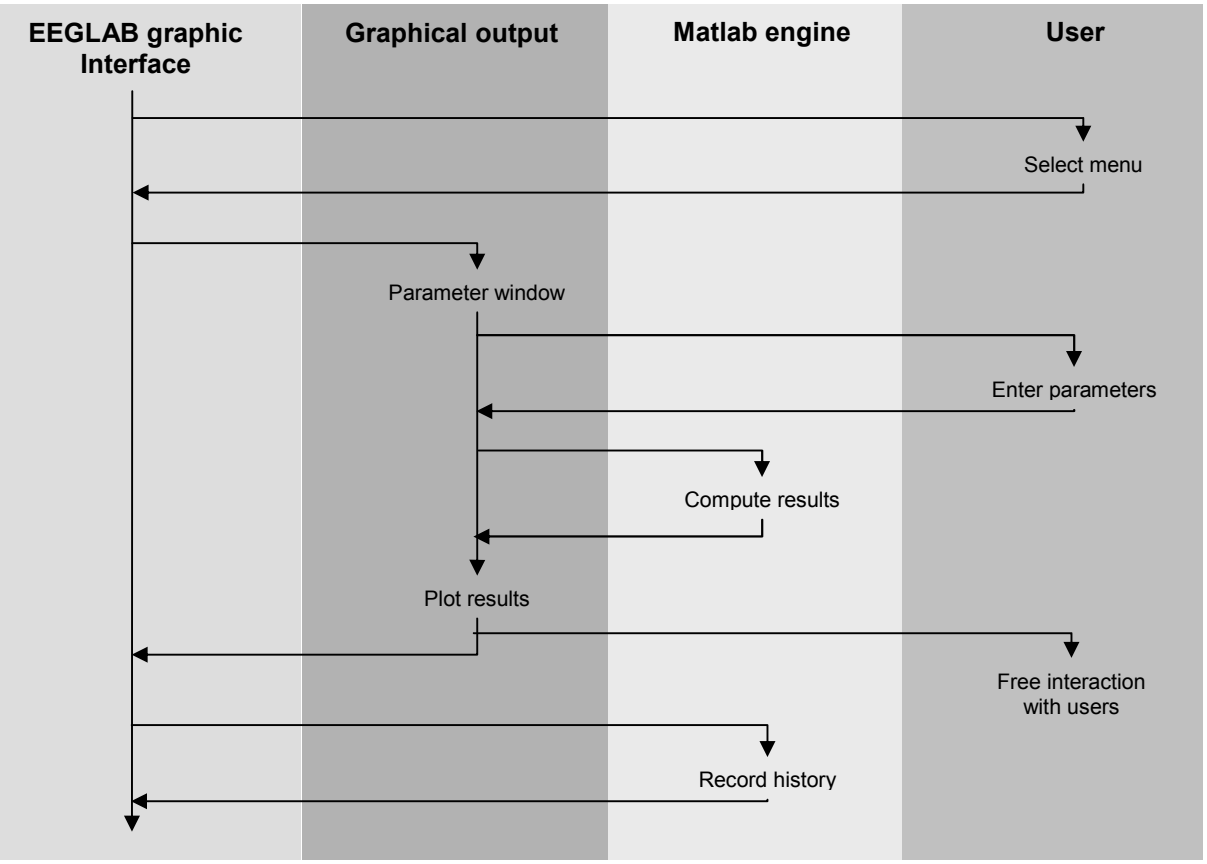
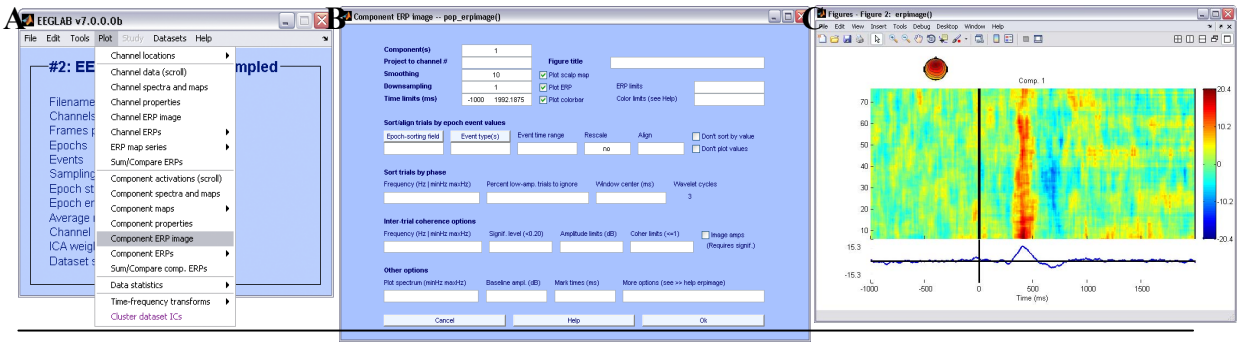


