Sleep: a Python open-source software for visualizing and scoring sleep data

Etienne Combrisson[1, 2]*, Raphael Vallat[2, 3]*, Jean-Baptiste Eichenlaub[4], Christian O'Reilly[5], Tarek Lajnef[1,6], Aymeric Guillot[2,7], Perrine Ruby[2,3,§] and Karim Jerbi[1,§]

[1] Département de Psychologie, Université de Montréal, Canada
[2] Lyon 1 University, Lyon, France
[3] Lyon Neuroscience Research Center, Brain Dynamics and Cognition, INSERM U1028, UMR 5292, Lyon University, France
[4] Department of Neurology, Massachusetts General Hospital, Harvard Medical School, Boston, MA, USA
[5] Blue Brain Project, École Polytechnique Fédérale de Lausanne, Geneva, Switzerland
[6] Center for Advanced Research in Sleep Medicine, Hôpital du Sacré-Coeur de Montréal, Montreal, Quebec, Canada
[7] Inter-University Laboratory of Human Movement Biology, 27-29 Boulevard du 11 Novembre 1918, F-69622, Villeurbanne cedex, France

* These authors contributed equally to this work.

§ These senior authors contributed equally to this work

Corresponding authors:
Etienne Combrisson (e.combrisson@gmail.com)
Raphaël Vallat (raphael.vallat@inserm.fr)

Abstract

We introduce *Sleep*, a new Python open-source graphical user interface (GUI) dedicated to visualization, scoring and analyses of sleep data. Among its most prominent features are: 1) Dynamic display of polysomnographic data, spectrogram, hypnogram and topographic maps with several customizable parameters, 2) Implementation of several automatic detection of sleep features such as spindles, K-complexes, slow waves and rapid eye movements, 3) Implementation of practical signal processing tools such as re-referencing or filtering, and 4) Display of main descriptive statistics including publication-ready tables and figures. The software package supports loading and reading raw EEG data from a standard file formats such as European Data Format, in addition to a range of commercial data formats. Most importantly, *Sleep* is built on top of the VisPy library, which provides GPU-based fast and high-level visualization. As a result, it is capable of efficiently handling and displaying large sleep datasets. *Sleep* is freely available (https://github.com/EtienneCmb/visbrain) and comes with sample datasets and an extensive documentation. Novel functionalities will continue to be added and open-science community efforts are expected to enhance the capacities of this module.

Keywords: Electroencephalography (EEG), sleep scoring, sleep graphoelements, automatic detection, visualization, hypnogram, python, vispy, opengl

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.

Introduction

Polysomnography provides a comprehensive recording of the major physiological changes associated with sleep and is hence the gold standard for modern sleep analysis, both in research and clinical settings. At its simplest, it consists of monitoring at least 2 electroencephalogram (EEG), an electro-oculogram (EOG) and a submental electromyogram (EMG), providing sufficient information to identify sleep stages (sleep scoring) according to standard international established guidelines. A first set of rules were published by Rechtschaffen and Kales (R&K) in 1968 (Rechtschaffen and Kales, 1968) and proposed to divide sleep into 5 stages with distinct electrophysiological properties, named rapid-eye movement (REM) and non-REM (NREM) stages 1, 2, 3, 4. This nomenclature was updated in 2007 by the American Academy of Sleep Medicine (Iber, 2007) and sleep stage 3 and 4 have been merged into stage N3. In humans, a normal night of sleep consists of a repetition of four or five cycles in which sleep stages tend to follow each other in a particular order. Sleep staging is generally done visually by inspecting consecutive polysomnographic segments of 30 seconds. It results in a hypnogram which represents the succession of sleep stages across time. Apart from being time-consuming, visual sleep scoring is subject to both inter and intra-rater variability and is thus far from being optimal. By contrast, automatic sleep scoring has the advantage of being fast, reproducible and with generally good agreement with visual scoring (Berthomier et al., 2007; Lajnef et al., 2015a), yet its usage is far from being widespread and most sleep laboratories still rely on visual scoring, using either commercial softwares or in-house packages. In many cases, these software tools come with their own data and hypnogram file formats, and this heterogeneity can represent a substantial obstacle for sharing of sleep data across laboratories or clinics. Some of the very few existing open sources graphical user interface (GUI) for reading and scoring sleep include Phypno[1], written in Python, and the MATLAB-based toolboxes sleepSMG[2] or SPiSOP[3].

With this in mind, we developed *Sleep*, an intuitive and efficient open-source graphical user interface dedicated to the visualization of polysomnographic recordings and scoring of sleep stages. *Sleep* supports a range of data file formats and provides several scoring aid including the detection of essential features of NREM and REM sleep such as spindles, K-complexes, slow waves and rapid eye movements. *Sleep* was written in Python, an easy-to-learn high-level programming language widely used in the scientific community. We developed *Sleep* on top of VisPy[4] (Campagnola et al., 2015), a Python scientific library based on OpenGL which offloads graphics rendering to the GPU in order to provide fast and high-quality visualization, even under heavy loads as is the case with large dataset. *Sleep* therefore benefits from the high performances provided by VisPy alongside Python's inherent qualities such as its portability and ease of use.

[1] https://github.com/gpiantoni/phypno
[2] http://sleepsmg.sourceforge.net/
[3] http://spisop.org/
[4] http://vispy.org/

Materials and Methods

Efficient scientific visualization often consists of finding the best possible way to explore the data and to illustrate results in an intuitive and straightforward manner. The huge variety of neuroscientific data types and acquisition modalities naturally requires a wide range of specific visualization tools. Ideally, the various software packages needed for the various applications should be free and capable of handling several types of brain data recordings. In this context, we are currently developing a Python package we called Visbrain[3] distributed under a BSD licence, which provides and centralizes a number of useful brain data visualization utilities. We found a lack of existing software that wrap together an efficient, portable and user-friendly interface for polysomnographic data visualization and edition. As a result, we combined our skills in sleep analysis and Python programming to provide an open-source module, included within Visbrain, and named *Sleep*.

The choice of Python and the project vision

The choice of the programming language naturally turned to Python as this high-level and open-source language benefits from many libraries, an extensive documentation and a dynamic community. From data analysis to the production of high-definition paper figures, Python offers all the tools needed by scientists, with the comfort of a clean and easy to read syntax. *Sleep* is a pure Python software built on top of NumPy, VisPy, PyQt4[4] and uses a limited number of functions from SciPy and Matplotlib. Thanks to the Python portability, the software can be installed and used on any platform. One of the initial objectives of the project was to provide a user-friendly and intuitive interface capable of efficiently load and display large sleep dataset. To this end, we paid a particular attention to avoid deep data copy and display only what is necessary. Therefore, even very large recordings with a consequent number of channels can be handled by *Sleep* on any modern laptop with snappy GUI response. From a programming perspective, we did our best to provide a clean, commented and high-quality code, with a NumPy style documentation and using static analysis tool, as recommended by PEP 8. *Sleep* is hosted on GitHub and we encourage Python programmers and sleep scientists to collaborate in order to collectively improve this software by extending its functionalities and data compatibilities.

