# Visbrain : hardware-accelerated visualization suite for neuroscientific data in Python

Etienne Combrisson[1, 2], [...], Aymeric Guillot, Karim Jerbi[1,2, 3]

[1] Psychology department, Université de Montréal, Canada
[2] Lyon 1 University, Lyon, France
[3] DYCOG Lab, Lyon Neuroscience Research Center, Lyon, France

**Corresponding authors:**
Etienne Combrisson (e.combrisson@gmail.com)

# Abstract

We present a Python open-source package called Visbrain which proposes a coherent visualization suite for neuroscientific data. The current version of Visbrain is articulated around four modules respectively dedicated to 1) data mining and basic plotting functions (*NdViz* module), 2) visualizations on a 3D standard MNI brain (*Brain* module), 3) polysomnographic data visualization and sleep analysis (*Sleep* module) and finally, 3) a module for page layout and export of high-quality figures (*Figure* module). The three first modules come with modulars and powerful graphical user interfaces built with PyQt. Each module has been developped in collaboration with neuroscientifics and specialists of the field and provides therefore a coherent and exhaustive set of functionalities. Visbrain is developed on top of VisPy which provides high performance graphics by offloading rendering to the graphic card. This package is available on Github[1] and comes with an extensive documentation[2] and examples datasets.



**Keywords:** visualization, neuroscience, python, open-source, brain, MNI, GPU, opengl, sleep, data mining


### Declarations


**Funding:** This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.


**Conflict of interest:** The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

[1] https://github.com/EtienneCmb/visbrain
[2] http://etiennecmb.github.io/visbrain/

# Introduction

Generally, scientific visualizations are used to illustrate complex results and make the data speak. For publication, those illustrations must be clear, concise and understanding at the first glance. In practice, it is not always simple to handle very large datasets, especially with the dimension problem. Publication figures are statics, definitive and concentrate hundreds or thousands of pre-figures that have been upstream used to understand the data. Neuroscience is no exception and the several research fields imply a large range of possible representations. Moreover, this range is further expanded when account is taken of the diversity of neural recordings types.

To date, Matlab represents one of the most widely-used software for brain data analysis and visualization, notably thanks to several toolboxes such as SPM, Brainstorm[3] (Tadel et al., 2011), EEGlab[4] (Delorme and Makeig, 2004) or Fieldtrip[5] (Oostenveld et al., 2011). On the other hand, only few Python alternatives exist so far, among which are MNE[6] (Gramfort et al., 2013), PySurfer[7] or Nilearn[8] (Abraham et al., 2014).

In this context, we propose a Python open-source software named Visbrain, distributed under a BSD licence and dedicated to the visualization of neuroscientific data Visbrain is built on top of VisPy (Campagnola et al., 2015) package, and beneficiate therefore from high-performances graphical rendering offloaded to the graphic card. This high plotting performance is assessed using the VisPy (Campagnola et al., 2015) package. The goals of the Visbrain project are three folds : 1) Merge together several pure Python neuroscientific visualization tools, 2) Each module should comes with a graphical user interface (GUI) for non-python experts and should provide most used and extended functionalities, 3) Bring the community together to collaborate and adapt the software to larger needs.

[3] http://neuroimage.usc.edu/brainstorm/
[4] https://sccn.ucsd.edu/eeglab/
[5] http://www.fieldtriptoolbox.org/
[6] http://mne-tools.github.io/stable/index.html#
[7] https://pysurfer.github.io/
[8] http://nilearn.github.io/

# Materials and Methods

The main philosophy behind Visbrain is to propose a package that centralize several visualization modules. Each module could comes from very different fields but all should answer to one specific visualization type and must comes with a modular and responsive GUI. In addition, neuroscientific datasets can potentially be large hence plotting methods should be efficient, with the ability to be updated in real-time through the GUI. This package do not intend to provide functions to analyse data except if those functions lead to a potentially useful illustration.

## Programming language and code guidelines

As any new programming project, the choice of the language was the first interrogation. We naturally consider Matlab (MathWorks, 2012) or Julia (Bezanson et al., 2017) as they both are high-level languages. We finally chose Python as this mature language benefits from a large range of high-quality packages with a clear syntax, a huge community and documentation and, even more importantly, because we try to share this idea of open science and free software solution. In addition, Python software are portables and cross-platform and are easily distributed.
From a programming perspective, we did our best to avoid deep data copy and be able to load and process big data. The full Visbrain package is hosted on Github and is described in a NumPy like documentation and we further provide examples and datasets to improve understanding of those functions. Code blocks are well commented and we used statics analysis tools as recommended by PEP8.

### Graphics on the highway

As said before, neuroscientific data could be large, very large. For example, eighth hours of sleep data EEG recordings could weigh several gigabits according to the sample frequency. Hence, the visualization solution should be very efficient and, worst, should able to be updated in real-time. For this latter point, Matplotlib (Hunter, 2007) has been discarded as this library is primarily designed to provide publication quality figure and is not well suited for handling large data and user interactions.
For those reasons, we digged into the VisPy package (Campagnola et al., 2015) to find if this software could be a solution and indeed, it was. Basically, VisPy make a bridge between the comfortable syntax of Python and OpenGL which, in short, is really performant but not as intuitive as Python. In addition to those performances, VisPy is build on four levels, from the highest level very close to the Matplotlib style but suffering from weakest performances, to the lowest level (OpenGL Object Oriented), more difficult to manage and configure but providing the best graphics rendering. As Visbrain evolve, we went down from the highest to the lowest level for certain modules that do not rely on the basic objects, already implemented into VisPy. As a consequence, because graphic rendering are offloaded to the GPU and with the improvements of recent graphical cards, any modern laptop should benefits from this graphic treatment.

## Graphical interface and user interactions

GUI are not a necessity, at least not for confirmed Python user but non-expert usually appreciate manual interactions into a dedicated interface. The fact is that it's easier for expert programmers to use an interface that for non-programmers to use command line functions and this is the primarily reason of interfaces based modules.
At this point of the development, we had to make a choice of how building those graphical user interface and, even more critically, if it was possible to embedded VisPy graphics in it. The cross-platform GUI toolkit Qt[9] provides Python binding tools, such as PyQt to the C++ Qt toolkit and therefore make possible to convert into a Python compatible GUI. In addition, the Qt designer has been extensively used to build modular and responsive GUI of each module.

## Documentation and examples

In addition to the Python package, we provide a detailed step-by-step documentation build with Sphinx[10] and hosted on github[11]. This documentation describe how to install Visbrain and use modules. We also described each GUI panel, each functionalities and inputs of each class module. Moreover, inside those interfaces, we also provide a description of each graphical element using tooltips which appear on mouseover.

[9] https://www.qt.io/
[10] http://www.sphinx-doc.org/en/stable/
[11] http://etiennecmb.github.io/visbrain/sleep.html

# Results

Currently, Visbrain architecture is based on four modules :

- *Ndviz* : for visualization of multidimensional data, basic plotting and data mining
- *Brain* : a 3D MNI standard brain centered module with a lot of controls and functionalities
- *Sleep* : this module is really specific to sleep scientist and propose to visualize polysomnographic recordings and hypnogram edition
- *Figure* : a Maplotlib based module that wrap together most used functions for publication-ready figures.

