Instructions for REMs analysis using automatic peak detection

Scripts required:

automatic_peak_detection.py

rems_analyse.py

rems_functions.py

Overview

What is automatic eye movement detection?

Eye movements in a signal can either be identified by a person marking the peaks in the data corresponding to eye movements (manual detection), or by an peak detection algorithm (automatic detection) using pre-defined thresholds.

Why are we doing automatic eye movement detection?

1. To make things quicker, more consistent and (in theory) easier

What do I need to perform automatic eye movement detection?

- A Python IDE (e.g. PyCharm)
- The following data files from a participant:
  - Hypnogram file (.csv format)
  - EEG data file (.edf format)
  - EOG data file (.edf format)
- The following python scripts downloaded into your PycharmProject folder:
  - automatic_peak_detection.py - you will 'interact' with this file by changing the paths at the start of the file
  - rems_analyse.py - you do not need to 'interact' with this file in any way
  - rems_functions.py - you do not need to 'interact' with this file in any way

Running SECTION 1

Section 1 imports modules, specifies file paths, aligns the sleep hypnogram to the EOG data & extracts REM periods

1. Open the manual_peak_detection.py file in PyCharm

2. Make sure you have all the modules imported (**lines 9-22**).
   - To import modules, open the PyCharm terminal & type ‘pip install [name_of_module]’ & press enter

3. Specify the paths of the files you will be analysing (**lines 25-31**)
   - Remember to check the spelling of your file names in PyCharm
   - Remember to include the file type in the string (e.g. ‘scoring_vid.csv’)

4. Copy **lines 9-34**, paste into the Python Console & press enter

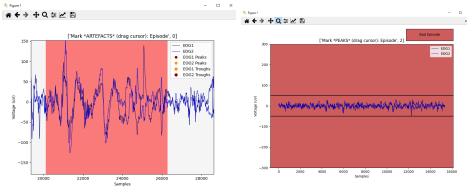
```
"""
script to automatically identify eye movements during REM sleep & analyse
author: @ggpr141]
last updated: 01/12/12
"""
# ----------------------------------------------------------------------------------------------------------------------
# SECTION 1  - import modules, specify file paths, align hypnogram to EOG file & extract REM periods

# import modules - pip install if missing
import os
from pathlib import Path
import mne
import numpy as np
import pandas
from matplotlib import pyplot as plt
import scipy
import neurokit2 as nk
import statistics
import matplotlib.widgets as mwidgets
from rems_analyse import *
from rems_functions import *
import warnings
warnings.filterwarnings("ignore")

# specify paths
path = Path('Y:/22qEEG/E006-1-1-1')  # define path to file
os.chdir(path)  # change working directory to path
hypnogram = 'scoring_outputs/E006-1-1-1_scoring_info_vid.csv'  # define hypnogram path
EEG = 'exported_data/E006-1-1-1_sleep_EEG_PREP.edf'  # define EEG data path (for hypnogram alignment)
EOG = 'exported_data/E006-1-1-1_PSG.edf'  # define EOG data path
sampling_freq = 256  # define sampling frequency for EOG data

# preprocess data & extract rem periods for each eog channel
rem_episodes_e1, rem_episodes_e2, rem_timings = extract_rem_episodes(EEG, EOG, hypnogram)
```

Running SECTION 2

Section 2 opens several plots of data so you can 1. manually identify eye movements 2. mark bad episodes 3. mark artefacts. The script then automatically extracts eye movement, cluster & microstate characteristics

1. Copy **lines 37-107**, paste into the Python Console & press enter

2. For each episode, a graph will open with the detected peaks & each episode's data plotted. You have 2 choices:
   - Manually mark artefact regions
   - Mark the entire episode as a 'bad' episode