Hardware accelerated graphics

In addition to an ergonomics and portable interface, another goal was to use an efficient plotting library able to support and process large sleep data. Matplotlib was an option, but although this package has been developed to produce publication quality figures, it is not well suited for plotting and interacting in real-time with large datasets. VisPy is a scientific visualization library based on NumPy and OpenGL and was primarily designed to provide both high performances with real-time interactions and publication quality figures. VisPy make a bridge between the intuitive Python syntax and modern shader-based OpenGL pipeline allowing to offload graphical rendering cost to the graphics processor (GPU). This

[3] https://github.com/EtienneCmb/visbrain
[4] https://riverbankcomputing.com/software/pyqt/intro

package has been well designed and is built on four levels, from a Matplotlib oriented one to the lowest-level (closer to OpenGL) being more flexible and efficient at the cost of a slower learning curve. Because all *Sleep* graphical elements are primitive 2D objects (line, points and images) it was not a necessity to go down to the lowest level of VisPy (`vispy.gloo`). Indeed, all required objects were already implemented into the `Visual` library. Hence, any modern computer equipped with a graphics processor unit should see the benefits of the hardware accelerated graphics implemented in *Sleep*.

Portable GUI through Python

Among the major cross-platform GUI toolkits that interface with Python, wxWidgets[5] (wxPython), Tcl/Tk[6] (TkInter) and Qt[7] (PyQt/PySide) are probably the most known and used. We choosed PyQt which is a python binding for the C++ Qt toolkit. In addition, Qt Designer was used to design the graphical user interface.

Taken together, VisPy provides high-performance rendering graphics that are well integrated in a portable, modular and responsive Qt GUI using Python PyQt package.

Automatic events detection

One of the main objectives of *Sleep* was to provide a complete and easy-to-use interface for analyzing and staging sleep data. To this purpose, we implemented several algorithms for the automatic detection of sleep features that are embedded within the software ("Detection" panels). This include detections of spindles, K-complexes, slow waves, rapid-eye movements, muscle twitches and signal peaks. With the exception of the latter, all those features represent the landmarks of specific sleep stages and can therefore be used to quickly and easily identify the repartition of sleep stages within a period of sleep (see **Figure 1**). The main characteristics of each of these features are listed below.

- Sleep spindles refer to burst of 12 to 14 Hz waves predominant over central scalp electrodes and lasting between 0.5 and 2 seconds. These bursts of oscillatory activity have been known as a defining characteristics of N2 sleep.

- K-complexes are defined as sharp negative waves followed by a positive component, prominent over frontal scalp electrodes and lasting more than 0.5 seconds. Along with spindles, they constitute one landmark of N2 sleep and should not be mistaken with slow waves that defined N3 sleep.

- Slow-waves (or delta waves) are high-amplitude (>75µV) and low-frequency (<3Hz) oscillations that are present during the deepest NREM sleep stage, i.e N3 sleep. According to the standard international guidelines, N3 sleep is defined by the presence of 20% or more slow waves in a given epoch..

[5] https://www.wxwidgets.org/
[6] http://www.tcl.tk/
[7] https://www.qt.io/

- As its name suggests, rapid eye movements (REM) sleep is characterized by rapid eye movements easily observable on the EOG channels. They consist of conjugate, irregular and sharply peaked eye movements, similar to some extent to those exhibited during wakefulness.

- Another fundamental aspect of REM sleep is its muscle atonia, as revealed by a low EMG activity.. However, some transient muscle activity or muscle twitchings (MTs) can also be observed. These short irregular bursts of EMG activity are superimposed on the background of low EMG activity.

- Finally, Sleep implements a signal peak detection algorithm that is useful for example to calculate the heart rate, provided that an ECG channel is present.

Altogether, the set of detectors implemented in our software offers a valuable help for scoring sleep stages by offering efficient algorithms to automatically identify the main features of each sleep stages. Detections can also be used for a more in-depth analysis of the sleep microstructure (e.g. Vallat et al., 2017). Comparisons of performances between our detections and visual scoring are reported for K-complexes and spindles in the Results section.

Signal processing tools

In addition to the automatic detection presented above, *Sleep* provide a few more signal processing tools such as signal demeaning, detrending and a filtering (using either Butterworth or Bessel filters) with the following band option : lowpass, highpass, bandpass or bandstop. In addition, after being filtered, further informations can be extracted from the Morlet wavelet complex decomposition (Tallon-Baudry et al., 1996) such as time-resolved and band-specific amplitude, power or phase. Each one of this signal processing tools are reversive and can therefore be activated and deactivated without altering the original data and without any data copy in order to minimize memory requirements. Finally, loaded data can be re-referenced directly from the interface by either re-referencing to a single channel or common-average (frequently used for scalp EEG datasets) or by using bipolarization, which consists of subtracting neural activity from consecutives sites (classically used in intracranial EEG, see Jerbi et al., 2009).

Documentation and examples

We are conscious that not all neuroscientists are familiar with Python programming. Therefore, we provided an extensive and detailed step-by-step documentation, build with Sphinx[8] and hosted on github[9]. This documentation include a description of the graphical components and the main functionalities of the software. A PDF version of the documentation can also be downloaded directly from the "Help" contextual menu of the software. For scientists or students that wish to familiarize themselves with *Sleep* but do not have a dataset, we also provide anonymous and free-to-use example datasets, including

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

the corresponding loading scripts. Finally, for a real-time documentation, we used the tooltips provided by PyQt to describe each element of the interface.

Results

Graphical User Interface

The *Sleep* GUI can be subdivided into six distinct components :

- Settings panel
- Time properties bar
- Hypnogram
- Electrophysiological time series
- Spectrogram
- Topographic map

As the user interface is built up in a modular way, each of these components can be hidden or displayed, depending on whether the user prefers a light or fully-featured interface. Using the contextual menu, users can save and subsequently load the current display properties in order to easily retrieve previous session.

Settings panel and navigation bar

All controls and properties are grouped in a settings panel. This panel is subdivided into five thematic tabs :

- Panels : manage the visibility and properties of each plotted canvas.
- Tools : bundle of signal processing tools.
- Infos : basic informations of the current recording (name, sampling rate) and sleep statistics computed using the hypnogram (sleep stage duration, latency, etc). Note that the statistics can be exported in *.csv or *.txt file and are automatically updated when the hypnogram is edited.
- Scoring : scoring table that can be used to inspect and edit where each stage start and finish. This panel represents one of the three methods available within the software to edit hypnogram (see hypnogram edition section) and may be useful for example to score long periods of continuous and homogenous sleep by just providing the starting and ending times.
- Detection : perform and manage the automatic detection of several sleep features.