## *Ndviz* : efficient data mining

A common first step when starting a new scientific study is to explore the dataset in order to inspect the shape of the neural signals or finding artefacts for trials that need to be rejected. And, for large datasets it is not always evident. In this Visbrain package, we included a module called *Ndviz*[12] dedicated to the visualization of multidimensional signals.

### Dispose your data into a grid

This idea of data mining module was induced by one of the VisPy example[13] in which thousands of signals, each one having thousands of points, can be instantly plotted by the use of graphics rendering offloaded to the GPU. Those signals are disposed into a two dimensional grid and the user can scroll on each one of them. We thought that this idea has a great potential for inspecting neuroscientific data. For example, 10 sessions of EEG recordings using 128 contacts can be visualized in a 10 by 128 grid and on each point of this grid seat the corresponding time series.
We embedded this example in a GUI (**Figure 1**), providing more controls on which dimension to pick. In addition, we also provide several ways to color those signals.

### Plotting forms

Then, each signal of the grid can also be visualized independently and in several forms (**Figure 2**). As a simple continuous line which is the first choice for time series. Then, as markers or cloud of points which, for example, might be useful for displaying features used in machine-learning in order to see if they are well separated. Thirdly, as an image as neuroscience representations frequently used them. Inspecting a large amount of time-frequency maps might be a dedicated scenario. Finally, it is also possible to compute the histogram or spectrogram and to visualize them. For also add basic “Previous” and “Next” buttons to quickly jump for a signal to another.

From a multidimensional signal, the user can pick the dimensions he wants to inspect the dataset, and visualize it either on a complex 2D grid or using simples to common

[12] http://etiennecmb.github.io/visbrain/ndviz.html
[13] https://github.com/vispy/vispy/blob/master/examples/demo/gloo/realtime_signals.py

representations. Thus, *Ndviz* is a good way to quickly inspect data, produce basics plot and familiarize with dimension switching. A small piece of Python code is provided in the **code snippet 1**.

# Brain : visualization on a standard 3D MNI brain

The main idea behind the *Brain*[14] module is to propose a GUI (**Figure 3**) able to perform complex visualizations using a 3D standard MNI brain. To this end, *Brain* is articulated around four distinct objects :

- The main brain template
- Deep sources
- Connectivity links between those sources
- Sub-structures

Importantly, those objects are independent from recording types can be added to the main canvas and can are highly configurables. We describe those objects in sections below.

## MNI brains

With the installation of Visbrain comes three default templates containing or not the cerebellum and more or less smooth. This limitation of three templates can be extending by the use of input parameters to directly pass vertices and faces of user specific templates (**Figure 4**). Importantly, this visualization is not a volume i.e. it only display the brain surface. As a compensation of this latter point, the user can control the color of the surface.
In addition to the provided templates, this module allows transparency control (from fully transparent to opaque). The user can also take brain slices over the (x, y, z) axis. This latter functionality can be interesting for medial wall inspection. Finally, *Brain* take the benefits of the VisPy cameras so that the brain can be rotate around the central point or in a airplane simulation way for deep visualization.

## Region of interest

Secondly, we refer to region of interest (ROI), labelled volumes that can be add to the canvas. *Brain* can extract those structures from two atlases: brodmann areas and Automated Anatomical Labeling (AAL). As an example, the user can add brodmann areas 4 and 6, which respectively contains primary and premotor areas as well as the Supplementary Motor Area extracted from the AAL volume (**Figure 5**). Those volume are converted into surface with the possibility be smoothed before being displayed. Finally, those mesh inherits the same methods as the brain such as transparency and slice control.

## Deep sources

Third object that can be add on the main canvas : sources. Here, the term sources can be understand as small balls inside the brain as, depending on the neuroscience field, it can be interpreted differently. For example, in MEG, those balls can be called source (source reconstruction) but insing intracranial data those sources are electrodes.

[14] http://etiennecmb.github.io/visbrain/brain.html

We used the standard MNI coordinate system but *Brain* also includes functions for Talairach coordinates conversion (Talairach and Tournoux, 1993). We offer further source's parameters such as size, color, shape (i.e. disc, square, diamond…). A text can also be add to each one of them. More interestingly, it is possible to attach values to each source and, in this case, ball's radius will be proportional to this value. As an example, each site can materialize the beta power. Moreover, *Brain* contains methods that can detect and hide sites that are not contained into the brain template volume.
Taken together, color and shape can be useful for representing, for example, the intracranial implantation per subject. The proportional ball's size modulation according to the attached activity is an interesting feature that can give a first impression. But, the imposed planar representation of a laptop screen does not restitute the depth. As a complement, we provided the ability to project the activity on the cortical surface. Indeed, we look for every brain vertices contains in a 10mm (this radius is controllable) sphere around the source. Then, the activity attach to this source is finally set to the vertices within the sphere. In addition to the cortical projection, we also provide a cortical repartition which measure the number of contributing sources per vertex. Finally, all of those projections can also be applied on ROI. Those source's related functions are summarized in the **Figure 6**.

## Connectivity

Last but not least object : the connectivity between sources can also be add and highly configured. An upper triangular connectivity array is passed to define the connectivity strength between those sources. In addition, using boolean mask on this array allow to control which connection need to be drawn. From the user side, line's width is also a parameter such as colors which can be either set by specifying one color per connectivity strength, or using colormap's based methods. Those methods automatically set a color according to the strength, to the number of connections per node or by density which look at the number of connections in a sphere of interest (**Figure 7**). A large number of connections often occurs a confused scene. In that case, *Brain* also provide the possibility to have a dynamic transparency where weaker connections are going to be more transparent.

## High-definition screenshots

The last element of the chain is the possibility to save the scene into a compatible high-definition image (*.png, *.jpg, *.tif...). Note that some image file format conserve the transparency and, as a consequence, a transparent brain in the interface will also be transparent in the exported image. Finally, the exported image can be cropped and we also provide a method that automatically crop the image to the closest non-background pixel.

## Class methods for a code-line control

We are frequently in the scenario where scientific figures needs several version and ajustements. While some of those figures are basics and do not take time to be redrawn, others might contains 20 brains, with the cortical projection on each one and finally organized in a grid. If for each brain, the GUI have to be opened, then run the projection and finally make a screenshot of it, thus the full process for having the final figure can be long. In this context, each methods or parameter can be set in command line and this figure process

can easily be embedded in a loop. All of those methods are referenced into the documentation[15].

Taken together, Brain groups a relatively large amount of possibilities summarized in **Figure 8**. We tried the write a clear documentation[16] and also provide several basics and advanced examples[17] and datasets.

