Toolkit Comparison

A comparison of different toolkits as they relate to PeakAffectDS

# Neurokit2

The emg\_process() function takes an EMG signal, and a sampling rate. The EMG signal is a list containing all the values for the EMG recording (EMG\_zyg or EMG\_cor in the data), and the sampling frequency is the number of samples recorded per second (1 divided by the difference in time between samples). It then produces a dataframe with the following columns:

* **EMG\_Raw** – The unaltered signal provided (same as whatever input was used for input signal)
* **EMG\_Clean** – Cleaned signal, result of passing the raw signal through the function emg\_clean()
* **EMG\_Amplitude** – The amplitude of the EMG, result of passing the cleaned signal from emg\_clean() through the function emg\_amplitude()
* **EMG\_Activity** – The activity of the EMG signal, calculated by passing the amplitude (result of emg\_amplitude) through the emg\_activation() function and using a default threshold. The “activity” is a series of 1/0 values that identify if the EMG muscle is being activated, determined by if the amplitude is above a threshold value (1 for active, 0 for inactive)
* **EMG\_Onsets** – Onsets of the amplitude, locations where the amplitude began exceeding the activation threshold
* **EMG\_Offsets** – Offsets of the amplitude, locations where the amplitude ended exceeding the activation threshold

The emg\_process() function appears to follow a pipeline of loading raw EMG data, then passing it through a sequence of functions (emg\_clean, emg\_amplitude, emg\_activation), saving each result as a column in the data frame provided in the result. It’s worth noting that the emg\_clean function only has one option which assumes you are working with a 100 Hz power source. Since we have already cleaned the data using our own methods, and are not using a 100 Hz power source, I moved on to explore the results of skipping the cleaning steps and putting our generated clean data through the emg\_amplitude() and emg\_activation() functions.

Amplitude graph using the raw data:

A picture containing text, diagram, screenshot, line

Description automatically generated

Amplitude graph using the cleaned data:

A picture containing text, screenshot, plot, diagram

Description automatically generated

The results appear nearly identical. My next thought is to explore the emg\_analyze() function. It has proven difficult to put the data into a form that can be used without using the emg\_clean function, so for the time being I am using the proprietary functions just to get results.

# PhysioData

**Setup**

While downloading the PhysioData Toolbox, I found that it wasn’t necessary to install the paid version of MATLAB. You only need to install a freely available MATLAB runtime. The introduction also notes that it is necessary to use a 3-button mouse (mouse with clickable scroll wheel) to make use of all the features offered.

**Loading The Data**

PhysioData can only read PhysioData files, a special format of MATLAB files with the “physioData” extension. To load data into PhysioData, you must first convert it using the file converter in the menu of the session manager. PhysioData can convert files of the following formats:

* BIOPAC
* VU-AMS
* LIBC Achieva MRI
* Biosemi
* EET Output
* EyeLink

If the data is not stored in any of these file formats, it is still possible to load the data using a custom MATLAB script. Since we are using .csv files, we will have to either convert to one of the available formats, or find/create such a custom MATLAB script.

PhysioData files are MATLAB MAT v7.3 files that are given the “physioData” extension. Data is stored using MATLAB types (structs, cell arrays, tables, etc.), and can hold additional metadata like timing. Notes about the file format:

* Cannot use strings, only character arrays (use single quotes). Arrays of strings are represented with cell arrays of characters.
* All signals must have the same sampling frequency and number of samples, with the sampling frequency (pdtData.data.signals.fs) being between 10 and 10,000 Hz.
* When loading and saving PhysioData files, the toolbox strips base variables not part of the file specifications

PhysioData file content outline:

* **pdtData:** [1 x 1 struct] A scalar struct holding the contents of a PhysioData file.
  + **data:** [1 x 1 struct] A scalar struct holding all raw data, including events.
    - **labels:** [1 x 1 struct] Labels (events) associated with the signals inside the PhysioData file. This can be omitted if no labels are present.
      * **t:** [k x 1 double] A single column vector with times (in seconds) of the labels.
      * **channels:** [1 x 1 cell] Single cell that holds a [k x 1 cell] cell array of char arrays, each element being the name/value of the label.
      * **channelUnits:** [1 x 1 cell] A single cell containing the char “-“ (labels don’t have units).
      * **channelNames:** [1 x 1 cell] A single cell containing the following char array: “Labels”.
    - **signals:** [1 x 1 struct] Scalar struct holding raw signal data in a collection of channels.
      * **fs:** [1 x 1 double] Scalar double indicating the sampling frequency of the signals in Hz. The time vector is calculated with the first sample at t=0 and a sample interval of 1/fs.
      * **channels:** [1 x k cell] Cell array holding one signal per cell (one time-series per channel), all channels must contain the same number of samples in a single column, and all signals must have the same time-vector.
      * **channelUnits:** [1 x k cell] Cell array with k char arrays indicating the units of each signal, can be an empty char if the signal has no unit.
      * **channelNames:** [1 x k cell] Cell array with k char arrays indicating the name of each signal.
      * **channelDescription:** [1 x k cell] Cell array with k chars indicating the description of each signal.
  + **epochs:** [1 x 1 struct] Scalar struct holding pre-generated epochs. Can be omitted if none are present.
    - **epochData:** [m x n table] Table containing pre-generated epochs. The table must have the following variables:

**startTime:** [m x 1 double] Specifies the start time of the epoch in seconds.

**endTime:** [m x 1 double] Specifies the end time of the epoch in seconds.

**epochName:** [m x 1 cell array of chars] Specifies the name of the epoch.

The table can also contain extra metadata columns, which are copied to the output file. Metadata columns must be a single column double vector, or single column cell array of chars.

“duration”, “dataSource”, “error”, etc. variables are added by the toolbox.

* + **physioDataInfo:** [1 x 1 struct] Scalar struct holding metadata about the file, it and each of its fields are entirely optional.
    - **rawDataSource:** [1 x m char] Name of the original raw file.
    - **pdtFileCreationDate:** [1 x m char] File creation timestamp (no specific time format required).
    - **pdtFileCreationUser:** [1 x m char] Name of the user that created the file.
  + **UserData:** [1 x 1 struct] Scalar struct holding user data. Copied to the output when generating a Data Archive. Can be omitted if irrelevant.
  + **physioAnalyzerTemplates:** [1 x 1 struct] Scalar struct holding the PhysioAnalyzer module’s settings and state, not intended to be user modifiable.

Documentation for PhysioData file contents: <https://physiodatatoolbox.leidenuniv.nl/docs/user-guide/physiodata-file-format.html>

Based on the description of the internal storage system of the physioData files, it seems to follow a similar approach as the .csv files we have already, so making a custom script to convert these files to the appropriate format seems to be a viable possibility.

Documentation for how to make a custom script to create physioData files: <https://github.com/physiodatatoolbox/code_examples/blob/main/Generating_PhysioData_Files/Basic_Example_1/main.m>

**Analysis**

Once the data is loaded, I then tried to generate an PhysioAnalyzer module for the EMG signals. PhysioAnalyzer modules have the following settings when being created:

* General settings (channel to apply analysis on, analyzer prefix, etc.)
* Preprocessing settings (optional)
* Smoothing settings – either a boxcar filter or lowpass FIR filter for smoothing

To produce an analysis, it is necessary to use one of the provided smoothing settings even if no smoothing is required. For my investigation, I used the default settings (100 Hz boxcar filter).

The analysis created the following graph:

A screen shot of a graph

Description automatically generated with medium confidence

The following result parameters were generated from the analysis:

* Epoch\_ID
* startTime (s)
* endTime (s)
* duration (s)
* epochName
* dataSource
* error
* epochSource
* startActualValue
* startActualOccur
* startActualDelay (s)
* startActualDur (s)
* endActualValue
* endActualOccur
* endActualDelay (s)
* endActualDur (s)
* Min\_Smooth\_EMG (mV)
* Max\_Smooth\_EMG (mV)
* Mean\_Smooth\_EMG (mV)
* BoundedArea\_Smooth\_EMG (mV/s)
* TimeToMax\_Smooth\_EMG (s)
* Min\_Filt\_EMG (mV)
* Max\_Filt\_EMG (mV)
* Mean\_Filt\_EMG [mV]
* Min\_FiltRect\_EMG [mV]
* Max\_FiltRect\_EMG [mV]
* Mean\_FiltRect\_EMG [mV]
* Interpolated\_Smooth\_EMG [%]

The following information parameters were generated from the analysis:

* File
* analyzerType
* dataError
* dataSource
* epochsToFind
* epochsResolved
* acceptance
* HPfilterFIR [Hz]
* LPfilterFIR [Hz]
* Channel
* epochSettings
* filtType
* gain [x]
* notchFilt [Hz]
* smoothDeg [ms or Hz]
* smoothSkips [s]

Most of the analysis parameters appear to be related to the produced “filtered”, “filtered and rectified”, and “filtered, rectified and smoothed” plots in the figure.