In addition to this setting panel, *Sleep* provides a navigation bar that can be used to set several temporal properties, such as the length of the current time window, time step between each window, time units and the use, if provided, of the absolute time of the current recording. This navigation bar also includes a grid toggle button that either hide or display the grid and a magnify option to enlarge short events (see **Figure 3**).

Electrophysiological time series

Sleep offers a dynamic control of the displayed polysomnographic time series and most of the settings are in the “Panels” tab. Indeed, each channel can be added or removed from the list of the currently displayed canvas. By default, *Sleep* displays the time series by frames of 30 seconds, which is the standard duration for stage scoring (Iber et al. 2007), but this value can be changed directly from the navigation bar. Furthermore, the amplitude of each channel can either be set independently, using a same range across all channels, or automatically adjusted according to the minimum/maximum of the currently displayed signals.

Time-frequency representation

As shown in **Figure 2**, the visibility and amplitude of each channel can be controlled from the GUI. The same applies for the spectrogram, which corresponds to a time-frequency representation of the entire recording performed on one channel. Among the definable parameters of the spectrogram are the channel on which it is computed, bottom and top limiting frequencies, length and overlap of the fast fourier transform and colormap properties. Finally, a topographic map based on the MNE and SCoT implementations (Billinger et al., 2014; Gramfort, 2013) can also be embedded inside the GUI for a full data inspection. The topological plot is computed from the mean of time window currently displayed and can be used to visualize the raw data, the amplitude or power in specific frequency bands.

Shortcuts

Navigation and operations inside a software can be sometimes repetitive. For that reason, *Sleep* comes with a bunch of native shortcuts to facilitate the visualization and stage scoring, a non-exhaustive list of which is provided in **Table 1**. For a complete list we refer the reader to the “Shortcuts” paragraph of the documentation.

Keys	Description
n	Move to the next window
b	Move to the previous window
-	Decrease amplitude across all channels
+	Increase amplitude across all channels
s	Display / hide spectrogram
t	Display / hide topographic representation
h	Display / hide hypnogram
p	Display / hide time bar
z	Enable / disable zooming
a	Scoring: set current window to Art (-1)
w	Scoring: set current window to Wake (0)
r	Scoring: set current window to REM (4)
1	Scoring: set current window to N1 (1)
2	Scoring: set current window to N2 (2)
3	Scoring: set current window to N3 (3)

Table 1. Main native shortcuts of the software.

Supported electrophysiological and hypnogram data formats

Sleep natively supports several standard electrophysiological file formats, including European Data Format (EDF *.edf), Micromed (*.trc), Brain Vision (*.eeg), and Elan (*.eeg). In addition, it is possible to load directly NumPy array or Matlab file using the command-line parameters.

The hypnogram of the corresponding dataset can also be loaded and then edited directly from the GUI. Accepted hypnogram file formats are *.txt, *.csv or *.hyp. There is a great heterogeneity among sleep laboratories in the hypnogram format. This represents an obvious barrier for data sharing. To overcome this problem, *Sleep* allows the user to specify the hypnogram format in a separate text file. This file should contain the names and integer values assigned to each sleep stages in the hypnogram file, as well as the number of values per seconds. During loading, the hypnogram file will be converted to *Sleep* native hypnogram format described in **Table 2**. An example description file can be found on the documentation.

Parameters	**Values**	**Description**
Time	1	Hypnogram file contains one value per second
Wake	0	The value assigned to Wake in the hypnogram is 0
N1	1	The value assigned to N1 sleep in the hypnogram is 1
N2	2	The value assigned to N2 sleep in the hypnogram is 2
N3	3	The value assigned to N3 sleep in the hypnogram is 2
REM	4	The value assigned to REM in the hypnogram is 4
Artefact	-1	The value assigned to Artefact in the hypnogram is -1

Table 2. Parameters of *Sleep* native hypnogram format.

Hypnogram edition

The hypnogram can be edited either from scratch or from an existing hypnogram file. There are three methods to edit the hypnogram using *Sleep* GUI:

- Using intuitive keyboard shortcuts (see Table 1). When a new stage is entered, the next window is shown.
- Using a table where each stage can be specified by it starting and ending time point.
- Using a drag and drop operation directly on the hypnogram canvas.

At any moment, the user can export the hypnogram or save a black and white publication-ready figure using the contextual menu **(Figure 2)**

GUI integration and validation of automatic events detection

The automatic events detection can be performed on any selected or visible channel. When the detection is completed, detected events are depicted directly on the selected channel using a specific color-code for each feature. In addition, the starting point, duration and stage of occurrence of each detected events are reported in the “Location table”. Users can then easily navigate between the detected events by clicking on a row, which automatically set the time so that the event is centered on the screen. Furthermore, this table can be exported to a *.csv or *.txt file. Users can perform an unlimited number of detections in a row on a single channel and then switch from one to another using the “Location” panel. Last but not least, the location of each detected events is reported on the hypnogram using specific visual cues for each detection types. Integration of the detection inside the GUI is shown in **Figure 4**.

To test how these detections performed on real datasets, we measured performances of the spindles and K-complexes detection methods using visually-annotated EEG segments of N2 sleep collected from full-night polysomnographic recordings of 14 participants (Eichenlaub et al., 2012, 2014, Ruby et al., 2013a, 2013b). Spindles and K-complexes were visually scored by an expert (JBE) as part of a previous work that focused specifically on the detection of these sleep features using machine-learning algorithm (Lajnef et al. 2015a).

To perform the detection methods using *Sleep* algorithm, all N2-sleep EEG segments were concatenated into a single file of 210 minutes with a single channel (C3) and with a sampling rate of 100Hz (native downsampling frequency of *Sleep*). Then, to evaluate the performances of our detection, we used two standards metrics: the sensitivity *(1)*, which measures the proportion of correctly identified detected events and the False Detection Rate (FDR) *(2)* which assess the proportion of incorrectly detected events.

$$(1)\ Sensitivity\ =\ \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$(2)\ False\ Detection\ Rate\ =\ \frac{False\ Positive}{False\ Positive + True\ Positive}$$

where `True Positive` refer to the events scored by the expert and correctly detected by our methods, `False Negative` refer to the events scored by the expert but not detected by our method and `False Positive` refer to the events detected by our methods but not scored by the expert.

Performances of our detection algorithm are reported in **Figure 7.** For both spindles and K-complexes**,** we used 25 different thresholds ranging from 0 to 5 with 0.2 steps**.** The optimal threshold was defined as the one that maximizes the difference between sensitivity and FDR (Lajnef et al. 2015a). Regarding spindles, the best performance of our algorithm was obtained at a threshold of 2.4 standard deviations, yielding a sensitivity of 77.2% and a FDR of 40.1%. Regarding K-complexes, a threshold of 1.0 resulted in the best performances with a sensitivity of 70.7% and a FDR of 27.2%. These figures are similar to those of previous detection methods (Lajnef et al. 2015a, Devuyst et al. 2011). Moreover, the time of execution of these two algorithms at a given threshold is less than a minute on any modern laptop computer.

