NeuroKit2: A Python Toolbox for Neurophysiological Signal Processing

Dominique Makowski [1,*], Tam Pham [1], Zen J. Lau [1], Jan C. Brammer [2], Hung Pham [3], Francois Lespinasse [4], Christopher Schölzel [5], & S.H. Annabel Chen [1, 6, 7]

[1] School of Social Sciences, Nanyang Technological University, Singapore

[2] ???

[3] ???

[4] Departement de psychologie, Universite de Montreal, Montreal, Canada

[5] Life Science Informatics, THM University of Applied Sciences, Gisslen, Germany

[6] Centre for Research and Development in Learning, Nanyang Technological University, Singapore

[7] Lee Kong Chian School of Medicine, Nanyang Technological University, Singapore

Author Note

* Correspondence concerning this article should be addressed to Dominique Makowski (HSS 04-18, 48 Nanyang Avenue, Singapore; dmakowski@ntu.edu.sg).

Abstract

The NeuroKit2 toolbox is an open-source Python package aimed at providing users with comprehensive and flexible functionality in neurophysiological signal processing. It developed from a collaborative project aimed at offering programming ease for both novice and advanced users to perform elaborate analyses of electrocardiogram (ECG), respiratory (RSP), electrodermal activity (EDA), and electromyography (EMG) data. It comprises of a consistent set of user-friendly, high-level functions that implements an all-in-one cleaning, preprocessing, and processing pipeline with sensible defaults. At the same time, greater flexibility and parametric control can be achieved by using Neurokit2's mid-level functions to build a custom analysis pipeline. (talk about novelty?)

Keywords: Neurophysiology, Biosignals, Python, ECG, EDA, EMG, RSP

Word count:

NeuroKit2: A Python Toolbox for Neurophysiological Signal Processing

Cognitive neuroscience and psychology is increasingly relying on neurophysiological methods to assess brain and bodily activity. Reasons can include low monetary cost (especially compared with other imaging techniques, such as MRI), high user convenience (e.g., portability, setup speed) and wide availability (e.g., in "smart" devices). At the same time, the fields of signal processing and computational data science continue to grow, pushing like never before the horizon of possibilities and opportunities. However, as these methods are often not easily accessible and user-friendly, neurophysiological data processing remains a challenge for many researchers without a formal training or experience in programming.

NeuroKit2 aims at addressing this gap by offering a free and user-friendly solution for neurophysiological data processing. It is an open-source Python package, developed in a collaborative environment that continues to welcome contributors from different countries and fields. Historically, *NeuroKit2* is the re-forged successor *NeuroKit.py* (*https://github.com/neuropsychology/NeuroKit.py*), a PhD side project that ended up attracting a lot of users and success (248 GitHub stars as of 09-04-2020). The new version takes on its best features and design choices, and re-implements them in a professional and well-thaught way. It aims at being 1) accessible, 2) well-documented, 2) reliable, 4) cutting-edge and 5) powerful.

The package is available for Python 3 (Van Rossum & Drake, 2009) and thus benefits from its important base of users, existing tutorials and large online community. It is also relatively lightweight, using mainly standard dependencies (Virtanen et al., 2020) such as *NumPy*, *pandas*, *SciPy*, *scikit-learn* and *MatplotLib* (with an additional system of optional dependencies), enabling its use as a dependency in other software. The package source code is available under a permissive license on GitHub (*https://github.com/neuropsychology/NeuroKit*); along with its documentation, automatically built and hosted at *https://neurokit2.readthedocs.io/*. Apart from instructions for installation and contribution, and a decription of the package's functions, the documentation also includes several "hands-on" examples and tutorials pro-

viding a walk-through on how to address specific issues (for instance, how to extract and visualize individual heartbeats, how to analyze event-related data, ...). New examples can be easily added by users simply by uploading a Python notebook file to the repository. This notebook file will be automatically transformed into a webpage and displayed on the website, ensuring a state of the art and evolutive documentation. The accessibility for newcomers is reinforced by the issue tracker of GitHub, where users can create public issues to inquire for help.