# *Sleep* : polysomnographic data visualization and edition

*Sleep*[18] is the Visbrain module dedicated to the visualization and edition of polysomnographic data. It allows the user to load a variety of standard electrophysiological file formats (Brain vision *.eeg, European Data Format *.edf, Micromed *.trc, Elan *.eeg). For non-supported files, we also provide the possibility to directly pass NumPy arrays.

## Data visualization

After loading the data, the GUI propose as many checkbox as the number of channels for controlling which one to display. In addition, the amplitude can either be set per channel or though all of them. By default, *Sleep* propose to visualize 30 seconds of polysomnographic data but this window length is configurable. For spectral investigations, we also embedded a highly configurable spectrogram and topographic representation. As any of the Visbrain module, Sleep inherit from this modularity and any panel can either be display or hide as shown in **Figure 9**.

## Hypnogram visualization and scoring

A standard procedure in the sleep domain consist of scoring full night recordings that means setting one the referenced sleep stage. We used the nomenclature of the American Academy of Sleep Medicine (**Iber, 2007**) which define 6 possible scoring states Artefact, Wake, Rapid-Eye Movement (REM) and non-REM stages 1, 2 and 3.
*Sleep* provide the ability to load an existing hypnogram or to start a new one. By default, those stage are displayed in the order : Art, REM, Wake, N1, N2, N3 but this order can be set when defining the *Sleep* instance, in the Python script. We also provide three scoring methods in the GUI:

- A drag and drop way : the user can directly interact with the hypnogram, insert new points by clicking on it and move those points. This is method is more appropriate for shorter data as 8 hours night requires precision that is difficult to obtains with a mouse.
- Using a scoring table : the user can manually specify where the sleep stage start and finish in an interactive table.
- Using keyboard shortcuts : we associate a shortcut for each step. Pressing on the key set the stage on  the hypnogram and the next time window is displayed.

[15] http://etiennecmb.github.io/visbrain/brain.html#user-functions
[16] http://etiennecmb.github.io/visbrain/brain.html#
[17] https://github.com/EtienneCmb/visbrain/tree/master/examples/brain
[18] http://etiennecmb.github.io/visbrain/sleep.html

At any moment, the hypnogram data can be saved or exported in a black and white publication-ready figure (**Figure 10**).

### Automatic events detection and signal processing tools

Sleep data contains microstructural events that are specific to sleep stages. For example, the [12, 14hz] spindle bursts are frequently found in N2 stage. Therefore, those events represent an important scoring help. *Sleep* actually contains 6 detection types : spindles, k-complex, REM, slow-waves, muscle twitchings and a peaks detection that can be used to compute heart rate. Those detections can be runned on any channel, from the GUI, and are directly add into the channel and on the hypnogram (**Figure 9**).
In addition to those detection algorithms, we add further reversible signal processing tools that allows a real-time filtering and the ability to extract the amplitude, the phase or the power within specific frequency bands. Finally, the user can also re-referenced the data from the software or bipolarized, a frequent procedure for intracranial data (**Jerbi, 2009**)

### GUI commodities

On top of the functionalities presented above, the entire GUI is controllable using keyboard shortcuts which enable a quick navigation. The list of those shortcuts can be opened from the Help menu of the interface.

## *Figure* : page layout of publication-ready complex figures

At the end of this chain of modules, we also provide a Matplotlib based tools that we called *Figure*[19] with the aim to simplify the layout of saved pictures. *Figure* do not provide any tools that is not already provided by Matplotlib, but instead wrap together functionalities that present a serious interest with the Visbrain modules and a full integration. In order to works, this module requires picture paths and then, can propose a simple grid representation on which it is off course possible to add titles, x and y labels and highly configurable colorbars that can either be set per figure or shared across subplots. The final produced figure can be exported in high-resolution with a controllable dpi level for publication integration. A layout example, produced with the **code snippet 3** is provided in **figure 11**.

[19] http://etiennecmb.github.io/visbrain/figure.html

# Discussion

The complexity of recents analysis procedures make neuroscientific visualizations demanding. We are currently trying to group a set of existing visualization tools, that should be transparent regardless to the recording types. This open-source Python package, Visbrain, benefits from graphics rendering offloaded to the GPU for efficient data plot.
This software actually includes 4 modules :

- *Ndviz* : for data mining and basic plotting. From a scientific pipeline perspective, *Ndviz* is the first step as it allows to quickly inspect large data, produce basic plotting (such as simple lines, cloud of points, images, histogram, spectrogram). Therefore, the multidimensional plotting capabilities enable to quickly identify bad signals and artefacts. We think that digging in fresh data with this software might be useful or for educational purposes.
- *Brain* : dedicated to visualizations using a 3D standard MNI brain such as intracranial or MEG source reconstruction representation. It is also possible to represent region of interest and to project source's activity on it or to illustrate 3D connectivity between those sources. We paid a very particular attention to color properties (colormaps, lmits, colors under/over threshold...) as this highly declinable feature is usualy the more explicit one in neuroscientific papers.
- *Sleep* : specific to polysomnographic data visualization and edition, this module is really addressed to sleep scientist. The main idea behind *Sleep* is to be able to visualize long night recordings and benefits from the OpenGL efficiency. We provide the ability to load standard file format, to configure which channel to display and provide further spectral tools such as spectrogram, filtering or topographic representations. Another main feature of *Sleep* is the possibility to score hypnogram, help by automatic events detection such as spindles or k-complex.
- *Figure* : for quality layout figure rendering, this tool close the chain of the Visbrain modules by offering a simple but easy to use way of organizing figures and quickly add additional labels, color background and colorbars.

If we set aside this last module, the three others comes with a modular and responsive graphical user interface designed with Qt. If both esthetic and ergonomy are not essentials, we still have paid attention to it, hence, provide comfortable softwares.
Considering the number of visualization possibilities, Visbrain is in it early stage of development but we think that the base is present and further programmers can contribute to make it grown and build a real Python open-source solution for Neuroscientific visualization needs.

## Future directions

The first thing on our planning is to move from PyQt4 to PyQt5 as this new version of PyQt is needed for newer Python versions. Generally, each module could be improved especially for preserving memory as big data are frequent. Finally, we planned to add new modules especially for planar connectivity illustrations and topographic plotting.

## Conclusion

The Python language brings portability and cross-platform installation to Visbrain. In addition, as researcher frequently move, we paid a particular attention to propose a solution that might be use on any computers. We try to build those modules with coherence, with intuitive interfaces and to provide consistent and extensive documentations.