Figure 2: action diagram indicating the flow of event when calling an EEGLAB menu.

data structure to the input format of the *resample()* function. It would also ensure that its output is formatted to fit in the software main data structure. Called with the EEG object as its only input, it will query the user for an additional parameter via a graphic interface (e.g., the new sampling frequency, say 128 Hz) and will then call the low-level resampling function *resample()*. Once the processing is terminated, the function will return the modified EEG dataset structure containing data sampled at the new sampling rate. *pop_resample()* would also return a Matlab command, e.g., "*EEG = pop_resample(EEG, 128)*" that if executed would reproduce the re-sampling process. The EEGLAB software main graphic interface automatically stores this command in a global history variable.

In Figures 2 and 3 we illustrate the process for a function that plots single-trial EEG activities. The user first selects a menu in the graphic interface (in this case the single-trial plotting menu as shown in Fig. 3A). A GUI window then pops up querying for additional parameters (in this case the *pop_erpimage()* function GUI in Fig. 3B). Once the user has entered the missing parameters, the data is being processed by the Matlab compute engine (using the lower-level *erpimage()* function) returns a plotted image depicting single-trial activities (Fig. 3C) and text output describing the processing being performed (Fig. 3D). Note that this text output contains the low level command call that was used to plot the function. This text may be copied and pasted to the command line but is not registered in the software. The *pop_erpimage()* function [5] also returns a line of script (for a complete call to itself) which is passed on to the main EEGLAB graphic interface and added to the software history (Fig. 3E).

Figure 2 shows the action diagrams representing the flow of information and user interaction. The user plays a role at three levels. He/she first calls the *pop_function* via the menu in the EEGLAB graphic interface. He/she then enters parameters in the *pop_function* GUI, and finally she/he interacts with the plotted output from the function (a number of Matlab features are immediately available for this plotted figure such as zooming and printing).



D

Command executed by pop_erpimage:

erpimage(EEG.icaact([1], :), ones(1, EEG.trials)*EEG.xmax*1000, linspace(EEG.xmin*1000, EEG.xmax*1000, EEG.pnts), 'Comp. 1', 10, 1 , 'yerplabel','', 'erp', 'on', 'cbar', 'on','topo', { mean(EEG.icawinv(:,[1]),2) EEG.chanlocs EEG.chaninfo });

Plotting input data as 80 epochs of 384 frames sampled at 128.0 Hz.
Sorting data on input sortvar.
Smoothing the sorted epochs with a 10-epoch moving window. and a decimation factor of 1
Output data will be 384 frames by 71 smoothed trials.
Outtrials: 6.00 to 76.00
The caxis range will be the sym. abs. data range -> [-20.3655,20.3655].
Data will be plotted between -1000 and 1992.19 ms.
Overplotting sorted sortvar on data.
Plotting the ERP trace below the ERP image
Plotting a topo map in upper left.
Done.