Sleep class inputs and code example

From a programming point of view, the high-level interface with our software is provided by the *Sleep* class. This class can take into account a few input arguments. Hence, loading sleep data can be assessed in three ways adapted to a range of users, from non-programmers to advanced users. As shown in the **code snippet 1**, running *Sleep* without further input arguments will ask the user to specify the path to a supported sleep dataset (*.eeg, *.edf or *.trc). In addition, the user can either use an existing hypnogram or start a new one from scratch. Alternatively, instead of using the interface to select the files, they can be directly passed as input arguments (**code snippet 2**). In this example, we also demonstrate how to change the default order of the sleep stages in the hypnogram using a simple command-line option. If this option is not specified, the default display of *Sleep* is as follows: *Art, REM, Wake, N1, N2, N3*. Finally, several others file formats such as EEGLab, Neuroscan, EGI, GDF and BDF can be loaded using MNE python package. We report in **code snippet 3** a method to pass data to *Sleep* after loading them using MNE python.

Discussion

We are currently developing an open-source Python software called Visbrain dedicated to the visualization of neuroscientific data. As a part of this project, we developed a module called *Sleep* designed to visualize and score sleep data.

Sleep comes with a graphical user interface in which we embedded high-quality plots with graphical rendering offloaded to the GPU. As a result, plotting and user interactions can be processed in real-time. The software is capable of loading several widely-used sleep data files format, such as European Data Format and BrainVision, and to stream efficiently all of the polysomnographic channels, even on an average modern laptop. On top of that, *Sleep* also provides the possibility to display time-frequency (spectrogram) and topographic representations of the data, with several adjustable parameters for each. Regarding sleep staging and hypnogram editing, *Sleep* offers intuitive manual scoring functionalities, signal processing tools and automatic detection of sleep features in order to facilitate this fastidious process. Once completed, users can export the scored hypnogram data and/or a publication-ready high-quality figure of the hypnogram in one click. Regarding the automatic detections, *Sleep* includes 6 computationally-efficient and robust algorithms for detecting some of the most prominent features of each sleep stages, including spindles, K-complexes, slow waves, rapid eye movements and muscle twitches. Spindles and K-complexes detection algorithms were validated on a visually scored dataset including 210 minutes of N2 sleep from 14 participants and resulted in performances similar to recent publications. Last but not least, these detections are implemented inside the GUI in an ergonomic and intuitive manner. We think that these detections may represent a valuable help not only in the process of staging sleep, but also for researchers that are interested in the microstructure of sleep.

Future directions

Regarding the programming environment, Visbrain, and therefore *Sleep*, is built on top of PyQt4 and not the latest version PyQt5. As a consequence, Sleep is currently not able to run on the newest Python version (3.6) and users that wish to install the software must first create a dedicated Python 3.5 environment with PyQt4. Second, we are also considering to extend the list of the default supported files and we encourage programmers or sleep scientists interest by this project to collaborate on it. Regarding sleep analysis we are working on an automatic scoring function based on machine-learning algorithms, according to our previous experience in this field (Combrisson et al., 2017; Combrisson and Jerbi, 2015; Lajnef et al., 2015b). Finally, as different users have differents needs, we are constantly improving the interface and functionalities of the software thanks to the feedbacks we receive.

Conclusion

We offer a portable and cross-platform software, installable and usable on most configuration. While there is still place for improvement, *Sleep* already provides a complete and intuitive interface designed by and for sleep scientists. We hope this software will contribute to a more open science and strengthen the idea that open-source software can be of professional quality.

Figures

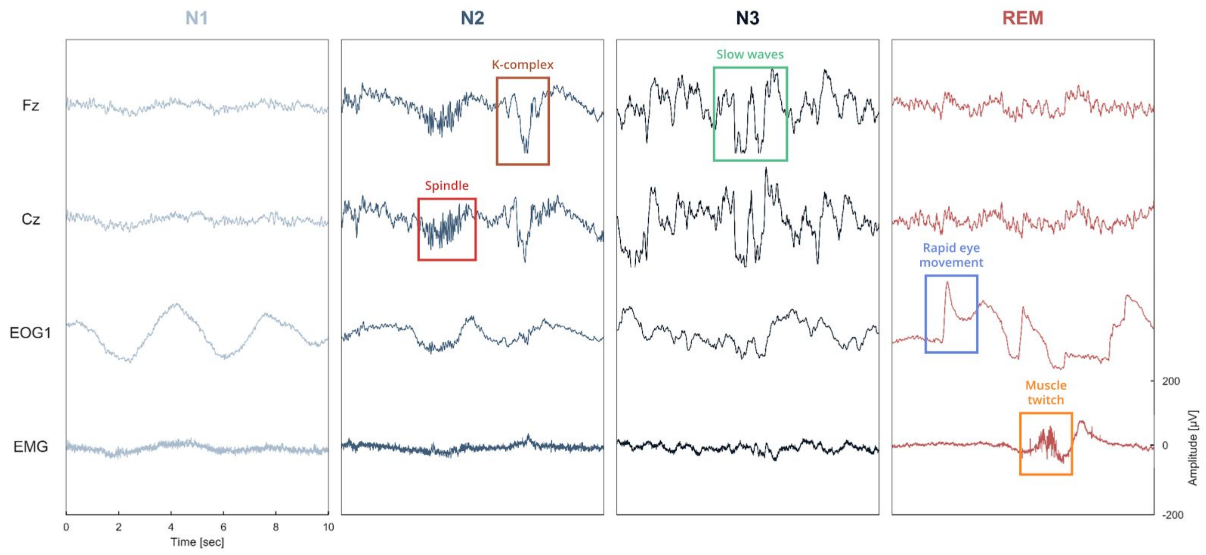
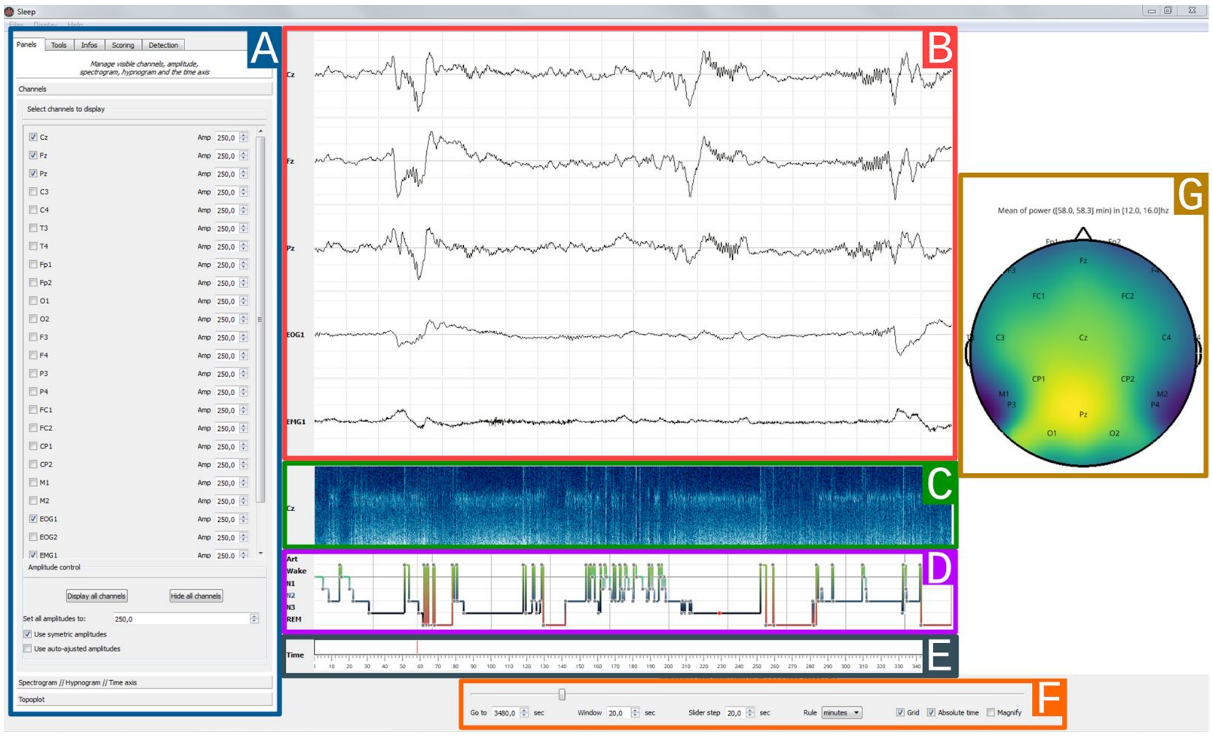