# Figures

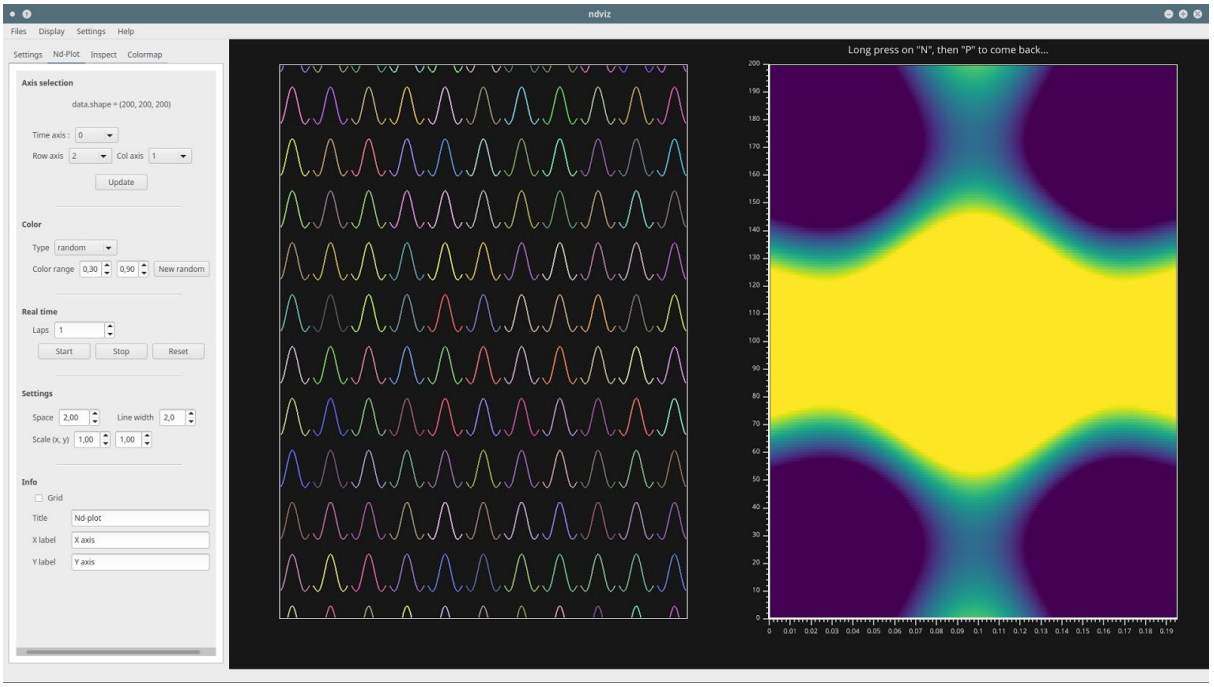


**Figure 1 :** Example of the graphical user interface of the *Ndviz* module. Leftmost is the setting panel, and side-by-side all of the sines disposed in a 2D grid and rightmost, the image formed by those sinaes.

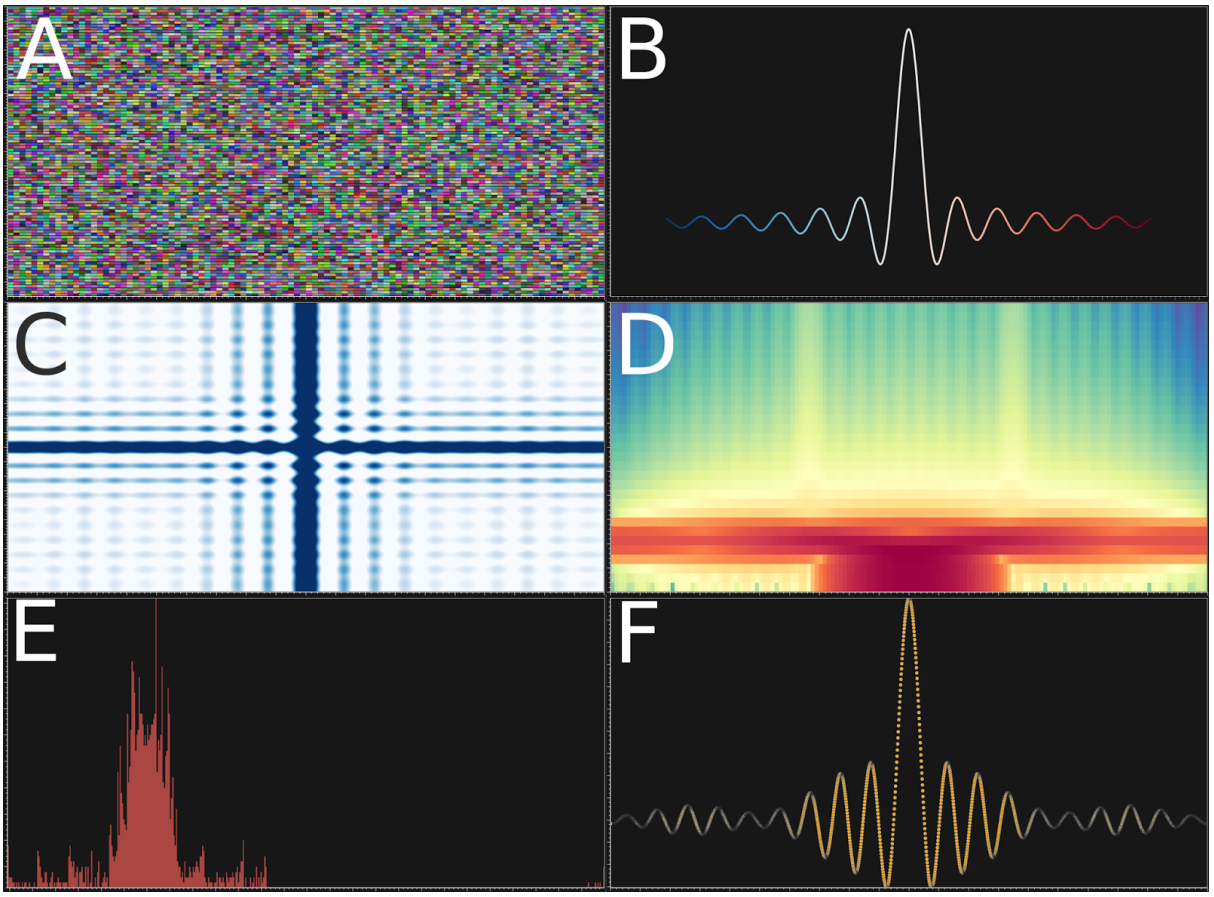


**Figure 2 :** Plotting functionalities from the *Ndviz* GUI, (*A*) Grid of 200x200 signals of 1000 time points each, visually differentiate with random color. Then, each one of those signals can be represented as a simple continuous line (*A*) as markers or points (*F*) or by picking two dimensions, the cardinal sine can be displayed as an image (*C*). Finally, it is also possible to compute and visualize the histogram (*E*) or the spectrogram (*D*).

