Major extrinsic sleep EEG artifact detection (sleep_EEG_loose_lead_detect)

Package Guideline created by
Dr. Anandanadarajah Nishanth

Project Group members
Dr. Amlan Talukdar, Dr. Deryck Yeung, Dr. Yuanyuan Li , Dr. David Umba,
Dr. Leping Li, Dr. Zheng (Jane) Fan, MD
October 1, 2024

Preface

This guide-line is **under construction**. The aim is to make the life easier for the package users. Thus the guideline is written in kind of discussion, so just present the general facts used by the research community.

This guide line is made for introduction to use the package. So it gives the broad picture of the overall sleep-EEG studies, and the proposed medthodologies and their parameters.

The main study is in review. Once it is published then the proper article will be shared. Use the package with knowing the **limitations of the package** to avoid the pithalls.

Apart from the peer-reviewed study, intial study and their findings with using the correlation combintations are presented in the following abstract. The package include the methodlogies used in the pilot study as well as the peer-reviewed study.

Abstract-pilot-study

Automatic Non-physiological artifact detection in sleep-EEG

Sleep-EEG data set non physiological (extrinsic) artifacts like loose-lead are unavoidable due to the EEG-recordings collected in subject's sleep for longer period. Due to growing dataset of sleep-EEG, manually identify and remove these structural artifacts from the EEG-sleep recordings nearly impossible due to the temporal and spatial presence of EEG data. The presence of the bigger extrinsic artifacts has higher chance to cause the biased finding in the sleep-EEG based studies. Earlier evidence supports, there is spectral correlation exist between the EEG-channels in sleep.

Channels' MT-spectrums' correlation based moving median window approaches are proposed to find outliers. Approach-1 consider channel-combinations independently, and Approach-2 uses the mean of the combinations, to find the outliers.

The detected outliers with the convolutional window on approaches, are used to point the suspected artifact present channel and temporal location. The sensitivity and specificity of the suspected artifacts can be adjusted by the parameters such as local-window size, threshold-factor, global-parameters, convolution window size, and combine strategy.

We conclude that correlation drop present between the spectrum of loose channel and good channel on the channel's loose period. Our proposed approaches will save the tremendous time of experts to go through the good EEG-recordings to check the structural artifact presence. Further our proposed index can be utilized to measure the performance on big EEG-sleep data without ground truth for this problem.

Contents

Chapter 1

Introduction

This introduction section gives broad picture of sleep EEG study.

The studies till now using the sleep EEG data, into main broad categories such as,

- ✓ Artifact detection/ removal
- ✓ Sleep feature extraction (e.g: spindle, PDR, etc.)
- ✓ Sleep stage prediction
- ✓ Disease classification
- ✓ Age prediction

These studies can be combined and build like multimodal analysis. One of the common example artifact removal and the feature extraction.

1.1 Artifact removal and Feature extraction

1. Extract the same sample rate -256 EEG data
2. Check the power line noise exist in the epoch, If so, notch filter used to remove the power noise.
3. Then band pass the sleep data (e.g: 0.5-30Hz), to remove the DC shift and the unwanted portions; though Dr. Jane suggested to use above 0.1 Hz; this is the purpose of visualise the signal in time domain effectively. This has to be done in the beginning stage of pre-processing, else we need 10 sec window to achieve this.
4. Then normalize the data to make them in unique way
5. Use the extracted feature for classification.

1.1.1 Main choices in pipeline

After removal of power line noise, the choices we make may impact on the results, such that the choices are categorised mainly as

- ✓ Using band-pass filter (choice of filter); and where we use this filter in the pipe-line
- ✓ Re-reference the channel.
- ✓ Artifact removal
- ✓ Normalisation method
- ✓ Feature extraction choice.

Here the choice of re-reference channel, or using the channel as it is also make some difference in performance. Normalised method like Z-normalization or min-max normalisation and the normalization procedure on the pipeline position. And finally the major choice of the feature extraction method, can be mainly categorised as,

- ✓ Time-domain features
- ✓ Frequency domain features
- ✓ Time-Frequency analysis

If we choose time-frequency analysis then that can be categorised further as

- ✓ Then use the multi spectral analysis to extract the EEG features
- ✓ wavelet transform as well for feature extraction.
- ✓ Maybe we can combine them

Apart from these choices EEG signal may have some relation between the age, and/or sex. So do we need to handle them accordingly. The following introduction shows the possible i/p choices and effects when the data used blindly with their effects. You can ignore the following section, this is shown to provide the overall effects representation.

1.1.2 Channel selection and Re-reference choice

Channel selection is based on the question of how we utilise the information spread across the channels, just averaging is not a good choice, if one channel has more information in specific rhythm, we may suppress that information of that channel. We can use deep learning to handle the weights of channels (not presented in the package). However the channel information we are going to present make impact. Such that re-referencing or not, further how and where (among the single epoch/ using the subjects full sleep data) normalise.

This is quiet tricky due to the choice we may loose/ gain some information. Like re-reference remove the dc-shift and artifacts by the subject movements. In trade of the noise on the reference channel

may suppress the useful information exist on the channel we are interested, as well as sometime reduce the channel power by re-referencing.

Let's go through some of the data to make the choice. In-order to do this first used the EEG signal pre-processed up-to band passed (0.5-25Hz) signal (including band-pass), obtained the 3 quantiles such as first q1 at 0.25, q2 at 0.5, and q3 at 0.75 quantiles value spread.

1.1.3 Variation of different choices via the histograms

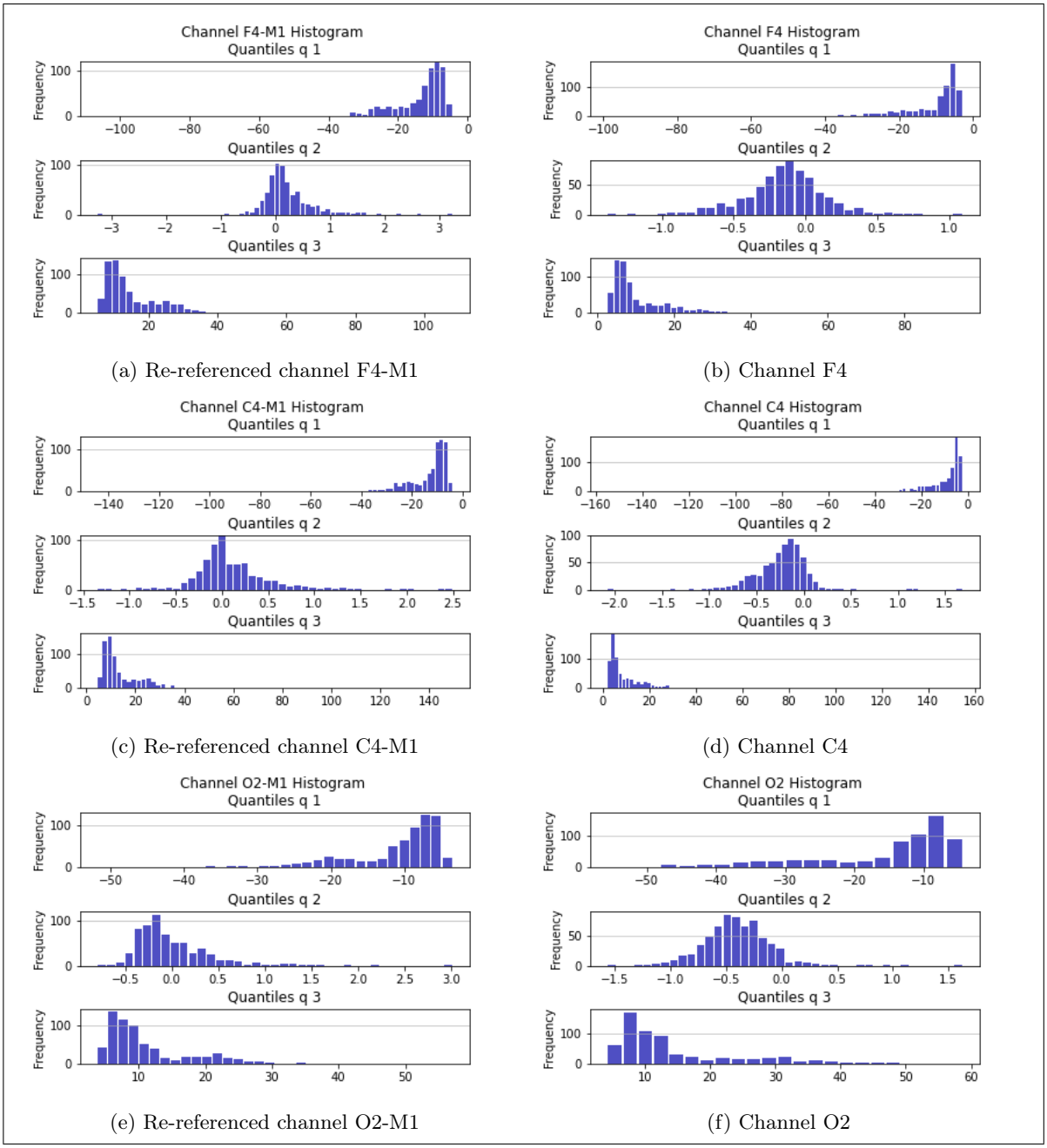
These values histograms are obtained separably to check the variation. If we normalise them blindly we may comparing the different kind of EEG signals via same scale. Such that this give us an idea about the re-referencing and averaging choice.

From the results of histograms in Figures. 1.1 there seems little shifts along the power values. It seems does not effect much in this view.

So we have initiated to check whether sex ("Male" subjects EEG signal vs Female Subjects EEG signal) histogram shows any difference. However their is not much, so not presented in this report.

Thus age-wise comparison is performed, in the age boundary at 15, and the results are shown in the Figure. 1.2. From this initial outcome from Fig. 1.2 clearly see the when the age get older the amplitude distribution decreases.

In-order to evaluate the hypothesis further, age wise separation histogram performed on 30 boundary. This time it has shown some above 30 shows some distribution power reduction.



(a) Re-referenced channel F4-M1

(b) Channel F4

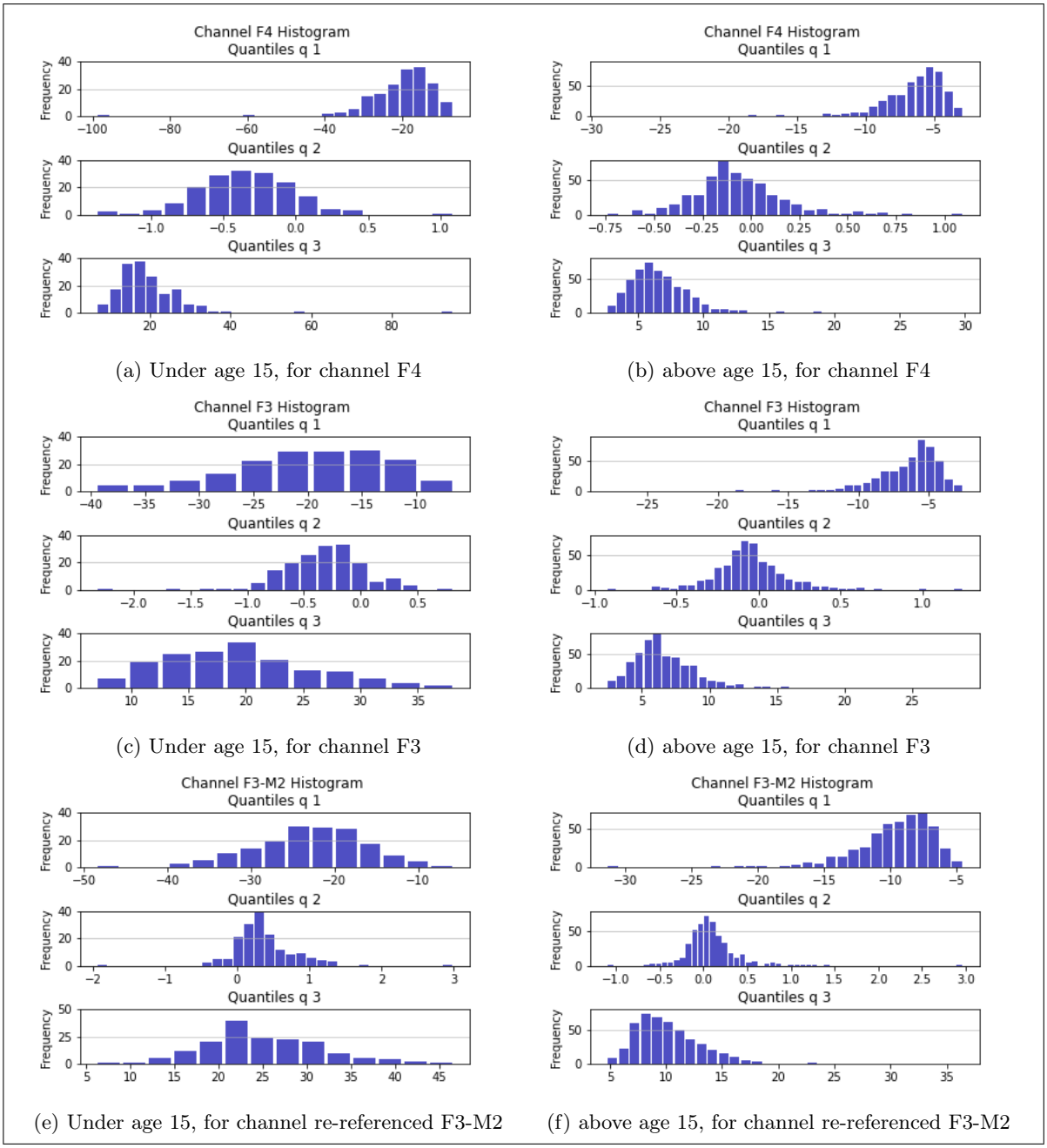
(c) Re-referenced channel C4-M1

(d) Channel C4

(e) Re-referenced channel O2-M1

(f) Channel O2

Figure 1.1: Histogram of the quantiles of channels with and without re-referencing based on channel M1 from the full control set



(a) Under age 15, for channel F4

(b) above age 15, for channel F4

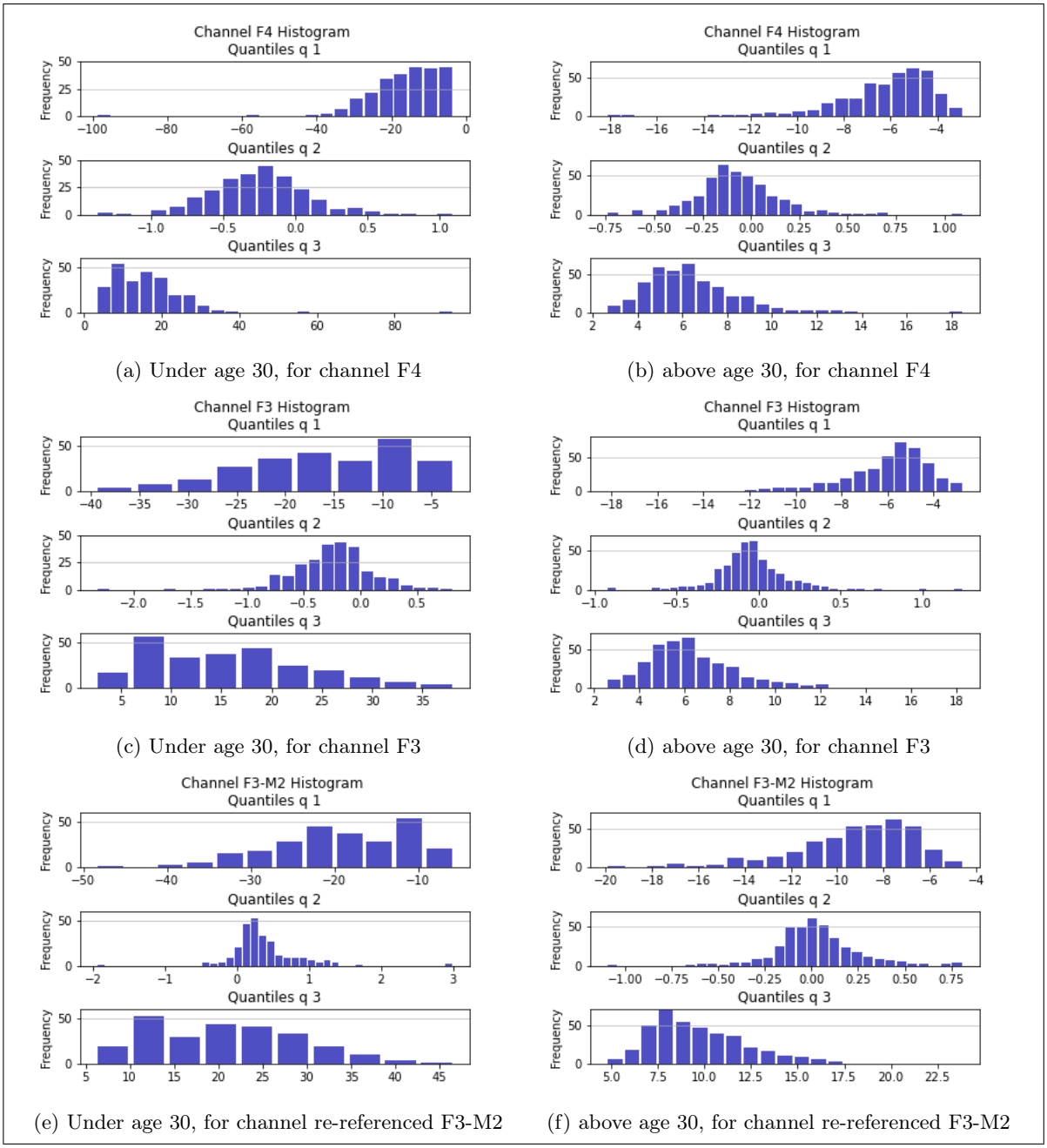
(c) Under age 15, for channel F3

(d) above age 15, for channel F3

(e) Under age 15, for channel re-referenced F3-M2

(f) above age 15, for channel re-referenced F3-M2

Figure 1.2: Histogram of the quantiles of channel F3 with F4, and F3 without re-referencing based on age wise separation on 15.



(a) Under age 30, for channel F4

(b) above age 30, for channel F4

(c) Under age 30, for channel F3

(d) above age 30, for channel F3

(e) Under age 30, for channel re-referenced F3-M2

(f) above age 30, for channel re-referenced F3-M2

Figure 1.3: Histogram of the quantiles of channel F3 with F4, and F3 without re-referencing based on age wise separation on 30.

Chapter 2

Methods and ideas with the package

Inorder to fully utilize the package, the user need to know the purpose of the package. through the proposed methodologies. The overall general methodolgy (pipeline) is captured in the Fig. 2.1.

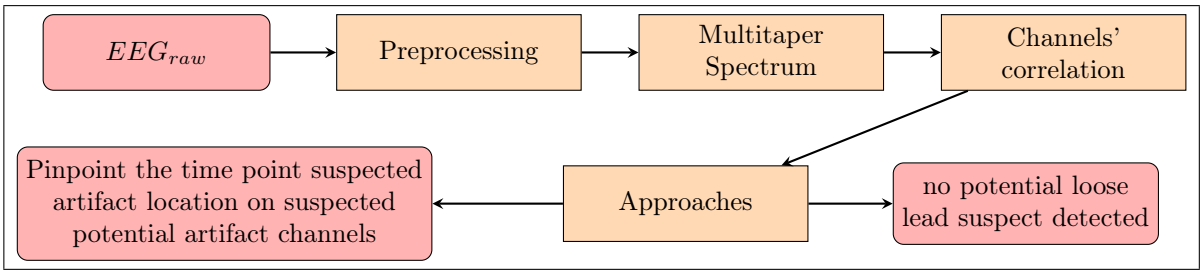


Figure 2.1: Overall workflow of the proposed method.

In this work we have used both domains "Time-Frequency analysis" kind of combined with "Spatial domain". In other words the "Spatial domain" features are captured by different re-referenced channels. Initialy the "Time domain" informations is used independently via the Preprocessing pipeline to remove the artifacts/ un-usable signals. In later part the "Multi spectral analysis" is chosen for optain the spectrogram ("Time-Frequency analysis").

With YASA [1] package , the spindles and/ or slow-waves are detected.

The variance in the spectrogram can be used to detect the artifact is analysed.

The correlation can be obtained from the spectorgram/ time domain. From analysis the time domain correlation need to analysed further for artifact detection. However, the correlation in spectral domian is easily utilised to detect the bigger artifacts like loose-leads.

In this methodolgy, I will show some part of the analysis other than presented in the **manuscript**. Focused on the spectral correlation-combinations usage in the outlier detection, while using independently vs mean. The analysis is carried out with this package. With their outcome so the user can get the feel of the package.

☞ There are **infinite possiblities** of using the package beyond these presented analysed approaches. Like using,

✓ correlation used in combinely or independently with the folcused sleep stages, etc.

- ✓ statitical properties: how we extract?; what we extract?
- ✓ Outlier detection technique: statistical based or fixed threshold on distribution, or portion or moving window etc.
- ✓ Even in the finalisation stage of the detected outlier makes difference, the package provide some choices of finalisation of detected outliers, etc.
- ✓ iterattive vs non-iterattive.
- ✓ combination of methods, ensampling /bagging/ voting
- ✓ Etc.

For an example, The Approach-1 also adapt the technique to iterively avoid the bad-cahnnel combintaion while pin-point the artifact regions. This iterattive analysis approach-1 ensambling with approach-2 can improve both sensitivity and specificity. But not explored in our analysis yet.

The package capabilities can be found in the following sections.

2.1 Preprocessing

This module is equipped with the basic Sleep-EEG$_{raw}$ extrinsic artifact removal.

Such that these basic extrinsic artifact-removal processing performed in this module with the following order,

1. "Nan" or "∞" values contain epochs-annotations are marked as "NaN in sleep stage".
2. Check the power line noise (60 Hz) exist in the epoch, If so, notch filter used to remove the power noise.
3. Then band pass the sleep data 0.5-32.5Hz.
4. Check the EEG signal and then the Nan segmentations annotated as NaN.
5. If $|EEGsignal| < 10^{-5}$ and $|EEGsignal| > 500$ are marked as "overly low/ high amplitude".
6. Epochs are chopped into $5sec$ intervals and check whether they are flat, such occurrence are annotated as flat.

The main choices in the Preprocessing going to be impact and change the overall outcome of the detected artifcats. Since we are using the statistical properties, the removal of signal in the earlier portion makes bigger difference in the later part. For an example, $|EEGsignal| > amplitude\_thres$ are marked as "overly low/ high amplitude". The choices are infinite, in the test scripts come along with the $amplitude\_thres$ as 500 or 2000, that going to be make bigger difference.

☞ **Each choices combination mostly going to produce totally different results**.

In one of the subject, in the preprocessing just changing only the $amplitude\_thres$ as 2000 to 500, mark 71 to 240 epochs as not useful as shown in the Fig. 5.10. Such that following analysis going to **skip** the **71** or **240 epochs** from this choice.

☞ Please check the Chapter. segment_EEG (5) for the **preprocessing choices** of the **parameter** and **functions**.

2.2 Multi-spectral analysis

Then the preprocessed Sleep-EEG signal's power-spectrum is obtained via the Multitaper Spectrum. As EEG-data is non-stationary process, using the spectral estimation questionable. Such that the study [2] detailed discription of how Mutitper-spectrum estimate suit for EEG-sleep data analysis in detail. Further as per the study [3] EEG-sleep data can be considered stationary for 4 sec (T = 4 sec). Such that in this study, we have used T (4-sec) window with seven-Slepian tapers for, Sleep-EEGs with sampling rate 256.

☞ **Blind choice** of Multitaper Spectrum parameters for the different sampling frequency will end up in (varianced or bias) **spectral estimation deviate from the original (true) spectrum**.

For more details please check the chapter. "Multitaper_class" (10)

2.3 Channels' correlation

As stated earlier the loose lead channel artifact presence will reduce the relation with the good-channels. Such that, it is going to reflect on the combination of these channels' Multitaper Spectrum's Spearman's rank correlation coefficient on that specific time.

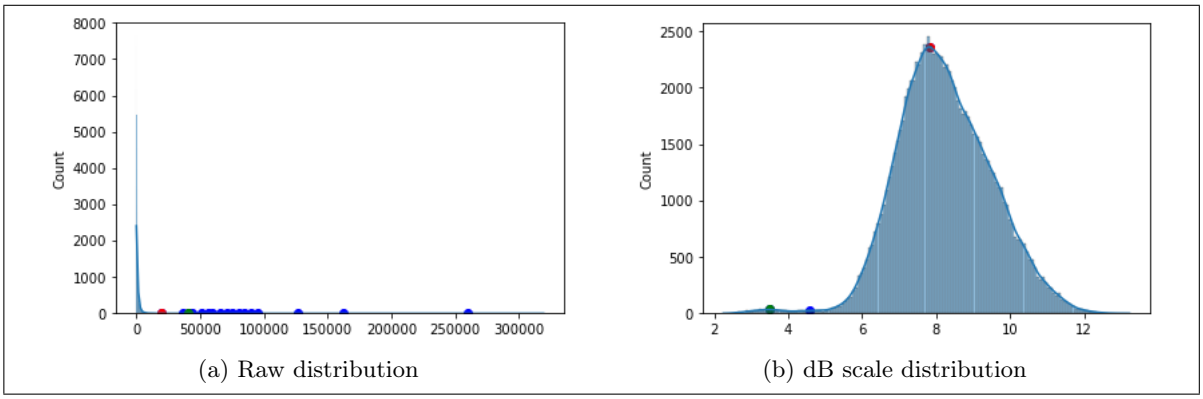
Inorder to avoid the ambiguity in channel combination names. Here onwards the channels' name means the re-referenced channels' name. For an example "F3" means F3-M2 channel and the "F3-F4" correlation means the Spearman's rank correlation coefficient between "F3-M2" and "F4-M1".

2.4 Variance based outlier detection

As we remove the flat signal from the time domain based on the varience in the selected region. Likewise we can remove the vertical spikes in the frequency domain. The intuition here is the vertical spikes appears can cause the power spread across the band which will reduce the variance.

Such that the potential outliers (mostly artifacts) are mostly clearly visible in the db scale spectrogram, the vertical red-lines or red-portions. Such that direct intutions is used the variance along the frequency axis to detect these outliers.

☞ This method has **limitations** like choices of the threshold (with the chosen scaling choices), etc. The example of the standard deviations values with the scaling choice for pooled σ's of raw vs dB scaling distribution is shown in Fig. 2.2. As it is clearly visible the dB scaling will be good choice, with the σ' threshold may be five for this subject. Please check the distribution of standard deviation



(a) Raw distribution (b) dB scale distribution

Figure 2.2: Pooling NREM sleep-stages' six channels' Multitaper Spectrum's σ's of raw vs dB scaling distribution

Such that the package provide an option to detect the artifacts with the variance (standard deviation). Please check the chapter. "MT_variance_for_structural_aritifact_funcs" (11) for more details.

2.5 Correlation based approaches for outlier detection

The final outcome of the detected artifact presence on the channel depends on corresponding channel's correlation-combinations handle in outlier-detection. The channel's correlation-combinations can be handled in two ways, such as

- ✓ Approach-1: If I broadly state channel's correlation-combinations are used independently to detect the outlieres and then combine the detected outliers from each combination. The overall pipeline of Approach-1 is shown in Fig. 2.3.
- ✓ Approach-2: The mean of channel's correlation-combinations are used detect the outlieres. The overall pipeline of Approach-2 is shown in Fig. 2.4.

Apart from the ways of using the combinations, detect the outliers also can be done in plenty of ways. In this package, mainly two ways (techniques) are presented such as,

- ✓ Way-1: Distribution based outlier detection.
- ✓ Way-2: Moving window based outlier detection.

☞ It is worth to point, the **Approach-2** is **simpler** than the **Approach-1**, since the Approach-2 combine the information from the combinations via the mean.

☞ The Approach-2 with the moving window technique for outlier detection method is so-intutive and effective; such the method alone is currently **under review**.

☞ In order to achive the **simplicity** the **Approach-2 miss** the **independent combination related information**. If more than 3-channels are bad (loose) in the same period the approach-2 definitely (or higher chance) of failure, however the **approach-1 has higher chance to detect that artifact**.

☞ Each of these **approaches** can be **applied independently**, and depends on the user preference. And later **these approaches can be combined**, or one selected approach results can be used.

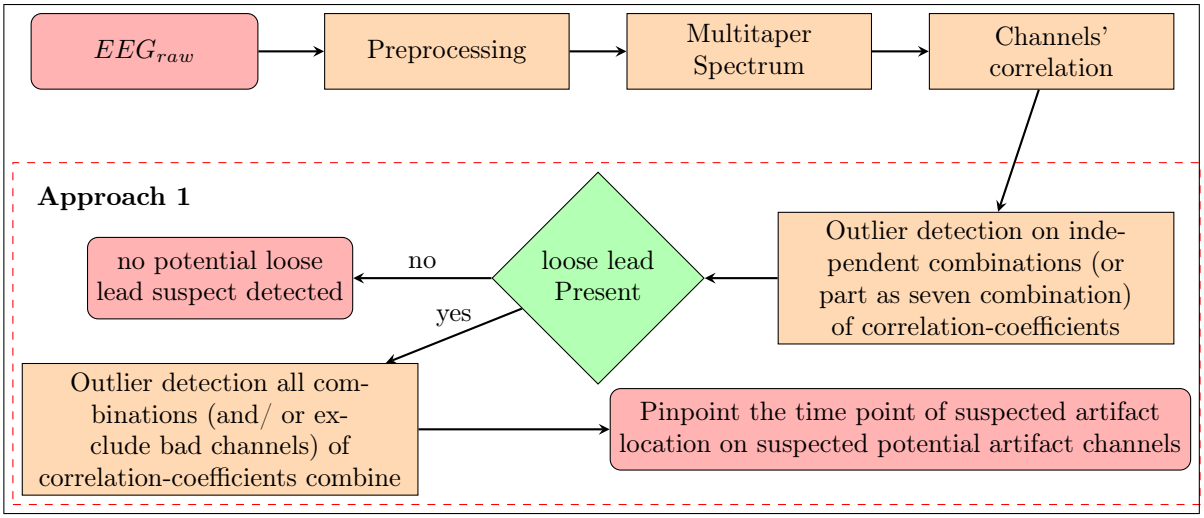


Figure 2.3: Overall workflow of the proposed method with Approach-1 presentd in the red-dashed line box.

In the research and analysis, the Approach-1 is Initialy proposed and analysed, later Approach-2 is developed in an iterattive manner to avoid the highly artifact channels in the mean correlation calculation. The combination outcome of approaches give more finite outcome in pinpointing the positions, with the cost of computing.

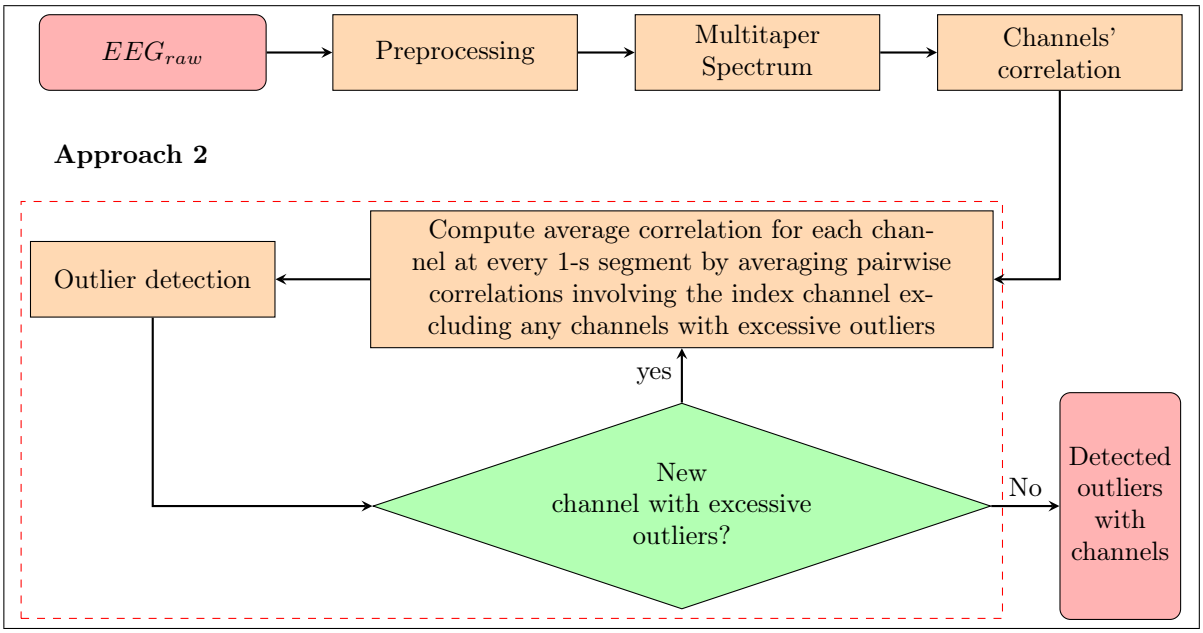


Figure 2.4: Overall workflow of the proposed method with Approach-2 presented in the red-dashed line box.

2.5.1 Main elements different among the approaches presented here

In the outlier detection analysis presented here only the Z-transformation procedure is different in Approach-1 and Approach-2. Because each approach handles the channel's correlation - combinations differently, such that the need for Z-transformation on outlier detection depends on approach.

- ✓ Approach-1: Distribution based outlier detection; handles the channel's correlation - combinations independently; such that the Z - transformation is essential.
- ✓ Approach-2: handles mean of the channel's correlation - combination to detect the outliers. Such that the Z-transformation should be applied on mean of the channel's correlation - combination to follow the procedure in outlier detection. However, most of the time mean of the channel's correlation - combination tend to have almost normal distribution, the Approach-2 works well without the Z-tarnsformation procedure.

Further Sleep-EEG structure is different for NREM and REM. In the outlier detection these sleep-stages need to be handled separately. For the loose-lead artifact detection the NREM and REM combined, or separate analysis does not show any difference at all. Here,

1. NREM: N1, N2 and N3 correlation values are pooled together.
2. REM: R correlation values are pooled together.
3. Combined: NREM and REM together (N1, N2, N3 and R together)

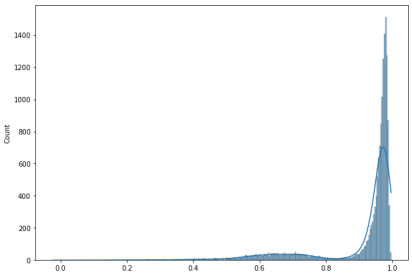
2.5.2 Distribution based outliers

Inorder to detect the potential outlier first the correlation-coefficients are mapped by Fisher's z-transformation as shown in the Fig. 2.5. The z-transformation ($corr_z$) obtained seperately for each sleep-stage groups of NREM and REM.

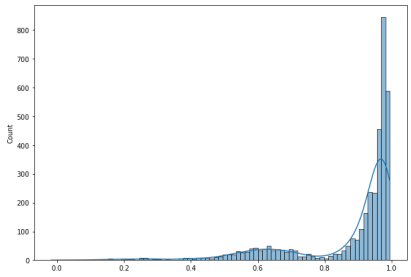
For an example, the Fig. 2.6 shows the effect on the distribution of channel combination F4-C4 correlation of the subjectafter the Z-transformation.

Figure 2.5 An algorithm (fun_z) to mapping and z-transformation; Please check the Chapter "poolers_Z_standardization_funcs" (12) for codes used for the purpose.

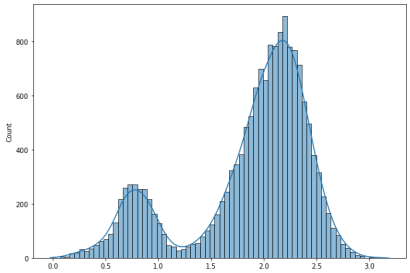
```
b_val=0.0001{The value to adjust in the bourdy conditions}
for corr in corr_pool do
   if corr == 1 then
      corr = corr − b_val
   else if corr == (−1) then
      corr = corr + b_val
   end if
   corr_z = arctanh(corr)
end for
```



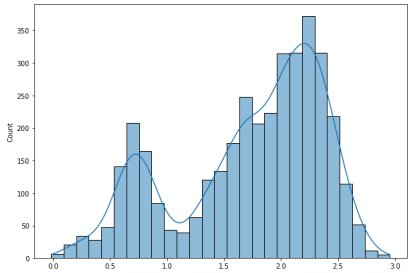
(a) NREM correlation distribution of re-referenced F4-C4



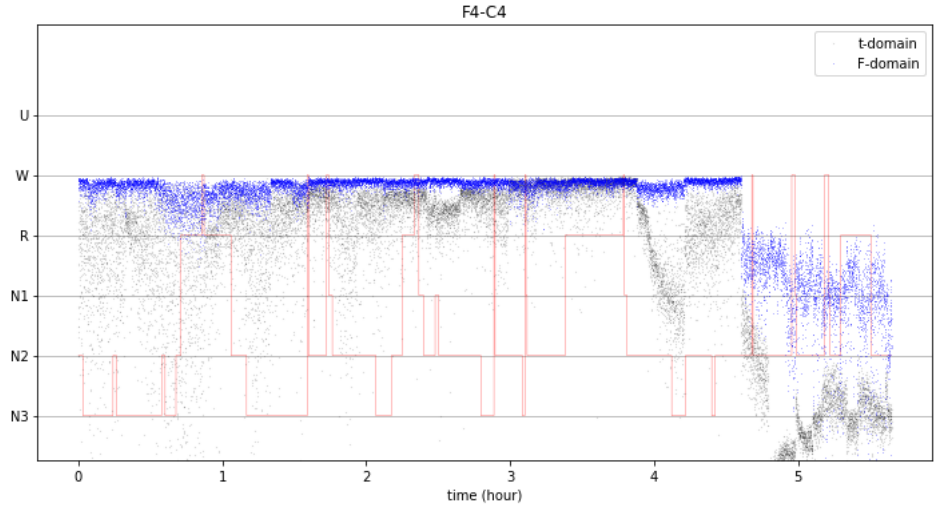
(b) REM correlation distribution of re-referenced F4-C4



(c) NREM z-transform distribution of re-referenced F4-C4



(d) REM z-transform distribution of re-referenced F4-C4



(e) Correlation of F4-C4 with the time stamps

Figure 2.6: Correlation with the time stamps and sleepstags; NREM and REM correlation distribution (histogram) before and after the transformation of re-referenced channels F4-C4 of 21-0995_F_9.7_1_di

☞ Please check the Chapter "poolers_Z_standardization_funcs" (12) for more codes used for the intermediate and/ or z-transformation, pooling correlation based on sleep stages, and obtain mean correltion.