Figure 1: Illustration of the different sleep features observed in a polysomnographic recording of one individual. Note that these events are just displayed for illustrative purposes. To see examples of automatic detection actually performed by our software, see Figure 4. Spindles and K-complexes are the landmarks of N2 sleep. Slow waves are present during N3 sleep (sometimes referred to as slow wave sleep). Rapid eye movements, observed in the EOG channel, and muscle twitches, observed on the EMG channel, are two essential features of rapid eye movement (REM) sleep.



Panels | Tools | Infos | Scoring | Detection

Manage visible channels, amplitude, spectrogram, hypnogram and the time axis

Channels

Select channels to display

[✓] Cz	Amp	250,0
[✓] Fz	Amp	250,0
[✓] Pz	Amp	250,0
[ ] C3	Amp	250,0
[ ] C4	Amp	250,0
[ ] T3	Amp	250,0
[ ] T4	Amp	250,0
[ ] Fp1	Amp	250,0
[ ] Fp2	Amp	250,0
[ ] O1	Amp	250,0
[ ] O2	Amp	250,0
[ ] F3	Amp	250,0
[ ] F4	Amp	250,0
[ ] P3	Amp	250,0
[ ] P4	Amp	250,0
[ ] FC1	Amp	250,0
[ ] FC2	Amp	250,0
[ ] CP1	Amp	250,0
[ ] CP2	Amp	250,0
[ ] M1	Amp	250,0
[ ] M2	Amp	250,0
[✓] EOG1	Amp	250,0
[ ] EOG2	Amp	250,0
[✓] EMG1	Amp	250,0

Amplitude control

Display all channels | Hide all channels

Set all amplitudes to: 250,0

[✓] Use symetric amplitudes

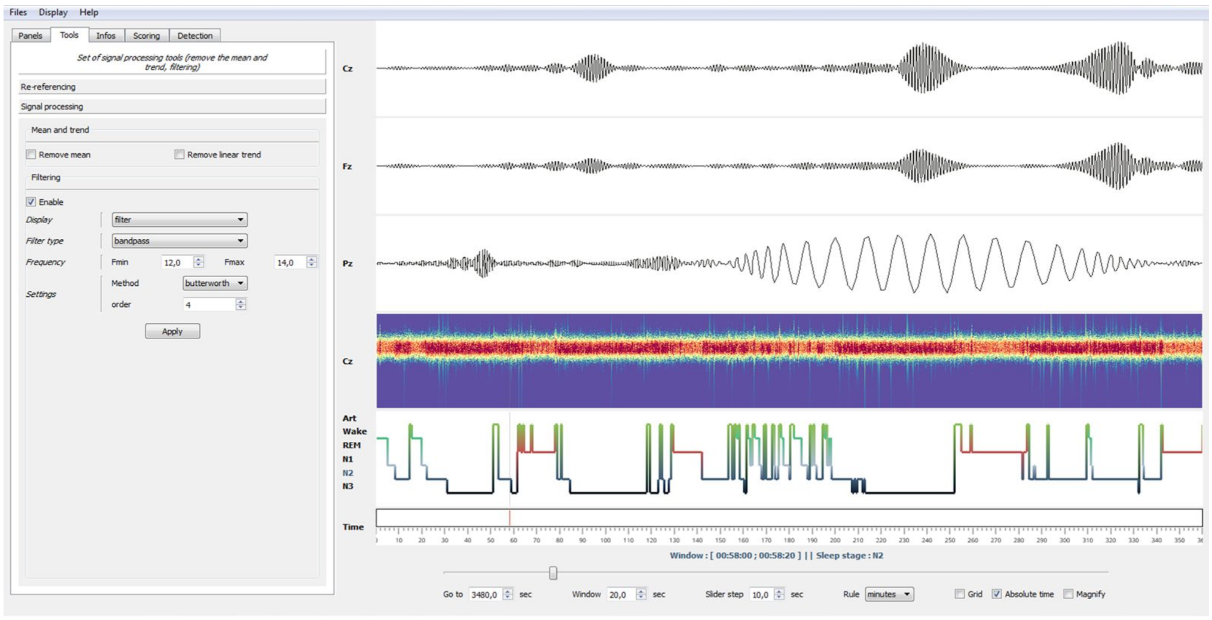
[ ] Use auto-ajusted amplitudes

Spectrogram // Hypnogram // Time axis

Topoplot

Go to 3480,0 sec | Window 20,0 sec | Slider step 20,0 sec | Rule minutes | [✓] Grid | [✓] Absolute time | [ ] Magnify

Figure 2 : *Sleep* main interface. Each element of the graphical user interface can either be displayed or hidden, (*A*) Settings panel containing all *Sleep* controls and parameters. The current displayed tab can be used to toggle channel visibility and to adjust individual amplitudes, (*B*) 30 seconds time window of electrophysiological data. Only 5 channels are currently displayed (Cz, Fz, Pz, EMG1), (*C*) The spectrogram displays the time-frequency representation of a specific channel for the entire recording, and can be useful to identify global changes in the spectral properties of the signal often associated with changes in sleep stages. Any channel can be picked and further time-frequency controls are available in the settings panel, (*D*) Hypnogram with one specific color per stage. The stage order can be changed from the default *Artefact, Wake, REM, N1, N2, N3* , (*E*) Time axis with visual indicator, (*F*) Navigation bar with time settings : window length and step size, unit (seconds/minutes/hours), (*G*) Topographic data representation