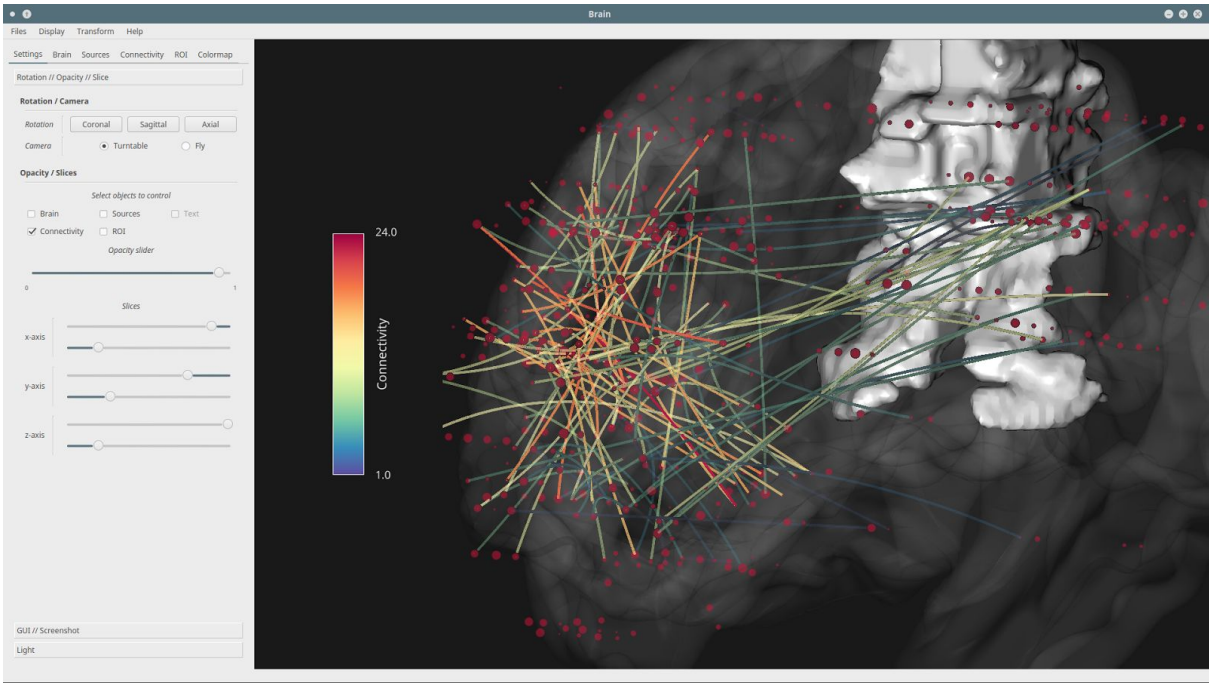


**Figure 3 :** Example of the graphical interface of the *Brain* module.

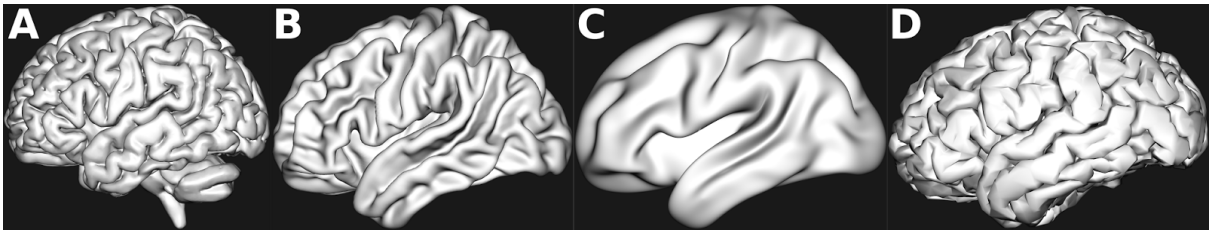


**Figure 4 :** Brain templates inside the *Brain* module. By default, *Brain* propose three templates (*A, B and C*) but using inputs parameters, the user can directly pass faces and vertices of his own template (*D*)

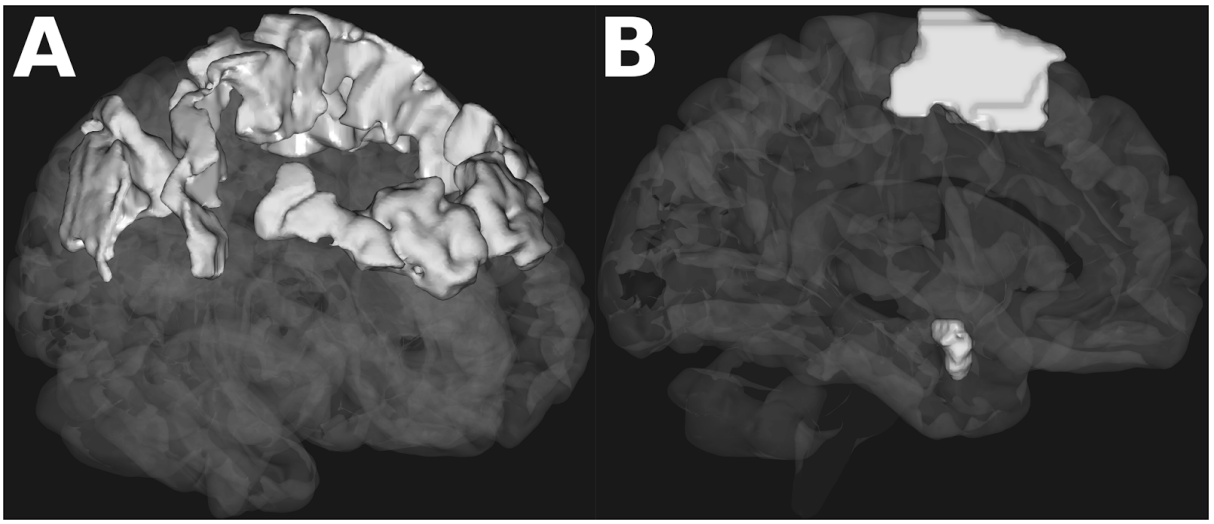


**Figure 5 :** Examples of region of interest displayed, (*A*) Brodman area's 4, 9 and 40, (*B*) Supplementary motor area and amygdala only for the left hemisphere.