3. If you mark the episode as 'bad' then that episode is ignored for all future analyses

4. All REMs characterisation is then calculated automatically for the episode

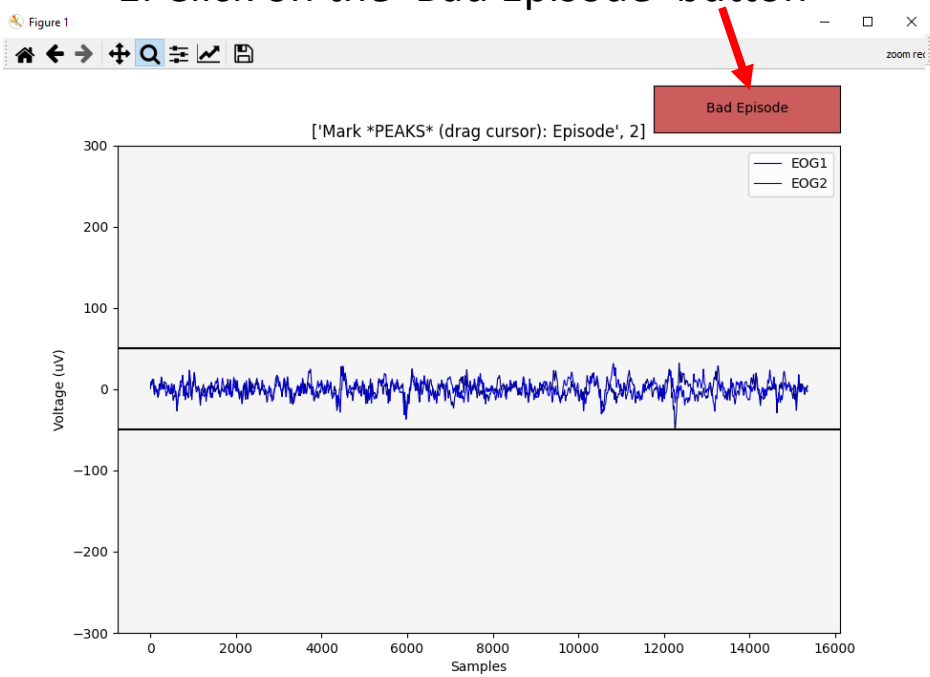
5. This process repeats automatically until all the episodes have been gone through

```
#-------------------------------------------------------------------------------------------------------------
# SECTION 2  - manual identification of peaks, mark bad episodes, mark artefacts, calculate rem, cluster & microstate characteristics

# initialise variables to append to later
sys.stdout = open('automatic_processing_output.txt', 'w')
episode_count = 0
rems_df = pandas.DataFrame()
rems_cluster_df = pandas.DataFrame()
episode_list = []
bad_episode_list = []
good_episode_list = []
ep_list = []  # initalise episode list
pp_list = []  # initialise phasic percentage list
tp_list = []  # initialise tonic percentage list
ap_list = []  # initialise artefact percentage list
td_list = []  # initialise total duration list
ttd_list = []  # initialise total ton. duration list
tpd_list = []  # initalise total phas. duration list
tad_list = []  # initialise total art duration list
rems_microstates_df = pandas.DataFrame()

# automatic peak detection for each REM episode - you mark 'bad episode' or artefacts. automatic analysis follows
for episode in range(len(rem_episodes_e1)):
    try:
        clean_e1, clean_e2, channel_crossings, e1_peaks, e1_troughs, e2_peaks, e2_troughs = \
            matched_peaks_detection(episode, rem_episodes_e1, rem_episodes_e2)

        start_art, end_art, bad_episode_list, good_episode_list = mark_bad_or_artefact(e1_peaks, e2_peaks, e1_troughs,
                                                                                        e2_troughs, clean_e1, clean_e2,
                                                                                        episode, bad_episode_list,
                                                                                        good_episode_list)
        episode_count += 1
        # if episode is marked 'bad' it is excluded from further analysis
        if episode in bad_episode_list:
            print('episode', episode, 'is a bad episode. no analysis done.')
        # if episode is marked 'good' automatic analysis follows.
        else:
            e1_peaks, e1_troughs, e2_peaks, e2_troughs = remove_art_peaks(e1_peaks, e2_peaks, e1_troughs, e2_troughs, start_art,
                                                                          end_art, episode)

            rems_df_e1p = rems_analyse(clean_e1, e1_peaks, channel_crossings, 'Left', 'Left', e1_peaks, e1_troughs, e2_peaks, e2_troughs,
                                       rem_episodes_e1, episode, invert=False)

            rems_df_e1t = rems_analyse(clean_e1, e1_troughs, channel_crossings, 'Left', 'Right', e1_peaks, e1_troughs, e2_peaks,
                                       e2_troughs, rem_episodes_e1, episode, invert=True)

            rems_df_e2p = rems_analyse(clean_e2, e2_peaks, channel_crossings, 'Right', 'Right', e1_peaks, e1_troughs, e2_peaks,
                                       e2_troughs, rem_episodes_e2, episode, invert=False)

            rems_df_e2t = rems_analyse(clean_e2, e2_troughs, channel_crossings, 'Right', 'Left', e1_peaks, e1_troughs, e2_peaks,
                                       e2_troughs, rem_episodes_e2, episode, invert=True)

            rems_df = rems_df.append([rems_df_e1p, rems_df_e1t, rems_df_e2p, rems_df_e2t])

            remgram, rems_clusters, tonic_percentage, phasic_percentage, art_percentage, total_duration, total_ton_dur, \
            total_phas_dur, total_art_dur = phasic_tonic_detections(e1_peaks, e2_peaks, e1_troughs, e2_troughs, clean_e1,
                                                                    episode, start_art, end_art)

            rems_cluster_df = rems_cluster_df.append(rems_clusters)

            ep_list, tp_list, pp_list, td_list, ttd_list, tpd_list, ap_list, tad_list = initialise_tp_micro(
                episode, tonic_percentage, phasic_percentage, ep_list, tp_list, pp_list, td_list, ttd_list, tpd_list,
                total_duration, total_ton_dur, total_phas_dur, ap_list, tad_list, art_percentage, total_art_dur)

            episode_count += 1

        rems_microstates_df = rems_microstates(ep_list, tp_list, pp_list, ap_list, td_list, ttd_list, tpd_list, tad_list)

    except:
        print('episode', episode, ': something went wrong in the code. No analysis done. Analyse independently')
        episode_count += 1
sys.stdout.close()
```