E

figure; pop_erpimage(EEG,0, [1],[],'Comp. 1',10,1,{},[]," , 'yerplabel','', 'erp', 'on', 'cbar', 'on','topo', { mean(EEG.icawinv(:,[1]),2) EEG.chanlocs EEG.chaninfo });

Figure 3. Graphical and text output for plotting single trial EEG images. A. Menu call (grayed). B. ERPIMAGE input parameter window. C. Output image. D. Output text providing information to the user as well as a low-level

3. History management

All menu interactions generate a line of script command that is recorded in a history variable. In the EEGLAB software, several menu items are dedicated to handling history information. There are two types of history: history linked to the current dataset being processed and history linked to handling different datasets. Since datasets may be loaded, merged, split and processed in many different ways. it is important for the user to keep track of changes involving multiple datasets via the session history. However such changes are not specific to single datasets and could confuse novice users. For novice users, it is easier to use the dataset history that is directly attached to a given dataset (for instance, recording that this dataset was imported from a binary file on disk, then it was preprocessed, and then plotted in some way) as shown in Figure 4.

The data set's own history is stored in a field within the dataset structure and may be saved separately using a dedicated EEGLAB menu item. Global history is stored independently of datasets and may also be saved in a separate file. Figure 4 illustrates a basic script (here, the actual history script has been made simpler for illustrative purposes, but the code shown is still functional). In (1), the data are being loaded from the hard drive. In (2), the data are being re-sampled to 128 Hz. In (3) the continuous data are being "sliced" into portions of data time-locked to specific events (in this case the "stimulus" events). In (4) the baseline is removed (the average of the portion of data preceding the time-locking experimental events). In (5) the data is plotted using the single-trial plotting function (the same function used in Fig. 3, although some parameters here assume default values).

EEG = pop_loadset('filename','dataset.set');

EEG = pop_resample(EEG, 128);

EEG = pop_epoch(EEG, { 'stimulus' }, [-0.2 0.8]);

EEG = pop_resample(EEG, [-200 0]);

figure; pop_erpimage(EEG,0, [1],[],'Comp. 1');

Figure 4. Example of simple history sequence.

Note that, we programmed all of these *pop_functions* and signal processing functions such that most of parameters are optional; listed default

values are used when a necessary parameter is not specified, either in the graphic interface or on the command line call.

To re-use saved scripts, users may simply enter their name on the Matlab command line (assuming they were saved with the .m file extension). Doing this will repeat all processing that had been performed via GUI interactions, although here no GUI will appear; instead, the parameters entered by the user via the GUI will be used). Users may also modify such scripts by, for example, changing the name of the subject input file. This is an easy way to process another subject's data using all the parameter used for a first subject. This is especially relevant for EEG research where the same pre-processing must typically be repeated for each subject's data. Users may also embed the saved history scripts into custom loops (for instance to process large groups of subjects in several conditions) and may easily integrate such history code in other custom scripts and functions.

3. Conclusion

The GIMBI framework we have described greatly facilitates transition from menu use to basic scripting. Users may process a dataset using menu commands, We have developed what we believe to be an elegant framework for automatically producing scripts to reproduce users' actions. Users may then automate processing on data from other subjects or conditions using the saved scripts. We believe that this GIMBI framework is one of the key factors in the world-wide success of the EEGLAB software environment.

[1] Ousterhout, K., *Scripting: higher level programming of the 21st century.* IEEE computer, 1998(March).

[2] Fogel, K., *Producing Open Source Software: How to Run a Successful Free Software Project*. 2005: O'Reilly Media, Inc.

[3] Delorme, A. and S. Makeig, *EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis.* J Neurosci Methods, 2004. 134(1): p. 9-21.

[4] Makeig, S., et al., *Independent component analysis of electroencephalographic data*, in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Editors. 1996. p. 145-151.

[5] Jung, T., et al., *Analysis and visualization of single-trial event-related potentials.* Hum Brain Mapp, 2001. 14(3): p. 166-85.