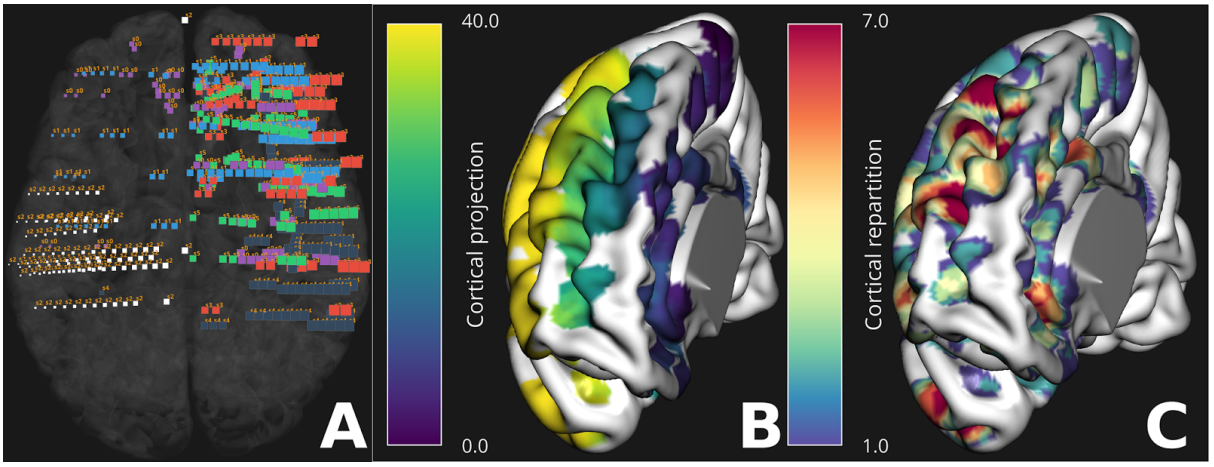


**Figure 6 :** Example of source objects and projections. (*A*) Sources obtained from the intracranial data of 6 subjects, each one having its own color. For the sake of this illustration, we set the activity of each source its x coordinate so that the diameter of each source increase along this axis. There's not neuroscientific meaning behind this. In addition, the name of each subject is added to each source, (*B*) Cortical projection of the source's activity only on the right hemisphere. Once again, according to the colorbar, the activity increase along the x axis, (*C*) Cortical repartition of this intracranial dataset. Accordingly to the colorbar, the blue means only one source contribute and in red, seven. this is particularly useful to see the distribution of the contributing sources on the cortical surface.

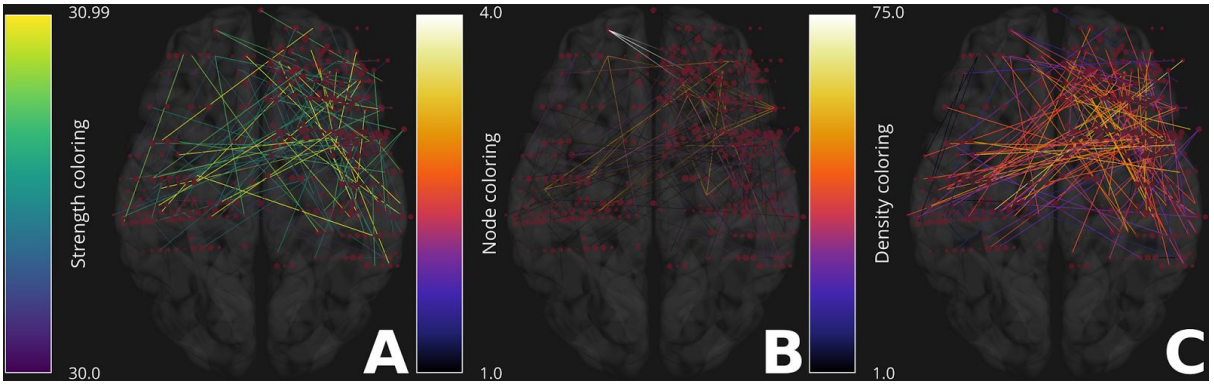


**Figure 7 :** Example of connectivity. Links are either colored according to the strength of connections (*A*) or by the number of connections per node (*B*) or finally according to the density i.e. the number of connections in a sphere of interest (*C*)

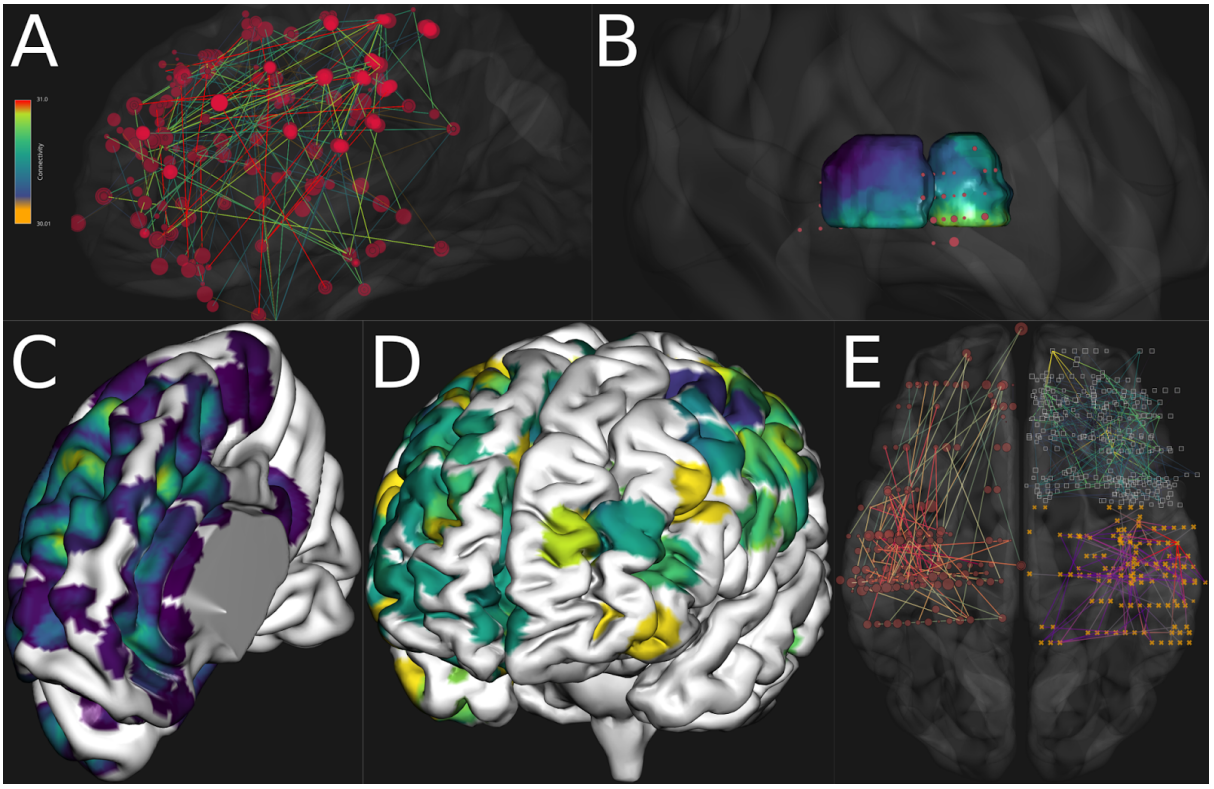


**Figure 8 :** Main functionalities of the *Brain* module, (*A*) Add deep sources inside the MNI brain and materialize connectivity, (*B*) Thalamus with the beta power of MEG source's reconstruction projected on it, (*C*) Cortical repartition i.e. number of contributing sources per vertex projected on the right brain hemisphere, (*D*) Cortical projection of high-gamma power recorded using intracranial data, (*E*) Complex scene made up of several objects including three types of sources (red disc, blue square and yellow cross) and connectivity link for each one of them.