Set of signal processing tools (remove the mean and trend, filtering)

☐ Remove mean ☐ Remove linear trend

☑ Enable

Display: filter

Filter type: bandpass

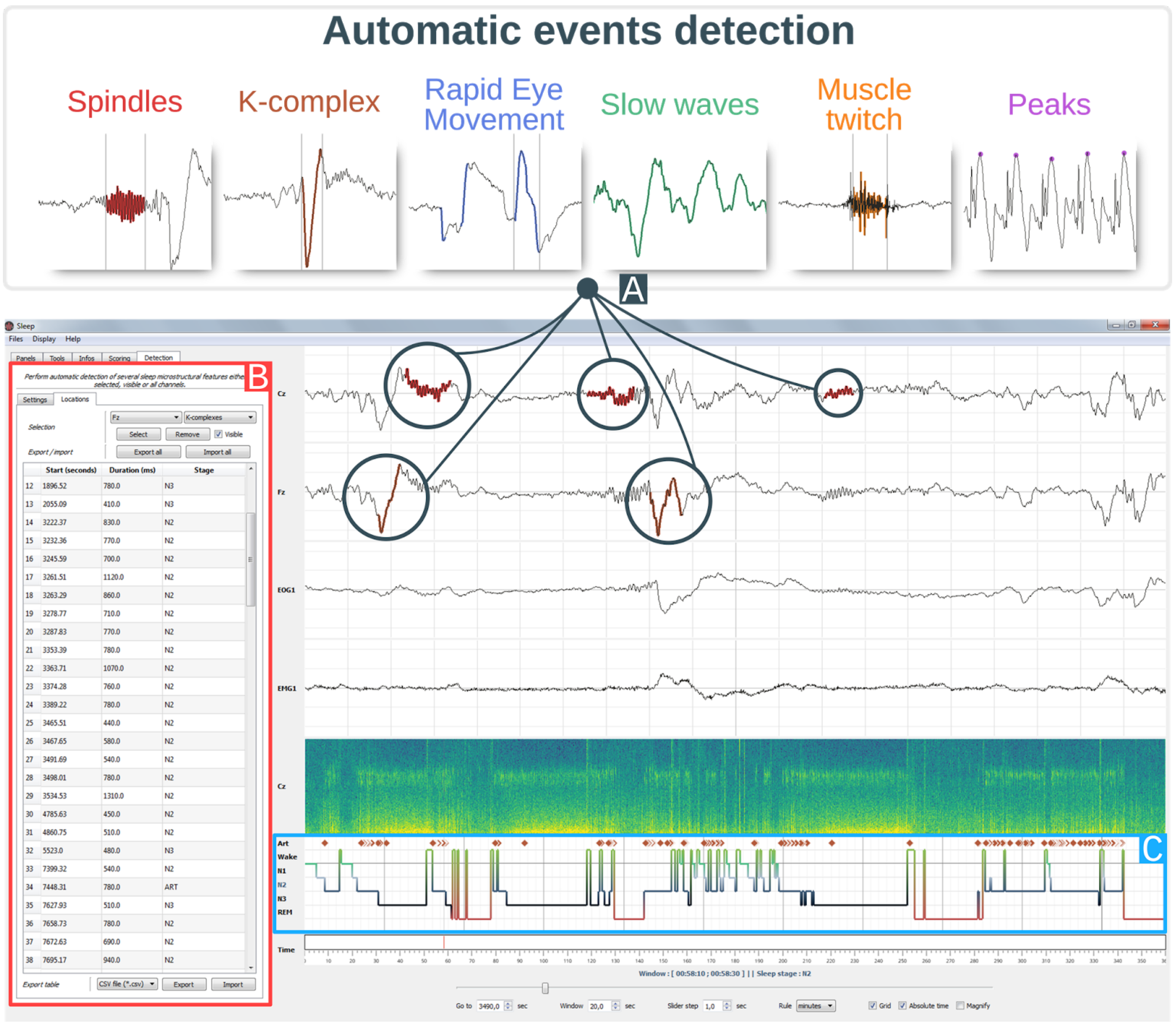
Frequency: Fmin 12,0 Fmax 14,0

Settings: Method butterworth; order 4

Window : [00:58:00 ; 00:58:20] | | Sleep stage : N2

Go to 3480,0 sec Window 20,0 sec Slider step 10,0 sec Rule minutes ☐ Grid ☑ Absolute time ☐ Magnify

Figure 3 : Example of bandpass filtering. Using the Tools panel (left), the EEG signals have been bandpass-filtered in the spindles frequency band (12-14 Hz, butterworth filter). Using the “Enable” checkbox of the panel, this filtering operation can be disabled at any moment to retrieve the original EEG signals. Finally, by left-clicking on a specific time point in a channel or selecting the Magnify tools (bottom), users can enlarge events. This was used in this example to enlarge a sleep spindle observed on channel Pz.



Perform automatic detection of several sleep microstructural features either selected, visible or all channels.

Selection: Fz — K-complexes — [✓ Visible]

Export / import

	Start (seconds)	Duration (ms)	Stage
12	1896.52	780.0	N3
13	2055.09	410.0	N3
14	3222.37	830.0	N2
15	3232.36	770.0	N2
16	3245.59	700.0	N2
17	3261.51	1120.0	N2
18	3263.29	860.0	N2
19	3278.77	710.0	N2
20	3287.83	770.0	N2
21	3353.39	780.0	N2
22	3363.71	1070.0	N2
23	3374.28	760.0	N2
24	3389.22	780.0	N2
25	3465.51	440.0	N2
26	3467.65	580.0	N2
27	3491.69	540.0	N2
28	3498.01	780.0	N2
29	3534.53	1310.0	N2
30	4785.63	450.0	N2
31	4860.75	510.0	N2
32	5523.0	480.0	N3
33	7399.32	540.0	N2
34	7448.31	780.0	ART
35	7627.93	510.0	N3
36	7658.73	780.0	N2
37	7672.63	690.0	N2
38	7695.17	940.0	N2

Export table: CSV file (*.csv)

Window : [00:58:10 ; 00:58:30] | | Sleep stage : N2

Go to 3490,0 sec — Window 20,0 sec — Slider step 1,0 sec — Rule minutes — [✓ Grid] [✓ Absolute time] Magnify

Figure 4 : GUI integration of the automatic event detection, (*A*) exemple of events detected on two channels. *Sleep* provide six detection types including spindles, k-complex, rapid eye movement, slow-waves, muscle twitch and peaks detection, and each one of them can be ran on any channel, (*B*) time location referencing all detected events and including the starting point, the duration and the sleep stage occurring during this event. A mouse click on a line center the corresponding event on the screen. This table can then be exported into *.csv or *.txt file, (*C*) hypnogram with reported events. Each detection type is identified using different symbols and colors.