F-domain (Multitaper Spectrum specral) correlation values of 7-channel groups are taken seperatly and annotation of the potential outliers are obtained.

Then from the $corr_z$ the outliers are detected based on the pseudocode in Fig. 2.7. First calculate

the quantiles (Q1, and Q3) then inter quantile range (IQR = Q3 - Q1) is calculated, with OLB=3 the outliers are annotated.

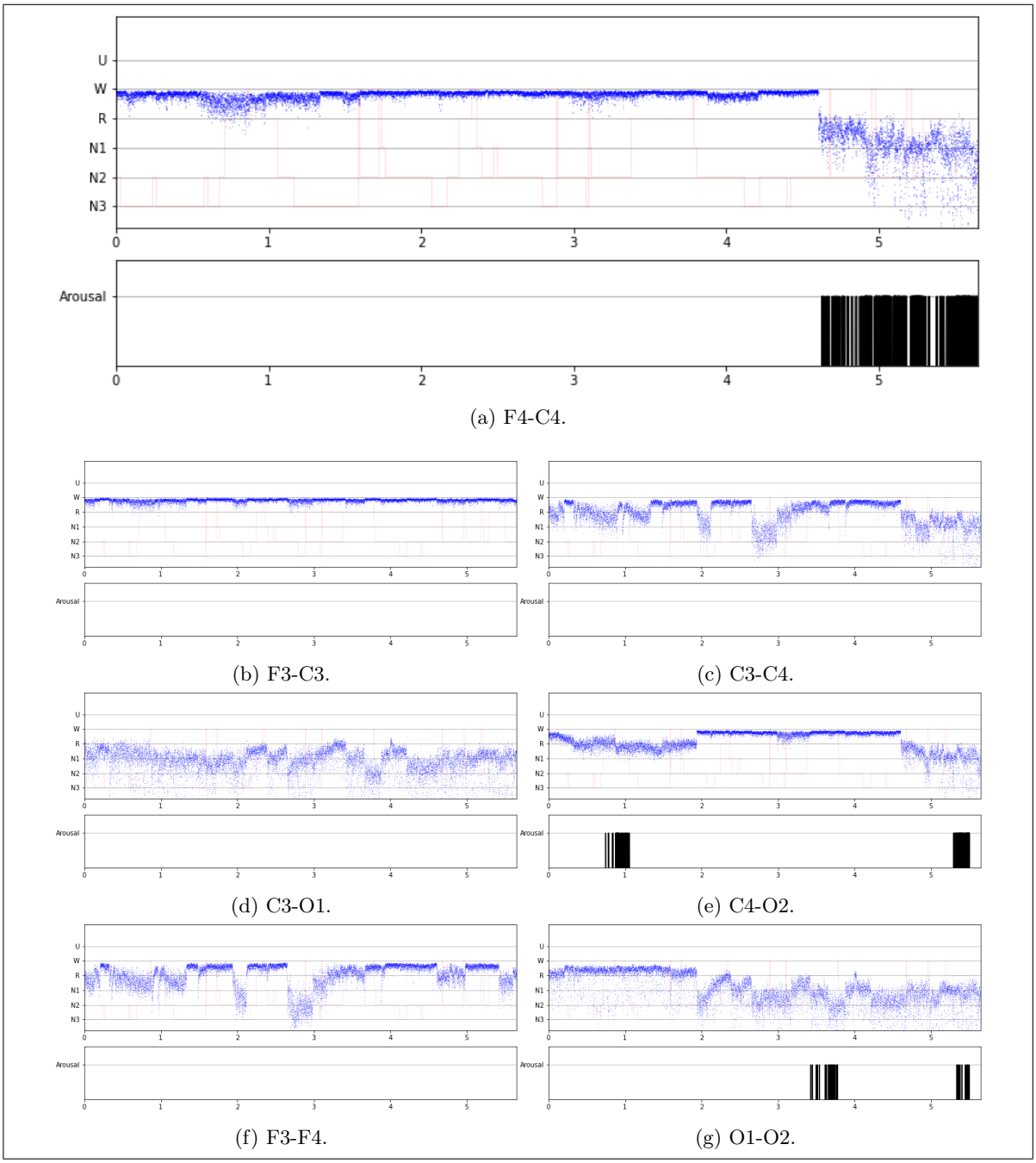
Figure 2.7 An algorithm to annotate the outlier based on outlier condition

1: $OL_{IQR} = OLB \times IQR$
2: **if** $corr_z < (Q1 - OL_{IQR})$ **then**
3: annotate as arousal
4: **else if** $corr_z > (Q3 + OL_{IQR})$ **then**
5: annotate as outlier
6: **else**
7: annotate as no outlier.
8: **end if**

Here the threshold value (OL_{IQR}) is obtained for NREM and REM seperately.

Like wise the T-domain (time only) can be used but need to be explored.

For Approach-1, with the six channels'15 correlation combination. Inorder to reduce the computational cost in the intial run, we need to have atleast seven correlation combination values pin-point the loose-lead as mentioned in the subsection. 2.5.5.



(a) F4-C4.

(b) F3-C3.

(c) C3-C4.

(d) C3-O1.

(e) C4-O2.

(f) F3-F4.

(g) O1-O2.

Figure 2.8: Interested seven channels with distribution based outlier based on MT-spectrum with 7-tapers of of 21-0995_F_9.7_1_di.

Issue due to Multimodes

As shown in the Fig. 2.8 among the C4 channels' combinations (in the checked seven combinations) as F3-C4, F4-C4, C3-C4, C4-O2 only the C3-C4 doesn't detected a single outlier. As we can see the correlation is varying too-much.

Lets check the Z-transform distribution as shown in the Fig. **??**. There if we check the C4's combinations as shown in subfigures,

- ✓ (c) F3-C4
- ✓ (j) C3-C4
- ✓ (n) C4-O2
- ✓ (g) F4-C4
- ✓ (m) C4-O1

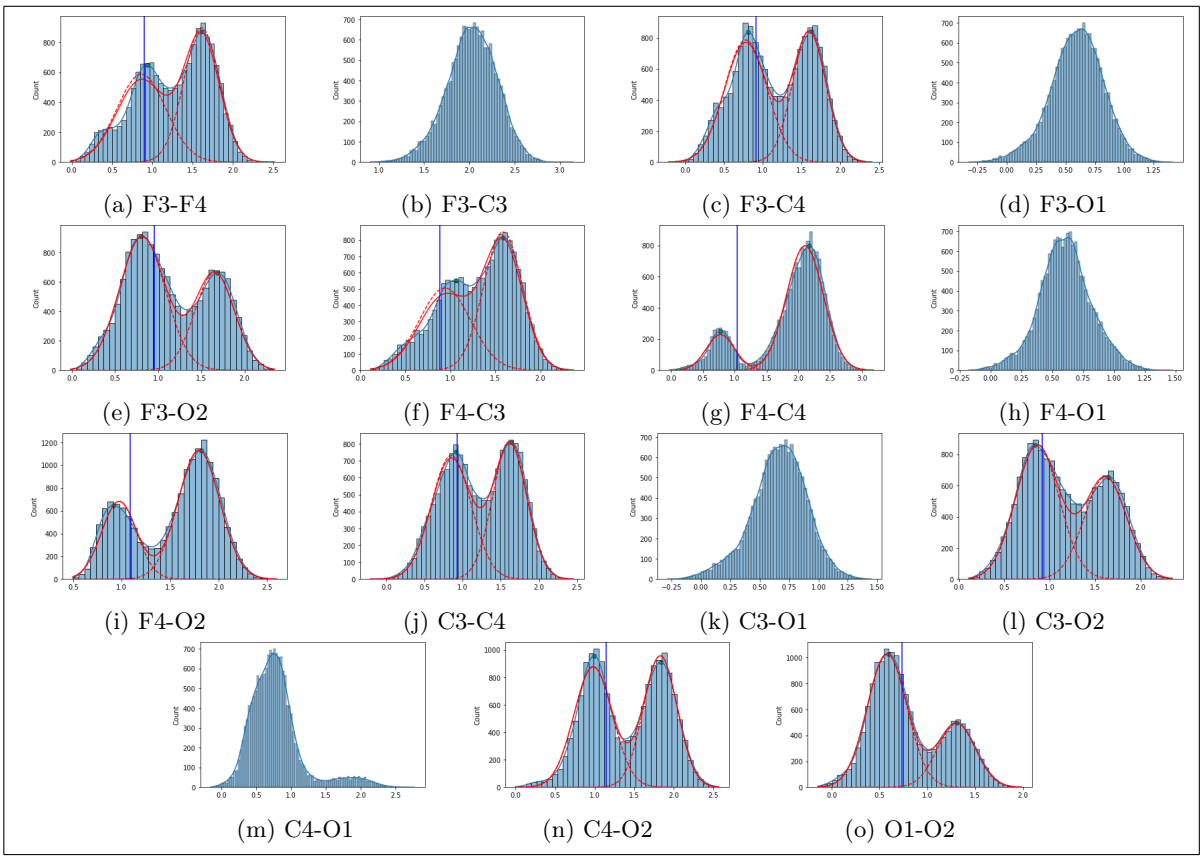
Almost all of them have two peaks (apart from C4-O1; due to O1 is mostly fully bad channel).

In this situation solution can be used the Gaussian mixture model (GMM) to extract the right side distribution. With the base assumption of higher correlation occurs mostly on the combination between good - good segments, compare to combination between bad - bad segments. Once we find the distributions, then we can annotate outliers with

- ✓ Use the righside distribution's probability of data points present with the left side distribution.
- ✓ Use the rightside distribution's parameters to caculate the interquantile based threshold (OL_{IQR}).
- ✓ There will be lot of choices not explored.

Incase if we using the righside distribution's probability of data points present with the left side distribution. To annotate the points fell in the left side of the threshold as outliers, the probability should be chosen carefully since the effect going to be exclude more data points based on the threshold For an example when we chosen probability as 0.1 vs 0.01 is shown in the Fig. 2.10. Here the blue vertical lines shows the threshold.

Such that here I prefer use the rightside distribution's parameters to caculate the interquantile based threshold (OL_{IQR}). The distribution based results presented in this guide-line obtained by this way (using GMM's right side parameter based OL_{IQR}).



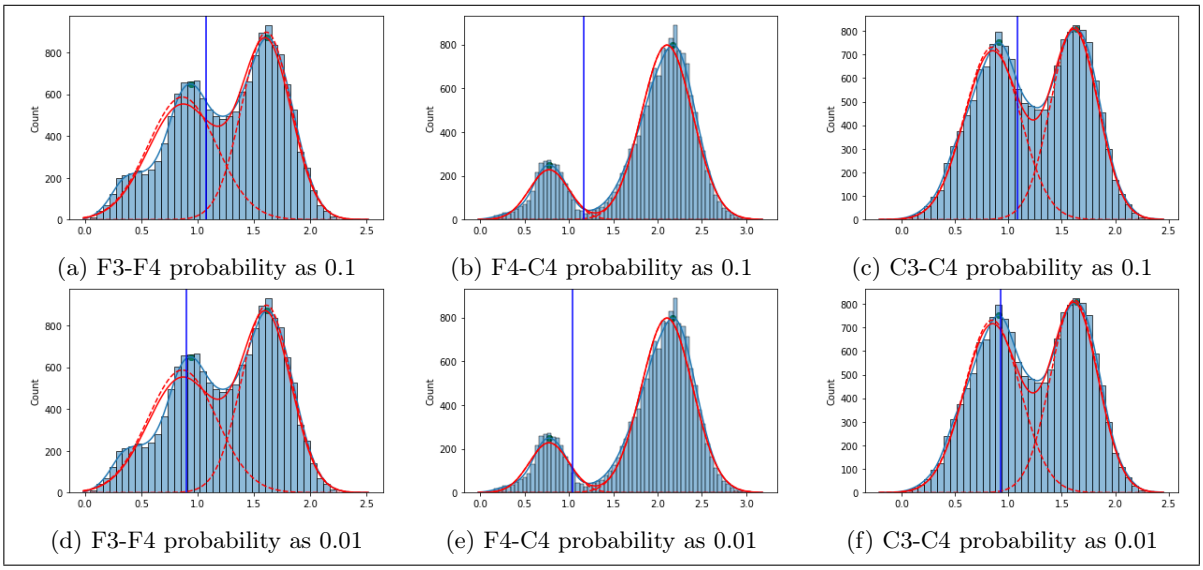
(a) F3-F4 (b) F3-C3 (c) F3-C4 (d) F3-O1

(e) F3-O2 (f) F4-C3 (g) F4-C4 (h) F4-O1

(i) F4-O2 (j) C3-C4 (k) C3-O1 (l) C3-O2

(m) C4-O1 (n) C4-O2 (o) O1-O2

Figure 2.9: Selected groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis GMM based model checked for mulitimodal distribution with the probability(0.01) threhold of the right side distribution



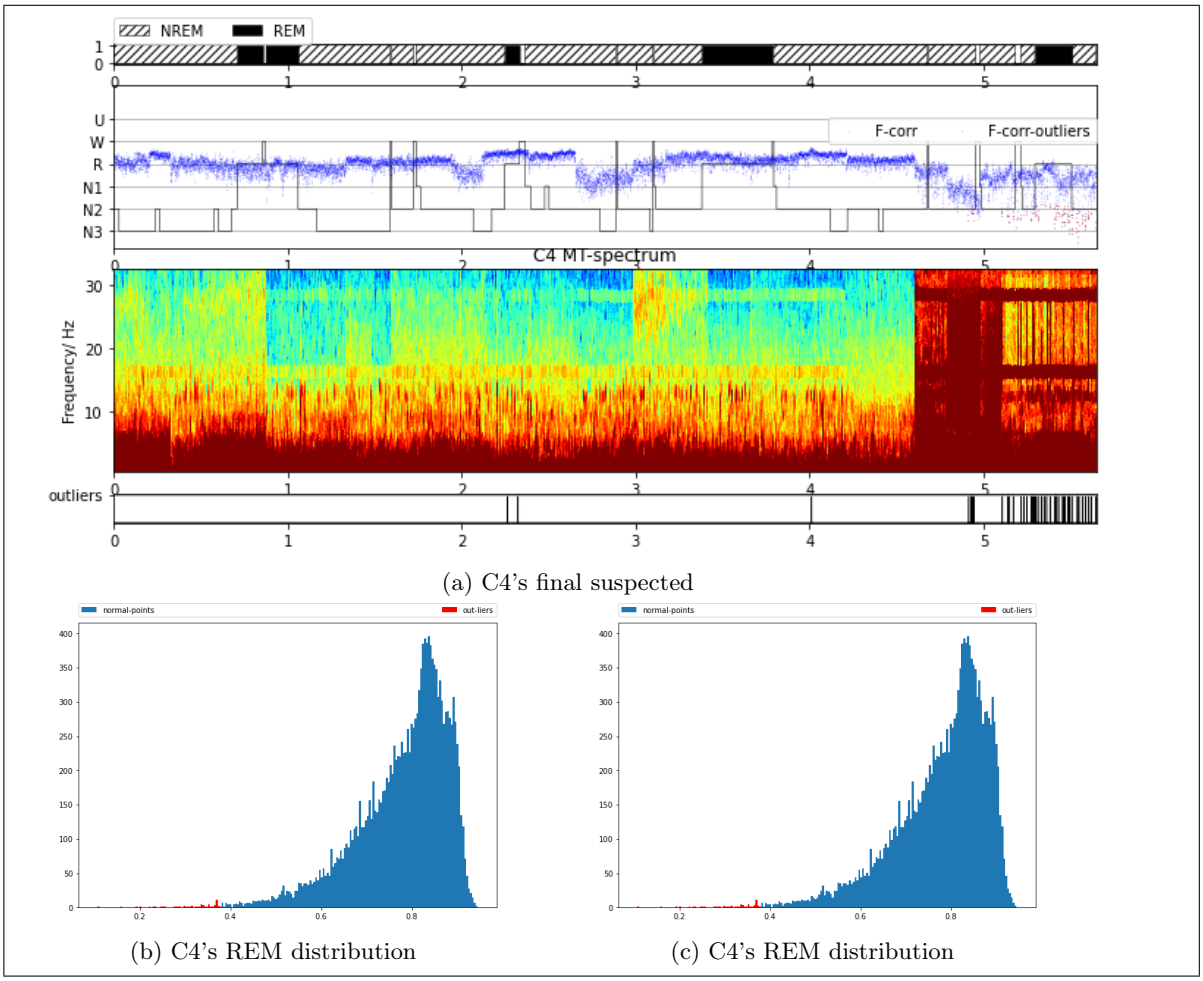
(a) F3-F4 probability as 0.1 (b) F4-C4 probability as 0.1 (c) C3-C4 probability as 0.1

(d) F3-F4 probability as 0.01 (e) F4-C4 probability as 0.01 (f) C3-C4 probability as 0.01

Figure 2.10: Selected groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis GMM based model checked for mulitimodal distribution with the probability(0.1) vs probability(0.01) threhold of the right side distribution

The Approach-2's (mean correlation combination) distribution based loose-lead detection selected channel results is presented in,

- ✓ For channel C4's mean **raw-correlation among with the with** OLB **as 1.5** is shown in Fig. 2.11.
- ✓ For channel C4's mean **Z-transformed correlation** among with the with OLB as 1.5 is shown in Fig. 2.11.
- ✓ For channel C3's mean **raw-correlation** among with the with OLB as 1.5 is shown inFig. 2.13

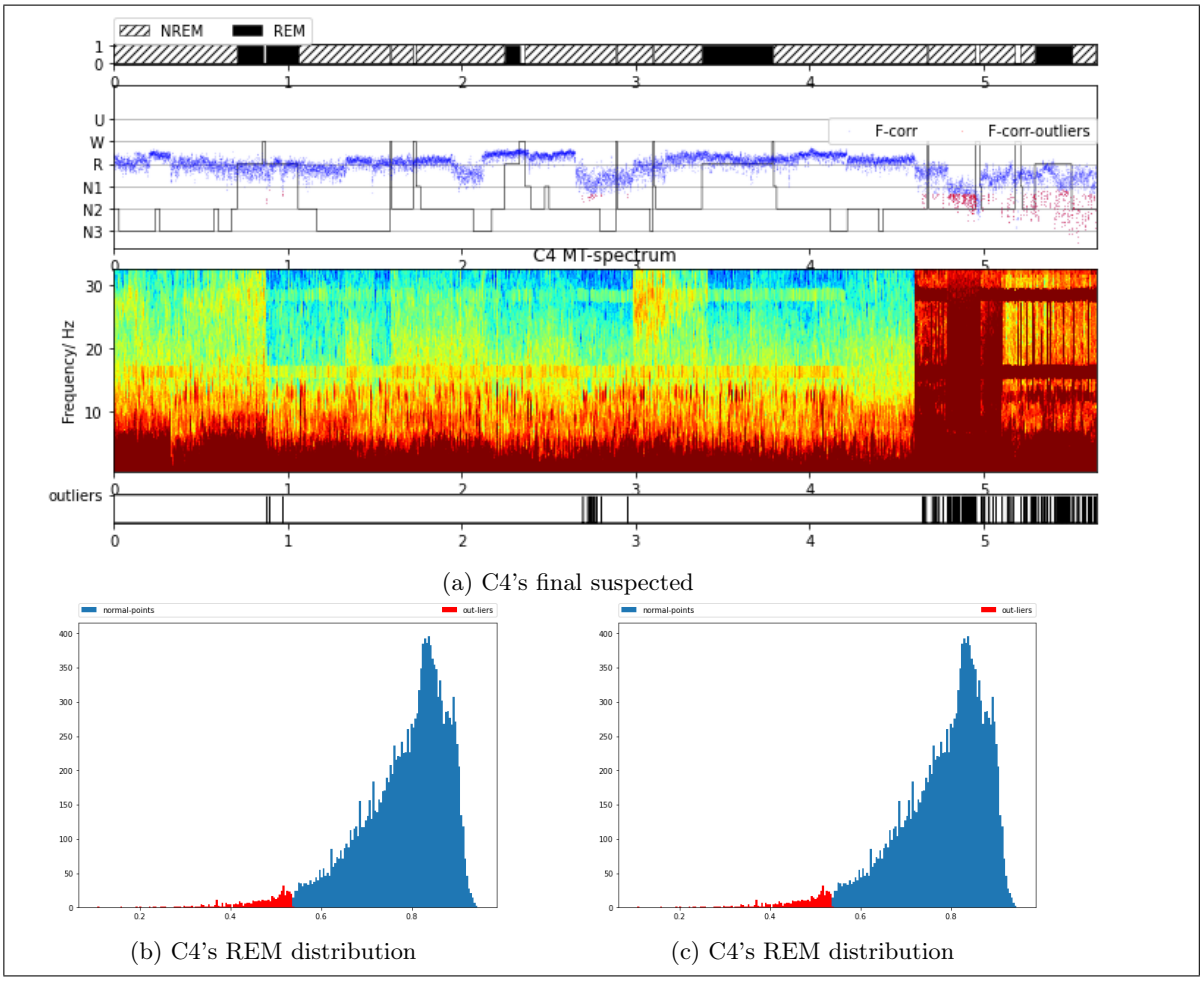


(a) C4's final suspected

(b) C4's REM distribution

(c) C4's REM distribution

Figure 2.11: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients on C4 by Z-transformed correlation coefficient mean distribution based model with outlier condition 1.5.

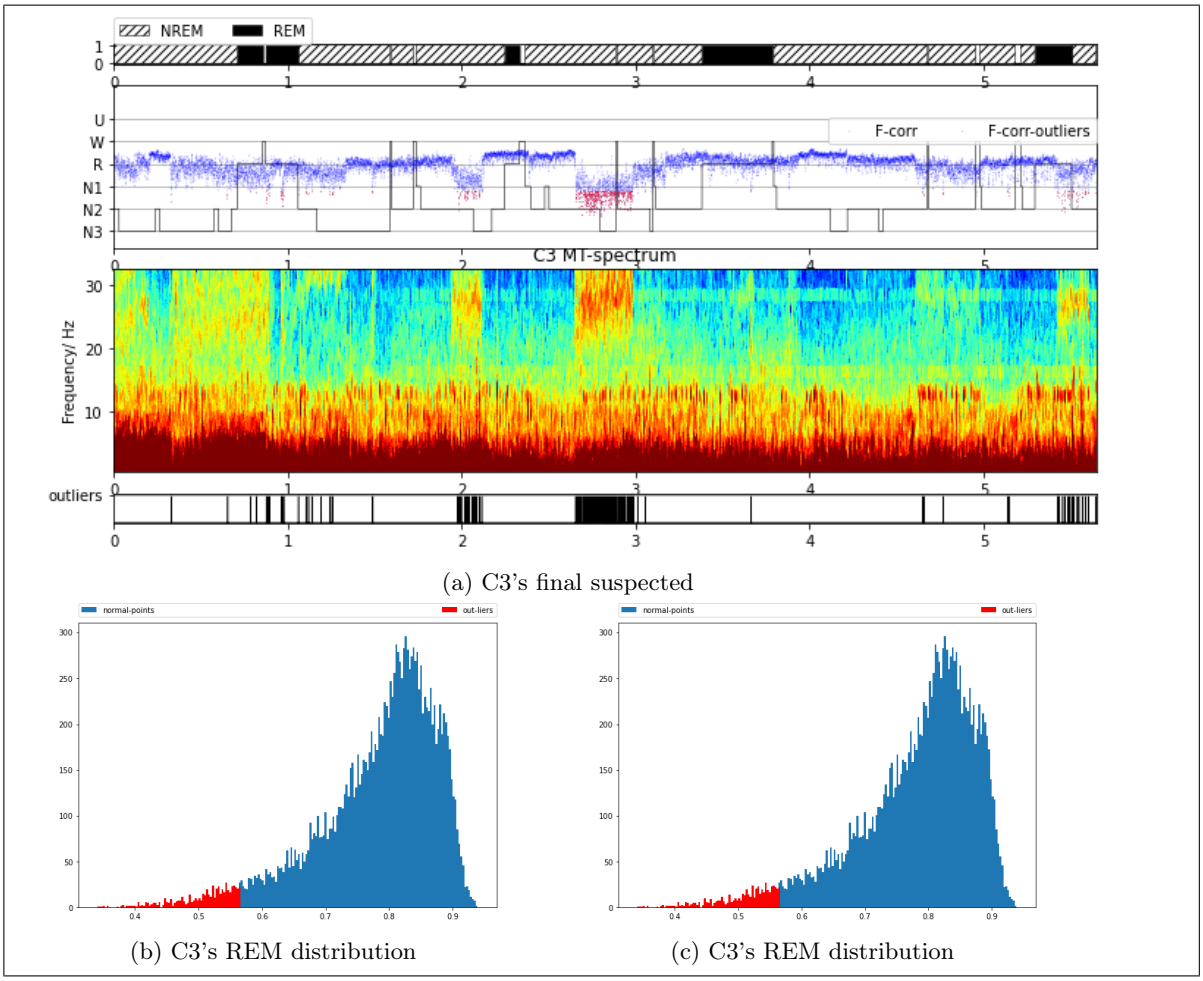


(a) C4's final suspected

(b) C4's REM distribution

(c) C4's REM distribution

Figure 2.12: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients on C4 by direct correlation coefficient mean distribution based model with outlier condition 1.5.



(a) C3's final suspected

(b) C3's REM distribution

(c) C3's REM distribution

Figure 2.13: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients on C3 by direct correlation coefficient mean distribution based model with outlier condition 1.5.

2.5.3 Moving window based methodolgies

The moving window based approach is already presented in the manuscript in **review**. Since the guide-line is part of the package some texts may appear as it presented in the manuscript. As **manuscript - review** presentation way is different, here the moving window based method is explored with explonation.

Approches in outlier detection with the moving-window

As stated in [4] the adaptive threshold choise is better than rigid threshold based approaches in Sleep-EEG. We use the adaptive threshold for outlier detection in all approaches. The overall outlier detection algorithm is shown in Fig. 2.14.

The main difference of the variables of Approach-1 and Approach-2 in the Fig. 2.14

- ✓ Approach-1 $corr_{pool}$ is from each channel correlation combination (com), and the corresponding detected outliers also for the channel - combinations (com).
- ✓ Approach-2 $corr_{pool}$ is from each, channel's mean of correlation combination (com), and the corresponding detected outliers is for the channel (com).

Figure 2.14 An algorithm to detect the outliers

for $corr_{pool}$ in ($channel\ correlation\ combinations$) **do**
  if Apply Z-transform **then**
    $corr_z = fun_z(corr_{pool})$ {function map correlation coefficients by Z-trasformation, Approach-1 prefered to use Z-transform}
  else
    $corr_z = corr_{pool}$ {Skip Z-transform, Approach-2 not prefered to use Z-transform}
  end if
  $annot_{local-outliers} = fun_{moving-local}(corr_z)$ {The function detect the outliers based on the local moving window method.}
  $annot_{outliers} = fun_{global}(corr_z, annot_{local-outliers})$ {The function detect the outliers with the temporal location based on the global threhold method, with the aid of already detected outliers by the local-moving window.}
  $dic_{annot}[com] = annot_{outliers}$ {This dictionary holds the predicted outliers with the combination}
end for

Local moving window

The study [5] used the local-moving window-based approach to detect the EMG artifacts. The study [5] pointed out the effectiveness of median in local-window to find the outliers, since the medians are less effected by extreme outliers than the means from the local-window. The local window-size is chosen as the $two \times epoch_{length}$ like study [5], such that if the $epoch_{length}$ is 30 sec, then the window size 60 sec (win_{size}) or greater is better. Such that we can find the outliers fell in the middle of two epochs, while sleep-stage annotation does not account this part [5].

The study [5] evluated the five threshold condtions 1.5, 2, 4, 10, and 40, found the threshold factor 4 is effective for the task. compares high frequency activity (26.25±32.0Hz) in each 4-s epoch thresholds, th40, th10, th4, th3, th2, and th1.5

The study [5] stated as *"Changing the threshold level allows adjustments in sensitivity and specificity of the artifact detection"*.

On a specific channel's mean of correlation combination ($corr_z$ as mentioned in the channel correlation section **??**) is checked for the correlation drop due to artifact presence. The algorithm scans for "bad" 1s segment using a local moving window. A segment is considered "bad" when its correlation is less than certain fraction (like half $corr_z < \frac{corr_z median}{threshold_{loc}}$ when $threshold_{loc} = 2$) of the median correlation of all segments in the window. Those that are above the cutoff are placed in a pool of "good" segments, which is used as good-pool.

Global threshold

Unlike the EMG artifact presence in Sleep-EEG, potential artifacts like loose-lead appear for longer period. The local-moving window-based detection fails in the artifact presence for longer period. This problem can be overcome by the defining global-threshold ($corr_z glob$) which annotate outliers in the second pass, as the mean correlation drop below the global-threshold. ($corr_z < corr_z glob$)

To obtain the global-threshold, we need a good-pool $corr_z$s. We can use the good segments obtained in the local window section 2.5.3 However, the local-moving window based procedure not only choose the good-pool $corr_z$s from non artifact local window's $corr_z median$, but also have some $corr_z$s from artifact region due to local window's limitation in longer period of artifact. On the other hand, we know the good $corr_z$s going to have higher value. Such that we can safely assign, the 75^{th} percentile of the obtained good-pool $corr_z$s by local moving window, as $corr_z glob$. Such that, a segment is declared as an outlier, when its correlation is less than fraction of the 75^{th} percentile of the good pool. ($corr_z < \frac{corr_z glob}{threshold_{glob}}$; we have used the $threshold_{glob} = 4$)

2.5.4 Loose-lead present check

Deciding the loose lead is going to be depends not only in detected outliers, but also the continuity presence check. To explain the outcome easier, this module is checking the period and continuity of outlier presence for each channel(com) in $dic_{annot}[com]$ obtained in outlier section 2.5.3 on Fig. 2.14. Each channel's predicted outliers are presented in one hot encoding, such that each channel is presented by a sequence of zeros, with the predicted outliers are only marked as ones.

The continuity check is performed on the detected outliers (one-hot encoded sequence) in two steps, such as

1. Mark the continuous outliers by unit weight convolution with stride one (sliding window sum obtain by the sliding in one sec resolution): The sum represents the number of predicted outliers present from the given time-point to sliding period.
2. Check the period of presence of marked continuous outliers. This step is depending on the previous step outcome.

The first step, marking the continuous outliers in the sequence can be done in two ways, such as,

1. Strick continuity check ($sliding_{sum} == sliding_{window}$): the outliers must present in the whole period in the sliding window. For an example, five seconds sliding window sequence, whole five second sequence should be outliers.

2. Relaxed continuity check($sliding_{sum} > threshold_{sec}$ here the $threshold_{sec} < sliding_{window}$): In the in the checking $sliding_{window}$ the total number of outlier presence is more than the certain number ($threshold_{sec}$). When the condition is more relaxed as $threshold_{sec} == 0$ and $sliding_{window} = 1$; the way just present the predicted outliers without any modification.

Once we obtain the stream of continuous outliers. In step 2; we check the period of the continuous outliers with some threshold as,

1. Percentage of continuous outliers ($> 5\%$) present in the selected moving window period (20 minutes with moving window of 5 minutes) is considered as loose-lead when this kind of occurrence repeated more than certain times (> 3; $(20/5) - 1 = 3$ this choice may avoid the same outliers in the sliding).

2. Percentage of continuous outliers present in the whole sleep-sequence.

3. Total time of continuous outliers.

In the **manuscript- review**, combine the outliers via checking the gap between them is used. We can fill the gap by checking the difference and using the convolotion to finalise, or in the other way. In other words finalisation itself contain infinite choices. In this package provide some options.

2.5.5 Approach-1's Loose-lead present check

This procedure is for need more attention for Approach-1, because this approach detect outliers present the each channel-combinations independently. Such that the independent detected outliers need to be combined to check the loose-channel (lead).

Loose-lead present is evaluated based on the outliers(dic_{annot}) combinations. The pseudo code shown in Fig. 2.15 shows potential loose lead presence based on the length of common combined-segments.

Figure 2.15 An algorithm to find the continiuos outlier condition to annotate the loose lead

1: $threh_{outlier}$= This is the threshold value to annotate the segment has loose lead. Intially the value is choosen as 15 sec (since epoch is 30 sec atleast half of it does not belong to outlier)
2: **for** *com* in (*channel combinations*) **do**
3: $seg_{chk_{outlier}}[com] = fun_{loose-com}(dic_{annot}[com])$ {Combined-segmentation obtained based on the selected channel combination outlier annotation results. The Convolutional window (like 5 minutes) is moved along the results of annotated outlier results, to combine the results of annotated outliers as combined-segments of outliers}
4: **end for**
5: **for** *channel* in *channels* **do**
6: $comm_{com-seg}$: if more than two combinations of the selected *channel* has same segment ($seg_{chk_{outlier}}$). Then that segment is marked as $comm_{com-seg}$ for the *channel*.
7: $loose_{channel}$: If atleast one $comm_{com-seg} > threh_{outlier}$ then that *channel*'s portion is marked as suspected loose-lead.
8: **end for**

Inorder to avoid the overall computaion cost the modules are evaluated in three stages. Here lets we explore the six sleep related channels, as stated earlier In the first stage, we must have atleast seven

combinations to detect the loose-lead. Such that in this study, three inter hemispheres (left vs right) channels F3-F4, C3-C4, and O1-O2 were checked. Along with four intra hemispheres channels such as,

1. left side 2-channels F3-C3 and O1-C3
2. right side 2-channels F4-C4 and O2-C4

were checked.

If the first stage detect atleast one loose lead suspect. In stage two, first stage results (using seven combinations) based selected combination with good-channels (from 15 combinations') is used to pinpoint the final suspected loose-lead with the temporal position. Here good channels can be choosen based on different ways,

1. Using all channels which has less than 20 minutes of $comm_{com-seg}$
2. First good channel which is the channel has least. If more than one channel have least-$comm_{com-seg}$ (like same length of $comm_{com-seg}$, or no $comm_{com-seg}$) then consider all good channels.
3. Percentage of presence, etc.

The final detected position can be emphasized by giving $comm_{com-seg}$ based on the presense of the combination.

1. Kind of present as probability from the combined combinations of the channel as shown in the equation 2.1.
2. Just like the step one the present only the $seg_{chk_{outlier}}$ in all combinations as shown in the equation 2.2.

$$comm_{com-seg} = \frac{\sum_{com}^{combinations} seg_{chk_{outlier}}[com]}{\#ofcombinations} \tag{2.1}$$

$$comm_{com-seg} = \prod_{com}^{combinations} seg_{chk_{outlier}}[com] \tag{2.2}$$

2.5.6 Approach-2's Loose-lead present check

The Approach-2's loose-lead is the direct outcome (for each channel(com) in $dic_{annot}[com]$) obtained from outlier section 2.5.3 on Fig. 2.14 with the mean of the channel's correlation combination with the moving-window based method. Then the detected outliers of each channel (com) is fed via the convolutional window based procedure to combine the outliers ($seg_{chk_{outlier}}[com]$) as shown in the pseudo code Fig. 2.15 line 2-4. This $seg_{chk_{outlier}}$ represent the suspected loose-lead location ($comm_{com-seg}$) for Approach-2.

Since the approach's results is already presented in the manuscript. The results obtained through different ways can also make changes in the final outcome is shown in the Appendix "Comparison between different ways on Approach-2 with the six channel's prediction on the selected subject plots" (B).

2.5.7 New loose-lead present

In approach-1 we are Intially running on the sub-sample (of seven out of 15 for six channels). If we detected a bad channel (loose-lead), then we go over the full combination. Such that in the re-iteration we can avoid the detected bad channel with the other channels while checking the loose-lead.

In approach-2 each channel's correlation mean is depending on the channel's combinations. Thus, any loose-lead present in the combinations going to affect the overall mean correlation; such that it finally impacts on outlier detection. To avoid the influence of loose-lead presence in the detection, the methodology is equipped with re-iteration while avoiding detected loose-lead/s in the mean calculation. Finally, if there is no new suspected loose-lead found then the last detected outliers are used to find the potential artifact and pinpoint the suspected loose-lead.

For an example, in both approaches when we re-iterate, Upto the last iteration we detected "F4" and "C4" as loose-channels, then the combintaions going to be,

- ✓ for F3 is with the C3, O1, and O2.
- ✓ for F4 is with the F3, C3, O1, and O2.
- ✓ for C3 is with the F3, O1, and O2.
- ✓ for C4 is with the F3, C3, O1, and O2.
- ✓ for O1 is with the F3, C3, and O2.
- ✓ for O2 is with the F3, C3, and O1.

Here "for F3 is with the C3, O1, and O2." means the when we check the channel F3, the combinations going to be F3-C3, F3-O1 and F3-O2.

2.5.8 Combine both approaches together

Each of these approaches handles the correlation-combintations differently, such that their predicted outliers by these approaches also going to be different.

This section summerises the ensembling approaches to finalise the loose-lead. First obtain the detected outliers as mentioned in the previous sections for Approach-1 ($comm_{com-seg}$) and Approach-2 ($seg_{chk_{outlier}} - App - 2$). Then combine the detected loose-leads via the same kind of way used to combine the results from the independent combintations of Approach-1.

The final detected position can be emphasized by giving $comm_{app-chan}$ based on the presense of the detected outliers by the approaches.

1. Kind of present as probability from the combined outliers of the channel as shown in the equation 2.3; since we have combine the results of two devide by two.
2. Check the $comm_{com-seg}(app)$ in all methods (here two approaches) as shown in the equation 2.4.

$$comm_{app-chan} = \sum_{app}^{2} comm_{com-seg}(app)/2 \tag{2.3}$$

$$comm_{app-chan} = \prod_{app}^{2} comm_{com-seg}(app) \tag{2.4}$$

2.5.9 Pinpoint loose-lead present

To pinpoint the loose-leads with the exact location of presence, we can use detected continuous outliers obtained from the section. 2.5.4 in the final iteration (no new loose lead detected).

To make decision of avoiding certain epochs, the continuous outliers are further annotated based on the tolerance period (tol_{period}).

Here any continuous outliers are separated less than tolerance period whole period between these outliers is annotated as loose-lead (potential extrinsic artifact). For an example of five minutes tolerance period going to annotate loose-leads presence for all the gaps between the outlier occurs less than five minute period, along with the continuous outliers.