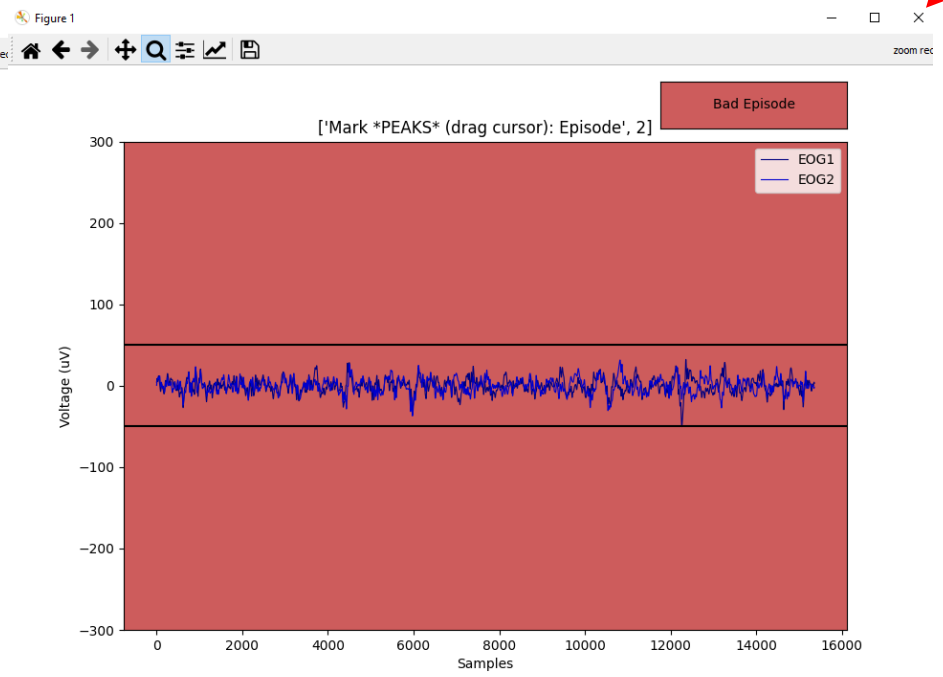
Spotlight on: Marking a Bad Episode

Bad episodes are where the majority/all of the data has an artefact (ECG bleed-through, for example) & you are unable to detect eye movements.