The package aims at being reliable and trustworthy, and its functions are tested against existing implementations of established reference software such as *BioSPPy* (Carreiras et al., 2015), *hrv under review*, *PySiology* (Gabrieli, Azhari, & Esposito, 2019), *HeartPy* (Gent, Farah, Nes, & Arem, 2019), *systole* (Legrand & Allen, 2020) or *nolds* (Schölzel, 2020). The code itself includes a comprehensive test suite to ensure stability and prevent error. Moreover, the issue tracker allows users to easily report any bugs and track their fixation. Thanks to its collaborative and open developpment, as well as its modular organization, *NeuroKit2* is being developped with a longterm perspective in mind, aiming at remaining cutting-edge through its ability to evolve, adapt, and integrate new methods as they are emerging.

Finally, we believe that the design philosophy behing *NeuroKit2* contributes to a powerful (allowing to achieve a lot with very few functions) yet flexible (enabling fine control and precision over what is done) user interface (API), which is described below.

Design Philosophy

NeuroKit2 aims at being accessible to beginners and, at the same time, offering a maximal level of control of experienced users. This is achieved via the implementation of 3 abstract levels of functions.

Low-level: Signal Processing Base Utilities

The basic building blocks are functions to facilitate general signal processing, i.e., to do filtering, resampling, interpolating, peak detection, etc. These functions are signal-agnostic, and include a lot of tweakable parameters. For instance, one can change the filtering method, frequencies, order etc. Most of these functions are based on validated algorithms present in *scipy* (Virtanen et al., 2020). Examples of such functions include `signal_filter()`, `signal_interpolate()`, `signal_resample()`, `signal_detrend()`. `signal_findpeaks()`.

Mid-level: Neurophysiological Processing Steps

The signal processing utilities are then used by functions specific to different types of signals. These functions aim at taking care of specific steps of physiological data processing, such as cleaning, peak detection, phase classification or rate computation. Critically, for each type of signals (ECG, RSP, EDA, EMG…), the same function names are called (in the form `signaltype_functiongoal()`) to achieve equivalent goals, such as `\*_clean()`, `\*_findpeaks()`, `\*_process()`, `\*_plot()` (replace the star with the signal type, e.g., `ecg_clean()`), making it intuitive and consistent to work with different signals.

For example, the `rsp_clean()` function uses `signal_filter()` and `signal_detrend()`, with different possible sets of default parameters that can be switched via a "method" argument (corresponding to different published or validated pipelines). For instance, setting `method=khodadad2018"` will use the cleaning workflow described in Khodadad et al. (2018). If a user wants to build its own custom cleaning function, he can reproduce the cleaning function but using the low-level signal processing tools with a specific set of parameters.

High-level Wrappers for Processing and Analysis

Finally, these steps are assembled in front-end "master" functions. For instance, the `ecg_process()` function uses `ecg_clean()`, `ecg_findpeaks()`, `ecg_rate()`, and the processing pipeline se-

lected via the `method` function is propagated throughout the different subsets. Last but not least, the package includes meta-functions (e.g., `bio_process`) that allows processing of multiple types of signals at once. As powerful as it might sound, this function basically combines the high-level function of each signal type into one output.

As a result, using *NeuroKit2* is very easy to use by beginners through the existence of these high-level functions, performing all of the steps of physiological preprocessing and processing with sensible defaults. Using one line of code (e.g., `bio_process(ecg=ecg_signal, eda=eda_signal)`), users can achieve a lot, which is rewarding and serves to demistify the usage of programming to newcommers. Importantly however, advanced users can very easily build their own custom analysis pipeline by using the mid-level functions that offer more control and flexibility over their parameters.

Example

We will present two examples that illustrate the most common use-cases. The first is an event-related paradigm, in which the interest lies in the momentarily short-term physiological changes related to specific stimuli, while the second shows how to extract the characteristics (features) of the physiological activity during a longer period of time (not necessarily tied to a specific and suddent event).