We can do this by multiple ways, in this package

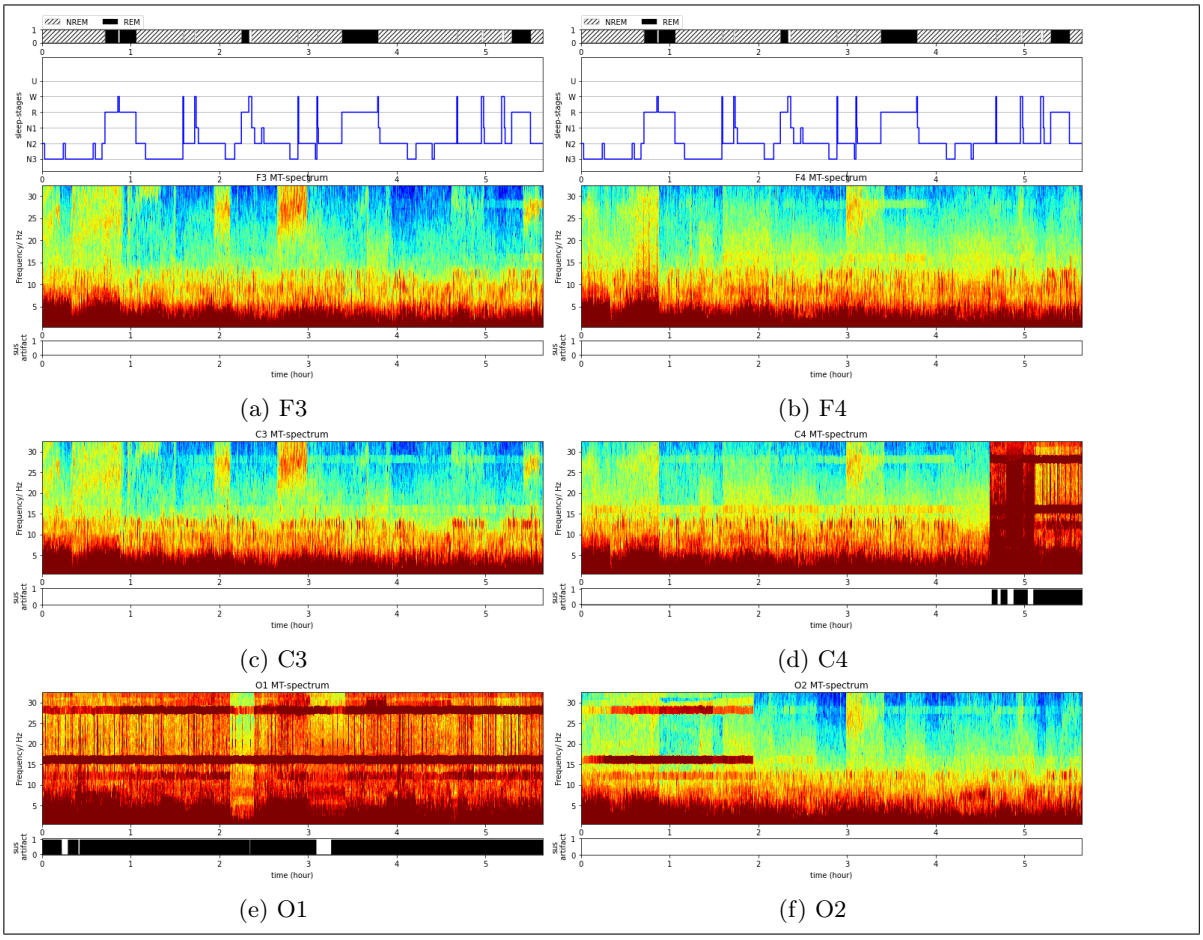
- ✓ Way_{diff}**-finalisation**: Check the difference between the current detected outlier to next occured outlier; achived by the locations of the outlier.
- ✓ Way_{sum}**-finalisation**: Count the the presence of outliers between the given period of time; achived by convolution (or moving sum)

There is going to be infinite choices. Due to combintations of finalisation and their order with their parameter on way of finalisation. Way_{diff}**-finalisation** is carried on the **manuscript - review**. Way_{sum}**-finalisation** is presented in this guideline's results; unless Way_{diff}**-finalisation** is explictly mentioned. Please check the Appendix section B.1.1 on ("Comparison between different ways on Approach-2 with the six channel's prediction on the selected subject plots" (B)).

2.5.10 Pinpoint the results with the combined approaches

Here the equations used for obtain the results to combine the approaches such as,

- ✓ Equation 2.4: Must present in both approaches as shown in the Fig. 2.16.
- ✓ Equation 2.3: probability of approaches as shown in the Fig. 2.17.

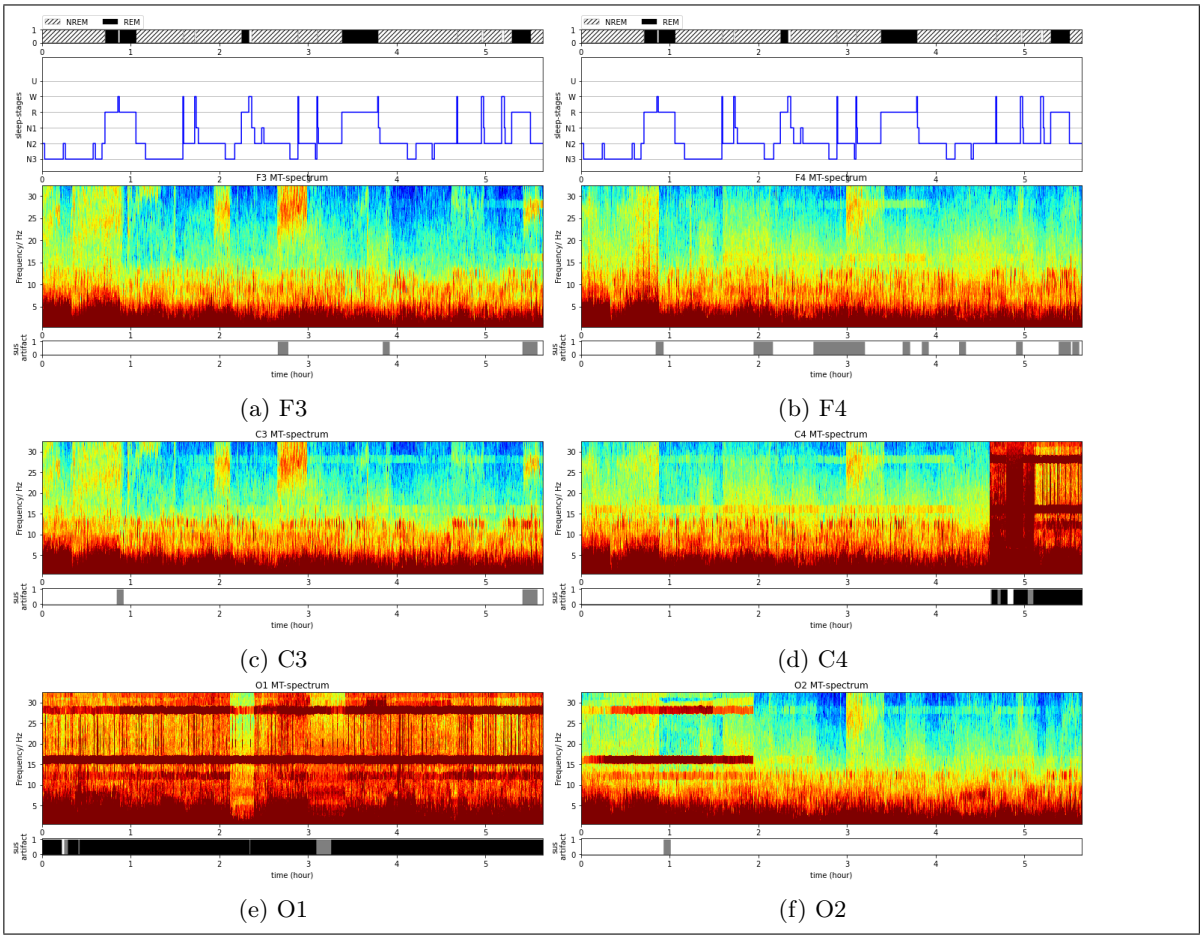


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure 2.16: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients based on with moving window with 60 and threshold factor 2 with global.with convolutional window ≈5 minutes in combined approach with must present in both.



(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure 2.17: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients based on with moving window with 60 and threshold factor 2 with global.with convolutional window ≈5 minutes in probability from approaches.

Chapter 3

Package capabilities

This package can identify the major artifacts (loose-leads in other words pop-leads) via the correlation of the spectogram (Multitaper Spectrum) with **only** using the **sleep-related channels** such as F3, F4, C3, C4, O1, and O2. Further this can be modified to use the available (or selected) channels. For an example in the 64 channels cap only the 6 electrodoes are only used for store the data, we only select the 6 for analysis.

This can be done via the -opt OPTIONS, – options OPTIONS. **Please check the input parameter for preprocessing for more details.** Like wise each parameters presented in the option controls the data selection to final outcome.

3.1 Input details

- h, –help show this help message and exit
- i INLOC, –inloc INLOC Input directory relative to the mount using docker file, this directory holds the .edf file or .edf files
- o OUTLOC, –outloc OUTLOC Main o/p directory relative to the muount using docker file, this dierctory will hold the tool obtained results
- opt OPTIONS, –options OPTIONS file name for options to the tools this can be used to vary the default parameters under construction

Eventhough this is not implemeted in the cmd line interface. The package based system this can be easily modified/ fed through the "optional_parameters" via the "parameter_assignment".

Such that lets check the "parameter_assignment" possible parameter assignments.

In other words file name "optional_parameters.py" with the object "parameter_assignment"

3.2 optional.parameter_assignment

In this section lets first check the intial parameters.

```
1   class parameter_assignment:
2
3       def __init__(self,
4                GUI_percentile=True,
5                save_events_origin = True,
6                sleep_stage_preprocess_origin = True,
7                channel_specific_preprocess=True,
8                pred_slow_waves=False,
9                pred_spindles =False, T=4,
10               amplitude_high_same_all_age=False,
11               avoid_spindle_loc=False,
12               verbose=False,
13               f_min_interst=0.5,f_max_interst=32.5,
14               epoch_length = 30,  line_freq = 60,
15               ch_names = ['F3', 'F4', 'C3', 'C4', 'O1', 'O2'],
16               b_val=0.001,
17               intersted_sleep_stages_REM=[4],
18               intersted_sleep_stages_NREM=[0,1,2,3],#N1,N2,N3,N4
19               intersted_sleep_stages_NREM_REM_comb=[0,1,2,3,4]):#N1,N2,N3,N4 and R)
```

Figure 3.1: parameter_assignment intiation code

The GUI_percentile in the line-4 of Fig. 3.1, shows the intial assignment. This can be later fed to the function seperately. The purpose of this is just to show the overall percentage of completed task for the given function. For an example, while extracting the Multitaper Spectrum spetrum from the preprocessed step, the progress of the extraction is shown in Fig. 3.2.

(a) loading edf GUI percentile completed 100%

(b) Preprocess GUI percentile completed 100%

(c) MT-Spectrum extraction GUI percentile completed ≈ 29%

Figure 3.2: GUI_percentile o/p

☞ In short **GUI_percentile** in any function can be used to check the progress (in Fig. 3.2).

Inputs related to the preprocessing save root data set o/ps

The save_events_origin in the line-5 of Fig. 3.1, is save the events annotation come along with the provided EEG file (".edf"). The o/ps extracted from the "preprocessing.load_EDF_dataset_events.load_root_dataset"

The sleep_stage_preprocess_origin in the line-6 of Fig. 3.1, is save the events annotation come along with the provided EEG file (".edf")

The channel_specific_preprocess in the line-7 of Fig. 3.1, is save the events annotation come along with the provided EEG file (".edf") with the preprocess detected artifacts (in time domain) with the channel infromation, if channel_specific_preprocess false, only the prepocess save the event with general detected artifact. For an example,

✓ Epoch "m's" annotation come in raw-EEG: Sleep stage W

- ✓ if preprocess only detect “C3” channel has higher amplitude for epoch “m”
- ✓ channel_specific_preprocess=True will annotate then the epoch “m’s” already provided event is as “Sleep stage W;overly high/low amplitude channels C3”
- ✓ channel_specific_preprocess=False will annotate then the epoch “m’s” already provided event is as “Sleep stage W;overly high/low amplitude channels”

Please check the Chapters, segment_EEG (5), preprocess_events_to_txt (6.1) and only_sleep_epoches_events_to
(6.2).

Save some extracted raw-EEG information

```
from sleep_EEG_loose_lead_detect.preprocessing.load_EDF_dataset_events \
import load_root_dataset
# Feeding the i/p to the load_root_dataset function and extract the o/ps
EEG_root, sleep_stages_or, EEG_channels, Fs, start_time_idx, \
sel_index_sl_st, whole_annotations, ev_or = load_root_dataset(in_edf, \
epoch_length, GUI_percentile=opt_paramters.GUI_percentile)

if save_events_origin:
    np.save(loading_dir_pre.out_loc_NREM_REM+ in_name_temp + "_sel_index_sl_st",
                                        sel_index_sl_st)
    np.save(loading_dir_pre.out_loc_NREM_REM+ in_name_temp + "_ev_or",ev_or)
```

Figure 3.3: Save the extracted raw-EEG information via the function (load_root_dataset)

The sel_index_sl_st is the starting indexes of the splited epochs (from the starting point of the event to end of the signal-length is splited into equal windows/ epochs) presented in the sleep aligned raw-EEG file, and the ev_or is the origin of the events come along with the aligned EEG (“.edf”) file. Please check the Chapter. 4 “load_EDF_dataset_events” for more details.

3.2.1 Extract the bad epoch information and save the information

```
bad_epochs = markBadEpochs(in_edf, in_bad_events, epoch_sec=epoch_length)
if loading_dir_pre.keep_signature_dic['bad_epochs']:
    np.save(loading_dir_pre.bad_epochs_folder+ in_name_temp + "_bad_epochs",bad_epochs
                                        )
```

Figure 3.4: bad epoch information extraction (markBadEpochs))

Some times edfs have rich information other than the sleep-stages like patients out for bathroom break, restroom, etc. The epoches fell into the bathroom break is just noise so these kind of noise present data supposed to be removed from the sleep-stages annotation

However each technician use their own terminolgy. we haven’t remove all the bad-events here, just some known events that is surely known as bad-events we have provided the bad events in the */docs/bad_events_bathroom.txt* files that can be updatable. Some potential events are purposefully

leaved like movenments, arousal etc. Since these information can be later used to find the subjects disease based studies etc. And the movements related artifacts like (poping leads), not always going to be present in the exact time annoated portion in all/ specific channels, however those information can be later useful, in the analysis.

Most of the time the bad events are related the alignment issue as shown in the subsubsection "Alignment on large .edf files" (4.9)

Preprocessing with extraction only sleep-related epochs and stats

The i/p shown in Fig. 3.5 for function segment_EEG (detailed explonantion) can be found in the Chapter. segment_EEG (5).

Since the segmentation function can modify the EEG_root and sleep_stages_or obtained from the Fig. 3.3 (detailed in Chapter. 4). The deepcopy of them (EEG_root and sleep_stages_or) are obtaied and fed through the function via EEG_root_copy and sleep_stages. Fs is the extracted information of sampling frequency of the raw EEG, like wise start_time_idx. For more details please check the Chapter. 4.

```
1  # Assigning prprocess paramter values
2  epoch_length, line_freq, bandpass_freq, normal_only, \
3  notch_freq_essential_checker, amplitude_thres = opt_paramters.preprocess_par()
4
5
6  EEG_root_copy=deepcopy(EEG_root)
7  sleep_stages=  deepcopy(sleep_stages_or)
8  # to feed the filterd EEG through the MT-make sure assign this as True
9  # basically the epoches points to the EEG_root_copies location
10 filtered_EEG_MT=True
11 epochs, EEG_root_copy, sleep_stages, epoch_start_idx_o,\
12 epoch_status, q1,q2,q3, notch_filter_skipped = segment_EEG(
13                     EEG_root_copy, sleep_stages,
14                     epoch_length, Fs,
15                     start_time_idx,
16                     notch_freq=line_freq,
17                     bandpass_freq=bandpass_freq,
18                     amplitude_thres=amplitude_thres,
19                     channel_specific_preprocess=channel_specific_preprocess,
20                     ch_names=ch_names,
21                     bad_epochs=bad_epochs,
22                     notch_freq_essential_checker=notch_freq_essential_checker,
23                     return_filtered_EEG=True,
24                     verbose=verbose)
```

Figure 3.5: Time series based preprocessing i/ps with segment_EEG's i/ps block

Other i/p parameters are obtained via the passed through the object's (opt_parameters) function preprocess_par() lines 2 and 3 of Fig. 3.5. Intial i/ps are intialised in the channel_specific_preprocess, Fig. 3.1.

- ✓ epoch_length: (intialised via the channel_specific_preprocess in the line-14 of Fig. 3.1 30) is same as the Fig. 3.3.
- ✓ notch_freq: Notch filter frequency for avoid power-line interfearence (60Hz) intialised chan-

nel_specific_preprocess in the line-14 of Fig. 3.1 3.1. For more details please check the section "power_noise_handler" 5.2.2.

- ✓ bandpass_freq: sleep-related events the region of interst going to be 0.5-32.5Hz such that here the bandpass_freq is assigned via f_min_interst =0.5 and f_max_interst = 32.5.
- ✓ amplitude_thres: the values assigned the amplitude threshold value for subject, the values presented in the scripts are 500 or 2000; since this threhold is based on the age. When the age is low, the brain EEG signal power is also high. The results are shown in the Figure. 1.2. From this initial outcome from Fig. 1.2 clearly see the when the age get older the amplitude distribution decreases. So these values should be chosen carefully to avoid loosing too-much signals.
- ✓ channel_specific_preprocess: When the preprocessing events annotation are created, to annotate the channel information with the events or not. channel_specific_preprocess=True will include the channel information, channel_specific_preprocess=False generalise event annaotaion.
- ✓ ch_names: This should be the signal's (EEG_root's) selected channels' order such that in default 'F3', 'F4', 'C3', 'C4', 'O1', and 'O2' are presented after re-referenced. Please check the chapter. 4.
- ✓ bad_epochs: The bad epochs and location extracted via the function "markBadEpochs" as shown in Fig. 3.4.
- ✓ notch_freq_essential_checker: Since the line noise presneted in the EEG signal may be varies, based on the apply the notch filter's need may be differs, inorder to check the line noise presence in the signal, this condition is given. If the user want to apply the notch filter for all this can be easily tuned off by assigning False.
- ✓ return_filtered_EEG: If this condition is True, then the filtered (bandpassed, and/or notch filtered) signal is returened by the function.

For more details please check the Chapter. 5.

Split into continious blocks and/ or extract spindle information via YASA

The spindle and slow-wave detection is done via YASA. If the detected spindles are used in the later analysis, the fuction spindle_detect_main is used for extract the spindle and slow-wave detection and o/p the continious segements. Since we are using the YASA, the unprocessed (not filtered) raw-EEG-data (EEG_root obtained via the function load_root_dataset as shown in Fig. 3.3) along with the prerocess information is used to extract the spindle information. As shown in the if block in line-5 to 27 as shown in Fig. 3.6. For more details please check the Chapter. "spindle_fun" (8).

If we don't need the spindle or slow-wave information, skip the part and directly extract the continious segments as shown in else block starts from line 29 of Fig. 3.6. For more details about the continious blocks please check the Chapter. "cont_segs_of_whole_EEG" (7).

```
#for the time being only the good ids
sel_ids = good_ids
#  When we are avoiding the spindle occuring location
#  while creating the spindle location the connious segmentation is done
if avoid_spindle_loc:
    #  for spindle detect ion we feed the unpreprocessed
    # (not filter applied) the EEG data
    data=deepcopy(EEG_root)

    # spindle detection intiated
    sw_columns_ordered =  ['Start', 'End', 'MidCrossing','Duration',
                'NegPeak' ,'PosPeak',  'ValNegPeak', 'ValPosPeak',
                'PTP', 'Slope', 'Frequency',  'IdxChannel']

    sp_columns_ordered =  ['Start', 'End', 'Duration', 'Peak', 'Amplitude', 'RMS',
     'AbsPower', 'RelPower', 'Frequency', 'Oscillations', 'Symmetry', 'IdxChannel']

    pred_slow_waves=True
    pred_spindles =True

    cont_EEG_segments,start_keys,cont_EEG_segments_np, sp_sw_dic_format =
        spindle_detect_main(in_name_temp, sel_ids, start_time_idx, data,
            sleep_stages_origin,save_spindle_loc, sel_id_name=sel_id_name,
            Fs=Fs, epoch_length=epoch_length, window_time=epoch_length,
            s_loc=0, e_loc=1,sw_columns_ordered=sw_columns_ordered,
            sp_columns_ordered=sp_columns_ordered,
            pred_slow_waves=pred_slow_waves, pred_spindles =pred_spindles)

else:
    # to obtain the continious segmentations for the loose lead detection
    # whether save the continious segments or not
    save_cont_seg=False
    cont_EEG_segments_use =  save_cont_segs(in_name_temp,
                        loading_dir_pre.out_loc_NREM_REM, sel_ids,
                        start_time_idx, Fs, sel_id_name='good_ids',
                        epoch_length=epoch_length, save_cont_seg=save_cont_seg)

    start_keys = list(cont_EEG_segments_use.keys())
    cont_EEG_segments_np = continious_seg_to_np(cont_EEG_segments_use, start_keys)
```

Figure 3.6: Split into continious segments and extract the spindle information

Choosing only the selected ids for the further processing

As shown in the Fig. 3.7, first save the raw-sleep stage and then only select the good_ids (annotated as "normal" by preprocessing). Then only the good_ids' signal and sleep stages are used for further analysis.

```
if normal_only:
sleep_stages_orgin = deepcopy(sleep_stages)
# saving the preprocessed sleep-stages
if sleep_stage_preprocess_origin:
    np.save(loading_dir_pre.out_loc_NREM_REM+ in_name_temp + '_sleep_stages_orgin',
                                                sleep_stages_orgin)

good_ids = np.where(epoch_status=='normal')[0]
sleep_stages = sleep_stages[good_ids]

epochs = epochs[good_ids]
try:
    epoch_start_idx = epoch_start_idx_o[good_ids]
except:
    epoch_start_idx =  np.array(epoch_start_idx_o)[good_ids.astype(int)]
```

Figure 3.7: Only extract the information of the "normal" annotated epoches in the preprocess for further analysis

MT-spectral estimation and correlation extraction

Overall preprocessing steps in time domain is completed. Here onwards a choice of extract the spectrogram via MT-estimation. Then the correlation can be extracted via the time domain or spectogram. And the correlation can be standardised (mapped) to Z-transform etc, for further analysis.

Let's check the possible inputs presented here with the combined extraction of MT-spectrogram and correlation together, via the function MT_based_correltion_calc_in_continious_segs (as shown in Fig. 3.8).

Inputs for the function "MT_based_correltion_calc_in_continious_segs",

- ✓ EEG_MT_feed: Either filterd EEG/ raw-EEG can be fed through the MT-estimation.
- ✓ Earlier extracted information such as sleep_stages, cont_EEG_segments_np, Fs,ch_names, and start_time_idx.
- ✓ T is an input related to MT-spectral estimation, to choose the window default (4sec for sampling rate Fs= 256).
- ✓ db_scale: False means the correlation values are calculated from the basic amplitude.
- ✓ f_min_interst=bandpass_freq[0]; this is the minimum frequency provided by the extracted MT-spectrum. Bandpass filtering frquency's lower band 0.5Hz is assigned as default.
- ✓ f_max_interst=bandpass_freq[1];; this is the minimum frequency provided by the extracted MT-spectrum. Bandpass filtering frquency's upper band 32.5Hz is assigned as default.
- ✓ window_time=epoch_length; combination of continious segments and window_time is used to extract only the interested MT-spectrum without loosing any useful information.
- ✓ padding=0; to add shift in the sample while spliting the sgements.
- ✓ sleep_stage_anoot_ext=True; while extracting MT-spectrum, extract their corresponding sleep-stage annaotaion for each seconds.

- ✓ root_data=True; split the given EEG-data's provided full channles as it is.
- ✓ interested_channels_index=[]; If the root_data False, this will choose the given channels.
- ✓ inter_mediate_transform=False; since the deafult using the extracted correlation as it is; this variable is false.
- ✓ b_val: If we do the Fisher-based standardisation (transform with tanh) on correlations inorder to satisfy the boundry conditions, like correlation (1 or -1), need to be adjusted with small negation or addition with b_val.
- ✓ optim_bandwidth=False; while choosing the MT-extraction tapers for spectral estimation, the paramter choices followed based on the publication pointed optimal bandwidth choice. Please check the chapter. 10
- ✓ save_MT_spectrum=True; the choice of return the MT-spectrum, when this true it will return.

In this function provide an option to the intermediate mapping on the correlations;

```
#this assign values are especillay for correlation based loose lead detection
# adjust the boundry values and mapped by tanh z_transform=True
b_val=opt_paramters.b_val
# either filterd EEG can be fed through the MT-estimation
if filtered_EEG_MT:
    EEG_MT_feed = EEG_root_copy
else:
    EEG_MT_feed = EEG_root
# as this functions is limited to 1 sec sliding through the MT-spectrum
#   as the corrrelation value of the temporal information is obtained in
# 1 sec resolution can be later down sampled like sliding 2 sec
#   by removing the one sample in the middle (kind of downsampling)
db_scale=False
correlation_flatten_MT_not_opt, sleep_stages_annot_flatten, MT_spec_not_db = \
    MT_based_correltion_calc_in_continious_segs(EEG_MT_feed, sleep_stages,
                cont_EEG_segments_np, Fs,ch_names, start_time_idx=start_time_idx,
                T=T, db_scale=db_scale,
                f_min_interst=bandpass_freq[0],f_max_interst=bandpass_freq[1],
                window_time=epoch_length, padding=0,  sleep_stage_anoot_ext=True,
                interested_channels_index=[],root_data=True,
                b_val=b_val,inter_mediate_transform=False,
                optim_bandwidth=False, save_MT_spectrum=True)
```

Figure 3.8: The main inputs for MT-extraction and correlation extraction together

For more details, please check the Chapter. "correlation_functions" (9) and Chapter. "Multitaper_class" (10).

Outlier detection in main scripts

Variance based method

Then the outlier detection can be done by the variance based method as mentioned in the

```
#  break the contuniious grops while consideriing the given spindle or
#  already predicted i/p given
break_spindle_flatten, break_flat_MT_flatten, z_transform,  inter_mediate_transform,\
                            Fisher_based, flat_MT_consider =
                            opt_paramters.intial_parameters_outlier_vertical_spikes(
                                    flat_MT_consider=opt_paramters.flat_MT_consider)
# since the  flat_MT need to be donebased on the region of standard deviation
#  that need to be selected whether region after preprcess or full
#  and the std_thres value need to be selected with caution
if flat_MT_consider:
    #to consider the detection as continious block or not
    cont_seg_wise=False
    # only_flat_MT=False
    # we can assign the strict threhold like this or we can assign the moving window
    # based approach
    std_thres=opt_paramters.std
    #this will only make the spikes this vlaue can be even lower like 3
    if not db_scale:
        #converting the MT_spec_raw to MT_spec_db
        MT_spec_db = 10*np.log10(np.concatenate(MT_spec_not_db,axis=2))
    else:
        MT_spec_db = np.concatenate(MT_spec_not_db,axis=2)
    # cont_seg_wise=True
    flat_MT = find_vertical_spikes(MT_spec_db,MT_spec_not_db,std_thres=std_thres,
                                                cont_seg_wise=cont_seg_wise)
else:
    flat_MT=[]
```

Figure 3.9: Variance based outler detection

♣ Codes - ✌ - explonation ♣
✍ Explonation (2.4, 11)
❄ Codes (11)

Figure 3.10: Veriance (standard deviation σ) based approach useful chapters and section for queick look-up

Correlation based methods

We can do the loose-lead detection (outlier detection) in plenty of ways as mentioned in the section "Correlation based approaches for outlier detection" (2.5). In this section just point the main chapters ralated to scripts related to the outlier detection.

But intialy the mean correlation iterative moving window based method (the **manuscript - review**) is presented. Such that first the hyper-parameters values for the function should be obtained as shown in the Fig. 3.11.

```
# Assigning the Hyper-parameters for the correlation based loose-lead detetion

# We have the choice to use the distribution or moving window.
# Any of these methods need the hyper-parameter assignment, for retrive the detect the
                                                    outlier segments
#  (possible potential loose-leads segments).
#  Here we used moving window based method.
tail_check_bin, GMM_based,    factor_check, threh_prob, outlier_basic_con,\
    moving_window_based, moving_window_size, th_fact,\
    sep, global_check, only_good_points,\
    sorted_median_cont_grp_comp_sort_median_cond, \
    sorted_median_cont_grp_comp_sort_max_cond, \
    sorted_median_cont_grp_comp_sort_quan_cond,\
    cont_seg_wise, cont_threh, threh_prob_artf_cont_seg, \
    with_conv, thresh_min_conv ,thresh_in_sec, \
    outlier_presene_con_lenth_th, with_fill_period\
        = opt_paramters.methodology_related_paramaters_for_outlier_detection()
#finalise the loose lead detection the following hyper parameters
# are just annotate the whole lead as loose-lead based on the conditions
loose_lead_period_min,\
    percentage_on_given_period_while_sliding,\
    overall_percent_check,\
    apply_conv_window,num_occurance, percent_check, loose_conv_wind,stride_size,
                                            conv_type = opt_paramters.
                                            assign_par_finalise_lead_loose_due_to_amoun_of_presente
                                            ()
```

Figure 3.11: Hyper parameters assignment for correlation based loose-lead detection

3.3 Output details

Lets see the possible outputs in details.

- ev {0,1} – event {0,1} file name for saving the final results in events with sleep-stage related epoches.
- dic {0,1}, – dictionary {0,1} ‘ Save the meta data from the edf in dictionary format as pickle
- b {0,1}, – bad_epochs {0,1} Check th ebad epoches in the provided edf to focus only in the sleep-related signals
- sleep {0,1}, – sleep_annot {0,1} save the sleep-annotation in npy file
- out {0,1}, – outlier {0,1} save the predicted outliers
- MT {0,1}, – MultiTaper {0,1} save the predicted multitapers outcome
- outNREM {0,1}, – outlierNREM_REM {0,1} save the predicted outliers as NREM and REM
- sp {0,1}, – spindles {0,1} First predict the spindles via the YASA, then avoid the spindles while predicting the outliers
- all {0,1}, – sel_all {0,1} select all the options for saving the directory

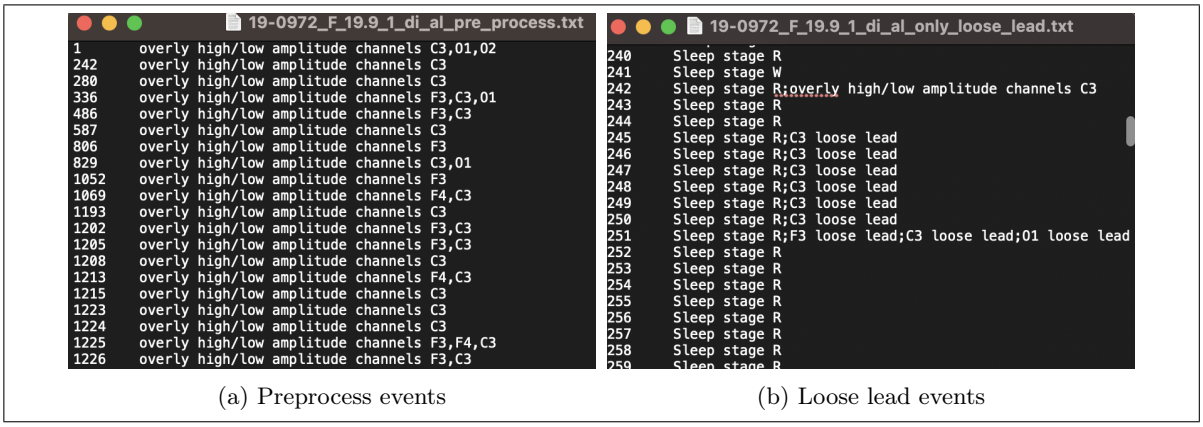
3.3.1 -ev or –event

This is the sleep related events from the sleep EEG (".edf") files. The events are annotated by the sleep-expert while recording the sleep-EEG. They used the guidelines to mark the sleep-related events like N1, N2, N3, R, W.

Since the experts uses different anotation methodlogies the sleep annaotaions can be annotated in different ways. For an example the N1 can be annotated as n1, 1, N-1, etc.

Further, the experts also annotate the events not related to sleep as well, like the patient movement, ligt on, bathroom break, disconnection of recording, etc.

The packge is designed to focus on the sleep-related epoches like NREM and/or REM to find the loose-leads, such that the outcomes for loose-leads mainly marked for NREM and/or REM. Rest is marked from the preprocessing. Example of prediction is shown in the Fig. 3.12.



(a) Preprocess events (b) Loose lead events

Figure 3.12: Events output from the package snip shots.

As shown in the Fig. 3.12a, the preprocess anotated events marks the prepocess detected events like overly high/lower amplitude channels information presented in the epoch along with the sleep-annotations. The final o/p events annotate the epoches with the loose-lead presneted in the sleep-related epochs with the preprocess annaotaions like shown in the Fig. 3.12b.

Chapter 4

load_EDF_dataset_events

This chapter covers the functions related to the preprocessing.load_EDF_dataset_events.py. The **main function** that uses the dependent libraries like (mne), to **load the ".edf" file** to **extract signals** and **relative events** positional information.

```
1  import numpy as np
2  import mne
3  import logging
4
5  from copy import deepcopy
6  from sleep_EEG_loose_lead_detect.GUI_interface.percentage_bar_vis \
7  import percent_complete
8
9  # logger intialisation
10 logging.basicConfig(level=logging.INFO)
11
12 logger = logging.getLogger("load_EDFS_with_events")
13 while logger.handlers:
14       logger.handlers.pop()
15 c_handler = logging.StreamHandler()
16 # link handler to logger
17 logger.addHandler(c_handler)
18 logger.setLevel(logging.INFO)
19 logger.propagate = False
```

Figure 4.1: load_root_dataset intialisation importing Dependencies

The Fig. 4.1 shows the intiation step for the function.

Since the load_root_dataset function is big, that is splited into three parts and presented in three Figures such as Fig. 4.2,Fig. 4.4 and Fig. 4.12.

The load_root_dataset in the line-24 of Fig. 4.2 shows the read the raw edf via "mne". Line-28 extract the sampling frequency information from the edf file. Since the "mne" load the edf in "V" to convert back to uV as shown in line-32.

```
def load_root_dataset(in_edf, epoch_sec, micro_volt_scaling=1e6,
                      re_reference_mastoid=True,verbose=False,GUI_percentile=True):

    '''
    I/P:
        in_edf: edf location and edf name
        epoch_sec: 30 sec

    re_reference_mastoid:    giving option to chooose the channels without
        re-refencing via the re_reference_mastoid

        re-referencing is always prefered due to aviod the channel miss placement
                                            issue
    '''
    ch_names = ['F3', 'F4', 'C3', 'C4', 'O1', 'O2', 'M1', 'M2']


    # Load edf file
    if verbose:
        logger.info('loading %s iniated',in_edf)
    if GUI_percentile:
        percent_complete(1, 100, bar_width=60, title="Loading-edf", print_perc=True)

    # try:
    edf = mne.io.read_raw_edf(in_edf, preload=False, verbose=False, stim_channel=None)
    # except:
    #     edf = mne.io.read_raw_edf(in_edf, preload=False, verbose=False, stim_channel
                                            =None, encoding='latin1')

    sampling_freq = edf.info['sfreq']
    signals = edf.get_data(picks=ch_names)

    # mne automatically converts to V, convert back to uV
    signals *= micro_volt_scaling

    if verbose:
        logger.info('annotation extaraction %s iniated',in_edf)
        logger.info('sampling frquency '+str(sampling_freq))

    if GUI_percentile:
        percent_complete(20, 100, bar_width=60, title="Loading-edf", print_perc=True)


    annotations = edf._annotations
    whole_annotations = np.concatenate([annotations.onset[:, None].astype(object),
                           annotations.duration[:, None].astype(object),
                           annotations.description[:, None].astype(object)], axis=1)
    # re-reference the channels
    if re_reference_mastoid:
        signals_root = np.array([
            signals[ch_names.index('F3')] - signals[ch_names.index('M2')],
            signals[ch_names.index('F4')] - signals[ch_names.index('M1')],
            signals[ch_names.index('C3')] - signals[ch_names.index('M2')],
            signals[ch_names.index('C4')] - signals[ch_names.index('M1')],
            signals[ch_names.index('O1')] - signals[ch_names.index('M2')],
            signals[ch_names.index('O2')] - signals[ch_names.index('M1')],
            ])
    else:
        signals_root = deepcopy(np.array([
        signals[ch_names.index('F3')],
        signals[ch_names.index('F4')],
        signals[ch_names.index('C3')],
        signals[ch_names.index('C4')],
        signals[ch_names.index('O2')]]))
        signals[ch_names.index('O1')],

    ch_names  = get_assigned_channels()
```

Figure 4.2: load_root_dataset part 1

4.1 Channels in raw EEG-file (.edf)

Channel selection details about the choise of re-referencing etc is already dissussed in the subsection 1.1.2. This section only covers about the package. Since the package is build to detect the sleep EEG present using the six channels such as,

1. F3
2. F4
3. C3
4. C4
5. O1
6. O2

With the the re-referencing channels such as M1, and M2. To select the EEG-channels and related signal inforation using the "mne" package.

The load_root_dataset in the line-14 of Fig. 4.2 shows the ch_names order going to be selected from the edf. Inorder to do this line-29 of Fig. 4.2 picks the channels with the provided channel names order. This function comes along with the predefined channels information. And the re-referencing is done on line-47 of Fig. 4.2 re_reference_mastoidchoice.

After the re-referencing. Just need the channel inforation. To make the life easier The re-referenced channels also named with the normal channels, such that "F3" represents the "F3-M2", and "F4" represents "F4-M2", etc. This part is done by line-65 of Fig. 4.2 via the get_assigned_channels.