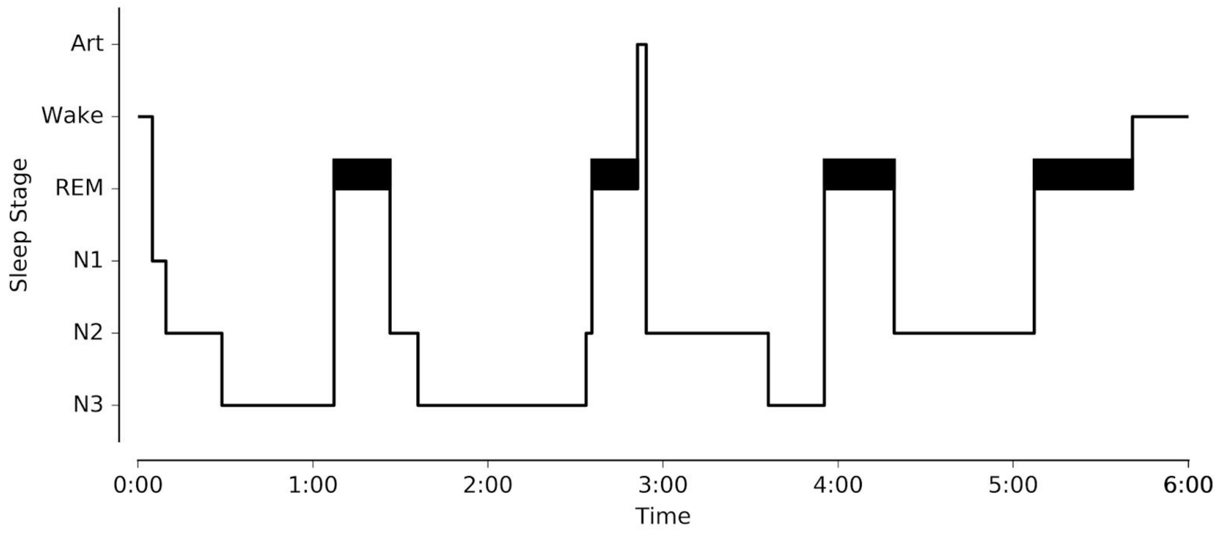


Figure 5 : Example of high-resolution publication-ready hypnogram figure exported using *Sleep* GUI.

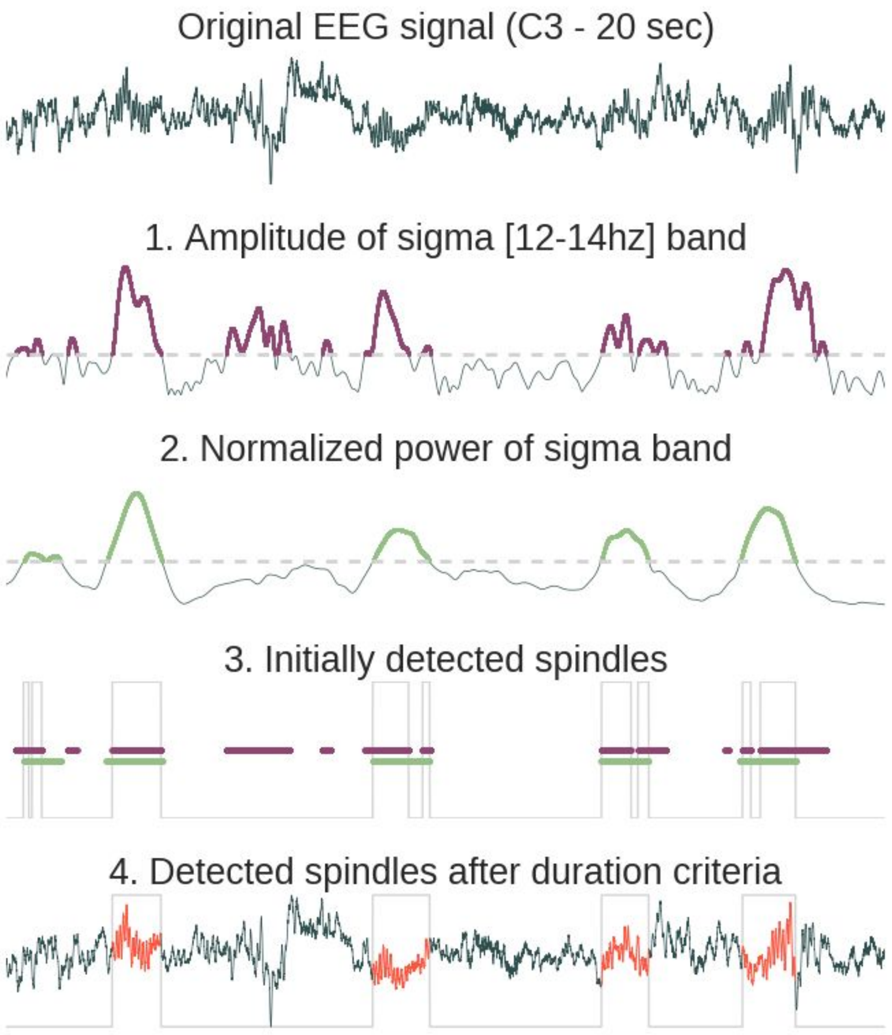


Figure 6. Method for the automatic sleep spindles detection. First, the original signal was convoluted with a Morlet wavelet centered in the spindles frequency band [12-14 Hz]. . From the resulting complex decomposition, we kept only the amplitude and find time indices where the amplitude exceeded the threshold (purple in *1.*). Then, we computed the normalized power in the sigma band and detected again time index where the power exceed a threshold (green in *2.*). The time location of the initial detected spindles (gray line in *3.*) is the result of the intersection of exceeding both the amplitude index (purple ligne) and the power index (green line). Finally, time gaps are filled only for proximate detected events (<500 ms) and a final duration criteria is applied in order to suppress events with a duration inferior to 500 ms or superior to 2000 ms (these thresholds are definable within *Sleep* interface, 4.).