1. Click on the ‘Bad Episode’ button



2. The entire graph will go red

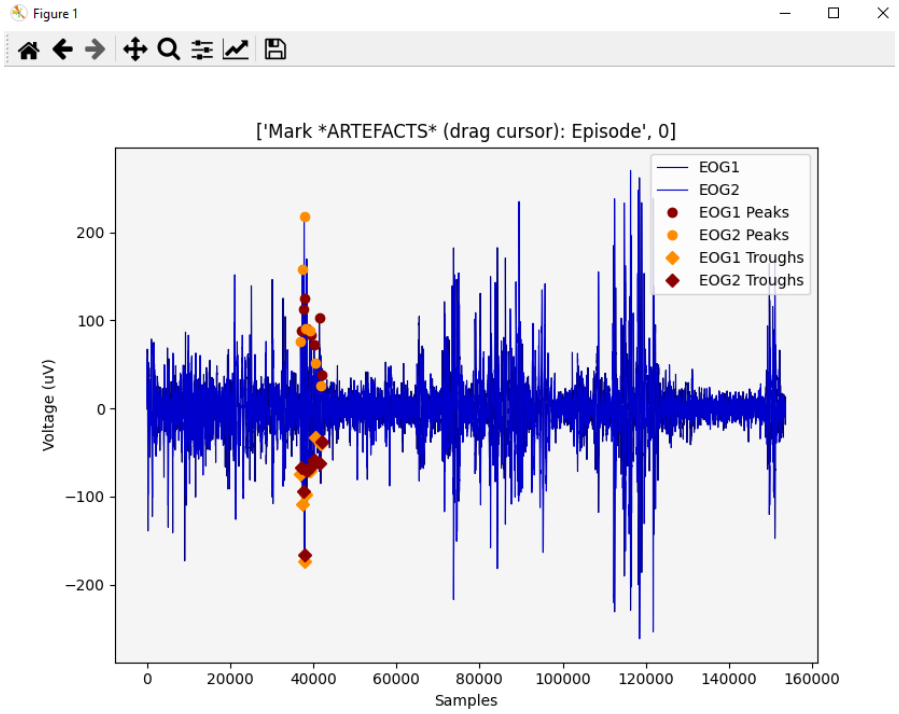


3. Exit the graph. This episode is now marked as ‘bad’ and will be ignored in all future analyses

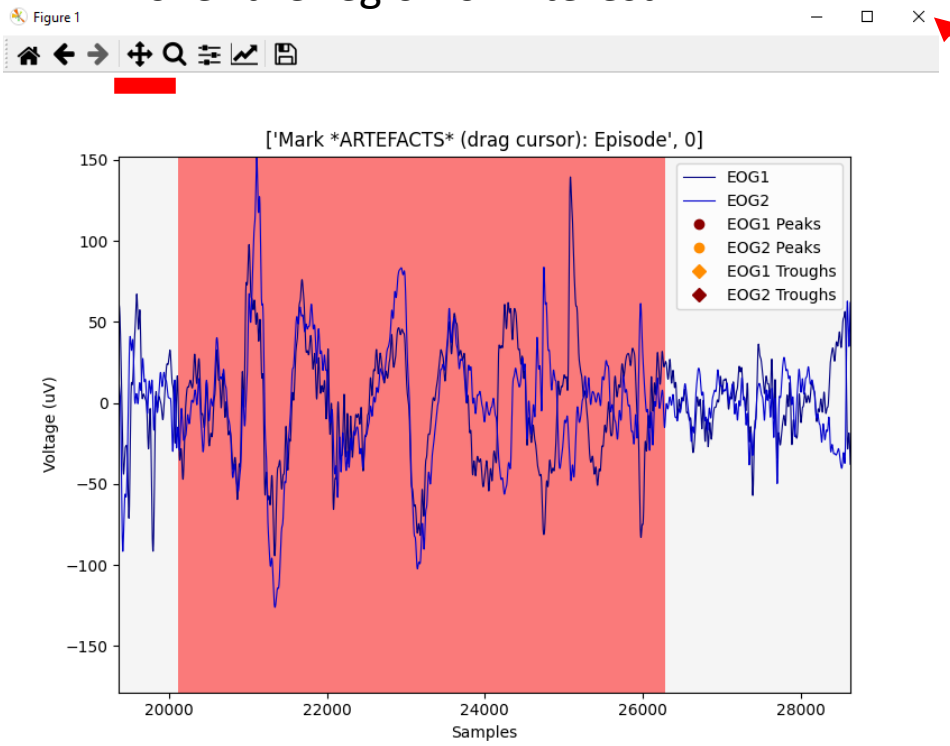
Spotlight on: Marking Artefacts

Manual identification of artefacts relies on you being confident in what an eye movement is- see Movement_V_Artefact_Identification.pdf for more guidance

The episode will be plotted & any manually identified eye movements will be plotted



Using the zoom & move functions of the GUI you can then visualise the data. Mark any artefact sections by dragging your cursor over the region of interest



* Once you have marked all the artefacts in the episode, exit the graph and the automatic analyses of the eye movements will happen

Running SECTION 3

Section 3 saves the output files at the end of the analyses, as well as a 'manual_processing_output.txt' file which details which episodes were bad & the percentage of peaks rejected due to artefacts/episode.

```
109  # ------------------------------------------------------------------------------------------------------------------------
110  # SECTION 3  - save outputs to csv files
111
112  # save individual eye movement, cluster & microstate characteristics to .csv file format into participant folder
113  rems_df.to_csv(path_or_buf='rems_characteristics.csv', index_label='Peak Number in Ep')
114  rems_cluster_df.to_csv(path_or_buf='rems_clusters.csv', index=False)
115  rems_microstates_df.to_csv(path_or_buf='rems_microstates.csv', index=False)
116
```

1. Copy **lines 110-115**, paste into the Python Console & press enter
   - The following files will be saved to the participant's folder:
     - automatic_rems_characteristics.csv - contains characteristics for each 'peak' (i.e. eye movement) during REM sleep episode