1. F3: F3-M2
2. F4: F4-M1
3. C3: C3-M2
4. C4: C4-M1
5. O1: O1-M2
6. O2: O2-M1

Inorder to change the channels inforation, and/or re-referencing choice. Like choosing the channels present in the edf file to load. This is obtained from the function get_root_channels Fig. 4.3. Basically calling mne's object to extract the channel details.

```
1        def get_root_channels(in_edf):
2            # Load channel information while loading the edf
3            edf = mne.io.read_raw_edf(in_edf, preload=False, verbose=False,
                                       stim_channel=None)
4            return edf.ch_names
5
```

Figure 4.3: Get the available channel information

From the extracted channels. With the EEG channels, other possible PSG channels, and other channels which can be including the "EMG" and/ or "EOG" electrodes with the EEG channels.

Possible "EOG" electrode annotation can be such as,

1. ROC
2. E1
3. LOC
4. E2

Possible "EMG" electrodes can be from the chin such as,

1. CHIN1
2. CHIN2
3. CHIN3

Possible "EMG" from the legs

1. RAT1
2. RAT2
3. LAT1
4. LAT2

Apart from these there are lot of possible channels with their differennt annaotaions as shown in the list below.

- ✓ F7
- ✓ O1
- ✓ DIF6
- ✓ X2
- ✓ PR
- ✓ DC1
- ✓ X6
- ✓ LAT2
- ✓ T3
- ✓ X8
- ✓ F8
- ✓ LOC
- ✓ E2
- ✓ POS
- ✓ C3
- ✓ X5
- ✓ DC7
- ✓ ABD
- ✓ DC4
- ✓ C4
- ✓ DC5
- ✓ PRES
- ✓ DC9
- ✓ F3
- ✓ LEAK
- ✓ X3
- ✓ NASAL
- ✓ ECGL
- ✓ ROC
- ✓ LAT1
- ✓ ORAL
- ✓ Pz
- ✓ ECGR
- ✓ CPRESS
- ✓ A1
- ✓ T5
- ✓ Fp1
- ✓ ARM
- ✓ ARMS
- ✓ X4
- ✓ C-FLOW
- ✓ O2
- ✓ Fpz
- ✓ Fz
- ✓ DIF5
- ✓ DIF1
- ✓ CHEST
- ✓ OSAT
- ✓ TCO2
- ✓ Cz
- ✓ SNORE
- ✓ DIF3
- ✓ DC3
- ✓ M2
- ✓ P4
- ✓ DC2
- ✓ A2
- ✓ DC6
- ✓ CFLOW
- ✓ PTAF
- ✓ X1
- ✓ E1
- ✓ CHIN2
- ✓ FLOW
- ✓ CHIN1
- ✓ DC8
- ✓ PTAF -
- ✓ F4
- ✓ DC10
- ✓ X7
- ✓ T6
- ✓ RAT1
- ✓ Fp2
- ✓ M1
- ✓ T4
- ✓ RAT2
- ✓ P3
- ✓ DIF2
- ✓ APNEA
- ✓ EtCO2
- ✓ DIF4
- ✓ TcCO2
- ✓ CHIN3
- ✓ EtCO2 Wave
- ✓ HEARTRATE
- ✓ Heart Rate

As you can see from the list “HEARTRATE” and “Heart Rate” shows the same information with different annotation. Some times some channels are not connected; if we already know the information, we can safely ignore those channels.

4.2 Events details from raw-EEG (.edf)

```
        # Get sleep stage annotations
        # These are all possible sleep stage events from the known data
        # made changes in the n4 stage is especially annotated by the 0
        # and the unknows are annotated by 6
        sleep_stage_event_to_id_mapping = {'sleep stage w': 5,
                                           'sleep stage r': 4,
                                           'sleep stage 1': 3,
                                           'sleep stage 2': 2,
                                           'sleep stage 3': 1,
                                           'sleep stage 4': 0,
                                           'sleep stage n1': 3,
                                           'sleep stage n2': 2,
                                           'sleep stage n3': 1,
                                           'sleep stage n4': 0,
                                           'sleep stage n': 6,
                                           'sleep stage ?': 6}

        ev_or, event_ids = mne.events_from_annotations(edf,verbose=False)

        relevant_sleep_events = {}
        for e_name in event_ids:
            if e_name.lower() in sleep_stage_event_to_id_mapping:
                relevant_sleep_events[e_name] = sleep_stage_event_to_id_mapping[e_name
                                                .lower()]

        events, _ = mne.events_from_annotations(edf, relevant_sleep_events,
                                                chunk_duration=epoch_sec,verbose=False)

        if verbose:
            logger.info('events are assigned')
        if GUI_percentile:
            percent_complete(50, 100, bar_width=60, title="Loading-edf", print_perc=
                                                True)

        window_size = int(epoch_sec*sampling_freq)

        '''
        we can directly use the events information
            to Calculate start position ids for each epoch starting from labeled sleep
                                                 stages

        '''
        # start_ids=events[:,0]
        len_signals = signals_root.shape[1] # (Channels x Data)
        start_ids = np.arange(events[0][0], len_signals-window_size+1, window_size)

        '''
        to trace the orgin of the edf events

        '''
        sel_index_sl_st, flag_other_start_id_extraction =  obtain_sel_index_sl_st(
                                                ev_or, events, start_ids)
```

Figure 4.4: load_root_dataset part 2

4.2.1 Mapping the Sleep-related-events presented in the edf file to common form

In the line-5 of create the mapper "*sleep_stage_event_to_id_mapping*" Fig. 4.4 map the raw-EEG sleep related events. In the line-18 of Fig. 4.4 get the annotated events presented in the raw-EEG (.edf) file. line-21 for loop going through the event_ids and assign the event_id to common form via *sleep_stage_event_to_id_mapping* (e.g: N3, n3, 3 all are representing NREM-3 mapped to 1), and create the new mapper for this corresponding raw-EEG (.edf) as *relevant_sleep_events*. Then the mapper (*relevant_sleep_events*) is fed in line 25 to get the events with the given mapped common annotation.

Some time the annotation presented in "*sleep_stage_event_to_id_mapping*" not covers the sleep-related/ other events interested by the user. Inorder to obtain/ check the presented events, use the function shown in Fig. 4.5. Since the captilisation make the same event can be presented in different ways, in this function, we make all in lower case. From this obtained events, we can easily create the key to map the event via the mapper "*sleep_stage_event_to_id_mapping*"; such that this leads to mapping via *relevant_sleep_events*.

```
1
2  def get_event_infor(in_edf):
3      # Load edf file with event information
4      edf = mne.io.read_raw_edf(in_edf, preload=False, verbose=False, stim_channel=None)
5      _, event_ids = mne.events_from_annotations(edf)
6
7      events_all =[]
8      for e_name in event_ids:
9          events_all.append(str(e_name.lower()))
10     return events_all
```

Figure 4.5: Get the available events information

```
1      def get_channel_event_infor(in_edf):
2          # Load edf file with event information
3          edf = mne.io.read_raw_edf(in_edf, preload=False, \
4                          verbose=False, stim_channel=None)
5          _, event_ids = mne.events_from_annotations(edf)
6
7          events_all =[]
8          try:
9              for e_name in event_ids:
10                 events_all.append(str(e_name.lower()))
11         except:
12             logger.warning(in_edf+ " event details issue")
13
14         try:
15             ch_names=edf.ch_names
16         except:
17             logger.warning(in_edf+ " channel names details issue")
18             ch_names=[]
19
20         return ch_names, events_all
```

Figure 4.6: Get the available channel and events information together

From one of the analysis extracting the information from the $\approx 8,500$ subjects end up in $\approx 35k$ annotations, this is attached in the file "events_all.txt" . These annotations include the not useful annotations as well.

Further, the histogram shown in the Fig. 4.7. Shows the length of characters' frequency of unique events among the $\approx 35k$ annotations.

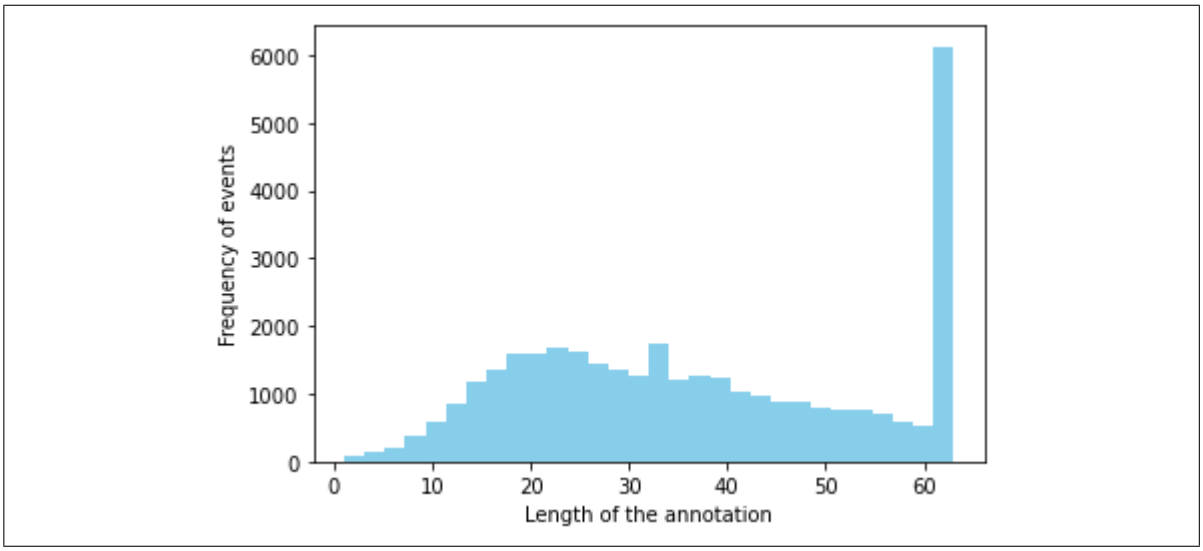


Figure 4.7: Histogram of unique events annotations

Examples of less than 5 characters, from the 139 obtained

✓ ti	✓ rem	✓ n2?	✓ plms	✓ gt	✓ tir'	✓ ?	✓ eeg
✓ to	✓ tir	✓ uars	✓ ekg	✓ ti	✓ p	✓ ??	
✓ rera	✓ ril	✓ wake	✓ tor	✓ pa	✓ csb?	✓ re	

Example of 63 characters, this is the maximum annotated annotation extracted via the mne library extracted via this package.

- ✓ leads are all over the place since pt keeps turning her head si
- ✓ mom wanted to know pt's hr and said she is giving her meds that
- ✓ mom was waving hand in front of pt face as if she thought she w
- ✓ pt irritated by cpap- i may have to keep her pressure low if sh
- ✓ on 0.2 l/m o2 she is still at 88-91% will watch for a while lon
- ✓ pt showing central apneas as well as apneas and hypopneas with
- ✓ pt mentioned before study she sleeps only on side due to hip pa
- ✓ pt. is s;ighly elevated in bed,...says she cannot sleep flat in

4.2.2 Seperate the signal's starting points with the relative events

In the line-32 of Fig. 4.4 shows the window_size (signal length) of the sleep-related epoch (in US according to ASSAM annotation currently the epoch_sec = 30 sec) multiplied by the sampling frequency (F_s) which endup in the signals samples.

In line 40 len_signals presents the overall signal length. Then in start_ids are split the overall signal into epoch sizes. However, if we already know the overall signal present in the raw-EEG (.edf) file only have the annotated sleep stages this works. Mostly we don't know this is true unless we record the edf, such that we need to check the events annotatio positions (ev_or extracted from the edf on line-18 of Fig. 4.4) come along with the raw-EEG (.edf) to track which is true.

In line 47 sel_index_sl_st is obtained via the function (obtain_sel_index_sl_st) as shown in the Fig. 4.8. Such that go through the start_ids with the ev_or positions and only select the releavant epochs, if atleast one of the start_id not present in the ev_or then flag (flag_other_start_id_extraction) the given methodolgy to split the signal not work for this raw-EEG (.edf). This will trigger alternative way (function in Fig. 4.10) to extract the sel_index_sl_st, and start_ids.

```
1  def obtain_sel_index_sl_st(ev_or, events, start_ids):
2      '''
3      since the algorithm
4      based on the sleep events the edf file must have the
5      annoated events with the intended sleep-events in the same duration
6      go through the start_ids and select only the selected events
7
8      need to hold this if we are going to provide the detected annotations map back to
                                                the edf
9
10     '''
11     flag_other_start_id_extraction=False
12
13     sel_index_sl_st=np.zeros((len(start_ids)))
14     p=0
15     for s in start_ids:
16         try:
17             sel_index_sl_st[p]=np.where(ev_or[:,0]==s)[0][0]
18         except:
19             flag_other_start_id_extraction=True
20             break
21         p+=1
22
23     # since the indexes are integer this whould be
24     sel_index_sl_st = sel_index_sl_st.astype(int)
25     if not flag_other_start_id_extraction:
26         if  not np.array_equal(ev_or[list(sel_index_sl_st),0],events[:,0]):
27             logger.error("Events orgin not match with the selected events, please
                                                report with the edf file to track the issue
                                                ")
28             flag_other_start_id_extraction=True
29
30
31     return sel_index_sl_st, flag_other_start_id_extraction
```

Figure 4.8: obtain_sel_index_sl_st function code

Alignment on large .edf files

☞ Due to the sleep-EEG (.edf) files are bigger inorder to save the files effectively, the signals are broken and left when the subject is left in the middle (like bathroom break etc.), this is dependent on the software used to save the ".edf" files and the technician save the file.

The EEG files are recorded over the night with the multiple channels, the software devoloped for save the ".edf" files with the technician present can optimally save the EEG-signals information; like avoiding the unwanted signals. For an example: The subject is dissconnected from the EEG-chhannels for bathroom break, can cause the unwanted signal. The software provided can easily avoid the bigger chunk of data, interuption.

The EEG (".edf") files are obtained in

- ✓ different nights of recording
- ✓ different technicians (perspective and knowledge)
- ✓ technician' s inputs. (their dedication, day, etc.)
- ✓ the subjects medical conditions and age, and the care-takers (for children), etc.
- ✓ Recordings are carried out for years, the software version updates, and the noise due to external power sources (added extra instruments, power sources, buildings, etc.).

These combinations makes difficult to define a single way of find the extracted signal properly.

These can be easily avoided when the recordings are in small scale, while manually check them. But when the recordings are in thousands and continously growing, this is time consuming and impossible.

☞ If the interuptions are not properly handled while re-create the signal with the annotations. Then the signal obatined may have the shift. Thus these shifts neeed to be identified and aligned for proper EEG-signal analysis.

The possible interuptions are mostly found in the bad-events, just some known events that is surely known as bad-events we have provided the bad events in the */docs/bad_events_bathroom.txt* Please check the intial subsection "Extract the bad epoch information and save the information " (3.2.1).

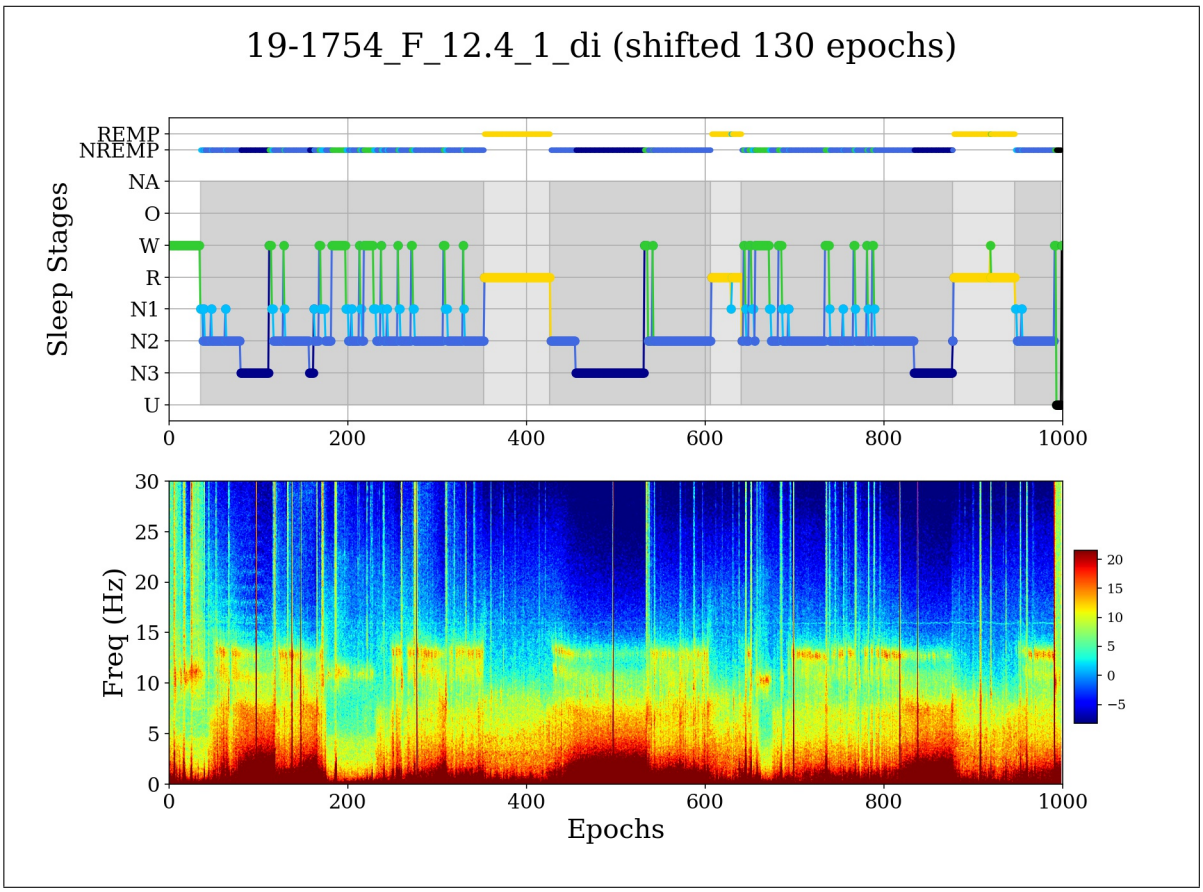


Figure 4.9: Image credit goes to Dr. Amlan, the obtained spectrum shows the subjects sleep-stages are shift by 130 epochs to obtain the final outcome.

We also encountered this miss alignment issue on sleep stages and the EEG spectrums when we start the project on the larger sleep=EEG data.

Dr. Amlan pointed out the cause of the issue and he provided the aligned outputs as well. Such the aligned spectrum with the issue is shown in the Fig. 4.9. Here the multi spectrum is obtained with 30 sec window.

A helper function (Fig. 4.11) is provided to check the alignment of the file. Basically checking the ".edf"files' annotated time and the signal length are same.

This is a helper function "re_intiate_manually_obtain_sel_index_sl_st" (in Fig. 4.10) is used on the assumption of aligned "edf" file is obtained by manual assignment.

Prefer, if you are not sure how to align the file, based on the number of files effected in the saved "edf"; Feel free to leave the files or check the aligned files and align them accordingly. ☞ This helper function only gives an idea about checking the alignment. Not align the provided file.

```python

def re_intiate_manually_obtain_sel_index_sl_st(whole_annotations,events,ev_or,
                                               sleep_stage_event_to_id_mapping,
                                               # start_id_manual_or,
                                               window_size,
                                               epoch_sec=30):
    '''
    recheck the sleep stage events for robust edf handling
    if any breaks happens in the middle lets reintiae and calculate
    '''
    # start_id_manual=deepcopy(start_id_manual_or)
    start_id_manual=[]
    sel_index_sl_st_manual=[]
    # sleep_stages_epoch_wise=[]
    i=-1
    for t, d, desc in whole_annotations:
        i+=1
        epoch_id = int(t//epoch_sec)

        # Check if this is a sleep stage annotation
        if desc.lower() not in sleep_stage_event_to_id_mapping:

            continue
        label = sleep_stage_event_to_id_mapping[desc.lower()]
        n_epochs = int(d//epoch_sec)

        for j in range(n_epochs):
            # sleep_stages_epoch_wise.append(label)
            index_raw = int((epoch_id + j) * window_size)

            #this will make sure to place the right startindex for sleep-stages
            # start_id_manual[i]=index_raw
            start_id_manual.append(index_raw)
        sel_index_sl_st_manual.append(i)

    if not np.array_equal(ev_or[sel_index_sl_st_manual,0],events[:,0]):
          logger.error("Events orgin not match with the selected events, please report
                                                with the edf file to track the issue")
    return np.array(sel_index_sl_st_manual),start_id_manual
```

Figure 4.10: re_intiate_manually_obtain_sel_index_sl_st function code

```python
def isEDFAligned(eeg_data_raw, annotations, sampling_freq, epoch_size):
    '''
    function credit goes to Dr. Amlan Talukdar
    This function just help to check the provided annotation
                              aligined with the given edf file

    As from the analysis some times the end criteria may not full-fill
     (use the function with your own-risk)

    '''
    whole_annotations = np.concatenate([annotations.onset[:, None].astype(object),
                                        annotations.duration[:, None].astype(object),
                                        annotations.description[:, None].astype(object
                                            )], axis=1)

    # Check if the last annotation time matches the last EEG epoch
    last_annot_time = whole_annotations[-1][0] + whole_annotations[-1][1]
    last_annot_epoch = int(last_annot_time//epoch_size)
    last_signal_epoch = int(eeg_data_raw.shape[1]//int(epoch_size * sampling_freq))

    return last_annot_epoch == last_signal_epoch
```

Figure 4.11: isEDFAligned function code

```
        '''
        if any break events occurs in the middle this is flagged and manusally extract
                                            the events
        '''
        if flag_other_start_id_extraction:
            logger.warn(in_edf+" Selected events start ids are not continious, seems
                                            some breaks may happened in the recording")
            logger.warning('Due to this missalignment due to time stamp missing; the
                                            down strean analysis is performed based on
                                            assumption of alignment')
            logger.warning('Assumption:  health check performed by our precheck tool
                                            with the EEG and the obtained aligned edf
                                            is used for the analysis')

            sel_index_sl_st, start_ids = re_intiate_manually_obtain_sel_index_sl_st(
                                            whole_annotations,events,ev_or,

                                            sleep_stage_event_to_id_mapping,

                                            window_size, epoch_sec=epoch_sec)
            logger.warning('Since the edf file missing the time stamp with the signal
                                            we cannot fully gurantee the outcomes')


        if verbose:
            logger.info('assigning the selected sleep-ids')
        if GUI_percentile:
            percent_complete(80, 100, bar_width=60, title="Loading-edf", print_perc=
                                            True)

        sleep_events = np.zeros(signals.shape[1]) + np.nan
        for start, _, stage in events:
            sleep_events[start:start+window_size] = stage

        '''
        # Write all sleep related events to file
        fname = out_loc + in_name + "_edfstage.txt"
        evt_df.to_csv(fname, sep='\t', header=True, index=True)
        '''
        if verbose:
            logger.info('loading edf completed')
        if GUI_percentile:
            percent_complete(100, 100, bar_width=60, title="Loading-edf", print_perc=
                                            True)
        logger.info('')
        logger.info('')

        return signals_root, sleep_events, ch_names, sampling_freq, start_ids,
                                            sel_index_sl_st, whole_annotations, ev_or
```

Figure 4.12: load_root_dataset part 3

4.3 Issue of sleep-signal and annotation (sleep-stage) miss alignment on large scale data

Such that in our analysis, we always check the signals are aligned with the annotations. Dr. Amlan provide the aligned signals and annotations.

Chapter 5

segment_EEG

The Fig. 5.1 shows the intialisation like loading packges and printing the statuses etc. The Fig. 5.2 is the events annotations predicted by the time domain preprocessing stage.

```
1  import logging
2  import numpy as np
3  from scipy.signal import detrend
4  from scipy.stats import mode
5  from mne.filter import filter_data, notch_filter
6
7  from sleep_EEG_loose_lead_detect.preprocessing.power_noise_handler \
8                          import powerline_noice_magnificant_checker
9  from sleep_EEG_loose_lead_detect.GUI_interface.percentage_bar_vis  \
10                         import percent_complete
11
12 # logger intialisation
13 logging.basicConfig(level=logging.INFO)
14
15 logger = logging.getLogger("preprocess_time")
16 while logger.handlers:
17       logger.handlers.pop()
18 c_handler = logging.StreamHandler()
19 # link handler to logger
20 logger.addHandler(c_handler)
21 logger.setLevel(logging.INFO)
22 logger.propagate = False
```

Figure 5.1: segment_EEG intialisation importing Dependencies

```
1  epoch_status_explanation = ['normal',
2                              'NaN in sleep stage',
3                              'NaN in EEG',
4                              'overly high/low amplitude',
5                              'flat signal',
6                              'bad events']
```

Figure 5.2: epoch_status_explanation

The Fig. 5.3 from line-47 shows the preprocessing annotate all epochs as “normal” (from the Fig.

5.2)

5.1 NaN in sleep stage

Intial NaN and Inf annotations based on the label-annotations. The Fig. 5.3 from line-30 to 54 shows the preprocessing annotate the infinity and nan values as “NaN in sleep stage”; based on checking the events (labels) annotations.

```
def segment_EEG(EEG, labels, window_time, Fs, start_ids,
    notch_freq=None,bandpass_freq=None,
    start_end_remove_window_num=0, amplitude_thres=2000,
    to_remove_mean=False, bad_epochs=None,  notch_freq_essential_checker=False,
    channel_specific_preprocess=False, ch_names=[],
    return_filtered_EEG=False, quantile_normalise_time=False,
    verbose=False,GUI_percentile=True ):
    # identify the flat signal and remove them
    std_thres = 0.2
    std_thres2 = 1.
    flat_seconds = 5
    padding = 0

    if to_remove_mean:
        EEG = EEG - np.nanmean(EEG,axis=1, keepdims=True)
    window_size = int(round(window_time*Fs))
    # ------------------
    # step_time : int
    # DESCRIPTION. used to check the flat signal
    #  while sliding window (with window time) with the sliding step step-time apart
    # ------------------
    # step_size = int(round(step_time*Fs))
    step_size = window_size
    flat_length = int(round(flat_seconds*Fs))

    # other preprocessing steps like remove the infinity and nan values etc.
    if verbose:
        logger.info('Nan and infinity check intiated')
    labels_ = []
    for si in start_ids:
        labels2 = labels[si:si+window_size]
        labels2[np.isinf(labels2)] = np.nan
        labels2[np.isnan(labels2)] = -1
        # to keep the same kind of syntax and compbility
        label__ = mode(labels2,keepdims=True).mode[0]
        if label__==-1:
            labels_.append(np.nan)
        else:
            labels_.append(label__)
    labels = np.array(labels_)

    if verbose:
        logger.info('Nan and infinity check assigned')
    if GUI_percentile:
        percent_complete(10, 100, bar_width=60, title="Preprocess", print_perc=True)
    # first assign normal to all epoch status
    epoch_status = [epoch_status_explanation[0]]*len(start_ids)
    # check nan sleep stage
    if np.any(np.isnan(labels)):
        ids = np.where(np.isnan(labels))[0]
        for i in ids:
            epoch_status[i] = epoch_status_explanation[1]
```

Figure 5.3: segment_EEG part-1

5.2 Filtering EEG-Signals

```
    # just a constant to check the notch filter status
    notch_filter_skipped=True
    if verbose:
        logger.info('Notch filter check intiated')

    # just check the essential ity of the notch filter applying need
    if Fs/2>notch_freq and bandpass_freq is not None \
                        and np.max(bandpass_freq)>=notch_freq:
        # notch_freq_essential_checker check the  signal's power line noise presence
        # if that powerline noise is already supressed by the recorder
        #       we can skip the power -line noise notch filter
        # this is performed by only checking by
        # one of the EEG channel's one epoch by applying FFT
        if notch_freq_essential_checker:
            ch_sel=1#for F4 channel in default input
            ids_normal = np.where(~np.isnan(labels))[0]
            EEG_segs_sel = EEG[ch_sel,list(map(lambda x:np.arange(x-padding,x+
                                        window_size+padding), start_ids[[ids_normal
                                        [0],ids_normal[len(ids_normal)//2]]]))]
            if powerline_noice_magnificant_checker(EEG_segs_sel,Fs):
                EEG = notch_filter(EEG, Fs, notch_freq, fir_design="firwin", verbose=
                                        False)
                notch_filter_skipped=False
            else:
                if verbose:
                    logger.info("power line noise is not much just left without Notch
                                        filter")
                if GUI_percentile:
                    percent_complete(30, 100, bar_width=60, title="Preprocess",
                                        print_perc=True)
        else:
            EEG = notch_filter(EEG, Fs, notch_freq, fir_design="firwin", verbose=False
                                        )
            notch_filter_skipped=False
            if verbose:
                logger.info('Notch filteration done ')
            if GUI_percentile:
                percent_complete(30, 100, bar_width=60, title="Preprocess", print_perc
                                        =True)

    #  Aplying the band pass filter
    if bandpass_freq is None:
        fmin = None
        fmax = None
    else:
        fmin = bandpass_freq[0]
        fmax = bandpass_freq[1]
        if fmax>=Fs/2:
            fmax = None

    if bandpass_freq is not None:
        if verbose:
            logger.info('Band pass filteration  begin with %.2f Hz, %.2f Hz',fmin,fmax
                                        )

        EEG = filter_data(EEG, Fs, fmin, fmax, fir_design="firwin", verbose=False)
```

Figure 5.4: segment_EEG part-2 focused on Filtering

This function considers all the signals present in the extracted signals are EEG. Otherwise the filter using bandpass will create a negative impact. So, make sure to check the other channels extracted/ or used for other purpose.

The code portion presented in Fig. 5.4 focused only the filteration. There are two main section of filtering, like the get-rid of power line noise. And get remove the unwanted signals via the bandpass filter.

5.2.1 Band pass filter

Since the bandpass filtering can skip the need for notch filter and other unwated noise presented in the signal, such that it is first presented in before notch filter. Here we are just calling the helper filter data provided by the mne package as shown in the line-48 Fig. 5.4. The default bandpass frequecy provided by this package is 0.5-32.5Hz.

5.2.2 power_noise_handler

The main input to avoid the power-line interfearence via the notch filter, since the power line frequency varies across the world (America and some part of Asia is 60Hz, large part of the world is 50Hz). Such that this notch_freq (line_freq: line frequency) should be chosen with the raw-EEG's origin.

Essentiality check for notch filter

Apart from this the before we use the digital notch filter, we need to check the imapct of using the notch filter. If the notch filter is included in the pipeline, it will just create a unneccessary notch filter's noice in the useful signal. Such that firt the check the need for notch filter if that already avoided by the less sampling rate, like Nyquist frequency (Fs/2 $<$ power line frequency).

The EEG signals related to sleep has higher importance in the lower frequency. The line frequency most of the time above the interested bands (normally 0-35Hz). Such that if we are using the bandpass frequency in the later, part in the pipeline, we can skip the notch filter.

As shown in the line 7 shown in Fig. 5.4, if atleast one of the above conditions (and) not met. Then, the notch filter can be applied.

```
1
2  import numpy as np
3  from scipy.fft import fft
4
5  def powerline_noice_magnificant_checker(EEG_segs_sel,Fs,noice_f=60,min_in_f=5,max_in_f
                                            =25):
6      '''
7      EEG_segs_sel: this the selected normal epoches of one channel in far away
8      noice_f: notch frequency; 60 Hz for the US powerline noice
9
10     to be more carefull avoid the noice by the DC shift and focus on the EEG sleep
                                            data interest portion
11
12     min_in_f: 5
13     max_in_f: 25
14
15     '''
16     avg_powers =np.zeros((np.size(EEG_segs_sel,axis=0),2))
17     for e in range(0,np.size(EEG_segs_sel,axis=0)):
18         y=EEG_segs_sel[e]
19         N=len(y)
20         dt=1/Fs
21         # classical FFT: since we are only cheking the power line noise exist so we
                                            can safely apply FFT
22         #   seems the scipy.fft is faster only that is chosen
23         yf = fft(y)
24         xf = np.linspace(0.0, 1.0/(2.0*dt), N//2)
25
26         avg_powers[e,:] = helper_for_fft_power_calc(xf,yf,noice_f, min_in_f, max_in_f)
27
28     avg_power_near_powerline, avg_power_not_powerline = np.mean(avg_powers,axis=0)
29
30     # then finally  says the signal need to go through the notch filter
31     if avg_power_near_powerline > avg_power_not_powerline:
32         return True
33     else:
34         return False
```

Figure 5.5: power_noise_handler part1 code

However, if the power line noise is already filtered by technician's effort by the hardware/ software (versions). Then re-filteration is just make extra filtered noice, such that in this package. I have provided an option (notch_freq_essential_checker) to check the need for applying notch filter from line 14-17 shown in Fig. 5.4 via the function powerline_noice_magnificant_checker in Fig. 5.5.

Here just sample the part of the signal (just one channel) is selected and check the power line power vs the other power presented in the signal while avoiding the DC-shift and lower frequencies. The default power line checker come with the 5-25Hz vs 60Hz. Inorder to do that first extract the powers via the function helper_for_fft_power_calc in Fig. 5.6, through the FFT. and their average absolute power.

```
def helper_for_fft_power_calc(xf,yf,noice_f, min_in_f, max_in_f):
    '''
    This function is for check the average power near to the powerline/ given
                                            frequnecy
    and the intended frequecy range

    xf: the ffts frequency axis
    yf: the ffts's powers
    '''
    indexes_near_to_powerline=np.where((xf>(noice_f-1)) & (xf<(noice_f+1)))[0]
    avg_power_near_powerline = sum(abs(yf[indexes_near_to_powerline])**2)/len(
                                            indexes_near_to_powerline)

    # this will sleect the inetersetd region default is designed to avoid the noice by
                                             the DC shift in EEG sleep analysis

    indexes_out_of_powerline_dc_shift_sleep_EEG = np.where((xf>min_in_f) & (xf <
                                            max_in_f))[0]
    avg_power_not_powerline = sum(abs(yf[indexes_out_of_powerline_dc_shift_sleep_EEG])
                                            **2)/len(
                                            indexes_out_of_powerline_dc_shift_sleep_EEG
                                            )

    return avg_power_near_powerline, avg_power_not_powerline


```

Figure 5.6: power_noise_handler code part2

If the user don't want to check the need to apply the notch filter with the provided function. They can skip the checker via the turnoff the option (notch_freq_essential_checker = False).

5.3 Segmentation

The main part of the function, is once the filterd signal is obtained. First extract the stats like quantiles Q1 (0.25%), Q2 (.5%) and Q3 (0.75) in line-4 of Fig. 5.7; from the overall all signals for each channels separably from the overall signal.

Then segment into epochs as shown in line-7 of Fig. 5.7.

```
    #  Assigning minimal value and get quantiles
    EEG2 = np.array(EEG)
    EEG2[np.abs(EEG2)<1e-5] = np.nan
    q1,q2,q3 = np.nanpercentile(EEG2, (25,50,75), axis=1)

    # Segment into epochs
    EEG_segs = EEG[:,list(map(lambda x:np.arange(x-padding,x+window_size+padding),
                                            start_ids))].transpose(1,0,2)

    if channel_specific_preprocess:
        if  len(ch_names)!=np.shape(EEG_segs)[1]:
            raise Exception("The ch_names should be the length of channels present in
                                            the EEG")


    if verbose:
        logger.info('segmentaion done')
    if GUI_percentile:
        percent_complete(60, 100, bar_width=60, title="Preprocess", print_perc=True)
```

Figure 5.7: segment_EEG part-3

5.4 Assign events status from the prepocess outcome

The Fig. 5.8 shows the annotate the events based on the filtered signal's presence. This portion (Fig. 5.8)marks (from epoch_status_explanation from Fig. 5.2) the events such as,

- ✓ 'NaN in EEG'
- ✓ 'overly high/low amplitude'
- ✓ 'flat signal'

```
    nan2d = np.any(np.isnan(EEG_segs), axis=2)
    nan1d = np.where(np.any(nan2d, axis=1))[0]
    for i in nan1d:
        epoch_status = _channel_specific_preprocess_events_handler(epoch_status, i,
                                        2, nan2d, channel_specific_preprocess,
                                        epoch_status_explanation, ch_names)

    amplitude_large2d = np.any(np.abs(EEG_segs)>amplitude_thres, axis=2)
    amplitude_large1d = np.where(np.any(amplitude_large2d, axis=1))[0]
    for i in amplitude_large1d:
        epoch_status = _channel_specific_preprocess_events_handler(epoch_status,i,
                                                3,amplitude_large2d,
                                                channel_specific_preprocess,
                                            epoch_status_explanation, ch_names)
    # if there is any flat signal with flat_length
    short_segs = EEG_segs.reshape(EEG_segs.shape[0], EEG_segs.shape[1], EEG_segs.shape
                                            [2]//flat_length, flat_length)
    flat2d = np.any(detrend(short_segs, axis=3).std(axis=3)<=std_thres, axis=2)
    flat2d = np.logical_or(flat2d, np.std(EEG_segs,axis=2)<=std_thres2)
    flat1d = np.where(np.any(flat2d, axis=1))[0]
    for i in flat1d:
        epoch_status = _channel_specific_preprocess_events_handler(epoch_status,i,
                                                4,flat2d,

                                                channel_specific_preprocess,
                                            epoch_status_explanation, ch_names)
```

Figure 5.8: segment_EEG part-4

5.4.1 Channel specific preprocess events

To annotate events with channel information or not is processed via the function "_channel_specific--_preprocess_events_handler" (as shown in Fig. 5.9) is used. The fuction (_channel_specific_pre--process_events_handler) return the annotated events with the channel details, based on the option (channel_specific_preprocess). If the channel_specific_preproces option is True, then the channel detected by the preprocessing procedure information is also included in the preprocessed events. Else, the epoch is marked with the detected event without the channel detail.

For an example, if the epoch "m" is Wake stage and Channel "C4" is detected with the higher amplitude through the preproces. Then the if channel_specific_preproces option is True, then the epoch "m" event is annotated with Sleep stage W;overly high/low amplitude channels C4. Else Sleep stage W;overly high/low amplitude channels. When the channel_specific_preproces option is False, we can't say which channel is bad after the preprocessing step is done. The channel information is missed in the preprocessing.

```
1     def _channel_specific_preprocess_events_handler(epoch_status,i,
2         ep_ex_indx, arr_2d, channel_specific_preprocess,
3         epoch_status_explanation, ch_names):
4         # Not channel specific such all the events are annotated
5         # without the channel specific info
6         # for an example flat signal not mention which channel is flat etc.
7         if not channel_specific_preprocess:
8             epoch_status[i] = epoch_status_explanation[ep_ex_indx]
9         else:
10            # Due to channel specific such all the previous  events
11            # and  channel specific preprocess info combined
12            #
13            # just cheeck the normal event annotaion
14            if  epoch_status[i]!=epoch_status_explanation[0]:
15                epoch_status[i] =  epoch_status[i]+'; '+epoch_status_explanation[
                                       ep_ex_indx]+ ' channels '+','.join([
                                       ch_names[x] for x in list(np.where(arr_2d[i
                                       ,:])[0])])
16            else:
17                epoch_status[i] = epoch_status_explanation[ep_ex_indx]+ ' channels '+'
                                       ,'.join([ch_names[x] for x in list(np.where
                                       (arr_2d[i,:])[0])])
18        return epoch_status
```

Figure 5.9: _channel_specific_preprocess_events_handler function

NaN in EEG

Line 4 of Fig. 5.8 annotate the "NaN in EEG" (from epoch_status_explanation from Fig. 5.2) From the detected NANs from line-1 to 2 of Fig. 5.8.

overly high/low amplitude

From line 8-9 of Fig. 5.8. Basicaly checking the amplitude (absolute power) of EEG signal above the amplitude_thres (default in the scripts are 500 or 2000) as artifact.