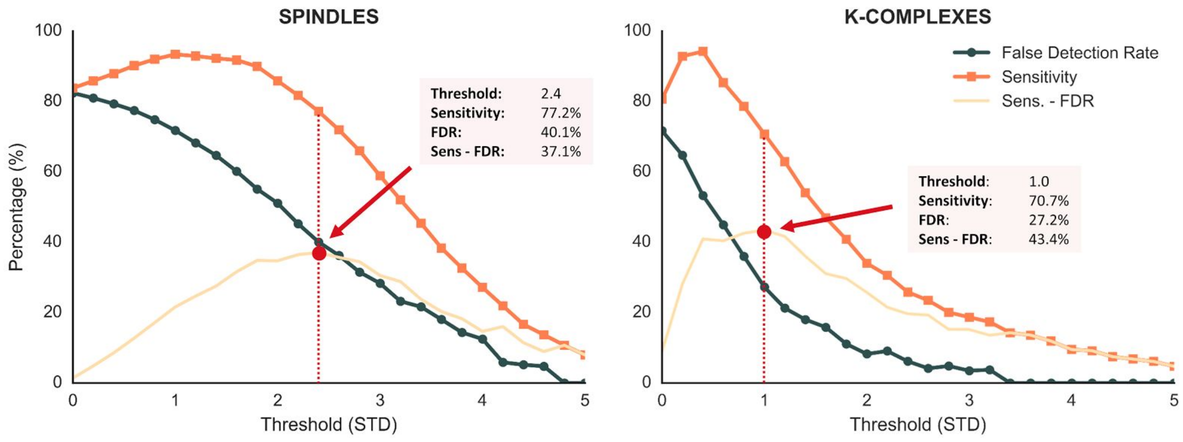


Figure 7. Performance metrics of our spindles and K-complexes detection methods evaluated at 25 different thresholds (range=0-5, step=0.2). Dark orange and blue lines depict the sensitivity and false detection rate (FDR), respectively. Light orange lines show the difference between sensitivity and FDR. Red dotted lines depict the threshold values that maximized this difference.

Code snippets

```python
# Load the Sleep module from visbrain :
from visbrain import Sleep

# Open the default Sleep window :
Sleep().show()
```

Code snippet 1: Simplest way to launch Sleep from a Python interpreter. This will open a window asking the user to select the EEG data and corresponding hypnogram.

```python
# Import the Sleep module from visbrain :
from visbrain import Sleep

# Define where the data are located :
dfile = '/home/perso/myfile.eeg'

# Define where the hypogram is located :
hfile = '/home/perso/hypno.hyp'
# hfile = None # Eventually, start from a fresh one

# Inverse the default sleep stage order :
norder = ['n3', 'n2', 'n1', 'rem', 'wake', 'art']

# Finally, pass both file to the class :
Sleep(file=dfile, hypno_file=hfile, href=norder).show()
```

Code snippet 2 : In this example, the paths to the EEG data and hypnogram are entered as inputs arguments of the main Sleep function, resulting in the software opening directly with the dataset and hypnogram loaded. We also show how to change the default display order of the hypnogram by changing the href argument of *Sleep* main function. The sleep stages will be displayed in the order defined in *norder* variable, with N3 on top and Art on bottom.

```
# Import the Sleep module and MNE:
from visbrain import Sleep
from mne import io

# - Biosemi Data Format (BDF)
raw = io.read_raw_edf('mybdffile.bdf', preload=True)

# - EGI format
# raw = io.read_raw_egi('myegifile.egi', preload=True)

# - EEGLab
# raw = io.read_raw_eeglab('myeeglabfile.set', preload=True)

# Extract data, sampling frequency and channels names
data, sf, chan = raw._data, raw.info['sfreq'], raw.info['ch_names']

# Now, pass all the arguments to the Sleep module :
Sleep(data=data, sf=sf, channels=chan).show()
```

Code snippet 3 : This example shows a method to pass data to *Sleep* after loading them using MNE-Python package (see http://martinos.org/mne/dev/manual/io.html for a full list of the data formats supported by MNE).

References

Berthomier, C., Drouot, X., Herman-Stoïca, M., Berthomier, P., Prado, J., Bokar-Thire, D., et al. (2007). Automatic analysis of single-channel sleep EEG: validation in healthy individuals. *Sleep* 30, 1587–1595.

Billinger, M., Brunner, C., and Müller-Putz, G. R. SCoT: a Python toolbox for EEG source connectivity. Available at: https://pdfs.semanticscholar.org/b196/7f587fbea9ecf4cb6be3f757a8136fc60ca8.pdf.

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].

Combrisson, E., and Jerbi, K. (2015). Exceeding chance level by chance: The caveat of theoretical chance levels in brain signal classification and statistical assessment of decoding accuracy. *Journal of Neuroscience Methods* 250, 126–136. doi:10.1016/j.jneumeth.2015.01.010.

Combrisson, E., Perrone-Bertolotti, M., Soto, J. L., Alamian, G., Kahane, P., Lachaux, J.-P., et al. (2017). From intentions to actions: Neural oscillations encode motor processes through phase, amplitude and phase-amplitude coupling. *NeuroImage* 147, 473–487. doi:10.1016/j.neuroimage.2016.11.042.

Devuyst, S., Dutoit, T., Stenuit, P., and Kerkhofs, M. (2011). Automatic sleep spindles detection—overview and development of a standard proposal assessment method. in *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE* (IEEE), 1713–1716.

Eichenlaub, J.-B., Bertrand, O., Morlet, D., and Ruby, P. (2014). Brain reactivity differentiates subjects with high and low dream recall frequencies during both sleep and wakefulness. *Cerebral Cortex* 24, 1206–1215.

Eichenlaub, J.-B., Ruby, P., and Morlet, D. (2012). What is the specificity of the response to the own first-name when presented as a novel in a passive oddball paradigm? An ERP study. *Brain research* 1447, 65–78.

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.

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.

Lajnef, T., Chaibi, S., Eichenlaub, J.-B., Ruby, P. M., Aguera, P.-E., Samet, M., et al. (2015a). Sleep spindle and K-complex detection using tunable Q-factor wavelet transform and morphological component analysis. *Frontiers in human neuroscience* 9.

Lajnef, T., Chaibi, S., Ruby, P., Aguera, P.-E., Eichenlaub, J.-B., Samet, M., et al. (2015b). Learning machines and sleeping brains: Automatic sleep stage classification using decision-tree multi-class support vector machines. *Journal of Neuroscience Methods* 250, 94–105. doi:10.1016/j.jneumeth.2015.01.022.

Rechtschaffen, A., and Kales, A. (1968). A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects.

Ruby, P., Blochet, C., Eichenlaub, J.-B., Bertrand, O., Morlet, D., and Bidet-Caulet, A. (2013a). Alpha reactivity to complex sounds differs during REM sleep and wakefulness. *PloS one* 8, e79989.

Ruby, P. M., Blochet, C., Eichenlaub, J.-B., Bertrand, O., Morlet, D., and Bidet-Caulet, A. (2013b). Alpha reactivity to first names differs in subjects with high and low dream recall frequency. *Frontiers in psychology* 4, 419.

Tallon-Baudry, C., Bertrand, O., Delpuech, C., and Pernier, J. (1996). Stimulus specificity of phase-locked and non-phase-locked 40 Hz visual responses in human. *The Journal of Neuroscience* 16, 4240–4249.

Vallat, R., Lajnef, T., Eichenlaub, J.-B., Berthomier, C., Jerbi, K., Morlet, D., et al. (2017). Increased Evoked Potentials to Arousing Auditory Stimuli during Sleep: Implication for the Understanding of Dream Recall. *Frontiers in Human Neuroscience* 11.