     - automatic _rems_clusters.csv - contains characteristics for each 'cluster' (i.e. group of eye movements) during a REM sleep episode
     - automatic _rems_microstates.csv - contains data on macro characteristics of each episode
     - automatic _processing_output.txt - contains information on the analysis status of each episode (e.g. bad episode, code bug & episode analysis skipped, analysis completed)

Questions

What if I make a mistake with my selections?
There is no ‘undo’ button on any of the graphs, so if you make a mistake you can either a) continue with the analyses or b) close the python console in PyCharm and start the entire analysis again. It is up to you which you do, if you are on the final episode it might be better to continue your analyses and make a note of your mistake.
The occasional incorrectly selected artefact shouldn’t impact the overall results as you will be calculating averages.

What if there is an error in the analysis pipeline?
There is a mechanism built in to the code so that if any error comes up (unforeseen calculation bugs etc) then the entire episode will be skipped and you will move onto the next REM episode. Sometimes strange errors appear due to small differences between participants and it is hard to predict. If an episode is skipped, this will be shown in the ‘automatic_processing_output.txt’ file. You can then analyse that episode individually using ‘automatic_peak_detection_single_episode.py’.

How does ‘automatic_peak_detection_single_episode.py’ work?
Exactly the same as automatic_peak_detections.py, except you only analyse one REM episode instead of looping through all the episodes.
All you need to do is define your paths (**lines 30-34**) & define which episode you want to analyse (**line 36**). Then copy & paste the code as before. All the outputs will be saved with the episode number specified in the file name.