The normal amplitude is totally dependent on the subject's brain power, however these conditions can be visible in big subsject pool, when the age getting higher the amplitude reduces. Which is presented in the subsection. 1.1.3. There "*age-wise comparison is performed, in the age boundary at 15, and the results are shown in the Figure. 1.2. From this initial outcome from Fig. 1.2 clearly see the when the age get older the amplitude distribution decreases.*"

This choice made bigger impact in the following analysis, since the portion of the EEG useful signal may be considered as bad signal if we choose the bad threshold. The impact of varing this condition threhold (amplitude_thres) is shown in the one of the subject's preprocess outcome Fig. 5.10. This subject has 1,848 epochs, among them when we chose the amplitude_thres as

- ✓ 2000: only $\approx 4\%$ (71 out of 1,848 epochs)
- ✓ 500: $\approx 13\%$ (240 out of 1,848 epochs)

are annotated as artifacts.

```
1    overly high/low amplitude channels C3,O1,O2
242  overly high/low amplitude channels C3
280  overly high/low amplitude channels C3
336  overly high/low amplitude channels F3,C3,O1
486  overly high/low amplitude channels F3,C3
587  overly high/low amplitude channels C3
806  overly high/low amplitude channels F3
```

(a) Snip of first 10% portion of preprocess part amplitude_thres as 2000

```
1    overly high/low amplitude channels C3,C4,O1,O2
3    overly high/low amplitude channels C3
4    overly high/low amplitude channels C3
16   overly high/low amplitude channels F3,F4,C3,C4
17   overly high/low amplitude channels C3
19   overly high/low amplitude channels F4,C3
21   overly high/low amplitude channels C4
22   overly high/low amplitude channels F3,C3,C4
24   overly high/low amplitude channels C4
25   overly high/low amplitude channels F3,C3
26   overly high/low amplitude channels C3
28   overly high/low amplitude channels C3,C4
29   overly high/low amplitude channels C3,C4
33   overly high/low amplitude channels F4
34   overly high/low amplitude channels F4
36   overly high/low amplitude channels C3,C4
37   overly high/low amplitude channels C3
39   overly high/low amplitude channels C3,C4
41   overly high/low amplitude channels F3,C3
42   overly high/low amplitude channels C3
43   overly high/low amplitude channels C3,C4
44   overly high/low amplitude channels C3
46   overly high/low amplitude channels C3
48   overly high/low amplitude channels C3
```

(b) Snip of first 10% portion of preprocess part amplitude_thres as 500

Figure 5.10: Preprocess impact due to the amplitude choice in events output from the package snip shots. For this perticular subject only varing this amplitude condition makes the bigger impact in the findings. Such that changing threhold mark 71, and 240 epochs cross the threhold conditions 2000 and 500 accordingly.

flat signal

Then we annotate the flat signals based on the standard deviation of the signal presented on the given periods of the time domain EEG signal. The choices of the paramters are shown in the Fig. 5.3, from line 9-24 as

$$\begin{aligned} std\_thres &= 0.2 \\ std\_thres2 &= 1. \\ flat\_seconds &= 5 \\ padding &= 0 \\ window\_size &= round(window\_time \times Fs) \\ step\_size &= window\_size \\ flat\_length &= flat\_seconds \times Fs \end{aligned}$$

Cheking the short time segments, via the first split the signal into short segments (short_segs = 5 sec) as shown in line-15 in Fig. 5.8; this is done by flat_length. The $starandarddeviation < 0.2$ for that short segment then the the portion contained epoch is marked as flat signal.

In the next stage check the overall epoch's $standarddeviation < 1$ then the epoch is marked as flat signal epoch. If either one of the condition is less then the epoch is marked as flat signal (for specific channel/ whole based on the _channel_specific_preprocess_events_handler choise)

bad events

```
    # Mark epochs with bad events
    epoch_status = np.array(epoch_status)
    if not (bad_epochs is None):
        indx = np.searchsorted(start_ids, bad_epochs)
        #  only annotate the channels with teh epoch-status not normal
        #  with channel specific information for preprocess step detections
        if len(bad_epochs)>0:
            for i in indx:
                if not channel_specific_preprocess and \
                                epoch_status[i]==epoch_status_explanation[0]:
                    epoch_status[i] = epoch_status_explanation[5]
                else:
                    epoch_status[i] = epoch_status[i]+'; '+epoch_status_explanation[5]


    # normalize signal
    if quantile_normalise_time:

        nch = EEG_segs.shape[1]
        EEG_segs = (EEG_segs - q2.reshape(1,nch,1)) / (q3.reshape(1,nch,1)-q1.reshape(
                                            1,nch,1))
    if verbose:
        logger.info('Preprocess done')
    if GUI_percentile:
        percent_complete(100, 100, bar_width=60, title="Preprocess", print_perc=True)

    # to return the bandpassed notch filtered EEG
    if return_filtered_EEG:
        return EEG_segs, EEG, labels, start_ids, epoch_status, q1,q2,q3,
                                            notch_filter_skipped
    else:
        return EEG_segs, labels, start_ids, epoch_status, q1,q2,q3,
                                            notch_filter_skipped
```

Figure 5.11: segment_EEG part-5

As shown the Fig. 5.11, the obtained bad_epochs and start_ids information is used to find the indexes for epoch location, using that information the final epoch status is marked with bad epoch informations as well.

Chapter 6

events_to_txt

This chapter covers the main portion of the o/p of the package. Once the events are annotated, they must be stored for user's access. The packge provide an option to store as a txt file to visually see the events with the annotated epoch stages. Two main function are provided such as preprocess_events_to_txt and only_sleep_epoches_events_to_txt.

6.1 preprocess_events_to_txt

```
    def preprocess_events_to_txt(prerocess_event_txt_file_name, saving_dir,\
        epoch_status,  preprocess_txt_only_detected_bad=True,comma_sep=False,\
        save_csv=False):
        # this function creates the events to txt funtcion after preprocessing
        # preprocess_txt_only_detected_bad: this only select the epoches effected in
                                            the events annotation
        # this holds the rows of events
        preprocess_events_txt=[]
        for e in range(0,len(epoch_status)):
            if preprocess_txt_only_detected_bad:
                if epoch_status[e]!='normal':
                    preprocess_events_txt.append([e+1, str(epoch_status[e])])
            else:
                preprocess_events_txt.append([e+1, str(epoch_status[e])])
        # assign txt format to save as txt file
        if prerocess_event_txt_file_name.split('.')[-1]!='txt':
            if save_csv and  prerocess_event_txt_file_name.split('.')[-1]!='csv':
                prerocess_event_txt_file_name = prerocess_event_txt_file_name+'.csv'
            else:
                prerocess_event_txt_file_name = prerocess_event_txt_file_name+'.txt'
        # whether txt file in comma seprated format
        if comma_sep:
            col_sep=','
        else:
            col_sep='\t'
        writeFile(saving_dir+prerocess_event_txt_file_name,formatDataTable(
                                            preprocess_events_txt,col_sep=col_sep))
        logger.info("saved in "+saving_dir+prerocess_event_txt_file_name)
```

Figure 6.1: preprocess_events_to_txt

To store the events after the preprocessing. This funtion offers an option to choose only the effected epochs predicted by the preprocessing. With such that only not normal epochs are going to be presented.

6.2 only_sleep_epoches_events_to_txt

To store the events after the final event annotation.

```
def only_sleep_epoches_events_to_txt(prerocess_event_txt_file_name, saving_dir,
                                     epoch_status, comma_sep=False, save_csv=
                                     False):
    # this function creates the events to txt function
    #  return the epoch in index start from 1

    # this holds the rows of events
    preprocess_events_txt=[]
    for e in range(0,len(epoch_status)):
        preprocess_events_txt.append([e+1, str(epoch_status[e])])

    # assign txt format to save as txt file
    if prerocess_event_txt_file_name.split('.')[-1]!='txt':
        if save_csv and  prerocess_event_txt_file_name.split('.')[-1]!='csv':
            prerocess_event_txt_file_name = prerocess_event_txt_file_name+'.csv'
        else:
            prerocess_event_txt_file_name = prerocess_event_txt_file_name+'.txt'

    # whether txt file in comma seprated format
    if comma_sep:
        col_sep=','
    else:
        col_sep='\t'

    writeFile(saving_dir+prerocess_event_txt_file_name,formatDataTable(
                                     preprocess_events_txt,col_sep=col_sep))
    logger.info("saved in "+saving_dir+prerocess_event_txt_file_name)
```

Figure 6.2: only_sleep_epoches_events_to_txt

Chapter 7

cont_segs_of_whole_EEG

This chapter covers the functions related to the preprocessing.cont_segs_of_whole_EEG.py.

Once the preprocess intial steps are done Due to the unmwanted signals of time domain signals not going to be used in the following pipeline. Such that these need to be seperated in to usefull signal blocks (continious chaunks) of the EEG data. These functions are intended to seperate the useful EEG signals (continious segmentation) from whole EEGs. Inorder to visaulise the effect of the preprocessing step.

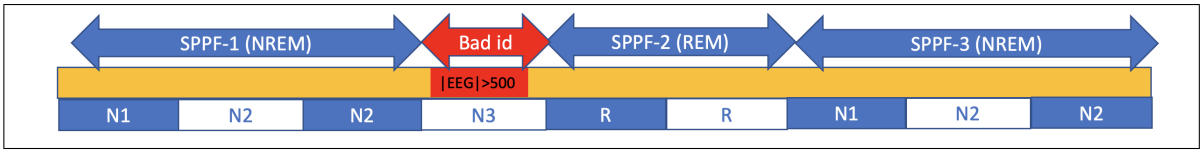


Figure 7.1: Extracted useful signal as continious block introduction due to the preprocessing

As shown in the Fig. 7.1. In the figure, I have used the sleep-pre-processed-fragment (SPPF) as uninterrupted preprocessed EEG-signal belongs to, groups such as,

✓ NREM: Group of N1, N2, N3

✓ REM: Group of R

In the package all the NREM and REM (N1, N2, N3 and R) is considered as one group. Such that the "wake" happend in the middle is splited as shown in the Fig. 7.2.

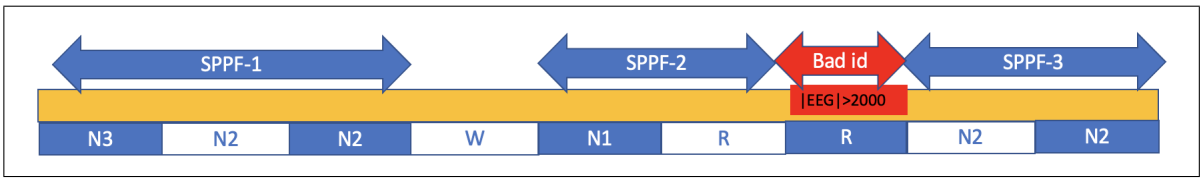


Figure 7.2: Extracted useful signal as continious block introduction due to the preprocessing

In this way the interested (focused sleep stages) continious preprocessed signal blocks are extracted.

```
import os
import logging

import numpy as np

from copy import deepcopy
from sleep_EEG_loose_lead_detect.GUI_interface.percentage_bar_vis  import
                                                    percent_complete

logger = logging.getLogger("cont_EEG_segs")
while logger.handlers:
     logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
logger.setLevel(logging.INFO)
logger.propagate = False
```

Figure 7.3: cont_segs_of_whole_EEG intialisation importing Dependencies

Inorder to directly save the extarcted continious segments (cont_EEG_segments); the functions save_cont_segs shown in Fig. 7.4 is used this directly call the function find_cont_EEG_segments (as shown in Fig. 7.5) while passing the epoch_groups_sample_size as,

$$epoch\_groups\_sample\_size = Fs \times epoch\_length$$

```
def save_cont_segs(in_name_temp, save_spindle_loc, sel_ids, start_time_idx, Fs,
                                                    sel_id_name='good_ids', epoch_length=30,
                    save_cont_seg=True,GUI_percentile=True):
    '''
    this function returns only the continious segments
    for the given epoch_groups_sample_size in secs (default=30sec)
    '''

    epoch_groups_sample_size=Fs*epoch_length
    cont_EEG_segments = find_cont_EEG_segments(sel_ids,start_time_idx,
                                                    epoch_groups_sample_size,GUI_percentile=
                                                    GUI_percentile)
    if save_cont_seg:
        os.chdir('/')
        os.chdir(save_spindle_loc)
        np.save(in_name_temp+'_cont_EEG_seg_info_'+sel_id_name,cont_EEG_segments)
    else:
        return cont_EEG_segments
```

Figure 7.4: save_cont_segs

Since the main find_cont_EEG_segments (as shown in Fig. 7.5) is the main function. The comments shown above each step (code) in Fig. 7.5 shows the purpose. Lets some explonantion, how the overall continuous segments are obtained via the facts.

Inorder to check the contniuous blocks first check the variables used in this function,

1. sel_ids: these are the interested selected ids (indexes); this must be in the assending order, such that is sorted in the line-17.

2. start_pos_sel_id_indx: To check the relative position continious block extraction among the sel_ids.

3. start_chunk_key_idex: As the named in the variable, this is the key to store the continious blocks. This variable itself present the index of starting position of continious block.

4. cont_EEG_segments: This the main variable store the positions of continious segments as dictionary format. cont_EEG_segment' key present the starting position of the continious block. And the cont_EEG_segment' value present the ending position of continious block.

To do this loop through the sel_ids and check the next sel_ids' distance with the current sel_id (start_pos_sel_id_indx) via the function chk_next_continious Fig. 7.6. In simple way

$$start\_time\_idx[sel\_ids[start\_pos\_sel\_id\_indx]] + epoch\_groups\_sample\_size == start\_time\_idx[sel\_ids[start\_pos\_sel\_id\_indx + 1]]$$

Such that if the next starting position is same then the function chk_next_continious return True, such that just increase the start_pos_sel_id_indx as shown in the line 29 of Fig. 7.5. Else assign the continious block as shown in line 34 of Fig. 7.5.

```
def find_cont_EEG_segments(sel_ids, start_time_idx, epoch_groups_sample_size=0,
                                    Fs=256,epoch_length=30,GUI_percentile=True):
    '''
    This function uses start_time_idx to obtain the continious segments to do this
    go-through the good-ids and starindex to find the continious EEG groups

    start_time_idx: starting positions of the sel_ids correspondiong epoches
        Fs= sampling freq
    '''
    if GUI_percentile:
        percent_complete(0, 100, bar_width=60, title="contniuous-grouping",
                    print_perc=True)

    if epoch_groups_sample_size==0:
        epoch_groups_sample_size = Fs*epoch_length

    sel_ids = np.sort(sel_ids)

    start_pos_sel_id_indx =0
    start_chunk_key_idex=0

    cont_EEG_segments={}
    while start_pos_sel_id_indx<len(sel_ids)-1:
        # based on the start_pos_sel_id_indx on the sel_ids,
        #  to decide the continuity
        # check the distance between the next epoch
        if chk_next_continious(sel_ids, start_pos_sel_id_indx,start_time_idx,
                                            epoch_groups_sample_size):
            #  if the next epoch is continious just go ahead and check the next epoch
            start_pos_sel_id_indx=start_pos_sel_id_indx+1
        else:
            # Else just assign the lastly enocunterd sel_ids[start_pos_sel_id_indx]
            # next index as ending of continious epoch
            # since the python last index is not included in the counting
            cont_EEG_segments[sel_ids[start_chunk_key_idex]]= deepcopy(sel_ids[
                                            start_pos_sel_id_indx]+1)
            start_pos_sel_id_indx=start_pos_sel_id_indx+1
            start_chunk_key_idex = deepcopy(start_pos_sel_id_indx)
        # shows the approximate percentage of the continious segmentation done
        # this part not gurantees thos much percentage exist for calculation
        if GUI_percentile:
            percent_complete(start_pos_sel_id_indx, len(sel_ids), bar_width=60, title=
                                            "contniuous-grouping", print_perc=True)
    if GUI_percentile:
        percent_complete(100, 100, bar_width=60, title="contniuous-grouping",
                                            print_perc=True)
    # finalise the border crteria
    if sel_ids[start_chunk_key_idex] not in list(cont_EEG_segments.keys()):
        cont_EEG_segments[sel_ids[start_chunk_key_idex]] = sel_ids[-1]+1
    return cont_EEG_segments
```

Figure 7.5: find_cont_EEG_segments is the main function, return the continious segments cont_EEG_segment in dictionary format. here the dictionary contains (keys and values) the values of the sel_ids. Please check the function_change_rel_idx_of_cont_segs function in Fig. 7.8.

```
def chk_next_continious(sel_ids,start_pos_sel_id_indx,start_time_idx,
                                                epoch_groups_sample_size):
    '''
    This function checks the next epoch's starting position iscontinous  for the
                                                selected_pos_idexes

    '''
    return start_time_idx[sel_ids[start_pos_sel_id_indx]]+epoch_groups_sample_size ==
                                                start_time_idx[sel_ids[
                                                start_pos_sel_id_indx+1]]
```

Figure 7.6: chk_next_continious

7.1 Representation of continious segments

The values presented in the contniuous block retrieval function **without proper understanding of these functions lead to false** outcomes.

1. cont_EEG_segments via the function find_cont_EEG_segments (in Fig. 7.5) presents the sel_ids direct values as key value pair.

2. cont_EEG_segment via the function continious_seg_to_np (in Fig. 7.7) is just the format changed to array.

3. cont_segmnets_rel_index via the function function_change_rel_idx_of_cont_segs (in Fig. 7.8) points the starting and ending postions relative to the sorted sel_ids' idexes.

```
def continious_seg_to_np(cont_EEG_segments, start_keys):
    '''
    This function converts the cont_EEG_segments to numpy format with
                                                start_epoch_index and end_epoch_index
    '''
    cont_EEG_segments_np = np.zeros((len(start_keys),2))
    for s_k_i in range(0,len(start_keys)):
        cont_EEG_segments_np[s_k_i,0]=start_keys[s_k_i]
        cont_EEG_segments_np[s_k_i,1]=cont_EEG_segments[start_keys[s_k_i]]
    return cont_EEG_segments_np.astype(int)
```

Figure 7.7: continious_seg_to_np the function change the cont_EEG_segment (dictionary) to cont_EEG_segments_np array (numpy) format like the keys and correspondiong values are presented in each row of an array.

```
def function_change_rel_idx_of_cont_segs(cont_EEG_segments):
    '''

    cont_EEG_segments : in the given EEG files segments

    '''
    # convert the continious segments to relative index
    cont_segmnets_rel_index =np.zeros((len(cont_EEG_segments),2))
    start_idx=0
    for i in range(0,len(cont_EEG_segments)):
        cont_segmnets_rel_index[i,0]=int(start_idx)
        start_idx = start_idx+cont_EEG_segments[i,1]-cont_EEG_segments[i,0]
        cont_segmnets_rel_index[i,1]=int(start_idx)

    return cont_segmnets_rel_index.astype(int)
```

Figure 7.8: function_change_rel_idx_of_cont_segs uses the cont_EEG_segments_np array (obtained through the function continious_seg_to_np) to convert the starting and ending positions are related to the sel_id's index.

Chapter 8

spindle_fun

☞ The spindle detection and slow-wave detection is done via the YASA, whole credit of obtained spindle and slow-waves information goes to YASA.

Since I am using the YASA package to extract the spindle and slow-wave information for assigning slow-wave (sw_columns_ordered), and splindle (sp_columns_ordered) extracted information. Please refer the YASA documentation.

In the following links,

✓ YASA-package-link

✓ YASA-Github-link.

If these links doesnot work due to updates, please check search the tool in search engine (e.g: google) with the keyword "yasa spindle detection".

I have followed the YASA package documentation while developing this package.

The purpose of the package (sleep_EEG_loose_lead_detect) is detect the biggest artifacts in the large EEG data (big-data). The functions are build for bigger data, with reduce the time-lags created for warnings/ etc. These can be activated by the user, for the spindle/ slow-wave extraction purpose, As shown in the function "spindle_detect_main" part-1 (Fig. 8.2) disable the loggers presented in the YASA and MNE. This will reduce the time-lag created while running through the bigger data. Such that the package (sleep_EEG_loose_lead_detect) stop the warnings/ information/ etc. from the YASA and MNE package.

☞ Since these warnings, etc are **turned-off**, use the package (sleep_EEG_loose_lead_detect) at your own **risk** to retrieve the spindles and slowwaves information.

The package (sleep_EEG_loose_lead_detect) is running through the large (in TBs) data, inorder to avoid the re-computation the package provided intermediate outcomes, like spindles/ slow-waves detected by the YASA.

Since the draft is underconstruction, this chapter details are not completed yet for the time-being.

```
import numpy as np
import os
import logging

from copy import deepcopy
from yasa import sw_detect,spindles_detect

from sleep_EEG_loose_lead_detect.preprocessing.cont_segs_of_whole_EEG import
                                                find_cont_EEG_segments,
                                                continious_seg_to_np
from sleep_EEG_loose_lead_detect.GUI_interface.percentage_bar_vis  import
                                                percent_complete

logger = logging.getLogger("spindle")
while logger.handlers:
     logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
# Set logging level to the logger
# logger.setLevel(logging.DEBUG) # <-- THIS!
logger.setLevel(logging.INFO)
logger.propagate = False

yasa_logger = logging.getLogger("yasa")
# to avoid pring the not detecting any spindles
yasa_logger.disabled = True


sw_columns_ordered =  ['Start', 'End', 'MidCrossing','Duration', 'NegPeak', 'PosPeak',
                                                'ValNegPeak', 'ValPosPeak',
                       'PTP', 'Slope', 'Frequency',  'IdxChannel']

sp_columns_ordered =  ['Start', 'End', 'Duration', 'Peak', 'Amplitude', 'RMS', '
                                                AbsPower',
        'RelPower', 'Frequency', 'Oscillations', 'Symmetry', 'IdxChannel']
```

Figure 8.1: spindle_fun importing dependencies and initialisation

```
1
2  def spindle_detect_main(in_name_temp, sel_ids, start_time_idx, data, sleep_stages,
3          save_spindle_loc, sel_id_name='good_ids', Fs=256, epoch_length=30,
4          window_time=30, s_loc=0, e_loc=1,sw_columns_ordered=sw_columns_ordered,
5          sp_columns_ordered=sp_columns_ordered, pred_slow_waves=True,
6          pred_spindles =True, save_pred_spindle=False,save_pred_slow_waves=False,
7          save_cont_seg=False, verbose=False, GUI_percentile=True, yasa_verbose=False):
8      '''
9      this function first break into continious segments
10     (to make sure to avoid the intereputions not effect)
11     '''
12     if not yasa_verbose:
13         yasa_logger = logging.getLogger("yasa")
14         yasa_logger.disabled = True
15         # to turnoff the MNE logger used by the YASA
16         mne_logger = logging.getLogger("mne")
17         # one selection here used across mne-python
18         mne_logger.disabled = True
19         logger.info("MNE verbose deactivated")
20
21         import warnings
22         warnings.filterwarnings("ignore", category=DeprecationWarning)
23     if verbose:
24         if pred_spindles:
25             logger.info("Running spindle detection  %s" , (in_name_temp))
26         if pred_slow_waves:
27             logger.info("Running slow-wave detection  %s" , (in_name_temp))
28
29
30     window_size = int(round(window_time*Fs))
31
32     # same as save continiuous segement part
33     epoch_groups_sample_size=Fs*epoch_length
34
35     cont_EEG_segments = find_cont_EEG_segments(sel_ids,start_time_idx,
                                                  epoch_groups_sample_size)
36
37     start_keys = list(cont_EEG_segments.keys())
38     cont_EEG_segments_np = continious_seg_to_np(cont_EEG_segments, start_keys)
39
40     logger.info("Running on continious EEG segments of = %i", len(start_keys))
```

Figure 8.2: spindle_detect_main part-1 disable the loggers presented in the YASA and MNE to reduce the time-lag created while running through the bigger data.

The variables used in the function is shown in the Fig. 8.3.

```
sw_unique_all=True
sp_unique_all=True

sw_sat =False
sp_sat=False

sw_group_of_interest_por=[]
sp_group_of_interest_por=[]


sw_eeg_seg_interest_pos = []
sp_eeg_seg_interest_pos = []
sw_co_occured=[]
```

Figure 8.3: spindle_detect_main part-2 variables

```
for sel_chk in range(0,len(start_keys)):
    # retrieve the data for applying spindle detection algorithm
    # select the continious EEG portion (chunk)
    chunk_start_exact_time_pos= start_time_idx[start_keys[sel_chk]]
    if cont_EEG_segments[start_keys[sel_chk]]==len(start_time_idx):
       chunk_end_exact_time_pos=int(start_time_idx[-1]+epoch_groups_sample_size)
    else:
       chunk_end_exact_time_pos=start_time_idx[cont_EEG_segments[start_keys[sel_chk]]]
    sel_EEG_chunk =  data[:,chunk_start_exact_time_pos:chunk_end_exact_time_pos]
    # need to get the hypnogram with the selected segments as it is uncomment next
    # corres_hyno_chunk = sleep_stages[chunk_start_exact_time_pos:chunk_end_...._pos]
    # here apply selected spindle detection algorithm/ slow-wave detction algorithms
    # based on the default par in 0.6.1 YASA
    if pred_spindles:
        sp = spindles_detect(sel_EEG_chunk, Fs,  include=(1, 2, 3), freq_sp=(12, 15),
                freq_broad=(1, 30), duration=(0.5, 2), min_distance=500,
                thresh={"rel_pow": 0.2, "corr": 0.65, "rms": 1.5},
                multi_only=False,    remove_outliers=False,     verbose=False)
        if not isinstance(sp, type(None)):
            sp_sat=True
            sp_group_of_interest_por, sp_eeg_seg_interest_pos, sp_masks, sp_unique_all
                =  sw_or_spindle_objest_postion_info_ret(sp,sp_columns_ordered,
                    sp_group_of_interest_por, sp_eeg_seg_interest_pos, sp_unique_all,
                    sleep_stages, chunk_start_exact_time_pos, start_time_idx,
                    window_size, Fs=Fs, s_loc=s_loc,e_loc=e_loc)

            sp_group_of_interest_por_com = np.vstack(sp_group_of_interest_por)
            sp_eeg_seg_interest_pos_com = np.vstack(sp_eeg_seg_interest_pos)
    # slow wave detection based on the default par in 0.6.1 YASA
    if pred_slow_waves:
        sw = sw_detect(sel_EEG_chunk, Fs, include=(2, 3), freq_sw=(0.3, 1.5),
                    dur_neg=(0.3, 1.5), dur_pos=(0.1, 1), amp_neg=(40, 200),
                    amp_pos=(10, 150), amp_ptp=(75, 350), coupling=False,
                    remove_outliers=False, verbose=False)
        # choose the slow_wave coloumns and spindle coloumns to be appear
        if not isinstance(sw, type(None)):
            sw_sat =True
            sw_group_of_interest_por, sw_eeg_seg_interest_pos, sw_masks, sw_unique_all
            = sw_or_spindle_objest_postion_info_ret(sw,sw_columns_ordered,
                sw_group_of_interest_por, sw_eeg_seg_interest_pos, sw_unique_all,
                sleep_stages, chunk_start_exact_time_pos, start_time_idx, window_size,
                Fs=Fs, s_loc=s_loc,e_loc=e_loc)

            sw_group_of_interest_por_com = np.vstack(sw_group_of_interest_por)
            sw_eeg_seg_interest_pos_com = np.vstack(sw_eeg_seg_interest_pos)
    # include the spindle and SW overlapped portions to find any relevant info
    if (sw_sat and sp_sat):
        try:
            sw_co_occured = sw.find_cooccurring_spindles(sp.summary(), lookaround=1.2)
            if not isinstance(sw_co_occured, type(None)):
                sw_co_occured.append([deepcopy(sw_co_occured),deepcopy(sw)])
        except:
            pass
    if verbose:
        if sel_chk%10==0:
            logger.info("Remaining %i continoius EEG segments  %s" %
                        (len(start_keys)-sel_chk,in_name_temp))
    if GUI_percentile:
        percent_complete((sel_chk/len(start_keys)*100), 100, bar_width=60, title="
                                            Spindle detectiion", print_perc=True)
```

Figure 8.4: spindle_detect_main part-3

```
if verbose:
    logger.info("Done spindle detection  %s" , (in_name_temp))
if GUI_percentile:
    percent_complete(100, 100, bar_width=60,
                     title="Spindle detectiion", print_perc=True)

if save_cont_seg or save_pred_slow_waves or save_pred_spindle:
    os.chdir('/')
    os.chdir(save_spindle_loc)

if save_cont_seg:
    np.save(in_name_temp+'_cont_EEG_seg_info_'+sel_id_name,cont_EEG_segments)


if save_pred_slow_waves and pred_slow_waves and sw_sat:
    np.save(in_name_temp+'_sw_fea_'+sel_id_name,sw_group_of_interest_por_com)
    np.save(in_name_temp+'_sw_sl_st_info_'+sel_id_name,sw_eeg_seg_interest_pos_com.
                                            astype(int))
elif pred_slow_waves and not sw_sat:
    logger.warning('No Slow wave is detected for %s',in_name_temp)
    sw_group_of_interest_por_com=[]
    sw_eeg_seg_interest_pos_com=[]

if save_pred_spindle and pred_spindles and sp_sat:

    np.save(in_name_temp+'_sp_fea_'+sel_id_name,sp_group_of_interest_por_com)
    np.save(in_name_temp+'_sp_sl_st_info_'+sel_id_name,sp_eeg_seg_interest_pos_com.
                                            astype(int))
elif pred_spindles and not sp_sat:
    logger.warning('No spindle detected for %s',in_name_temp)
    sp_group_of_interest_por_com=[]
    sp_eeg_seg_interest_pos_com=[]

if not pred_spindles:
    sp_group_of_interest_por_com=[]
    sp_eeg_seg_interest_pos_com=[]

if not pred_slow_waves:
    sw_group_of_interest_por_com=[]
    sw_eeg_seg_interest_pos_com=[]
#  this will hold all the values of spindles in one dictionary format
sp_sw_dic_format={}
sp_sw_dic_format['sw_sat']=sw_sat
sp_sw_dic_format['sp_sat']=sp_sat
sp_sw_dic_format['sp_eeg_seg_interest_pos_com']=sp_eeg_seg_interest_pos_com
sp_sw_dic_format['sp_group_of_interest_por_com']=sp_group_of_interest_por_com
sp_sw_dic_format['sw_eeg_seg_interest_pos_com']=sw_eeg_seg_interest_pos_com
sp_sw_dic_format['sw_group_of_interest_por_com']=sw_group_of_interest_por_com
# to turn-ON the MNE logger
if not yasa_verbose:
    mne_logger = logging.getLogger("mne")  # one selection here used across mne-python
    mne_logger.disabled = False
return cont_EEG_segments,start_keys,cont_EEG_segments_np, sp_sw_dic_format
```

Figure 8.5: spindle_detect_main part-4 finalising the spindle detection and save, the extracted outcomes

```
def int_por_in_the_chunk(ev_np,chunk_start_exact_time_pos,Fs,s_loc=0,e_loc=2):
    # This function gives the exact event_location in overall EEG group
    # s_loc= Start coloumn index
    # e_loc = End
    ev_np[:,[s_loc,e_loc]] = (np.copy(ev_np[:,[s_loc,e_loc]]) *Fs)+
                                            chunk_start_exact_time_pos
    return ev_np
```

Figure 8.6: int_por_in_the_chunk

```
def find_epoch_position_relevant_to_spindle_or_slow_wave(exact_pos_int,start_time_idx,
                                            window_size):
    # this function finds the segmented epoche"s location relevant to spindle
    eph_split_chk_ind = -1
    while exact_pos_int < start_time_idx[eph_split_chk_ind]:
        eph_split_chk_ind=eph_split_chk_ind-1

    eph_split_chk_ind = len(start_time_idx)+eph_split_chk_ind
    if not exact_pos_int<start_time_idx[eph_split_chk_ind]+window_size:
        raise("some thing wrong in epoch position sel for spindle")

    return eph_split_chk_ind
```

Figure 8.7: find_epoch_position_relevant_to_spindle_or_slow_wave

```
def sw_or_spindle_objest_postion_info_ret(ob,sel_cols,group_of_interest_por,
     eeg_seg_interest_pos, unique_all, sleep_stages, chunk_start_exact_time_pos,
     start_time_idx, window_size, Fs=256,s_loc=0,e_loc=1):
    # This is the main function retrieve the information of epoches
    # incase no-spindle found in the algorithm
    events =  ob.summary()
    ev_np = events[sel_cols].to_numpy()

    ev_np = int_por_in_the_chunk(ev_np,chunk_start_exact_time_pos,
                                 Fs, s_loc=s_loc, e_loc=e_loc)

    masks_all = deepcopy(ob.get_mask())
    group_of_interest_por.append(deepcopy(ev_np))

    #first two indexes point the epoch of the segments last two the hypnos sleep
    eeg_seg_interest_pos_np=np.zeros((np.shape(ev_np)[0],4))
    for ev in range(np.shape(ev_np)[0]):
        # to find the EEG_seg portion's spindle/ sw location
        eeg_seg_interest_pos_np[ev,0] = \
            find_epoch_position_relevant_to_spindle_or_slow_wave(ev_np[ev,s_loc],
            start_time_idx,window_size)
        eeg_seg_interest_pos_np[ev,1] = \
            find_epoch_position_relevant_to_spindle_or_slow_wave(ev_np[ev,e_loc],
            start_time_idx,window_size)

        eeg_seg_interest_pos_np[ev,2]=sleep_stages[int(ev_np[ev,s_loc])]
        eeg_seg_interest_pos_np[ev,3]=sleep_stages[int(ev_np[ev,e_loc])]

        if len(np.unique(eeg_seg_interest_pos_np[ev,[0,1]]))!=1 and \
                len(np.unique(eeg_seg_interest_pos_np[ev,[2,3]]))!=1:
            unique_all=False
    eeg_seg_interest_pos.append(deepcopy(eeg_seg_interest_pos_np))
    return group_of_interest_por, eeg_seg_interest_pos, masks_all, unique_all
```

Figure 8.8: sw_or_spindle_objest_postion_info_ret

Chapter 9

correlation_functions

```python
import logging
import os
import numpy as np
import matplotlib.pyplot as plt

from scipy.stats import spearmanr

from copy import deepcopy

from sleep_EEG_loose_lead_detect.channel_correlation_outlier.\
poolers_Z_standardization_funcs import inter_mediate_mapping_correlation
from sleep_EEG_loose_lead_detect.MT_spectrum.Multitaper_class\
    import taper_eigen_extractor_optim_bandwidth, taper_eigen_extractor
from sleep_EEG_loose_lead_detect.MT_spectrum.Multitaper_class\
    import overlap_window_1sec_fixed_slide_spectrogram_given_freq_res
from sleep_EEG_loose_lead_detect.GUI_interface.percentage_bar_vis\
    import percent_complete

'''
https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html
    on Mon Jan 16 09:00:49 2023
the pearson product-moment correlation coefficients are only calculated by numpy

when we calculate the correlation coefficient via numpy
the rows should be variables
(in EEG here herethe rows should be different EEG-channels)
coloumns are the observations of these variables
'''
logger = logging.getLogger("correlation_funcs")
while logger.handlers:
        logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
# Set logging level to the logger
logger.setLevel(logging.INFO)
logger.propagate = False
```

Figure 9.1: correlation_functions importing dependencies and initialisation

9.1 Channels' correlation

For correlation calculation also we have this choices, such as

- ✓ spearman
- ✓ Pearson

And the calculation domains such as,

- ✓ Time
- ✓ Spectral

In default settings the correlation is calculated in the spectral domain (in the scripts in direct spectrums, not db scale). Spearman correlation between the channels are calculated.

This package is focused on the loose lead channel artifact detection based on the higher artifcat presence will reduce the relation with the channels. Good-segments' (no artifaccts) correlation between the channels going to endup in higher correlation.

Such that, it is going to reflect on the combination of these channels' Multitaper Spectrum's Spearman's rank correlation coefficient on that specific time.

Inorder to avoid the ambiguity in channel combination names. Here onwards the channels' name means the re-referenced channels' name. For an example "F3" means F3-M2 channel and the "F3-F4" correlation means the Spearman's rank correlation coefficient between "F3-M2" and "F4-M1".

Since we have six channels (re-referenced), we are going to end up in $\binom{6}{2} = 15$ correlation combinations. Please check the chapter overall methodology in Chapter. 2 to understand the overall package and pipeline intutively.

9.2 MT_based_correltion_calc_in_continious_segs function initialisation

Since we already explored the overall inputs details about the function in the section "MT-spectral estimation and correlation extraction" of chapter. "Package capabilities" (3).

Such that this portion lets walk through the function and check the purpose of the blocks.