Event-related Paradigm

The data corresponds to ...

```
# Load the package
import neurokit2 as nk


# Download example dataset
data = nk.data("bio_eventrelated_100hz")
```

```
# Process the data
df, info = nk.bio_process(ecg=data["ECG"],
                          rsp=data["RSP"],
                          eda=data["EDA"],
                          sampling_rate=100)

# Find events
conditions = ["Negative", "Neutral", "Neutral", "Negative"]
events = nk.events_find(event_channel=data["Photosensor"],
                        threshold_keep='below',
                        event_conditions=conditions)

# Epoch the data
epochs = nk.epochs_create(data=df,
                          events=events,
                          sampling_rate=100,
                          epochs_start=-0.1,
                          epochs_end=4)

# Extract event related features
results = nk.bio_analyze(epochs)

# Show subset of results
results[["Condition", "ECG_Rate_Mean", "RSP_Rate_Mean", "EDA_Peak_Amplitude"]]
```

This extracts features like **blabla**.

Table 1

Subset of the ouput related to event-related analysis characterizing the pattern of physiological changes related to specific stimuli.

Condition	ECG_Rate_Mean	RSP_Rate_Mean	EDA_Peak_Amplitude
Negative	-1.94	-0.22	NaN
Neutral	-4.36	1.57	NaN
Neutral	1.02	-0.30	NaN
Negative	-3.61	2.22	1.68

Resting-state Features

The data corresponds to ...

```
# Load the package
import neurokit2 as nk


# Download example dataset
data = nk.data("bio_resting_5min_100hz")


# Process the data
df, info = nk.bio_process(ecg=data["ECG"],
                          rsp=data["RSP"],
                          sampling_rate=100)


# Extract features
results = nk.bio_analyze(df)


# Show subset of results
```

```
results[["ECG_Rate_Mean", "ECG_HRV_RMSSD", "RSP_Rate_Mean", "RSA_P2T_Mean"]]
```

Table 2

Subset of properties characterizing the physiological activity over a period of 5 minutes of resting-state.

ECG_Rate_Mean	ECG_HRV_RMSSD	RSP_Rate_Mean	RSA_P2T_Mean
86.42	4.28	15.86	0.01

This extracts features like **blabla**.

Conclusion and Future Directions

Despite not having a Graphical User Interface (GUI), *NeuroKit2* is accessible to people with very little knowledge of python or programming in general, thanks to its design choices focusing on user-experience.

Future evolution will mostly be driven by the community and the advances in the field. Possible directions include extending the support for other types of bodily signals (e.g., electrogastrography -EGG, electrooculography - EOG) and strenghtening the efficiency of the code to obtain performance gains for large datasets.

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.

Acknowledgements

We would like to thank all the contributors (https://neurokit2.readthedocs.io/en/latest/authors.html), and the users for their support.

References

Carreiras, C., Alves, A. P., Lourenço, A., Canento, F., Silva, H., Fred, A., & others. (2015). BioSPPy: Biosignal processing in Python. Retrieved from https://github.com/PIA-Group/BioSPPy/

Gabrieli, G., Azhari, A., & Esposito, G. (2019). PySiology: A python package for physiological feature extraction. In *Neural approaches to dynamics of signal exchanges* (pp. 395–402). Springer Singapore. https://doi.org/10.1007/978-981-13-8950-4_35

Gent, P. van, Farah, H., Nes, N. van, & Arem, B. van. (2019). HeartPy: A novel heart rate algorithm for the analysis of noisy signals. *Transportation Research Part F: Traffic Psychology and Behaviour*, *66*, 368–378. https://doi.org/10.1016/j.trf.2019.09.015

Khodadad, D., Nordebo, S., Mueller, B., Waldmann, A., Yerworth, R., Becher, T., … others. (2018). Optimized breath detection algorithm in electrical impedance tomography. *Physiological Measurement*, *39*(9), 094001.

Legrand, N., & Allen, M. (2020). Systole: A python toolbox for preprocessing, analyzing, and synchronizing cardiac data. Retrieved from https://github.com/embodied-computation-group/systole

Schölzel, C. (2020). NOnLinear measures for dynamical systems (nolds). Retrieved from https://github.com/CSchoel/nolds

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … Contributors, S. 1. 0. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/https://doi.org/10.1038/s41592-019-0686-2