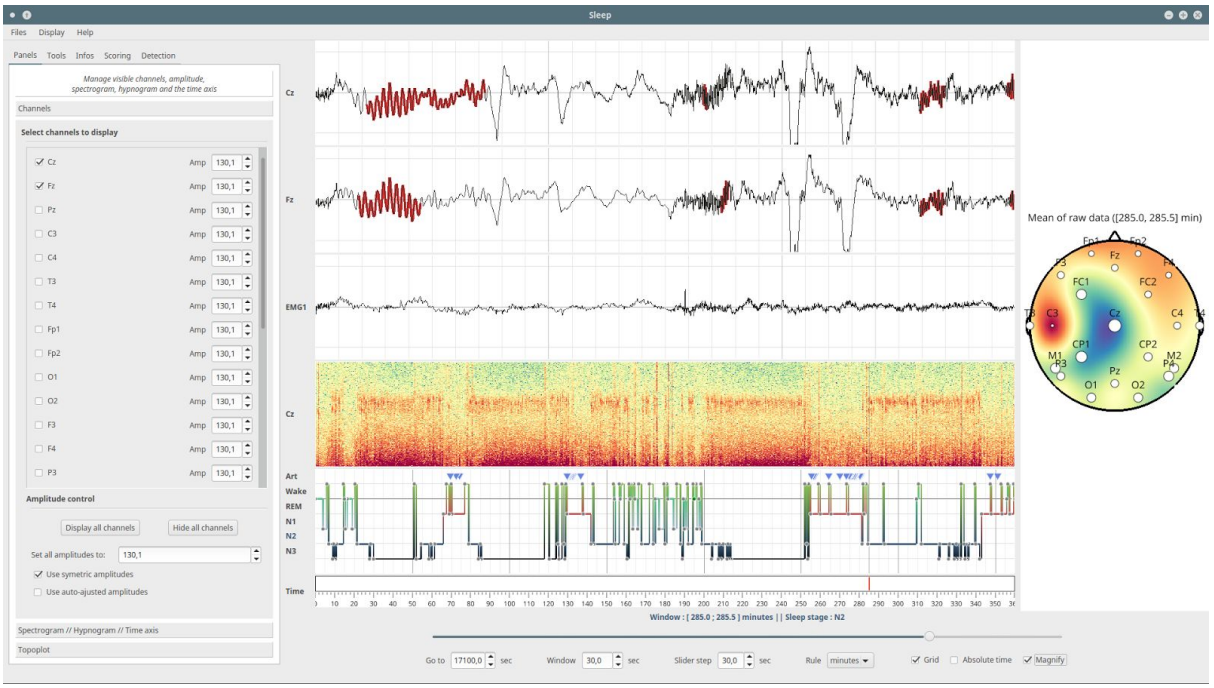


**Figure 9 :** Example of the graphical interface of the *Sleep* module. Leftmost is the setting panel that include all of plotting and signal processing controls and at the bottom of the figure, the navigation bar with time setting properties. Sleep data of three channels are plotted (Cz, Fz and EOG1) and in red, an example of GUI integration of microstructural event detection. *Sleep* also enable to add a spectrogram and provide further time-frequency settings. Finally, we also plot the hypnogram and the small markers on top of the hypnogram represent the time location of detected events. Finally, rightmost is an example of topographic representation.