Extract the parameters for MT-spectral estimation

Inorder to estimate spectogram (MT-spectrum) via Multiple tapers, we need to have the parameters such as tapers, eigen, d_f, and N. This can be obtained from the specialised package for MT-estimation, while giving the hypoer parameters. Please check the chapter. "Multitaper_class" (10).

```
1
2  def MT_based_correltion_calc_in_continious_segs(EEG_data_given, sleep_stages,
3      cont_EEG_segments, Fs,ch_names, start_time_idx=[], T=4, f_min_interst=0.5,
4      f_max_interst=32.5, window_time=30, padding=0, sleep_stage_anoot_ext=False,
5      db_scale=False, save_db_scale=False, pearson=False,spearman=True,
6      interested_channels_index=[], root_data=True, b_val=0.0001,
7      inter_mediate_transform=True, optim_bandwidth=True, save_MT_spectrum = False,
8      return_flatten=True, verbose=False, GUI_percentile=True):
9
10     if  verbose:
11         logger.info("intialising the MT-extraction")
12
13     if optim_bandwidth:
14         tapers, eigen, d_f, N = taper_eigen_extractor_optim_bandwidth(4,Fs,verbose=
                                           verbose)
15     else:
16         tapers, eigen, d_f, N = taper_eigen_extractor(4,Fs,verbose=verbose)
17
18     if  verbose:
19         logger.info("MT spectrums' applying tapers calculated ")
20     # since we are using the
21     #       overlap_window_1sec_fixed_slide_spectrogram_given_freq_res function
22     # that is fixed for 1 sec sliding window
23     sliding_size=1
24
25     N=int(T*Fs)#window size
26     Fs=int(Fs)
27
28     d_f=1/T
29     # The correlation is calulated based on th egiven range of the frequency
30     f_min_index = int(f_min_interst/d_f)
31     f_max_index = int(f_max_interst/d_f)
32
33     window_size = int(round(window_time*Fs))
34
35     if root_data:
36         EEG_segs = EEG_data_given[:,list(map(lambda x:np.arange(x-padding,x+
                                           window_size+padding), start_time_idx))]
37     else:
38         EEG_segs = EEG_data_given[:,interested_channels_index,:].transpose(1,0,2)
39
40     if  verbose:
41         logger.info("EEG data prepared for MT-extraction, correlation calculation
                                           intiated")
42
43     corr_coeff_time=[]
44
45     start_point =0
46     end_point=0
47     sleep_stages_annot = []
48
49     if save_MT_spectrum:
50         spectrogram_col_p_all=[]
51     #  calculate teh eprcentile for GUI interface
52     # to ensure the prediction satisfies only with the single continious segment
53     if GUI_percentile:
54         if np.size(cont_EEG_segments,axis=0)==1:
55             size_for_per=1
56         else:
57             size_for_per = np.size(cont_EEG_segments,axis=0)-1
```

Figure 9.2: MT_based_correltion_calc_in_continious_segs part-1

Using the obtained parameters to MT-spectral estimation

Once the parameters (tapers, eigen, d_f, and N) are obtained, those details are enough to extract the MT-spectrums while move alogn the windows. This is done on the for loop in the Fig. 9.4. Here the each continious segments' MT-spectrums are obtained via the main for loop (begin in line one of Fig. 9.4).

Here instead of manually calculate the size of the selected continious segment's size (due to the segment size varies all the time). First using only the first channel's selected continious segment is used to findout the corrsponding continious segment's spectogram size.

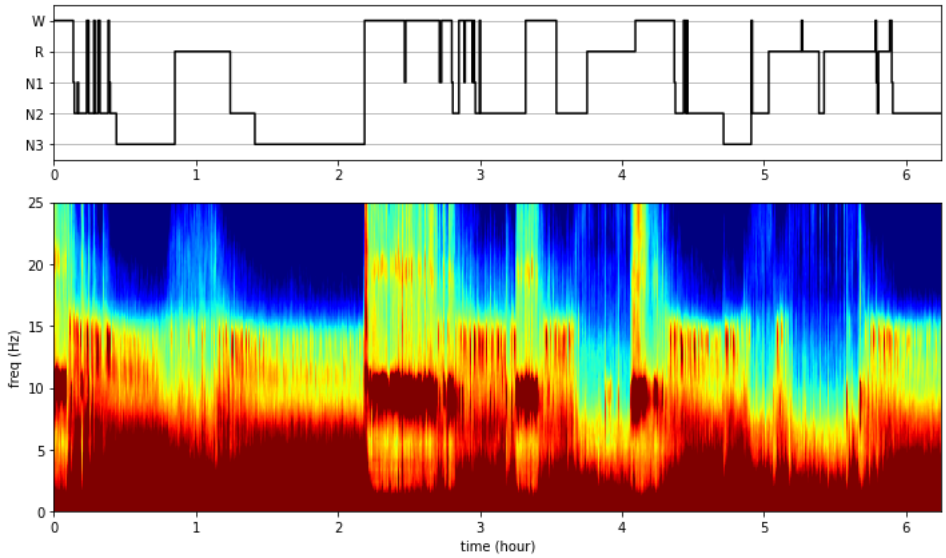
Then assign the spectrogram in the correponding channel placeholder. In this function, I have using only one function "overlap_window_1sec_fixed_slide_spectrogram_given_freq_res" (as shown in Fig. 10.4) to obtain the spectrograms as shown in line 10 and 19 (of the for loop of channels) as shown in Fig. 9.4. This fuction sliding one-sec along the given time domain signal and obtain the spectrogram for it, please check the section. "One-sec fixed sliding MT-spectogram" as shown in Fig. 10.3.

Inshort sliding a 1 sec of with T (4sec default) window to obatin the MT-spectrums on the each continious segments for each channels.

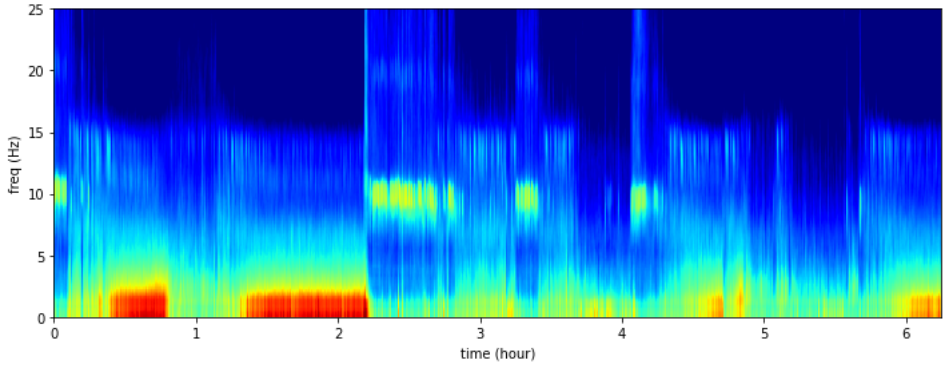
Choice of representation

Once the MT-spectrums are obtained, we have the choice of convert them into dB scale or other scales. The scale makes the way of representation of the obtained spectrum, in another way we can see the normalisation also a kind of scaling. The scaling may lead to loose information, or kind of emphasizing the one portion of the signal.

☞ A single scaling (representation) cannot capture all information. A portion of MT-spectral-focused-report's representation effect on the same spectrum is shown in the Fig. 9.3



(a) Normalisation minimum is selected as Vmin quatile 0.1 value, and Vmax as quantile 0.8



(b) Normalisation minimum is selected as Vmin quatile 0.25 value, and Vmax as quantile 0.999

Figure 9.3: To show the impact of scaling via the visual representation on same normalisation method with different choices of value for noramalisation performed on the single spectogram after the dB scale transformation.

As we can make the normalisation choices as arbitary like such as,

✓ -12db to +12db

✓ -15db to +15db

✓ parameter based normalisation (choices of different quantile for minimum and maximum)

A a parameter based normalisation how the choices make bigger diffrence in outcome is shown in the example figures (Fig. 9.3).

☞ Improper representation makes bigger impact in the visual. Like wise for computer vision also get/ miss the information.

The effect of choices makes the difference range of statical values like variance (standard deviation/ σ) as shown in the section. “Compare the standard distribution choices.” (11.1).

```
for co in range(0,np.size(cont_EEG_segments,axis=0)):
    sel_chk=EEG_segs[:,cont_EEG_segments[co][0]:cont_EEG_segments[co][1],:]
    ex_b = np.squeeze(np.concatenate(np.hsplit(sel_chk, np.size(sel_chk, axis=1)),
                                                                    axis=2))
    # first calculate the MT-spectrum for six channels
    # then check the correlation between them
    channel=0
    c=ex_b[channel,:]#.flatten()
    spectrogram_col_g, t, xf = \
        overlap_window_1sec_fixed_slide_spectrogram_given_freq_res(N,c,tapers,eigen,
                                            d_f,Fs)

    spectrogram_col_t=spectrogram_col_g[f_min_index:f_max_index,:]
    spectrogram_col_p = np.zeros((len(ch_names),f_max_index-f_min_index,np.size(
                                            spectrogram_col_t,axis=1)))
    spectrogram_col_p[channel,:,:]=deepcopy(spectrogram_col_t)

    for channel in range(1,len(ch_names)):
        c=ex_b[channel,:]#.flatten()
        spectrogram_col_g, t, xf = \
            overlap_window_1sec_fixed_slide_spectrogram_given_freq_res(N,c,tapers,
                                            eigen,d_f,Fs)
        spectrogram_col_p[channel,:,:]=deepcopy(spectrogram_col_g[f_min_index:
                                            f_max_index,:])

    if save_MT_spectrum and (not save_db_scale):
        spectrogram_col_p_all.append(deepcopy(spectrogram_col_p))
    # convert the obtained MT-spectrum to log-scale to calculate the correlation
    if db_scale:
        spectrogram_col_p = 10*np.log10(spectrogram_col_p)

        if save_MT_spectrum and save_db_scale:
            spectrogram_col_p_all.append(deepcopy(spectrogram_col_p))
    else:
            if save_MT_spectrum and save_db_scale:
                spectrogram_col_p_all.append(deepcopy(10*np.log10(spectrogram_col_p)))
    # then using the pre-calculated MT-spectrum values to calcultate the
    #  pearson correlation coefficient
    corr_coeff_time_t=np.zeros((len(list(range(0,np.size(ex_b,axis=1)+1-N,\
                sliding_size*Fs))),np.size(ex_b,axis=0),np.size(ex_b,axis=0)))
    # https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html
    # when each row represent the variable for spearman correlation
    for cor in range(0,np.size(spectrogram_col_p,axis=2)):
        if pearson:
            corr_coeff_time_t[cor,:,:]=np.corrcoef(spectrogram_col_p[:,:,cor])
        elif spearman:
            corr_coeff_time_t[cor,:,:],_=spearmanr(spectrogram_col_p[:,:,cor],axis=1)

    corr_coeff_time.append(deepcopy(corr_coeff_time_t))
    # relevant to to sleep-stage annotation
    if sleep_stage_anoot_ext:
        end_point+=(cont_EEG_segments[co][1]-cont_EEG_segments[co][0])

        sleep_stages_annot_t=[]
        for k in range(0,(end_point-start_point)-1):
            sleep_stages_annot_t.append(np.ones((window_time-(T-sliding_size))) *
                                            sleep_stages[start_point+k])
            # assign the next sleep-stage
            sleep_stages_annot_t.append(np.ones(((T-sliding_size))) *sleep_stages[
                                            start_point+k])

        sleep_stages_annot_t.append(np.ones((window_time-(T-sliding_size))) *
                                            sleep_stages[end_point-1])
        sleep_stages_annot.append(deepcopy(np.concatenate(\
                                sleep_stages_annot_t,axis=0)))
        start_point+=(cont_EEG_segments[co][1]-cont_EEG_segments[co][0])
```

Figure 9.4: MT_based_correltion_calc_in_continious_segs part-2

```
if  verbose:
    logger.info("EEG data prepared for MT-extraction, correlation calculation done")

if return_flatten:
    correlation_flatten = np.concatenate(corr_coeff_time,axis=0)
    if inter_mediate_transform:
        correlation_flatten = inter_mediate_mapping_correlation(correlation_flatten,
                                                b_val=b_val)
else:
    if inter_mediate_transform:
        raise Exception("inter_mediate_mapping_correlation is built for flatten \
                        please do it explicitly make inter_mediate_transform=False")


if save_MT_spectrum:
    if sleep_stage_anoot_ext:
        if return_flatten:
            sleep_stages_annot_flatten = np.concatenate(sleep_stages_annot,axis=0)
            return correlation_flatten, sleep_stages_annot_flatten,
                                                spectrogram_col_p_all
        else:
            return corr_coeff_time, sleep_stages_annot, spectrogram_col_p_all

    else:
        if return_flatten:
            return correlation_flatten, spectrogram_col_p_all
        else:
            return corr_coeff_time, spectrogram_col_p_all
else:
    if sleep_stage_anoot_ext:
        if return_flatten:
            sleep_stages_annot_flatten = np.concatenate(sleep_stages_annot,axis=0)
            return correlation_flatten, sleep_stages_annot_flatten
        else:
            return corr_coeff_time, sleep_stages_annot
    else:
        if return_flatten:
            return correlation_flatten
        else:
            return corr_coeff_time
```

Figure 9.5: MT_based_correltion_calc_in_continious_segs part-3

Assign sleep-stages flatten for extracted MT-spectrum and correlation

Since we are extracting the MT-spectrum and correlation from the flatten continious segments. Inorder to find the relative sleep-stages without any issues the function return the correponding flatten sleep-stages as well.

Incase if we want to extract the sleep-stages only for the MT-spectrum and correlation , the function "sleep_annot_retriever_sep_MT" as shown in the Fig. 9.6 can be directly used.

☞ "sleep_annot_retriever_sep_MT"function only output the fixed sliding of one second. Such that 4 sec window has 3 sec overlap.

```
def sleep_annot_retriever_sep_MT(cont_EEG_segments,sleep_stages, T=4,
                                                    window_time=30):

    '''
    This function assign the sleep-stage annotation for MT

    since we are using the
    overlap_window_1sec_fixed_slide_spectrogram_given_freq_res function
    that is fixed for 1 sec sliding window
    '''
    sliding_size=1
    start_point =0
    end_point=0
    sleep_stages_annot = []
    for co in range(0,np.size(cont_EEG_segments,axis=0)):
        # relevant to sleep-stage annotation
        end_point+=(cont_EEG_segments[co][1]-cont_EEG_segments[co][0])

        sleep_stages_annot_t=[]
        for k in range(0,(end_point-start_point)-1):
            sleep_stages_annot_t.append(np.ones((window_time-(T-sliding_size))) *
                                        sleep_stages[start_point+k])
            # assign unknown
            sleep_stages_annot_t.append(np.ones(((T-sliding_size))) *sleep_stages[
                                        start_point+k])

        sleep_stages_annot_t.append(np.ones((window_time-(T-sliding_size))) *
                                    sleep_stages[end_point-1])
        sleep_stages_annot.append(deepcopy(np.concatenate(sleep_stages_annot_t,axis=0)
                                    ))
        start_point+=(cont_EEG_segments[co][1]-cont_EEG_segments[co][0])

    sleep_stages_annot_flatten = np.concatenate(sleep_stages_annot,axis=0)

    return sleep_stages_annot_flatten
```

Figure 9.6: Flatten sleep-stage retrival for continious segments MT-spectrums and their correponding correlation. This fucntion return the slidning is fixed as one second.

9.3 Correlation intermediate transformation/ representation

One of the kind of intermediate mapping on correlation used in the function ca be found in "inter_mediate_mapping_correlation" fucntion in chapter. 12.

9.4 Time domain correlation

The time based correlation is directly obtain between the samples. Inorder to compare the MT-spectral correlation vs the time domain correlation, the following functions are designed to obtain the correlation between the samples of 4sec segements with sliding of one second.

```
def time_correlation_coefficient_retriever_for_cont_seg_main(EEG_data_given,
        sleep_stages, cont_EEG_segments,  Fs, start_time_idx=[], padding=0,
        window_time=30,  T=4, sliding_size=1, interested_channels_index=[],
        root_data=True, leaving_part_com=False, leaving_indexes_count=3, b_val=0.0001
        ,inter_mediate_transform=True, return_flatten=True,verbose=False):

    '''
    root_data: Uses the main six channels

    the last index of the continius segmentation is not part of the
    continious segmenation use regular python index directly

    Inorder to compare MT and time correlation in same plane
        leaving_part_com=This decide the last three indexes are going to be
        leaving_indexes_count=3 how many indexes left in the continious segment
    '''
    window_size = int(round(window_time*Fs))
    if verbose:
        logger.info("EEG_data_given shape: {}".format(' x '.join(map(str, np.shape(
                                            EEG_data_given)))))
    if root_data:
        logger.warning('Initiated on Root EEG with no-referenced chanels')
        if len(start_time_idx)==0:
            raise ValueError("Please assign start_time_idx")
        EEG_segs = EEG_data_given[:,list(map(lambda x:np.arange(x-padding,x+
                                            window_size+padding), start_time_idx))]
    else:
        logger.warning('Initiated on processed channels with referenced chanels')

        EEG_segs = EEG_data_given[:,interested_channels_index,:].transpose(1,0,2)
    if leaving_part_com:
        logger.warning('Purposely leaving %i sec in each continiou segments for
                                            comparison'%leaving_indexes_count)

    if verbose:
        logger.info('Time window is with %f sec'%T)
    corr_coeff_time=[]

    start_point =0
    end_point=0
    sleep_stages_annot = []
    if verbose:
        logger.info("EEG_segs shape: {}".format(' x '.join(map(str, np.shape(EEG_segs)
                                            ))))
        logger.info("cont_EEG_segments shape: {}".format(' x '.join(map(str, np.shape(
                                            cont_EEG_segments)))))
```

Figure 9.7: time_correlation_coefficient_retriever_for_cont_seg_main part-1

```
for i in range(0,np.size(cont_EEG_segments,axis=0)):

    sel_chk=EEG_segs[:,cont_EEG_segments[i][0]:cont_EEG_segments[i][1],:]
    end_point+=(cont_EEG_segments[i][1]-cont_EEG_segments[i][0])
    logger.debug("start point: %i  end point: %i", start_point, end_point)

    sleep_stages_annot_t=[]
    for k in range(0,(end_point-start_point)-1):
        sleep_stages_annot_t.append(np.ones((window_time-(T-sliding_size))) *
                                    sleep_stages[start_point+k])
        if sleep_stages[start_point+k]== sleep_stages[start_point+k+1]:
            sleep_stages_annot_t.append(np.ones(((T-sliding_size))) *sleep_stages[
                                    start_point+k])#assign unknown
        else:
            sleep_stages_annot_t.append(np.ones(((T-sliding_size))) *((sleep_stages[
                                    start_point+k]+sleep_stages[start_point+k+1
                                    ])/2))#assign unknown
        logger.debug("start_point+k:' %i  ",start_point+k)
    logger.debug("end_point-1:' %i  ",end_point-1)

    if leaving_part_com:
        sleep_stages_annot_t.append(np.ones((window_time-leaving_indexes_count)) *
                                    sleep_stages[end_point-1])
    else:
        sleep_stages_annot_t.append(np.ones((window_time-(T-sliding_size))) *
                                    sleep_stages[end_point-1])

    # then hstack the continious segmentations
    ex_b = np.squeeze(np.concatenate(np.hsplit(sel_chk, np.size(sel_chk, axis=1)),axis
                                    =2))
    corr_coeff_time_t = time_correlation_coefficient_retriever_for_cont_seg(ex_b,Fs,T=
                                    T,sliding_size=sliding_size)
    if leaving_part_com:
        corr_coeff_time.append(deepcopy(corr_coeff_time_t[0:(-leaving_indexes_count),:
                                    ,:]))
    else:
        corr_coeff_time.append(deepcopy(corr_coeff_time_t))

    sleep_stages_annot.append(deepcopy(np.concatenate(sleep_stages_annot_t,axis=0)))

    start_point+=(cont_EEG_segments[i][1]-cont_EEG_segments[i][0])

if return_flatten:
    correlation_flatten = np.concatenate(corr_coeff_time,axis=0)
    if inter_mediate_transform:
        correlation_flatten = inter_mediate_mapping_correlation(correlation_flatten,
                                    b_val=b_val)

    sleep_stages_annot_flatten = np.concatenate(sleep_stages_annot,axis=0)
    return correlation_flatten, sleep_stages_annot_flatten
else:
    return corr_coeff_time, sleep_stages_annot
```

Figure 9.8: time_correlation_coefficient_retriever_for_cont_seg_main part-2

The main function "time_correlation_coefficient_retriever_for_cont_seg" used to calculate the correlation is shown in Fig. 9.9.

```
def time_correlation_coefficient_retriever_for_cont_seg(ex_b,Fs,T=4,sliding_size=1,
                                        pearson=False,spearman=True):
    """
    ex_b = given signal to calculate the correlation, shape should be
        channels  x flatten axis

    Fs= sampling frquencncy

    T=4; if we take the 4 sec window for calculating the correlation coefficient
    sliding_size=1; 1sec sliding the window is sliding with 1 sec, means 4 sec window
    going to have 3 sec overlap

    number of combinations
    if we consider all 6 channels for check correlation calcultaion end up in 15
    6C2 = 6!/(4!x2!)=15
    """
    # window size
    N=int(T*Fs)
    Fs=int(Fs)
    # this is sliding cretion with the sampling size (Fs)
    # ----------------------------------------------------------------------------
    corr_coeff_time_t=np.zeros((len(list(range(0,np.size(ex_b,axis=1)+1-N,sliding_size
                                        *Fs))),np.size(ex_b,axis=0),np.size(ex_b,
                                        axis=0)))
    a=0
    for j in range(0,np.size(ex_b,axis=1)+1-N,sliding_size*Fs):
        if pearson:
            corr_coeff_time_t[a,:,:]=np.corrcoef(ex_b[:,j:j+N])
        elif spearman:
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html
# when each row represnet the variable
            corr_coeff_time_t[a,:,:],_= spearmanr(ex_b[:,j:j+N],axis=1)

        a=a+1
    return corr_coeff_time_t
```

Figure 9.9: Main function used for time segments' correlation

Chapter 10

Multitaper_class

Our lab adapted MT-spectral estimation due to the EEG-sleep-data spectral estimation can be found in Harvard's publication and tutorial Multitaper Spectral Analysis for Sleep EEG

The package for MT-spectral estimation is obtained via the spectral estimation focused library "spectrum", Please check the attached links,

- ✓ spectrum-main-page
- ✓ spectrum-Github

The following portion of the introduction and parameter slection are dierctly copied from the MT-spectral-focused.

As EEG-data is non-stationary process, using the spectral estimation quetionable. Such that the study [2] detailed discription of how Mutitper-spectrum estimate suit for EEG-sleep data analysis in detail. Further as per the study [3] EEG-sleep data can be considered stationary for 4sec (T=4sec).

10.1 Background and parameter selection

Since the frequency domain is well known in sleep stage predictions [1, 6, 7]. Then the question arises how we extract the frequency domain information; the study [3] pointed out the main advantage of using Multi taper over the spectrum estimation techniques, and the study [2] shows the usage in sleep EEG analysis.

Here different papers use different scales, and the way these parameters calculated depends on the scale we have chosen, parameters choices for Multiptaper, according to the annotations of [8] in their annotation scales

- ✓ W: bandwidth in normalised scale ($1 < 1/2$)
- ✓ N (int): desired window length (number of samples)

For the sampling frequency 250Hz, according to the study [3],

"Since during sleep transient events occur on a time scale of a few seconds, we choose $T = 4s$. We also choose a spectral resolution of 2 Hz, since the frequency bands of interest during sleep are typically

wider than 2 Hz. Finally, we choose a sliding length of $\tau = 1s$. The spectrogram representation of the EEG data was computed through the following steps"

according to the annotations of [3], they simplified for the experts in other areas to utilise the MT analysis. Such that according to their parameter selection

- ✓ T: Duration in Seconds (s)
- ✓ N: size of the data in seconds at which the data are thought to be stationary.
- ✓ ΔR Spectral resolution Bandwidth (in Hz) of the main lobe in the spectral estimate, which controls the minimum distance between peaks that we can resolve
- ✓ TW: time half bandwidth product $N\Delta R/2$

L (int): the number of tapers; first L Slepian sequences. Which is calculated based on the equation $\lfloor 2TW \rfloor - 1$ This calculation is based on the [2]. Such that we are end up in seven tapers.

For more details about the MT-extraction and normalisation please check the MT-spectral-focused.

```
import matplotlib.pyplot as plt
import logging
import os
import pickle

from spectrum import pmtm,dpss
import numpy as np
from scipy.fft import fft
from copy import deepcopy
'''
to include the depreciated function to satify the spectral estimation
'''
from scipy.signal._signaltools import _centered
import  scipy.signal.signaltools
scipy.signal.signaltools._centered = _centered

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger("MT_spectrum")
while logger.handlers:
    logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
logger.setLevel(logging.INFO)
logger.propagate = False
```

Figure 10.1: Multitaper_class importing dependencies and initialisation

10.2 Hyper parameters for MT-spectral estimation

Hyper parameters selection for taper extractor impact on the estimated spectrum thus these hyper parameter values should be chosen based on the sampling rate (Fs) and spectral estimation bandwidth and variance. Please check the paper [9] "Optimal Bandwidth for Multitaper Spectrum Estimation"main-link or main-pdf-link before you choose your default values, since this may cause unintended outputs in the spectrum.

☞ The optimal bandwidth choice with the assumption of Lorentzian may provide some estimation. As shown in the Fig. 10.3.

```
def taper_eigen_extractor(T=4,Fs=256, f_max_interst=0, bw=2,verbose=False):
    '''
    T: in seconds to choose the window

    N (int) desired window length
    L (int) returns the first L Slepian sequences.

    The default values of this taper in chosen from somewhat near value based on
                                        the window size (4 x 256 =1024 ~ 1000)
    Such a way the Time half bandwth is near to 15

    And this 4 sec window with 2Hz bandwidth is chosen based on the paper
        EEG also show the sleep data characterstics as mentioned in the paper

    A Review of Multitaper Spectral Analysis
        Behtash Babadi, Member, IEEE, and Emery N. Brown, Fellow, IEEE

    '''

    # ------------------------------------------------------------------------
    # when the f_max interest is not chosen this is automatically the maximum
                                        vlaue of Niquist samopling frequency
    # ------------------------------------------------------------------------

    if f_max_interst==0:
        f_max_interst=int(Fs/2)

    # ------------------------------------------------------------------------
    # d_f: frequency resolution
    # ------------------------------------------------------------------------
    d_f = 1/T

    # ------------------------------------------------------------------------
    # window length used to calculate the spectrum
    # ------------------------------------------------------------------------
    N = int(Fs*T)
    # ------------------------------------------------------------------------
    # The multitapered method
    # ------------------------------------------------------------------------

    TW= bw*T/2
    L= int(2*TW-1)
    [tapers, eigen] = dpss(N, TW, L)
    if verbose:
        logger.info("number of samples: %i",N)
        logger.info('TW calculataed based on the Time half bandwidth: %f',TW)
        logger.info('number of tapers going to be %i',  L)

    return tapers, eigen, d_f, N

```

Figure 10.2: taper_eigen_extractor the default parameters are chosen for sampling rate = 256 based on the paper [3].

Note: The number of tapers with the default of function is different among the functions, such that,

✓ taper_eigen_extractor shown in Fig. 10.2: seven tapers, inorder to optain the bandwidth as

2Hz.

✓ taper_eigen_extractor_optim_bandwidth shown in Fig. 10.3: 29 tapers, endup in rough optimal bandwidth as $\approx 7.7Hz$ for Lorentzian spectrum.

```
1
2  def taper_eigen_extractor_optim_bandwidth(T,Fs, f_max_interst=0,TW=4,TW_opt_default=
                                                 True,verbose=False):
3      '''
4      The default values of this taper in chosen from somewhat near value based on the
                                                 window size (4 x 256 =1024 ~ 1000)
5      Such a way the Time half bandwth is near to 15
6
7      And this 4 sec window with 2Hz badwidth is chosen based on the paper
8          EEG also show the sleep data characterstics as mentioned in the paper
9
10     This function kind of mimic the basic assumption of
11     Lorentzian of spectral property while kind of placing the hyperparter rough choice
                                                 , not guranteed their output
12     '''
13     # when the f_max interest is not chosen this is automatically the maximum value of
                                                  Niquist sampling frequency
14     if f_max_interst==0:
15         f_max_interst=int(Fs/2)
16     #    d_f: frequency resolution
17     d_f = 1/T
18     # window length used to calculate the spectrum
19     N = int(Fs*T)
20
21     if TW_opt_default:
22         logging.warning("these default value not gurantee optimal solution just a
                                                 rough calculation to find the near optimal
                                                 spectral estimation")
23         # following values are roughly calculated based on the Fig.2 of the paper
24         if 100<=N<=500:
25             TW= N/50
26         elif 500<N<=1500:
27             TW= ((7*N)/500)+1
28         elif 1500<N:
29             logging.warning("Not derived from the paper; just using the rough
                                                 calculation of in range 500-1500 assume NW
                                                 is lie on that line")
30             TW= ((7*N)/500)+1
31         else:
32             raise("The N should be greater than or equal to 100")
33     # The multitapered method
34     # TW= bw*T/2
35     L= int(2*TW-1)
36     [tapers, eigen] = dpss(N, TW, L)
37     if verbose:
38         logger.info("number of samples: %i",N)
39         logger.info('Time half bandwidth: %f',TW)
40         logger.info('Band width calculated based on the Time half bandwidth: %f',TW*2/
                                                 T)
41         logger.info('Number of tapers going to be %i',  L)
42     return tapers, eigen, d_f, N
```

Figure 10.3: Since the value selection for taper extractor impact on the estimated spectrum thus these hyper parameter values should be chosen based on the sampling rate (Fs) and spectral estimation bandwidth and variance. Please check the paper [9] "Optimal Bandwidth for Multitaper Spectrum Estimation"main-link or main-pdf-link before you choose your default values, since this may cause unintended outputs in the spectrum.

10.3 One-sec fixed sliding MT-spectogram

```
def overlap_window_1sec_fixed_slide_spectrogram_given_freq_res(N,c,tapers,eigen,d_f,Fs
    , extracted_spectrums_of_tapers=False, method='unity', normalisation_full=True):
    '''
    To create the overlapping window
        d_f: spectral resolution in (Hz)
        c: raw time domian signal need to extract the frequency features
    method= 'eigen'   or 'unity'

    extracted_spectrums_of_tapers: If we want the spectrums as it is from the tapers
    so we can use deep learning to weight each spectrums and frequency accordingly

    xf: The spectral frequency bins corresponding extracted spectrogram.
    spectrogram_col: Extracted estimated MT-spectrogram.
    '''
    xf = np.linspace(0.0, int((Fs/2)/d_f), int((N//2)/d_f))
    spectrogram_col=np.zeros((N//2,int((len(c)-N)/Fs)+1))
    if extracted_spectrums_of_tapers:
        spectrogram_col=np.zeros((int((len(c)-N)/Fs),len(eigen),N//2))

    dt=1/Fs
    a=0
    t=[]
    for i in range(0,len(c)+1-N,Fs):
        y=c[i:i+N]

        Sk_complex, weights, eigenvalues=pmtm(y, e=eigen, v=tapers, NFFT=N,
                                              show=False,method=method)
        Sk = abs(Sk_complex)**2
        if extracted_spectrums_of_tapers:
            if normalisation_full:
                spectrogram_col[a,:,:] = deepcopy(Sk[:,0:N//2])* dt
            else:
                spectrogram_col[a,:,:] = deepcopy(Sk[:,0:N//2])

        else:
            # adapted from the library
            #  https://pyspectrum.readthedocs.io/en/latest/_modules/spectrum/mtm.html
            # on Feb-10-2022 at 15.22p.m
            if method == "adapt":
                Sk = Sk.transpose()
                Sk = np.mean(Sk * weights, axis=1)
            else:
                Sk = np.mean(Sk * weights, axis=0)

            newpsd  = Sk[0:N//2] * 2
            if normalisation_full:
                spectrogram_col[:,a] = deepcopy(newpsd)*dt
            else:
                spectrogram_col[:,a] = deepcopy(newpsd)
        a=a+1
        t.append(a)
    return spectrogram_col, t, xf
```

Figure 10.4: Sliding one sec through the given signal with the given MT-spectral estimation parameters to obtain the interested frequency portion's MT-spectogram.

The function "overlap_window_1sec_fixed_slide_spectrogram_given_freq_res" shown in the Fig. 10.4. Here we feed a time domain signal and extract the estimated MT-spectrum with the provided

parameters while sliding one sec at each time (Fs samples) with the window length of (N).

10.4 Extract MT-spectogram on given time domain signal

The following function directly extract the spectrogram for the given time domain signal.

```
1
2  def build_for_comp_purpose_spectrogram_given_freq_res(N,c,tapers,eigen,d_f,Fs,
                                                  extracted_spectrums_of_tapers=False,
                                                  f_maximum_inters=30,method='adapt'):
3      '''
4      To create the overlapping window
5          d_f: spectral resolution in (Hz)
6          c: raw time domain signal need to extract the frequency features
7
8      extracted_spectrums_of_tapers: If we want the spectrums as it is from the tapers
                                                  so we can use deep learning to weight each
                                                  spectrums and frequency accordingly
9      '''
10     xf = np.linspace(0.0, int((Fs/2)/d_f), int((N//2)/d_f))
11     spectrogram_col=np.zeros((N//2,int((len(c)-N)/Fs)))
12
13     dt=1/Fs
14     a=0
15     y=c
16     Sk_complex, weights, eigenvalues=pmtm(y, e=eigen, v=tapers, NFFT=N, show=False,
                                                  method=method)
17     Sk = abs(Sk_complex)**2
18     if extracted_spectrums_of_tapers:
19         spectrogram_col[a,:,:] = deepcopy(Sk[:,0:N//2])* dt
20     else:
21         # adapted from the library
22         #  https://pyspectrum.readthedocs.io/en/latest/_modules/spectrum/mtm.html on
23         #  Feb-10-2022 at 15.22p.m
24         if method == "adapt":
25             Sk = Sk.transpose()
26             Sk = np.mean(Sk * weights, axis=1)
27         else:
28             Sk = np.mean(Sk * weights, axis=0)
29
30         newpsd  = (Sk[0:N//2] * 2)*dt
31
32     return newpsd, xf
```

Figure 10.5: Directly using the spectral estimation on given time domain signal.

Chapter 11

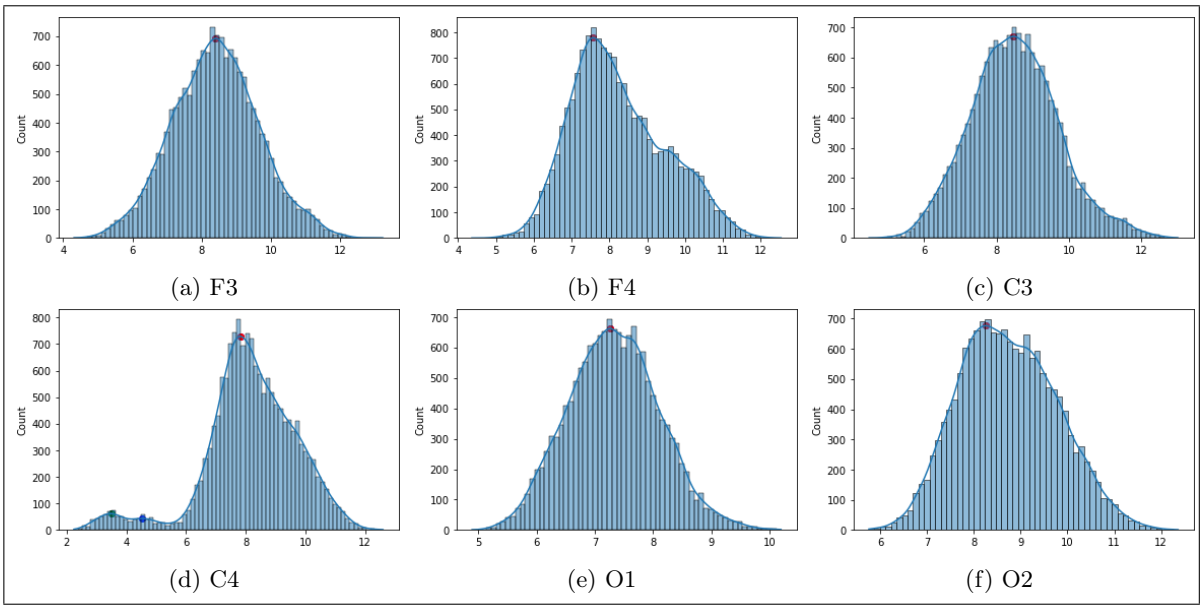
MT_variance_for_structural_aritifact_fu

Lets check the idea through visually before we go into details via the section 11.1.

Then lets check the functions.

11.1 Compare the standard distribution choices.

From the selected subject's dB scale Multitaper Spectrum's standard deviation (σ) of NREM stage comparison between the channels as shown in the Fig. 11.1, the threhold five would be an good choise to get rid of the left side distribution of channel C4.



(a) F3 (b) F4 (c) C3

(d) C4 (e) O1 (f) O2

Figure 11.1: 21-0995_F_9.7_1_di subjects MT spectrum's σ's distribution with different channels' sleep-stage NREM pooled

However when we check the Multitaper Spectrum of the subject with the predicted outliers (labeled as "Arousal") based on the the standard deviation threhold is shown in the Fig. 11.2. Inorder to show the different ways of annotating the outliers are inlcuded with this handpicked threshold. In the Fig.

11.2 the outliers detected by these methods are labeled as "Arousal". Since the correlation based methodolgies need atleast two channels, and one shoud have good segment regions while the other channel has artifact. (as shown in the Fig. 11.2 after 4.5 hours channel C4 has bad segments while F4 has good segments).

In the Fig. 11.2, first coloumn's rows are in the order

1. Sleep stage annotation with the correltion (blue dots) of F4 and C4 among the spectral dimention.
2. F4's Multitaper Spectrum
3. Only standard deviation (σ) along frequency to annotate outliers (labeled as "Arousal") if segment's $\sigma < 5$ (threhold as 5 for this subject). While annotating Arousals avoiding (exclude) the spindle segements (presented in the next row). Means if the spindle segment is annotated as 1 by the variance based threhold, then reassign 0 for avoiding spindle location.
4. Spindles: the spindles of channel C4 and F4 are obtained by the YASA, and spindle presence of C4 and F4 are one-hot seperately. If atleast one of the channel has spindle then the segment is marked as "1" in the marking; in other words both spindles are combined with the logically "OR" between the one hot encoding.
5. EEG arousal: the arousal (kind of movement, etc.) marking come along with the raw EEG annotation's by the technicians/ software while recording the sleep-EEG.
6. Only standard deviation (σ) along frequency including spindle location in thresholds analysis: as we can clealy see the inclusion of spindle location inclreases the prediction by the variance based method.
7. Varaiance threhold detection OR Correlation based approach including the spindle location.
8. Only correlation based approach avoiding spindle location.
9. Only correlation based approach including spindle location.

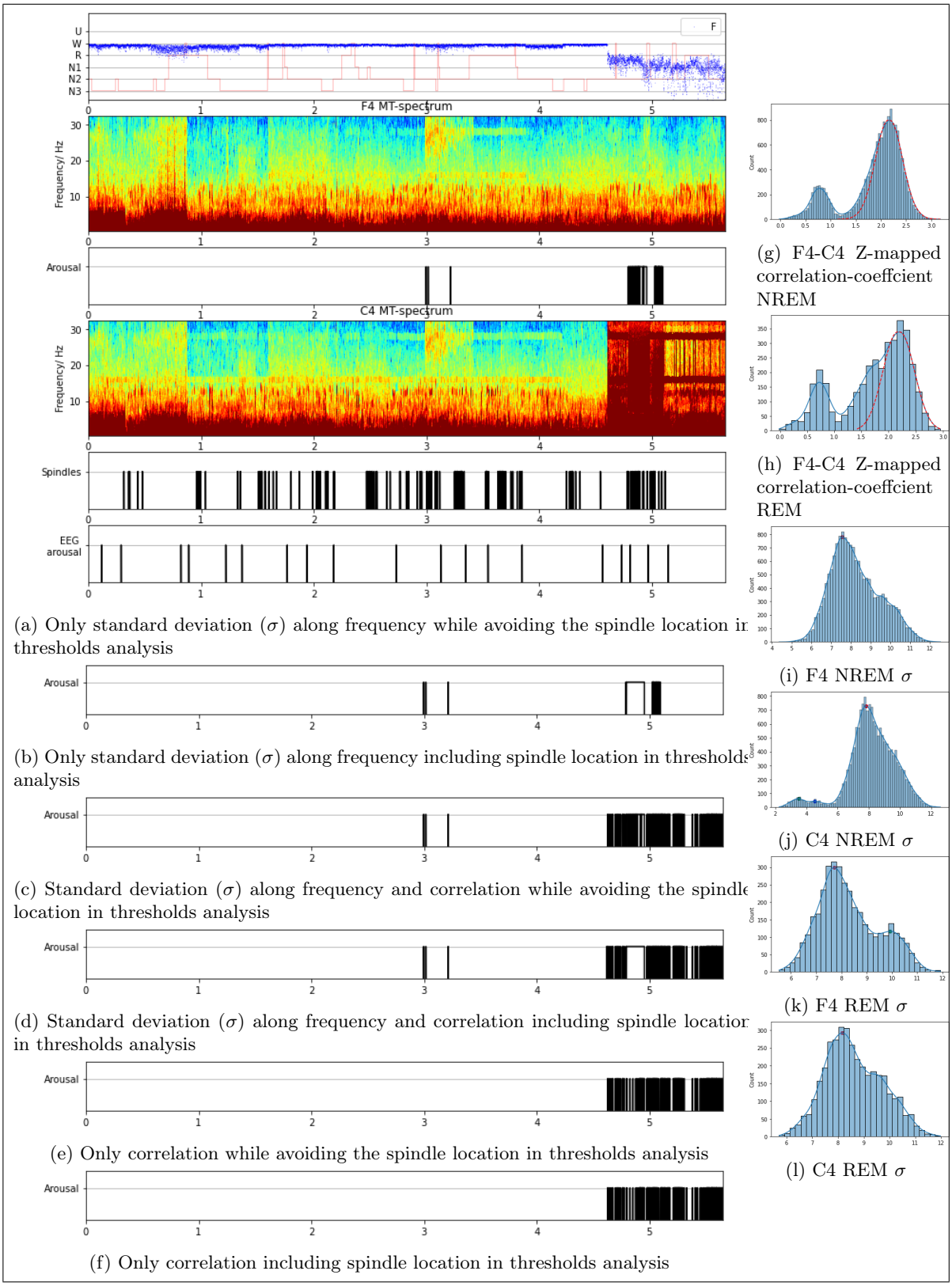
☞ It is clearly visible the correlation based approach outperforms the varaince (standard deviation) based approach.

Choices impact

It is cearly visible between the $4.5 < time < 5$, the only variance based detection while including spindles mark a chunk of segements as "Arousal" (outlier). Thus the choices make impact in the Arousal detection.

Scaling choice impact

☞ The user must know scaling choice (dB vs raw) made in the Multitaper Spectrum estimation with the numerical threshold based method like, variance based detect threhold as 5, will entirely change the outcome. Oneway of dealing with as using statitical methods to define the threhold from the distribution; But this need to be explored.



(a) Only standard deviation (σ) along frequency while avoiding the spindle location in thresholds analysis

(b) Only standard deviation (σ) along frequency including spindle location in thresholds analysis

(c) Standard deviation (σ) along frequency and correlation while avoiding the spindle location in thresholds analysis

(d) Standard deviation (σ) along frequency and correlation including spindle location in thresholds analysis

(e) Only correlation while avoiding the spindle location in thresholds analysis

(f) Only correlation including spindle location in thresholds analysis

(g) F4-C4 Z-mapped correlation-coeffcient NREM

(h) F4-C4 Z-mapped correlation-coeffcient REM

(i) F4 NREM σ

(j) C4 NREM σ

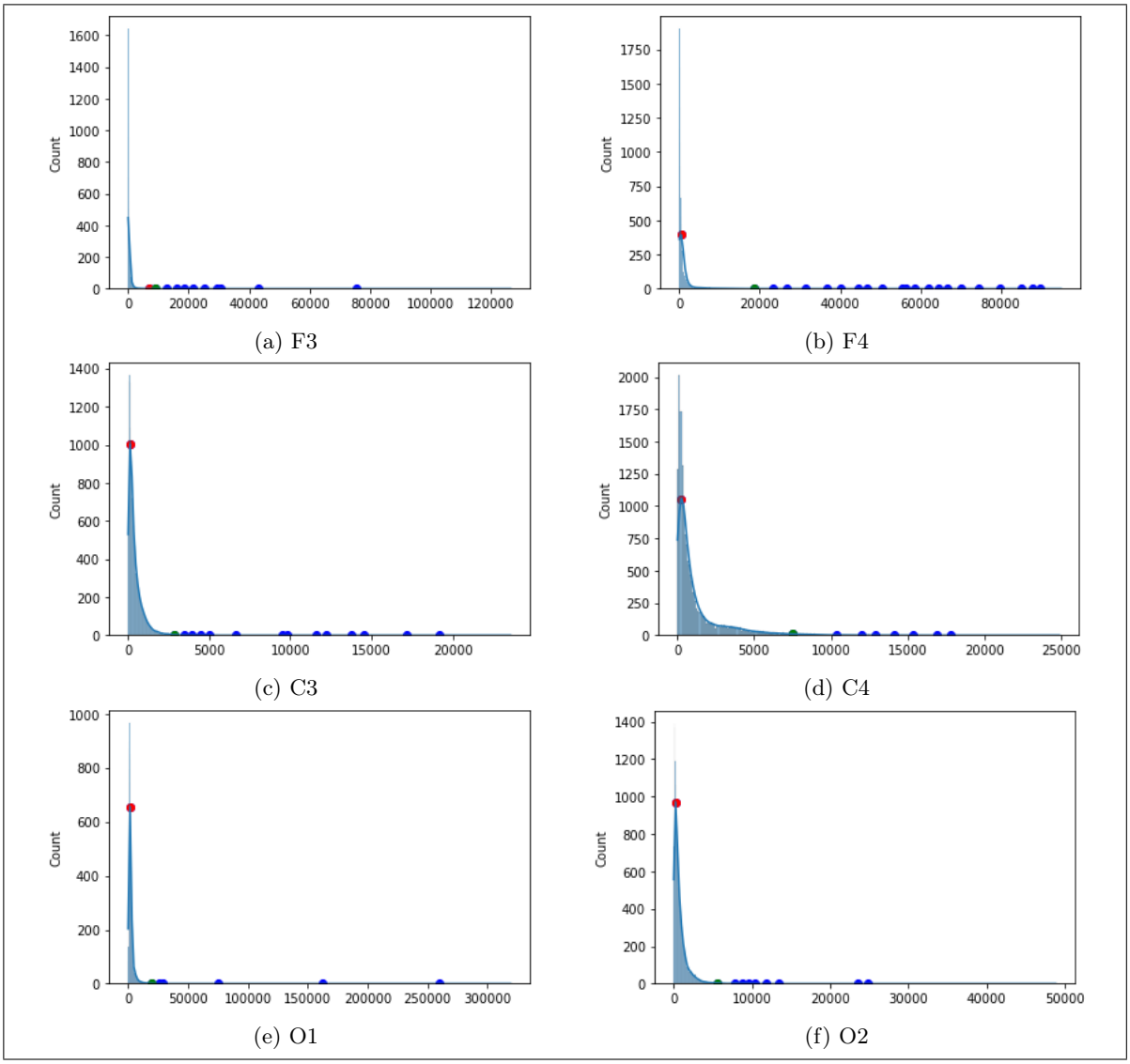
(k) F4 REM σ

(l) C4 REM σ

Figure 11.2: Comparison of different outlier arousal annotations of subject 21-0995_F_9.7_1_di with selected reref onside F4-C4 groups channels channel with σ distribution (histograms) on F4-C4.

Range of standard deviation (σ) are totally different for even a single sleep-group of NREM such that,

- ✓ **dB**: from $\approx 0 to \approx 13$ Fig. 11.1
- ✓ **Raw**: from $\approx 0 to \approx 300,000$ as shown in Fig. 11.3



(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure 11.3: MT spectrum's σ's raw distribution with different channels' sleep-stage NREM pooled

11.2 Functions of varance/ standard deviation based approach

```python
import numpy as np
import logging
import matplotlib.pyplot as plt

from copy import deepcopy
from sleep_EEG_loose_lead_detect.outlier_common.outlier_based_on_distribution_fucns
                                                    import outlier_stat_finder

logger = logging.getLogger("variance_std_div_threh")

while logger.handlers:
     logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
# Set logging level to the logger
# logger.setLevel(logging.DEBUG) # <-- THIS!
logger.setLevel(logging.INFO)
logger.propagate = False
```

Figure 11.4: MT_variance_for_structural_aritifact_funcs importing dependencies and initialisation

11.2.1 Find stats for vertical spikes

Since the fixed numerical values not going to work effective for subjects in different ages/ sex/ diseases, etc. Omway to deal with this find the statistics based threhold to treat all them in the same way. The function Fig. 11.5 presents some ways.

Since the function itself has comments to walkthrough, for the time being, I haven't present more details here.

```python
def find_statistics_for_vertical_spikes(MT_spec_db,outlier_basic_con):

    '''
    take the MT_raw and check their variance among the frequency distribution

    Here we check among the frequency band
    Further we are checking the variance didtribution while pooling
    all the channels information together (without Z-standisation)

    '''
    #inorder to pick the data using only one subject then combine different
    #  of subjects finalised threhold can be derived
    pooled_variance = np.transpose(MT_spec_db.std(axis=1),(1,0))

    #if any of the channels have higher standard deviation means there is a
    # continious block has an issue
    # standard_dev = np.std(pooled_variance,axis=0)
    mean = np.mean(pooled_variance,axis=0)

    #here we are not deviding by standard deviation, but subtract the mean to
    # centralise due to noise channel mean value going to be changed
    standardized_var = (pooled_variance-mean)#/standard_dev
    # standardized_var = pooled_variance

    standardized_var_pooled =standardized_var.flatten()
    min_cutt, max_cutt = outlier_stat_finder(standardized_var_pooled,
                                             outlier_basic_con=outlier_basic_con)
    channel_dic={}
    for ch in range(0,6):
        check=[]
        for sl_pos in range(0,len(pooled_variance)):
            if standardized_var[sl_pos,ch] <min_cutt or
                                            standardized_var[sl_pos,ch] >max_cutt:
                check.append(pooled_variance[sl_pos,ch])
        channel_dic[ch]=deepcopy(check)
    return channel_dic
```

Figure 11.5: Find statistics for vertical spikes

11.2.2 Find vertical spikes

```
def find_vertical_spikes(MT_spec_db,MT_spec_raw,std_thres=5,cont_seg_wise=False,
                                                return_var=False, centralise_std=False):
    '''
        std_thres=5 mostly if that below ~8 looks concerns
    '''
    logger.warning("choose the std threhold wisely we don't gurantee the default value
                                                 provided will find the spikes")
    logger.warning("The default value 5 only selected for dB scale MT-spectrum")

    #this can be done by just pooling all channels variance and find the outlier
    pooled_variance = np.transpose(MT_spec_db.std(axis=1),(1,0))

    if centralise_std:
        #here we are not deviding by standard deviation, but subtract the mean to
        # centralise due to noise channel mean value going to be changed
        mean = np.mean(pooled_variance,axis=0)
        pooled_variance = (pooled_variance-mean)#/standard_dev

    if  cont_seg_wise:
        # flat_MT = np.zeros((len(MT_spec_raw),6))
        flat_MT=np.ones((np.shape(pooled_variance)[0],np.shape(pooled_variance)[1]))
        s_pos =0
        for cont_seg in range(0,len(MT_spec_raw)):
            MT_db=  10*np.log10(MT_spec_raw[cont_seg])
            e_pos = s_pos+np.shape(MT_db)[2]
            flat_MT[s_pos:e_pos,:] = np.any(MT_db.std(axis=1)<=std_thres, axis=1)
            s_pos = s_pos+np.shape(MT_db)[2]
    else:
        # considering all the MT-frequency's variance
        flat_MT=np.ones((np.shape(pooled_variance)[0],np.shape(pooled_variance)[1]))
        flat_MT = np.where(pooled_variance<std_thres,flat_MT,0)
    if return_var:
        return flat_MT, pooled_variance
    else:
        return flat_MT
```

Figure 11.6: Find vertical spikes

11.2.3 Plot vertical spikes

```
def plot_vertical_spike_founder(fname,MT_spec_db,
        flat_MT, ch1,ch_names, sleep_stages_annot_flatten,
        markersize=[0.25,0.1], f_min_interst=0.5, f_max_interst=32.5,
        vmin=-15, vmax=15, save_fig=True, db_scale_given=False):

    sleep_stages_annot_plot_flatten=sleep_stages_annot_flatten/5

    tt = np.arange(len(sleep_stages_annot_flatten))/3600

    fig = plt.figure(figsize=(12,5.5))

    gs = fig.add_gridspec(3, 1, height_ratios=[1.5,2,1])

    ax_ss = fig.add_subplot(gs[0])

    ax_ss.step(tt, sleep_stages_annot_plot_flatten, color='r',linewidth=markersize[0])
    ax_ss.yaxis.grid(True)
    ax_ss.set_ylim([0.05,1.5])
    ax_ss.set_xlim([0,tt[-1]])

    sleep_bound =list(np.array([1,2,3,4,5,6])/5)
    ax_ss.set_yticks(sleep_bound)
    ax_ss.set_yticklabels(['N3', 'N2', 'N1', 'R', 'W','U'])
    ax_ss.legend()

    ax_mt1 = fig.add_subplot(gs[1])
    ax_mt1.imshow(MT_spec_db[ch1,:,:], cmap='jet', origin='lower', extent=[0,tt[-1],
                    f_min_interst,f_max_interst], interpolation='none',aspect='auto',
                    vmin=vmin,vmax=vmax)

    ax_mt1.set_ylabel('Frequency/ Hz')
    ax_mt1.set_xlabel('time (hour)')
    ax_mt1.set_title(ch_names[ch1]+' MT-spectrum')


    ax_s1 = fig.add_subplot(gs[2])
    ax_s1.step(tt, flat_MT[:,ch1], color='k')
    ax_s1.yaxis.grid(True)
    ax_s1.set_ylim([0.05,1.5])
    ax_s1.set_yticks([1])
    ax_s1.set_yticklabels(['Arousal'])
    ax_s1.set_xlim([0,tt[-1]])

    if save_fig:
        plt.savefig(ch_names[ch1]+'_vertical_spike_'+texfile_name+'.png',
                        bbox_inches='tight', pad_inches=0.05)
```

Figure 11.7: Plot vertical spikes

Chapter 12

poolers_Z_standardization_funcs

Functions presented here can be used transform/ map the correlation. And used to pool them together based on the events, like selected sleep-stages combination. These functions are used as mentioned in the subsection. 2.5.2 for mapping and pooling. There,

1. the function "inter_mediate_mapping_correlation" as shown in Fig. 12.4 is used for intermediate mapping to avoid the boudry condition issue for fisher z-transformation.
2. Fisher z-transformation is obtained by function "z_standardization" as shown in Fig. 12.5 on raw-correlation.
3. The function "obtain_mean_corr" as shown in Fig. 12.6 can be used for obtain the mean along the channles (for Approach-2).
4. Then the function "correlation_pooler" as shown in Fig. 12.3 is used to pool the raw/ mapped correlations together as distribution. This function causes the temporal information loss. Since we are obtaining the distibution of sleep-related events the number of pooled samples mimimum number is checked as 100 as default by threhold_pooled_samples as 99. Further more can be done,

Please check the each functions' section seperately for more details regarding their capabilities.

```
import logging
import numpy as np

logger = logging.getLogger("poolers_z")

while logger.handlers:
    logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
# Set logging level to the logger
# logger.setLevel(logging.DEBUG)
logger.setLevel(logging.INFO)
logger.propagate = False
```

Figure 12.1: poolers_Z_standardization_funcs intialisation

12.1 Cross correlation dictionary

In default we have six channels (re-referenced), we are going to end up in $\binom{6}{2} = 15$ correlation combinations. However the correlation obtained by the packages have redundant correltions as well. For an example the correlation coefficient between "F3-F4" and the "F4-F3" going to be same. such that this function only extract the useful information via the dictinoary assignment.

```
def cross_correltion_dic(ch_names = ['F3', 'F4', 'C3', 'C4', 'O1', 'O2']):
    '''
    Parameters
    ----------
    ch_names : TYPE, optional
        DESCRIPTION. The default is ['F3', 'F4', 'C3', 'C4', 'O1', 'O2'].

    Returns
    -------
    cross_correlation_ref_dic : relaive correlation as dictionary format with indexes

    '''
    cross_correlation_ref_dic={}
    a=0
    for ch1 in range(0,len(ch_names)-1):
        for ch2 in range(ch1+1,len(ch_names)):
            cross_correlation_ref_dic[ch_names[ch1]+'-'+ch_names[ch2]]=[[ch1,ch2],a]
            a=a+1

    return cross_correlation_ref_dic
```

Figure 12.2: This dictionary points the indexes of the cross-correlation between the channels obtained; default coross correlation amond the six channels are obtained.

12.2 Pool the correlations together

Based on the sleep stages we need to pool all correlation values together while providing some conditions. Inorder to understand the usage of the function we need to understand the methodology (especialy the approach-1 as mentioned in the Chapter. 2). And the avoiding conditions places due to the known artifacts encodings, and spindles etc.

```
def correlation_pooler(correlation_flatten, sleep_stages_annot_flatten,
    intersted_sleep_stages, flat_MT_consider=False,flat_MT_ch=[],
    threhold_pooled_samples=99, avoid_spindle_loc = False, spindle_enc_ch =[]):
    '''
    This function pool the selected sleep-stages with their correlation values

    threhold_pooled_samples: if the pooles samples above the given threhold
    only considered here 99 means,
        atleast 100 samples should be exist to consider in outlier detection

    avoid_spindle_loc: True will omit the correlation values occured in the
                    spindle occured places
    in this case already skipped flat_MT_ch (like presented other algorithms/
    known annotated outliers/ artifacts)
    '''
    corr_pool=[]
    first=True
    for sl_st_in in range(0,len(sleep_stages_annot_flatten)):
        if sleep_stages_annot_flatten[sl_st_in] in intersted_sleep_stages:
            # if we considering both variance
            # (/ already presented outlier annotations) and avoid spindle location
            if avoid_spindle_loc and flat_MT_consider:
                if np.sum(flat_MT_ch[sl_st_in,:])==0 and np.sum(spindle_enc_ch[
                                                sl_st_in,:])==0:
                    corr_pool.append(correlation_flatten[sl_st_in])
            # if we only avoid spindle location
            elif avoid_spindle_loc:
                if np.sum(spindle_enc_ch[sl_st_in,:])==0:
                    corr_pool.append(correlation_flatten[sl_st_in])
                    if first:
                        logger.warning("considering any single present in any channel
                                                leave that option out, this can be
                                                developed further")
                        first=False
            # if we considering variance/ alreday presented outlier annotations
            elif flat_MT_consider:
                if np.sum(flat_MT_ch[sl_st_in,:])==0:
                    corr_pool.append(correlation_flatten[sl_st_in])
            # if we do not consider neither variance nor avoid spindle location
            else:
                corr_pool.append(correlation_flatten[sl_st_in])

    if len(corr_pool)>threhold_pooled_samples:
        corr_pool = np.stack(corr_pool,axis=0)
        correlation_pool_sat = True
    else:
        correlation_pool_sat = False
        logger.warning("Due to lack of samples (%i) the pooling not satisfied the
                                            samples need to be above %i",len(corr_pool)
                                            ,threhold_pooled_samples)

    return corr_pool, correlation_pool_sat
```

Figure 12.3: This function pool the correlations together based on the given condition

12.3 Mapping/ transforming the correlation

Here the two standardization techniques are provided with the Z-transformation. Such that one with the tanh (Fisher transformation) or the mean and standard deviation based. Please check the psudo code as shown in Fig. 12.

```
def inter_mediate_mapping_correlation(corr_coeff_t,b_val=0.0001):
    '''
        corr_coeff_t: dimention time x ch x ch
            or
        corr_coeff_t: dimention time x ch
            or
        corr_coeff_t: dimention time

    this function is can map the correlation values depends on the input given
    '''
    #this is modified to accept the single group
    if len(np.shape(corr_coeff_t))==3:
        for a in range(0,np.shape(corr_coeff_t)[0]):
            for ch1 in range(0,np.shape(corr_coeff_t)[1]):
                for ch2 in range(0,np.shape(corr_coeff_t)[2]):
                    if corr_coeff_t[a,ch1,ch2]==1:
                        corr_coeff_t[a,ch1,ch2]=corr_coeff_t[a,ch1,ch2]-b_val
                    elif corr_coeff_t[a,ch1,ch2]==-1:
                        corr_coeff_t[a,ch1,ch2]=corr_coeff_t[a,ch1,ch2]+b_val

    elif  len(np.shape(corr_coeff_t))==2:
        for a in range(0,np.shape(corr_coeff_t)[0]):
            for ch1 in range(0,np.shape(corr_coeff_t)[1]):
                if corr_coeff_t[a,ch1]==1:
                    corr_coeff_t[a,ch1]=corr_coeff_t[a,ch1]-b_val
                elif corr_coeff_t[a,ch1]==-1:
                    corr_coeff_t[a,ch1]=corr_coeff_t[a,ch1]+b_val
    else:
        for a in range(0,np.shape(corr_coeff_t)[0]):
            if corr_coeff_t[a]==1:
                corr_coeff_t[a]=corr_coeff_t[a]-b_val
            elif corr_coeff_t[a]==-1:
                corr_coeff_t[a]=corr_coeff_t[a]+b_val
    return corr_coeff_t
```

Figure 12.4: A helper function to explointing numerical value due to mapping by tanh

```
def z_standardization(corr_pool,Fisher_based=True):
    '''
    corr_pool: dimention time x ...
    such that this z-standize along the time axis

    Fisher_based: standardize with arctanh function
    '''
    if Fisher_based:
        correlation_z=np.arctanh(corr_pool)
    else:
        standard_dev = np.std(corr_pool,axis=0)
        mean = np.mean(corr_pool,axis=0)
        correlation_z = (corr_pool-mean)/standard_dev
    return correlation_z
```

Figure 12.5: Z-standardisation based on Fishers' or with the mean and standard deviation

12.4 Mean correlation

Inorder to calculate the mean correlation along the good-channels. Means avoiding the "loose_lead_channels" in while calculate the correlation with the other channel. For an example: If the C4 channel is loose

among the six (F3 , F4 , C3 , C4 , O1 , O2) channels, Then the mean correlation obtained as,

- ✓ F3 obtained from correlations of F3-F4, F3-C3, F3-O1, and F3-O2.
- ✓ F4 obtained from correlations of F3-F4, F4-C3, F4-O1, and F4-O2.
- ✓ C3 obtained from correlations of F3- C3, F4-C3, C3-O1, and C3-O2.
- ✓ C4 obtained from correlations of F3- C4, F4-C4, C4-O1, and C4-O2.
- ✓ O1 obtained from correlations of F3-O1, , F4-O1, O1-C3, and O1-O2.
- ✓ O2 obtained from correlations of F3-O2, , F4-O2, O2-C3, and O1-O2.

Since we are avoiding the loose-channels in the mean correlation calculation, we need to have atleast one good channel for all the other channels get the mean for the loose-lead detection.

```
def obtain_mean_corr(bm_corr_pool_given,
        ch_names = ['F3', 'F4', 'C3', 'C4', 'O1', 'O2'],loose_lead_channels=[]):
    '''
    this function creates the final mean of the correlation pool group
    while avoiding the diagonal correlation
    '''
    corr_pool_mean_ob = np.zeros((np.shape(bm_corr_pool_given)[0],len(ch_names)))
    for ch1 in range(0,len(ch_names)):
        temp_ch_indexes = list(range(0,len(ch_names)))
        # to avoid the self correlation
        temp_ch_indexes.remove(ch1)
        #  avoid the mean of correlations with the loose-leads
        for l_ch in loose_lead_channels:
            if l_ch in temp_ch_indexes:
                temp_ch_indexes.remove(l_ch)
        corr_pool_mean_ob[:,ch1] = np.mean(bm_corr_pool_given[:,ch1,temp_ch_indexes],
                                           axis=1)

    return corr_pool_mean_ob
```

Figure 12.6: This function obtain the mean correlation along the channel axis

```
def loose_lead_channel_count_check(ch_names,loose_lead_channels):
    if len(list(set(ch_names)))<(len(list(set(loose_lead_channels)))+2):
        raise ValueError("Atleast one good-channel need to be create the mean for each
                                           channel")
```

Figure 12.7: To check the number of good channel for mean of correlation along the good-channels calculation.

Chapter 13

out_lier_based_fun

```
import logging
import numpy as np
from copy import deepcopy

from sleep_EEG_loose_lead_detect.outlier_common.moving_window_funcs \
    import moving_window_based_out_lier_annotator_channel_mean
from sleep_EEG_loose_lead_detect.outlier_common.outlier_based_on_distribution_fucns\
    import out_lier_annotator_channel_mean


# from outlier_common.cont_segment_artofact_annotation import calc_artifact_start_end,
                                                          \
#  finalise_artifact_start_end, calc_per_loose_lead, loose_con_seg_main_index

# from outlier_common.moving_window_funcs import
                                            break_cont_segments_intersetd_sleep_stages
                                            ,\
#  median_with_moving_window_in_broken_cont_groups#, local_outlier_detection

logger = logging.getLogger("outlier_funcs")

while logger.handlers:
     logger.handlers.pop()
c_handler = logging.StreamHandler()
# link handler to logger
logger.addHandler(c_handler)
# Set logging level to the logger
# logger.setLevel(logging.DEBUG) # <-- THIS!
logger.setLevel(logging.INFO)
logger.propagate = False
```

Figure 13.1: out_lier_based_fun intialisation

As clearly seen from the initialisation, the main functions to detection of outliers from the distribution/ moving based function. The functions presented here are mainly based on the Approach-2 (mean-correlation).

The independent correlation based outlier detection (Approach-1) based function is in commented due to the function need to be optimized, and cleaned.

The function "find_loose_leads_based_mean" shown in two parts in Fig. 13.2 and Fig. 13.3.

Since inputs belongs to the function "find_loose_leads_based_mean"; are fed through the function related such as,

- ✓ Moving window based method: "moving_window_based_out_lier_annotator_channel_mean" is now on mentined as mov_{fun}
- ✓ Distribution based method: "out_lier_annotator_channel_mean" is now on mentined as dis_{fun}

Lets first see the inputs with their function,

- ✓ correlation_flatten_MT_not_opt: The correlations

```
def find_loose_leads_based_mean(correlation_flatten_MT_not_opt,
    sleep_stages_annot_flatten, ch_names,cross_correlation_ref_dic,
    intersted_sleep_stages_NREM_REM, outlier_basic_con=3, b_val=0.0001,
    inter_mediate_transform=True, z_transform=True,Fisher_based=True,
    flat_MT_consider=True, flat_MT=[], avoid_spindle_loc=False, spindle_enc=[],
    spindle_possible_channels =[],  break_spindle_flatten=True,
    break_flat_MT_flatten=True, num_bins=20, density=True,#unimodal vs multimodal
    no_smoothing=False, persent_consider =20, tail_check_bin=True, factor_check=10,
    factor_check_consider_second=100, GMM_based=True, threh_prob = 0.01,
    cont_EEG_segments_np=[], cont_threh=4, o_p_adjuster=3, ep_length=30,
    threh_prob_artf_cont_seg=0.5,
    moving_window_based=False,#moving window based parameters
    moving_window_size=60, th_fact=4, global_check=False, only_good_points=False,
    sorted_median_cont_grp_comp_sort_median_cond=[True, 10],
    sorted_median_cont_grp_comp_sort_max_cond=[True,10],
    sorted_median_cont_grp_comp_sort_quan_cond=[True,10,0.75],
    loose_lead_channels=[], verbose=False,GUI_percentile=True):


    # while avoiding already presented outlier annotations
    #  for an example this can be the varaince based outlier flat_MT
    if flat_MT_consider:
        flat_MT_ch =flat_MT
        arousal_annot =deepcopy(flat_MT)
    else:
        arousal_annot = np.zeros((len(sleep_stages_annot_flatten),len(ch_names)))
        flat_MT_ch=[]

    # while avoiding the spindle occuring locations
    if avoid_spindle_loc:
        # correlation_pooler will avoid the location of the spindles when it pools
        #  so this will be omitted in the distribution based detection
        # default consider the spindles of all the channels
        if len(spindle_possible_channels)==0:
            spindle_enc_ch =spindle_enc
        else:
            # if the user provide the spindle_possible_channels
            #  then concentrate only on these spindle_possible_channels whether the
            # spindles exist or not asssign the rest of the channels no-spidles exist
            spindle_enc_ch= np.zeros((len(sleep_stages_annot_flatten),len(ch_names)))
            for ch1 in range(0,len(ch_names)):
                if (ch1 in spindle_possible_channels):
                    spindle_enc_ch[:,ch1] = spindle_enc[:,ch1]
    else:
        spindle_enc_ch=[]
```

Figure 13.2: Main function predicts the outliers based on the mean of correlation function to cover given sleep stage group (both NREM and REM together or just one sleep-stage group) since the function is bigger part-1 is presented here.

```
    # then the outlier detection choises
    # of how we are going to use the mean to find the outliers using
    # either the moving window based approach or the distribution based approach
    if moving_window_based:
        if len(cont_EEG_segments_np)==0:
            raise ValueError("this needs to atleast one contious-segments {sleep-pre-
                                        processed-fragments (SPPF)}  ")
        arousal_annot=  moving_window_based_out_lier_annotator_channel_mean(
            correlation_flatten_MT_not_opt,sleep_stages_annot_flatten,
            intersted_sleep_stages_NREM_REM, ch_names, arousal_annot,
            cont_EEG_segments_np, b_val=b_val,
            inter_mediate_transform =inter_mediate_transform,z_transform=z_transform,
            flat_MT_consider=flat_MT_consider,flat_MT_ch=flat_MT_ch,
            intersted_sleep_stages_term='NREM_REM ',
            avoid_spindle_loc=avoid_spindle_loc, spindle_enc_ch=spindle_enc_ch,
            break_spindle_flatten=break_spindle_flatten,
            break_flat_MT_flatten=break_flat_MT_flatten,
            moving_window_size=moving_window_size, th_fact=th_fact,
            o_p_adjuster=o_p_adjuster, ep_length=ep_length,
            global_check = global_check, only_good_points=only_good_points,
            sorted_median_cont_grp_comp_sort_median_cond=\
                        sorted_median_cont_grp_comp_sort_median_cond,
            sorted_median_cont_grp_comp_sort_max_cond=\
                        sorted_median_cont_grp_comp_sort_max_cond,
            sorted_median_cont_grp_comp_sort_quan_cond=\
                        sorted_median_cont_grp_comp_sort_quan_cond,
            loose_lead_channels=loose_lead_channels, verbose=verbose,\
                        GUI_percentile=GUI_percentile)
    else:
        arousal_annot = out_lier_annotator_channel_mean(correlation_flatten_MT_not_opt
            ,sleep_stages_annot_flatten, intersted_sleep_stages_NREM_REM,
            ch_names, arousal_annot,outlier_basic_con=outlier_basic_con,
            b_val=b_val,inter_mediate_transform=inter_mediate_transform,
            z_transform=z_transform,Fisher_based=Fisher_based,
            flat_MT_consider=flat_MT_consider,flat_MT_ch=flat_MT_ch,
            intersted_sleep_stages_term='NREM_REM ',
            avoid_spindle_loc=avoid_spindle_loc, spindle_enc_ch=spindle_enc_ch,
            num_bins = num_bins, density=density,
            no_smoothing = no_smoothing, persent_consider =persent_consider,
            tail_check_bin= tail_check_bin, factor_check=factor_check,
            factor_check_consider_second=factor_check_consider_second,
            GMM_based=GMM_based, threh_prob =threh_prob,
            loose_lead_channels=loose_lead_channels)

    # make sure avoiding the spindle occuring locations
    if avoid_spindle_loc:
        '''
            to skip the spindle present in the arousal_annot anotation
        '''
        #first find the intersept
        _arousal_annot= spindle_enc_ch*arousal_annot

        #then negate the common occurances
        arousal_annot = arousal_annot -  _arousal_annot

    return arousal_annot
```

Figure 13.3: find_loose_leads_based_mean part-2

Chapter 14

moving_window_funcs

This have the functions related to moving window.

14.1 Intialisation

```
1
2  import logging
3  import numpy as np
4
5  from copy import deepcopy
6
7  from sleep_EEG_loose_lead_detect.channel_correlation_outlier.\
8      poolers_Z_standardization_funcs import obtain_mean_corr, \
9      correlation_pooler, inter_mediate_mapping_correlation,z_standardization
10 from sleep_EEG_loose_lead_detect.outlier_common.\
11     cont_segment_artifact_annotation import cont_EEG_segments_np_len
12 from sleep_EEG_loose_lead_detect.GUI_interface.percentage_bar_vis \
13     import percent_complete
14
15
16 logger = logging.getLogger("moving_window_local_outlier_funcs")
17 while logger.handlers:
18      logger.handlers.pop()
19 c_handler = logging.StreamHandler()
20 # link handler to logger
21 logger.addHandler(c_handler)
22 # Set logging level to the logger
23 # logger.setLevel(logging.DEBUG) # <-- THIS!
24 logger.setLevel(logging.INFO)
25 logger.propagate = False
```

Figure 14.1: Moving window based functions intialisation.

14.2 Correlation-mean based moving window outlier detection

The Approach-2 as mentioned in the **manuscript - review** in chapter "Methods and ideas with the package " (2).

The main function used for mean correlation with the moving window based outlier annotaion is "moving_window_based_out_lier_annotator_channel_mean". Since this function covers the option of outlier detection in different ways. The function is big, such that it is shown in three-parts such as Fig. 14.2, Fig. 14.3 and Fig. 14.4.

Since the main for block (shown in Fig. 14.3) is bigger in the function, such that for loop's body is shown in the Fig. 14.4.

As shown in the Fig. 14.2, first the correlation values are initialised, like obtain the mean correlations (in line 47), then check the enough number of correlation values present (line 53) for the interseted sleep-groups for moving window based method. Incase not enough samples then the given annotations are return without detection, with the warning as shown in Fig. 14.3.

If enough samples presented then based on the transformation conditions given by the user, in the lines of of Fig. 14.3 as

- ✓ inter_mediate_transform: line 10
- ✓ z_transform: line 13

Then based on the continiuous segementation's (SPFFs') interested sleep relate segments (fragments) are going to be only by moving window detetcion. Inorder to break the continious segments into fragments the function "break_cont_segments_intersetd_sleep_stages" is used.