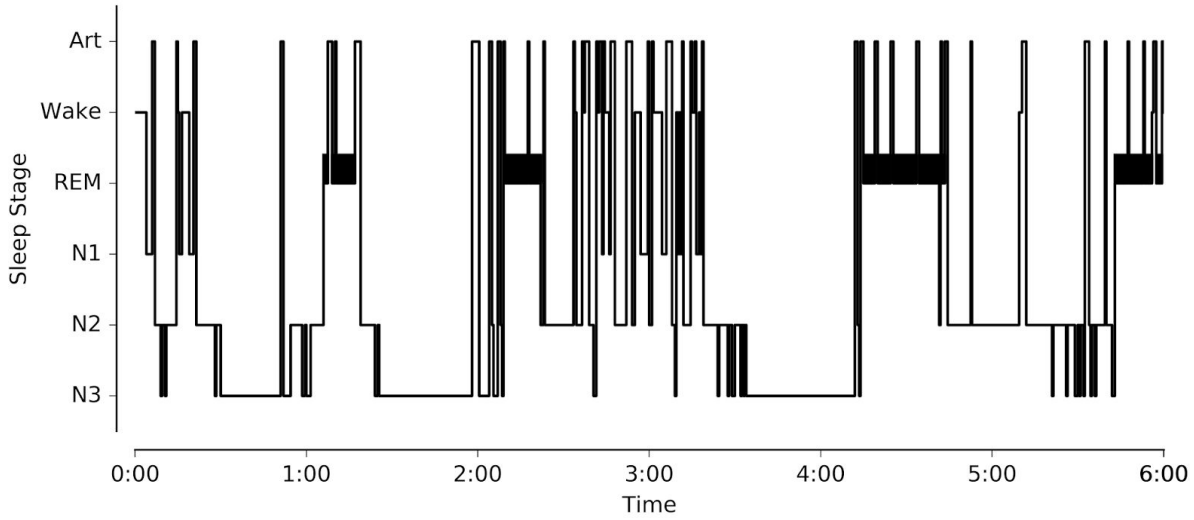


**Figure 10 :** Example of black and white exported hypnogram.

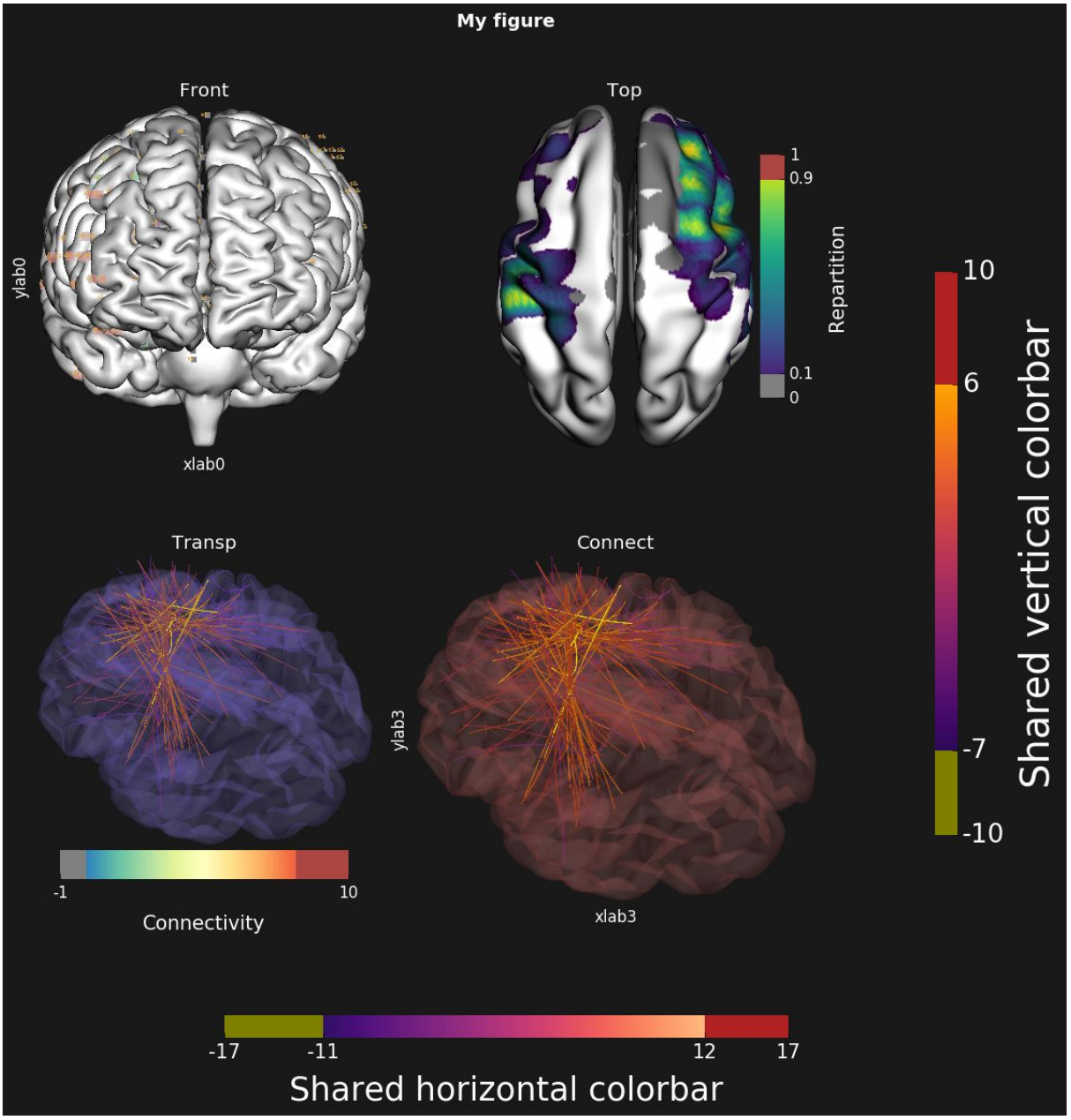


**Figure 11:** layout example produced with the *Figure* module.

# Code snippets

```
import numpy as np
from visbrain import Ndviz

# Sampling frequency :
sf = 1024.
npts = 200
time = np.arange(-npts/2, npts/2)/1024.
# Create a 2d signal :
y = np.sinc(2*10*time).astype(np.float32)
y = y.reshape(len(y), 1) + y
# Add a little bit of noise :
y = y**2 + np.random.rand(*y.shape) / 10
Ndviz(y, sf=sf).show()
```

**Code snippet 1 :** generate 200 cardinal sine of 200 points each and visualize them inside the *Ndviz* module. This code generate the **figure x**.

```
# Import the Figure module :
from visbrain import Figure

# Set the list of files to load :
files = ['front.png', 'top.png', 'connect.png', 'connect.png']

# Define titles, xlabels and ylabels :
title = ['Front', 'Top', 'Transp', 'Connect']
xlabels = ['xlab0', None, None, 'xlab3']
ylabels = ['ylab0', None, None, 'ylab3']

# Define the background color of each axis : the two last pictures are
# transparent brains. By setting the background color to 'slateblue' and
# '#ab4642', brains turn respectively to blue and red :
ax_bgcolor = [None, None, 'slateblue', '#ab4642']

# Define the Figure object :
f = Figure(files, titles=titles, figtitle='My figure', xlabels=xlabels,
           ylabels=ylabels, grid=(2, 2), ax_bgcolor=ax_bgcolor, y=1.,
           fig_bgcolor=(0.098, 0.098, 0.098), figsize=(12, 12),
           text_color='white', auto_crop=True)

# Add a colorbar only to the second axis :
f.colorbar_to_axis(1, (0, 1), 'viridis', title='Repartition', ticks='complete',
                   vmax=.9, over='#ab4642', fz_ticks=12,
                   vmin=.1, under='gray')

# Add a colorbar only to the third axis :
f.colorbar_to_axis(2, (-1, 10), 'Spectral_r', title='Connectivity', ticks='minmax',
                   vmax=8, over='#ab4642', fz_ticks=12,
                   vmin=0, under='gray', orientation='horizontal')

# Add a vertical shared colormap :
f.shared_colorbar((-10, 10), 'inferno', fz_title=30, vmin=-7, vmax=6,
                  under='olive', over='firebrick', position='right',
                  title='Shared vertical colorbar', fz_ticks=20, pltmargin=.1,
                  figmargin=.1)

# Add a horizontal shared colormap :
f.shared_colorbar(cmap='magma', clim=(-17, 17), fz_title=25, vmin=-11, vmax=12,
                  under='olive', over='firebrick', position='bottom',
                  title='Shared horizontal colorbar', fz_ticks=15, pltmargin=.1)

# Save the final figure in 600 dpi:
f.save('figlayout.png', dpi=600)

# Alternatively, you can show the figure :
f.show()
```

**Code snippet 3 :** generate 200 cardinal sine of 200 points each and visualize them inside the Ndviz module. This code was used to generate the **figure x**.

# References


Abraham, A., Pedregosa, F., Eickenberg, M., Gervais, P., Muller, A., Kossaifi, J., et al. (2014). Machine learning for neuroimaging with scikit-learn. *arXiv preprint arXiv:1412.3919*.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review* 59, 65–98.

Campagnola, L., Klein, A., Larson, E., Rossant, C., and Rougier, N. P. (2015). VisPy: Harnessing The GPU For Fast, High-Level Visualization. in *Proceedings of the 14th Python in Science Conference* Available at: https://hal.inria.fr/hal-01208191/ [Accessed May 23, 2017].

Delorme, A., and Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of neuroscience methods* 134, 9–21.

Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., et al. (2013). MEG and EEG data analysis with MNE-Python. *Frontiers in neuroscience* 7, 267.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing In Science & Engineering* 9, 90–95. doi:10.1109/MCSE.2007.55.

Iber, C., Ancoli-Israel, S., Chesson, A., Quan, S. F., and others (2007). *The AASM manual for the scoring of sleep and associated events: rules, terminology and technical specifications*. American Academy of Sleep Medicine Westchester, IL.

Jerbi, K., Ossandón, T., Hamamé, C. M., Senova, S., Dalal, S. S., Jung, J., et al. (2009). Task-related gamma-band dynamics from an intracerebral perspective: Review and implications for surface EEG and MEG. *Human Brain Mapping* 30, 1758–1771. doi:10.1002/hbm.20750.

MathWorks, I. (2012). MATLAB and Statistics Toolbox Release.

Oostenveld, R., Fries, P., Maris, E., and Schoffelen, J.-M. (2011). FieldTrip: open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. *Computational intelligence and neuroscience* 2011, 1.

Tadel, F., Baillet, S., Mosher, J. C., Pantazis, D., and Leahy, R. M. (2011). Brainstorm: a user-friendly application for MEG/EEG analysis. *Computational intelligence and neuroscience* 2011, 8.

Talairach, J., and Tournoux, P. (1993). Referentially oriented cerebral MRI anatomy: an atlas of stereotaxic anatomical correlations for gray and white matter. Thieme.