Even though the fragements are already created, then if the user want to break the fragements further to avoid the interuption caused by the outliers/ spindles. Then they are fragmented by function "break_flat_MT_from_broken_sleep_st_segments_intersetd_sleep_stages" Please check the section 14.2.

```
1
2  def moving_window_based_out_lier_annotator_channel_mean(
3      corr_check_given,sleep_stages_annot_flatten, intersted_sleep_stages,
4      ch_names, arousal_annot,
5      cont_EEG_segments_np,
6      b_val=0.0001,inter_mediate_transform=True,z_transform=True,Fisher_based=True,
7      flat_MT_consider=True,flat_MT_ch=[],intersted_sleep_stages_term='',
8      break_spindle_flatten=True,break_flat_MT_flatten=True,
9      avoid_spindle_loc=False, spindle_enc_ch=[],
10     moving_window_size=60, th_fact=1.5, o_p_adjuster=3, ep_length=30,
11     global_check=False, only_good_points=False,
12     sorted_median_cont_grp_comp_sort_median_cond=[True, 10],
13     sorted_median_cont_grp_comp_sort_max_cond=[True,10],
14     sorted_median_cont_grp_comp_sort_quan_cond=[True,10,0.75],
15     loose_lead_channels=[], SPFF_return_int_sleep=False,
16     verbose=False, GUI_percentile=True):
17     '''
18     corr_check : this contain the correlation value flatten x channel x channel
19     intersted_sleep_stages : the interested sleep stages that going to be selected
20      for outlier annotations
21
22
23     only_good_points: means the global threshold is obtained based only from the
24      good-distribution( that are not anotated as local condition as bad segment)
25
26     o/p:  arousal_annot : anotated as 1 if they fell in outlier based condition
27
28     break_spindle_flatten=True,break_flat_MT_flatten==True,
29         this will break on the given flat_MT conditions and spindle condition
30          as continious blocks
31     '''
32     if verbose:
33         logger.info("Moving window based outlier detection intitated")
34     if GUI_percentile:
35         logger.warning('Intiation warning')
36
37         if global_check:
38             if only_good_points:
39                 logger.warning('The good pool of values obtained by the local window %
                                       i is used for global threshold',
                                       moving_window_size)
40             else:
41                 logger.warning('The median values obtained by the local window %i used
                                        for global threshold', moving_window_size)
42         # percent_complete(1, 100, bar_width=60, title="mov-window-outlier-detection",
                                        print_perc=True)
43     # get the copies to avoid the edition later
44     # all channels are used
45     corr_check, bm_corr_check_mapped = get_deep_copies(corr_check_given)
46
47     corr_check_mapped = obtain_mean_corr(bm_corr_check_mapped, ch_names = ch_names,
48                                          loose_lead_channels=loose_lead_channels)
49
50     # only Z-transform mapping influenced by the spindle inclusion
51     #the following obtained before the mean of the correlation,
52     # just check the minimum numexist
53     _, correlation_pool_sat = correlation_pooler(corr_check,
54         sleep_stages_annot_flatten,   intersted_sleep_stages,
55          flat_MT_consider=flat_MT_consider, flat_MT_ch=flat_MT_ch,
56         avoid_spindle_loc = avoid_spindle_loc, spindle_enc_ch =spindle_enc_ch)
57
58
```

Figure 14.2: moving_window_based_out_lier_annotator_channel_mean part-1

```
# Z-standardisation is performed on the full data
# arousal annotation handle the wanted sleep-stage or not
if correlation_pool_sat:
    if verbose:
        logger.info("Pooling the correltaion is done ")
    if GUI_percentile:
        percent_complete(10, 100, bar_width=60, title="mov-window-outlier-detection",\
                                                                    print_perc=True)
    if inter_mediate_transform:
        corr_check_mapped =  inter_mediate_mapping_correlation(corr_check_mapped,
                                                                    b_val=b_val)
    if z_transform:
        corr_check_mapped=z_standardization(corr_check_mapped,
                                                        Fisher_based=Fisher_based)
    # then using the SPFF's continious segments start end position to check the
    # windowing kind of threhold
    # then based on the continious groups check the sleep-stages groups seperatey
    interseted_sleep_stage_cont_groups_start_end_index_or, \
        len_interseted_sleep_stage_cont_groups_or = \
        break_cont_segments_intersetd_sleep_stages(cont_EEG_segments_np, \
                    sleep_stages_annot_flatten,intersted_sleep_stages, \
                    o_p_adjuster=o_p_adjuster, ep_length= ep_length)

    #  if not having flat_MT previous annpotation only using the continious segments
    if not ((flat_MT_consider and break_flat_MT_flatten) or \
                    (avoid_spindle_loc and break_spindle_flatten)):
        interseted_sleep_stage_cont_groups_start_end_index = deepcopy( \
            interseted_sleep_stage_cont_groups_start_end_index_or)
        len_interseted_sleep_stage_cont_groups = deepcopy(\
                    len_interseted_sleep_stage_cont_groups_or)

    warn_first=True
    if GUI_percentile:
        # since this is warning is implemented above
        warn_first=False
        # here the grp is used to represent the single channel's correlation
                                                    coefficient
    for grp in range(0,len(ch_names)):
```

Calling the part in the Fig. 14.4

```
else:
    logger.warning("correlation pool of "+intersted_sleep_stages_term+" not satisfied
                                                    return the given arousal_annot")

# return the interested sleep-stage continious index information
#   interseted_sleep_stage_cont_groups_start_end_index: contian the start and end
                                                    positions of intersted sleep-groups in
                                                    SPFFs
#   len_interseted_sleep_stage_cont_groups: length of that gruops
if SPFF_return_int_sleep:
    # return arousal_annot, interseted_sleep_stage_cont_groups_start_end_index,
                                                    len_interseted_sleep_stage_cont_groups
    return arousal_annot, interseted_sleep_stage_cont_groups_start_end_index_new,
                                                    len_interseted_sleep_stage_cont_groups_new
else:
    return arousal_annot
```

Figure 14.3: moving_window_based_out_lier_annotator_channel_mean part-2

Then check the fragments are correctly obtained via the function "check_the_break_segment_len" as shown in the section. 14.3. This will make sure the fragments are only presented as ending position

is followed by the starting position.

As shown in the line-6 else group, will force all the annotation greatert than one to one since the spindle need to be avoid in the influence in moving window detection that place is broken in the continous segementation.

In line-21 one time effectively sort the medians and keep them the function median_with_moving_window_in_b effciently sort the elements one time to find the medians via the moving window

line-29 if block, only using the good -points obtained by the local moving window to decide the global based outlier finalisation

line-42 else block, only using the medians of the local moving winodw to decide the global threshold since we already obtianed the medians just local and global can be annotate together

```
if  (flat_MT_consider and break_flat_MT_flatten) or (avoid_spindle_loc and
                                        break_spindle_flatten):
    if (not avoid_spindle_loc) and break_flat_MT_flatten:
        flat_MT_flatten = flat_MT_ch[:,grp]
    elif (not flat_MT_consider) and break_spindle_flatten:
        flat_MT_flatten = spindle_enc_ch[:,grp]
    else:
        #  this will force all the annotation greatert than one to one
        _flat_MT_flatten =  flat_MT_ch[:,grp]+spindle_enc_ch[:,grp]
        flat_MT_flatten = np.where(_flat_MT_flatten<1,_flat_MT_flatten,1)

    interseted_sleep_stage_cont_groups_start_end_index,\
        len_interseted_sleep_stage_cont_groups =\
        break_flat_MT_from_broken_sleep_st_segments_intersetd_sleep_stages(
                interseted_sleep_stage_cont_groups_start_end_index_or,
                len_interseted_sleep_stage_cont_groups_or, flat_MT_flatten)

interseted_sleep_stage_cont_groups_start_end_index_new,\
    len_interseted_sleep_stage_cont_groups_new =\
    check_the_break_segment_len(interseted_sleep_stage_cont_groups_start_end_index,
    len_interseted_sleep_stage_cont_groups)
# one time effectively sort the medians and keep them
sorted_median_cont_grp = median_with_moving_window_in_broken_cont_groups(
    corr_check_mapped, grp,
    interseted_sleep_stage_cont_groups_start_end_index_new,
    len_interseted_sleep_stage_cont_groups_new,
    moving_window_size=moving_window_size)
if global_check:
    if only_good_points:
        # only good-points (from local moving window) to decide  global based outlier
        arousal_annot[:,grp] =  local_and_global_based_outlier_detection_with_loc_good
            (corr_check_mapped, grp, sorted_median_cont_grp,
            interseted_sleep_stage_cont_groups_start_end_index_new,
            len_interseted_sleep_stage_cont_groups_new,
        arousal_annot[:,grp], moving_window_size=moving_window_size, th_fact = th_fact
        , sorted_median_cont_grp_comp_sort_median_cond =\
                        sorted_median_cont_grp_comp_sort_median_cond,
            sorted_median_cont_grp_comp_sort_max_cond = \
                        sorted_median_cont_grp_comp_sort_max_cond,
            sorted_median_cont_grp_comp_sort_quan_cond = \
                        sorted_median_cont_grp_comp_sort_quan_cond,
            verbose=verbose, warn_first=warn_first, GUI_percentile=GUI_percentile)
    else:
        #  The medians of the local moving winodw to decide the global threshold
        arousal_annot[:,grp]  =  local_and_global_based_outlier_detection_with_all
        (corr_check_mapped, grp,  sorted_median_cont_grp,
        interseted_sleep_stage_cont_groups_start_end_index_new,
        len_interseted_sleep_stage_cont_groups_new, arousal_annot[:,grp],
         moving_window_size=moving_window_size, th_fact = th_fact
            ,sorted_median_cont_grp_comp_sort_median_cond = \
                        sorted_median_cont_grp_comp_sort_median_cond,
            sorted_median_cont_grp_comp_sort_max_cond = \
                        sorted_median_cont_grp_comp_sort_max_cond,
            sorted_median_cont_grp_comp_sort_quan_cond = \
                        sorted_median_cont_grp_comp_sort_quan_cond,\
        verbose=verbose, warn_first=warn_first)
    warn_first =False
else:
    # no global condition just follow the same 1996 paper kind of moving window method
    arousal_annot[:,grp]  =  local_outlier_detection(corr_check_mapped, grp,
        sorted_median_cont_grp, interseted_sleep_stage_cont_groups_start_end_index_new
        , len_interseted_sleep_stage_cont_groups_new, arousal_annot[:,grp],
        moving_window_size=moving_window_size, th_fact = th_fact)
if verbose:
    logger.info("Outlier detection done on "+ch_names[grp])
if GUI_percentile:
    percent_complete(10+int(90*((grp+1)/len(ch_names))), 100, bar_width=60,\
                     title="mov-window-outlier-detection", print_perc=True)
```

Figure 14.4: moving_window_based_out_lier_annotator_channel_mean part-3

Break the sleep-blocks into fragments

Based on the continiuous segementation's (SPFFs') interested sleep relate segments (fragments) are going to be only by moving window detetcion.

Please check the chapter "cont_segs_of_whole_EEG" (Fig. 7)

For an **example**, in the Fig. 7.1 from contionus blocks like SPFF-1, SPFF-2, SPFF-3 If the intersted sleep-stages are N2 and N3. Then,

- ✓ SPFF-1 contains N1, N2, N2; is broken to two fragment only the last fragment N2, N2 is used.
- ✓ SPFF-2 is R so it is excluded.
- ✓ SPFF-3 contains N1, N2, N2; is broken to two fragment only the last fragment N2, N2 is used.

In default both NREM and REM are used thus this will return the fragments as the given contniuous sgements.

Then based on the continiuous segementation's (SPFFs') interested sleep relate segments are going to be only by moving window detection.

This functions shown in the Fig. 14.6, and Fig. **??** with the comments to the code's line purpose.

```
def break_cont_segments_intersetd_sleep_stages(cont_EEG_segments_np,
     sleep_stages_annot_flatten,intersted_sleep_stages, o_p_adjuster=3, ep_length= 30)
                                                :
    '''
    annotate continiius segment-np with
    this will annotatte the sleep-pre-processed-fragment (SPPF): the fragmnet term may
    confuce so we use this term to avoid the ambiguity

    this function will select the epoches from the SPFFs with the intersted sleep-
                                                stages
    continious segments this function will further group inside the SPFFs into
    fragmented contioius group of intersted sleep-stages

    For an example if we intersted in NREM sleep stages:
        such that this function can output preprocessed NREM continious fragements

        interseted_sleep_stage_cont_groups_start_end_index: contian the start and
                            end point of the preprocessed continoius NREM segments
        len_interseted_sleep_stage_cont_groups: length of the preprocessed
                            continoius NREM segments
    '''

    len_cont_seg, cont_seg_start_end_index = cont_EEG_segments_np_len(
        cont_EEG_segments_np, o_p_adjuster=o_p_adjuster, ep_length=ep_length)
    # then break further into goups the SPPF segments
    # now we have the cont_seg_start_end_index of starting and ending positions
    # using this we can break the continious segments
    interseted_sleep_stage_cont_groups_start_end_index =[]
    for cn_idx in range(0,len(len_cont_seg)):
        # Just assign the starting and ending position of the preprocess
        s_p = cont_seg_start_end_index[cn_idx][0]
        e_p = cont_seg_start_end_index[cn_idx][1]
        # this will check the sl_st_index poistion relative to continoius segment
        sl_st_index_interset_cont_start = s_p
        # sl_st_index_interset_cont_end: SPFFs end point
        # boundry conditions between the SPFFs
        # the carry over is used for continius blocks boundry condition
        start_check = True
        carry=False
```

Figure 14.5: break_cont_segments_intersetd_sleep_stages part-1

```
    #  go through one SPFF and break further
    # the groups only contian the interested sleep-stages e.g: like NREM (N1, N2, N3)
    for sl_st_index in range(s_p,e_p,ep_length):
        logger.debug('sl_st_index %i:', sl_st_index)
        # the following line is added to check intiate ending point after first
        # full-for-loop
        sl_st_index_interset_cont_end=sl_st_index
        # Go through the sleep_stages_annot_flatten and check the
        # SPFF's selected epoch indexes fell into the interested sleep-stages or not
        # if the sl_st_index not fell break the group of continuity and append the
        # continous list
        # since the sl_st_index is not in the intersted group this can be safely
        # ignored not considered for next starting point
        if not (sleep_stages_annot_flatten[sl_st_index] in intersted_sleep_stages):
            #  this will skip the first index in the current for loop
            #  since it is assigned as sl_st_index_interset_cont_end=sl_st_index
            if not (sl_st_index_interset_cont_end==sl_st_index_interset_cont_start):
                # the following line is added to check whether the broken interested
                                                SPFF
                # belong to the interested sleep-stage
                if sleep_stages_annot_flatten[sl_st_index_interset_cont_start]
                                                    in intersted_sleep_stages:
                    interseted_sleep_stage_cont_groups_start_end_index.append(
                        [sl_st_index_interset_cont_start,sl_st_index_interset_cont_end
                                                ])
                    # since the sleep stage fell into intersted keep on carry
                    # to avoid the boundry condtion fell agin in the continous broken
                                                group
                    carry=False
            #  assign the starting poition to new group to avoid continiously adding
            # the unwanted sleep-continous SPFFS
            sl_st_index_interset_cont_start =sl_st_index
            # to capture the new continious groups strating position
            start_check = True
        else:
            # since the sleep stage fell into intersted keep on carry
            carry=True
            #the following portion is attached to assign the SPFF's starting position
            # first time fell into the group we have hold for starting position
            if  start_check:
                sl_st_index_interset_cont_start =sl_st_index
                # when this goes through first time with the sleep-stage interested
                                                this
                #  is turn offed
                #  to maintian the contiusly check
                start_check = False

    # if it carry only the atleast once it goes through the loop
    if  carry:
        if sleep_stages_annot_flatten[sl_st_index_interset_cont_start] in
                                                            intersted_sleep_stages
                                                :
            interseted_sleep_stage_cont_groups_start_end_index.append(
                                                    [sl_st_index_interset_cont_start,e_p])


interseted_sleep_stage_cont_groups_start_end_index = np.array(
                        interseted_sleep_stage_cont_groups_start_end_index,dtype=int)
len_interseted_sleep_stage_cont_groups = \
    interseted_sleep_stage_cont_groups_start_end_index[:,1] \
                            -interseted_sleep_stage_cont_groups_start_end_index[:,0]
return interseted_sleep_stage_cont_groups_start_end_index, \
                                    len_interseted_sleep_stage_cont_groups
```

Figure 14.6: break_cont_segments_intersetd_sleep_stages part-1

Break the fragments into further fragments based on given spindles/ outliers

if the user want to break the fragements further to avoid the interuption caused by the outliers and/or spindles. The further the fragmented as,

- ✓ If we have the **already found outliers** ("flat_MT_ch") and we consider the interuption by them via "flat_MT_consider" as True.
- ✓ If we consider the **spindles** ("break_spindle_flatten") and we consider the interuption by them via "avoid_spindle_loc" as True.
- ✓ If we consider both then both are combined by "flat_MT_ch" or "break_spindle_flatten".

```
def break_flat_MT_from_broken_sleep_st_segments_intersetd_sleep_stages(
    interseted_sleep_stage_cont_groups_start_end_index_or,
    len_interseted_sleep_stage_cont_groups_or, flat_MT_flatten):
    #  using the already broken sleep-continiuos blocks to break them further to
    #  avoid the loose-lead kind of events
    interseted_sleep_stage_cont_groups_start_end_index_cp = deepcopy(
                      interseted_sleep_stage_cont_groups_start_end_index_or)
    len_interseted_sleep_stage_cont_groups_cp = deepcopy(
                                    len_interseted_sleep_stage_cont_groups_or)
    #  then find the continious region of flat MT
    privious_annot_break_points = np.where(flat_MT_flatten>0)[0]
    if len(privious_annot_break_points)>0:
        privious_annot_break_points_diff = privious_annot_break_points[1:] \
                                           -privious_annot_break_points[0:-1]
        cont_flat_MT_t = np.where(privious_annot_break_points_diff==1)[0]
        cont_flat_MT_t2 = [[privious_annot_break_points[cont_flat_MT_t[x]],\
            privious_annot_break_points[cont_flat_MT_t[x]+1]] for x in range(0,len(
                                          cont_flat_MT_t))]

        cont_flat_MT=[]
        # this will avoid the continious flat MT there or not
        if len(cont_flat_MT_t2)>0:
            cont_flat_MT_h=cont_flat_MT_t2[0][0]
            for i in range(1,len(cont_flat_MT_t2)):
                if cont_flat_MT_t2[i][0]-cont_flat_MT_t2[i-1][1] !=0:
                    # inorder to force the index included by python 0 index
                    cont_flat_MT.append([cont_flat_MT_h,cont_flat_MT_t2[i-1][1]+1])
                    cont_flat_MT_h = cont_flat_MT_t2[i][0]

        cont_flat_MT_flatten=[]
        for cn_flat_MT_inx in range(0,len(cont_flat_MT)):
            cont_flat_MT_flatten.append(list(range(cont_flat_MT[cn_flat_MT_inx][0], \
                                                   cont_flat_MT[cn_flat_MT_inx][1])))
        cont_flat_MT_flatten = list(sum(cont_flat_MT_flatten,[]))
        #  then using the broken interested sleep-stages are further breaked
        # based on the flat_MT annotation this will avoid the continious flat MT there
                                                    or not
        cn_flat_MT_inx=0
        # to check the the continoius break points exists
        #  once we strated to check th econtinious block check the next continoiuly
                                                annotated
        #  flatten channel exists to check further in continious blocks else leave it
        cont_exists=True
        break_point_inx=0
        break_temp=[]
        for cn_idx in range(0,len(len_interseted_sleep_stage_cont_groups_cp)):

```

Calling the part in the Fig. 14.8

```
        interseted_sleep_stage_cont_groups_start_end_index = np.array(break_temp,dtype
                                                =int)
        len_interseted_sleep_stage_cont_groups = \
                interseted_sleep_stage_cont_groups_start_end_index[:,1]\
                -interseted_sleep_stage_cont_groups_start_end_index[:,0]
    else:
        interseted_sleep_stage_cont_groups_start_end_index = \
                interseted_sleep_stage_cont_groups_start_end_index_cp
        len_interseted_sleep_stage_cont_groups = \
                len_interseted_sleep_stage_cont_groups_cp
    return interseted_sleep_stage_cont_groups_start_end_index,
                                            len_interseted_sleep_stage_cont_groups

```

Figure 14.7: break_flat_MT_from_broken_sleep_st_segments_intersetd_sleep_stages part without the for-block

```
# Just assign the starting and ending position of the prepreoc
s_p = interseted_sleep_stage_cont_groups_start_end_index_cp[cn_idx][0]
e_p = interseted_sleep_stage_cont_groups_start_end_index_cp[cn_idx][1]
# just need to make sure the flat_MT is above the current satrting point
#  for increasing thats why -1
while break_point_inx<(len(privious_annot_break_points)-1) and \
                           privious_annot_break_points[break_point_inx]<s_p:
    break_point_inx+=1

if  s_p<= privious_annot_break_points[break_point_inx]<e_p:
    while  break_point_inx<len(privious_annot_break_points) and \
                s_p<= privious_annot_break_points[break_point_inx]<e_p:
        #  check the break points in the continuius flat - MT like previuosly detected
        #  loose-lead, artifact, etc.
        break_temp, break_point_inx, break_return, cont_exists, cn_flat_MT_inx, s_p= \
            helper_break_sleep_stage_flat_MT(cont_exists,privious_annot_break_points,
                                 break_point_inx,cont_flat_MT_flatten,
                                 cont_flat_MT,cn_flat_MT_inx,s_p,e_p,break_temp)
        if break_return:
            break
    # check the last privious_annot_break_points index make sure
    # boundry condition related continious sgement not missed
    if  break_point_inx>0 and s_p <=privious_annot_break_points[break_point_inx-1]+1 \
                                                                        <e_p:
        if len(break_temp)>0 and (s_p != break_temp[-1][0]):
            break_temp, break_point_inx, break_return, cont_exists, cn_flat_MT_inx,
                                            s_p \
                = helper_break_sleep_stage_flat_MT_boundry_check(cont_exists,
                    privious_annot_break_points,break_point_inx,cont_flat_MT_flatten,
                    cont_flat_MT,cn_flat_MT_inx,s_p,e_p,break_temp)
else:
    if s_p != e_p:
        break_temp.append([s_p,e_p])
```

Figure 14.8: break_flat_MT_from_broken_sleep_st_segments_intersetd_sleep_stages's for block in part-1

Helper function for further fragmentation

Since the fragments are broken into small chunks, the while loop is used. With the two helper functions, here one helper function is only used to check the boundry condition as shown in the Fig. 14.10. Which is commeneted near to the else group, which is only different from helper function shown in Fig. 14.9.

Since these python functions are lead by comments, the figures are enough to walk through them.

```
def helper_break_sleep_stage_flat_MT(cont_exists,privious_annot_break_points,
    break_point_inx,cont_flat_MT_flatten, cont_flat_MT,cn_flat_MT_inx,
    s_p,e_p,break_temp):
    break_return = False
    # to check the the continoius break points exists; once we stratied to check
    # the continious block check the next continoiuly annotated
    # flatten channel exists to check further in continious blocks else leave it
    if cont_exists  and privious_annot_break_points[break_point_inx] in
                                            cont_flat_MT_flatten:
        # if the continous block fell inside one full sleep-group break into two
        # pieces; since we know it is in continious part; we can safely increase
        # when we encounter there is no continious blocks exists with the
        # flatten annotation just turn off the cont_exists
        #  if any of the already annotation not fell in the
        # just an extra check
        while cont_exists and s_p > cont_flat_MT[cn_flat_MT_inx][0]:
            cn_flat_MT_inx+=1
            if len(cont_flat_MT)==cn_flat_MT_inx:
                cont_exists=False
        # if the size is small only we assign else that block is left
        if cont_exists:
            #  if the continious segment interfered by the given annotation
            #  break into two pieces
            if s_p <= cont_flat_MT[cn_flat_MT_inx][0]:
                break_temp.append([s_p,cont_flat_MT[cn_flat_MT_inx][0]])
            if cont_flat_MT[cn_flat_MT_inx][1] < e_p:
                #  since the next starting point will be the end of the current
                                            continiuos block
                s_p = cont_flat_MT[cn_flat_MT_inx][1]
                # then assign the hext index after the brak-point in the continiou
                                            block to check
                cn_flat_MT_inx+=1
                #   while assigning the cn_flat_MT_inx
                #  make sure the continious blocks exists
                if len(cont_flat_MT)==cn_flat_MT_inx:
                    cont_exists=False
                    #  if this is the last block just attach
                    break_temp.append([cont_flat_MT[cn_flat_MT_inx-1][1],e_p])
            else:
                #  Just assign the increase the breaking points
                if cont_exists:
                    break_point_inx = np.where(privious_annot_break_points == (
                                            cont_flat_MT[cn_flat_MT_inx][1]-1))[0][0]+1
                break_return = True
    else:
        # just need to check the region
        if  s_p <=privious_annot_break_points[break_point_inx]<e_p:
            break_temp.append([s_p,privious_annot_break_points[break_point_inx]])
            if (privious_annot_break_points[break_point_inx]+1)< e_p:
                s_p= privious_annot_break_points[break_point_inx]+1
                #  make sure the starting point not fell in the flat_MT annotation
                while (s_p in privious_annot_break_points) and s_p<e_p:
                    s_p+=1
                    break_point_inx+=1
    break_point_inx+=1
    if break_point_inx==len(privious_annot_break_points):
        # lets break the while-loop
        break_point_inx= len(privious_annot_break_points)-1
        break_return = True
    return break_temp, break_point_inx, break_return, cont_exists, cn_flat_MT_inx, s_p
```

Figure 14.9: helper_break_sleep_stage_flat_MT

```
def helper_break_sleep_stage_flat_MT_boundry_check(cont_exists,
        privious_annot_break_points,break_point_inx,
        cont_flat_MT_flatten, cont_flat_MT,cn_flat_MT_inx,
        s_p,e_p,break_temp):
    break_return = False
    if cont_exists  and privious_annot_break_points[break_point_inx] in \
                                                        cont_flat_MT_flatten:

        while cont_exists and s_p > cont_flat_MT[cn_flat_MT_inx][0]:
            cn_flat_MT_inx+=1
            if len(cont_flat_MT)==cn_flat_MT_inx:
                cont_exists=False
        if cont_exists:
            if s_p <= cont_flat_MT[cn_flat_MT_inx][0]:
                break_temp.append([s_p,cont_flat_MT[cn_flat_MT_inx][0]])

            if cont_flat_MT[cn_flat_MT_inx][1] < e_p:
                s_p = cont_flat_MT[cn_flat_MT_inx][1]
                cn_flat_MT_inx+=1
                if len(cont_flat_MT)==cn_flat_MT_inx:
                    cont_exists=False
                    break_temp.append([cont_flat_MT[cn_flat_MT_inx-1][1],e_p])
            else:
                if cont_exists:
                    break_point_inx = np.where(privious_annot_break_points == (
                                    cont_flat_MT[cn_flat_MT_inx][1]-1))[0][0]+1
                break_return = True
    else:
        #this is the portion different for boundry condition
        if  s_p <=privious_annot_break_points[break_point_inx]+1<e_p:
            break_temp.append([s_p,e_p])

    break_point_inx+=1
    if break_point_inx==len(privious_annot_break_points):
        break_point_inx= len(privious_annot_break_points)-1
        break_return = True

    return break_temp, break_point_inx, break_return, cont_exists, cn_flat_MT_inx, s_p
```

Figure 14.10: helper_break_sleep_stage_flat_MT_boundry_check fucntion only check the boundry condition such that only the commented block in the else block is different from the Fig. 14.9.

14.3 Sanity check of fragments

This is for boundry conditions and small portions comes via the fragmentation, need to be explored more.

```
def check_the_break_segment_len(interseted_sleep_stage_cont_groups_start_end_index,
                                    len_interseted_sleep_stage_cont_groups):
    #  sanity check the size of the break segmentation
    len_interseted_sleep_stage_cont_groups_new =[]
    interseted_sleep_stage_cont_groups_start_end_index_new= []
    for cn_indx in range(0,len(len_interseted_sleep_stage_cont_groups)):
        if len_interseted_sleep_stage_cont_groups[cn_indx]>0:
            interseted_sleep_stage_cont_groups_start_end_index_new.append(
                interseted_sleep_stage_cont_groups_start_end_index[cn_indx,:])
            len_interseted_sleep_stage_cont_groups_new.append(
                        len_interseted_sleep_stage_cont_groups[cn_indx])

    return interseted_sleep_stage_cont_groups_start_end_index_new,
                               len_interseted_sleep_stage_cont_groups_new
```

Figure 14.11: Sanity check for the fragments are performed correctly

14.4 Obtaining the medians

To sort the elements is continious blocks in the moving window corr_given: is the correlation elements of the different channels either this can be obtained from Z-mapped or the direct values

moving_window_size= 60

the algorithm is implemented efficiently to moving window sorting once the moving window started since we are sliding in one index only that element is added to assign the new median

Take the group of values and sort them

when the len_grp_size - moving_window_size will be the size of above window_size; due to once sec advance ovet the moving window size

grp: is used interchangely as channel combination for grp (optional) or for average correlation this is used to slected the specific channel

Direct correlation in signle vector can avoid the need of providing the grp information for selection

These following function is going to detect the moving window based medians for approch-1 (means for the given grp) as shown in the Fig. **??**.

```
def median_with_moving_window_in_broken_cont_groups(corr_given, grp,
    interseted_sleep_stage_cont_groups_start_end_index,
                                            len_interseted_sleep_stage_cont_groups,
    moving_window_size=60):
    # if the length is less than the moving window
    # For the default value window with size 60
    #      then the median is assigned only from that epoch/ two epochs like
    #        27 sec's or 57 secs else the moving window is used till the staring to
                                            end of the continious blocks
    sorted_median_cont_grp =[]
    for cn_indx in range(0,len(len_interseted_sleep_stage_cont_groups)):
        sorted_median_cont_grp_t=[]
        #  getting the starting and ending position of interested continious group
        s_p_intial = interseted_sleep_stage_cont_groups_start_end_index[cn_indx][0]
        e_p_intial = interseted_sleep_stage_cont_groups_start_end_index[cn_indx][1]
        # modified to accept correlation coefficient maps or single correlation
                                            coefficient maps
        # even though the correlation are fed sperately by the main functions
        # user can provide the input via the grp or directly
        if len(np.shape(corr_given))==3:
            #  For approach 1 with all the correlation groups together
            group_check_intial = corr_given[s_p_intial:e_p_intial,grp[0][0],grp[0][1]]
        elif len(np.shape(corr_given))==2:
            #  For approach 2 with all the mean of the channels together
            # feed the channel information via the group
            group_check_intial = corr_given[s_p_intial:e_p_intial,grp]
            if np.isnan(group_check_intial).all():
                logger.debug('s_p_intial %i:',s_p_intial)
                logger.debug('e_p_intial %i:',e_p_intial)
        else:
            group_check_intial = corr_given[s_p_intial:e_p_intial]

        s_p = 0
        if len_interseted_sleep_stage_cont_groups[cn_indx]<=moving_window_size:
            e_p = e_p_intial - s_p_intial
        else:
            e_p = moving_window_size

        logger.debug('s_p %i',s_p)

        # select the portion to slide the moving window
        group_check = deepcopy(group_check_intial[s_p:e_p])

        # do the first time sorting
        # find the median value from the sorted list
        goup_sorted = np.sort(group_check,kind='quicksort')
        sorted_median_cont_grp_t.append(deepcopy(np.median(goup_sorted)))
```

Figure 14.12: median_with_moving_window_in_broken_cont_groups part-1

```
# the folowing while loop can be presented as for loop but need to have litlle more
                                                variables
# with
# if e_p< len_interseted_sleep_stage_cont_groups[cn_indx]:

# to adjust the while statement e_p condition

e_p = e_p+1
while e_p < len_interseted_sleep_stage_cont_groups[cn_indx]:
    # remove the element that matches from the sorted list
    # we need to remove the s_p related elements and
    for indx in range(0,len(goup_sorted)):
        if goup_sorted[indx] == group_check_intial[s_p]:
            logger.debug('indx %i:',indx)
            break

    s_p = s_p+1
    logger.debug("goup_sorted are: {}".format(' '.join(map(str, goup_sorted))))
    goup_sorted = np.delete(goup_sorted, [indx])
    logger.debug("goup_sorted are: {}".format(' '.join(map(str, goup_sorted))))

    # since the sorting list already avaliable just do the part of insertion sort with
    # pivot element as the new in the group
    pivot_ele = group_check_intial[e_p-1]
    for indx in range(0,len(goup_sorted)):
        # to accomadate the python indexing
        if goup_sorted[indx]<=pivot_ele:
            break
    goup_sorted = np.insert(goup_sorted, indx, pivot_ele)

    sorted_median_cont_grp_t.append(deepcopy(np.median(goup_sorted)))

    e_p=e_p+1

sorted_median_cont_grp.append(deepcopy(sorted_median_cont_grp_t))

return sorted_median_cont_grp
```

Figure 14.13: median_with_moving_window_in_broken_cont_groups part-2

14.5 Local moving window detection

The function shown in Fig. 14.14 detect the outliers based on the local moving wibdow based outlier detection.

```
def local_outlier_detection(corr_given, grp, sorted_median_cont_grp,
    interseted_sleep_stage_cont_groups_start_end_index,
    len_interseted_sleep_stage_cont_groups,
    arousal_annot, moving_window_size=60, th_fact = 1.5):
    '''
    local outlier detection

    annotate arousal based on the sliding medians
    six threhold conditions evaluated 1996 paper
    such factors as
    40,10,4,3,2, 1.5
    but here we check the factor with smmaler values that can be inverse of these
                                            factors

    these factors are applied seperately for each continious segments seperately

    the values prsent in the continious segmets beggining part
      less than the moving_vindow is checked with only the first median value
      (consider as belonging to the first moving window)
    '''

    for cn_indx in range(0,len(len_interseted_sleep_stage_cont_groups)):

        s_p_intial = interseted_sleep_stage_cont_groups_start_end_index[cn_indx][0]
        e_p_intial = interseted_sleep_stage_cont_groups_start_end_index[cn_indx][1]

        sorted_median_cont_grp_t = np.array(sorted_median_cont_grp[cn_indx])/th_fact

        '''
        modified to accept correlation coefficient maps or single correlation
                                            coefficient maps
        '''
        if len(np.shape(corr_given))==3:
            group_check_intial = corr_given[s_p_intial:e_p_intial,grp[0][0],grp[0][1]]
        elif len(np.shape(corr_given))==2:
            group_check_intial = corr_given[s_p_intial:e_p_intial,grp]
        else:
            group_check_intial = corr_given[s_p_intial:e_p_intial]


        s_p = 0
        if len_interseted_sleep_stage_cont_groups[cn_indx]<=moving_window_size:
            e_p = e_p_intial - s_p_intial
        else:
            e_p = moving_window_size


        for chk_val_indx in range(s_p,e_p):
            if group_check_intial[chk_val_indx] < sorted_median_cont_grp_t[0]:
                arousal_annot[chk_val_indx+s_p_intial]= 1


        if e_p< len_interseted_sleep_stage_cont_groups[cn_indx]:
            for s_p in range(1,len_interseted_sleep_stage_cont_groups[cn_indx]-
                                            moving_window_size):
                chk_val_indx = e_p +s_p
                if group_check_intial[chk_val_indx] < sorted_median_cont_grp_t[s_p]:
                    arousal_annot[chk_val_indx+s_p_intial]= 1

    return arousal_annot
```

Figure 14.14: local_outlier_detection function

14.6 local and global outlier detection

Annotate arousal based on the sliding medians six threhold conditions evaluated paper [5] such factors as 40,10,4,3,2, 1.5 but here we check the factor with smaler values that can be inverse of these factors these factors are applied seperately for each continious segments seperately the values present in the continious segmets beggining part less than the moving_window is checked with only the first median value (consider as belonging to the first moving window)

- ✓ sorted_median_cont_grp_comp_sort_median_cond=[True, 10] these conditions give an option whether to consider the threhold or not with their corresponding factor

✓ sorted_median_cont_grp_comp_sort$_{q}uan = [True, 10, 0.75]thelastoneisthequantilevaluetodecidethemainvaluef$

Define global threshold different of ways since mean is interfered by the outliers, it can be done in two main ways like how we use the samples to decide the outliers,

1. local_and_global_based_outlier_detection_with_loc_good: after removing the outliers using the good-pool of values to decide the threshold.
2. local_and_global_based_outlier_detection_with_all: instead of good-pool here we are using local-moving window distribution to decide the global threhold such that global-detection can be parally run with local-moving-window.

Appendix A

moving_window_based_artifact_vin1

Presentation Apr-20-2023 regadring the different combinations with the distributions in the bottom. May be inclue like minipage next to each to show as in the presentation.

Showing the outcome of the local moving choice effect in the distribution

Include the index based results.

Appendix B

Comparison between different ways on Approach-2 with the six channel's prediction on the selected subject plots

Approach-2 is based on the mean correlation. Please check the chapter "Methods and ideas with the package" (2) for more details. Especially check the section "Correlation based approaches for outlier detection" (2.5).

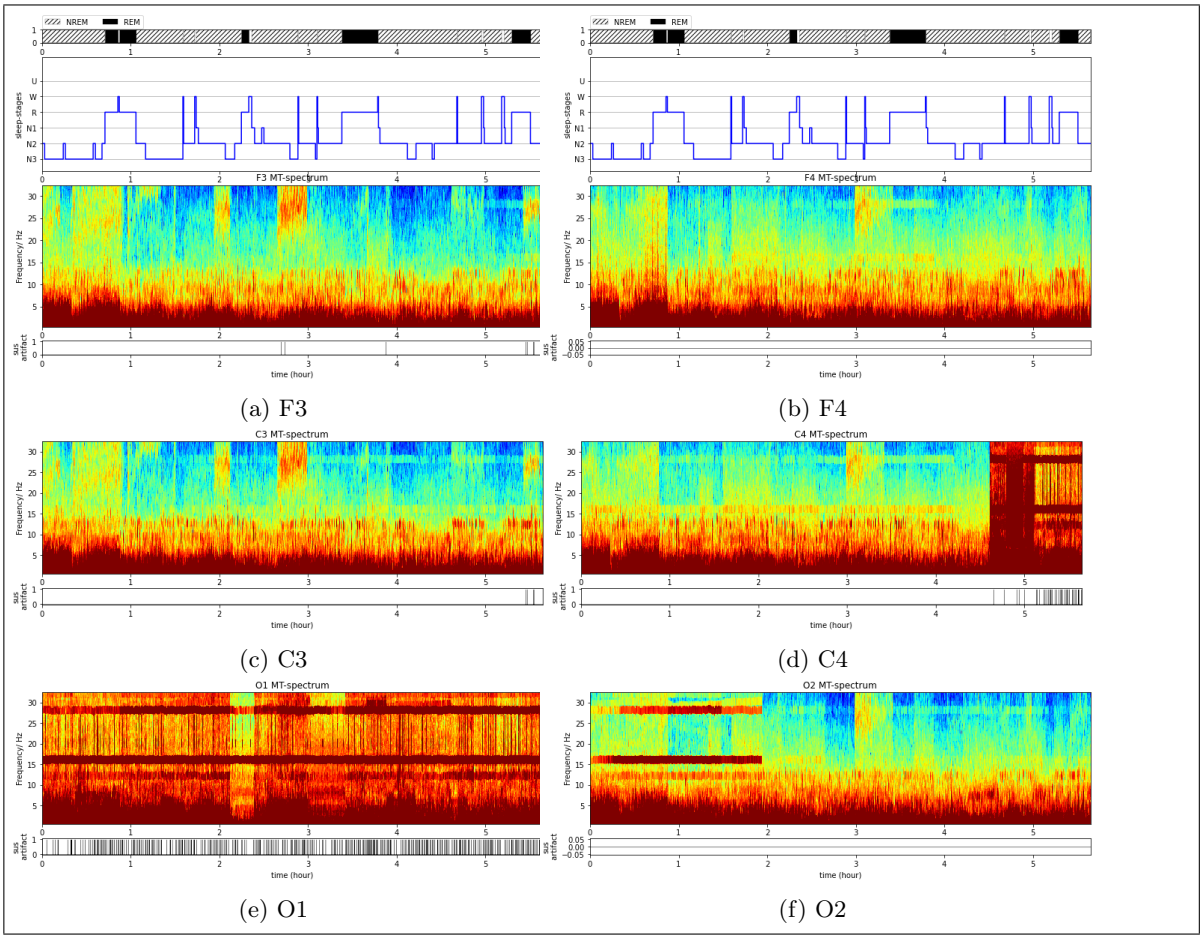
B.1 Moving window based results

B.1.1 Comparison between different finishing conditions with convolution

The following Figures present the outcome while using the moving-window (local and global) based method, while only changing the last finalisation condition's parameter effect the finalised outcome as,

- ✓ Fig. B.1 as direct outcome.
- ✓ Fig. B.2 as convolutional window as two minutes to unify the outcome.
- ✓ Fig. B.3 as convolutional window as five minutes to unify the outcome.

As it is clearly visible the convolution(Way_{sum}**-finalisation**) as mentioned in the section. "Pinpoint loose-lead present" (2.5.9). based finalisation just increase the region of outliers. The convolution window size used for finalisation stated in the caption of the results.

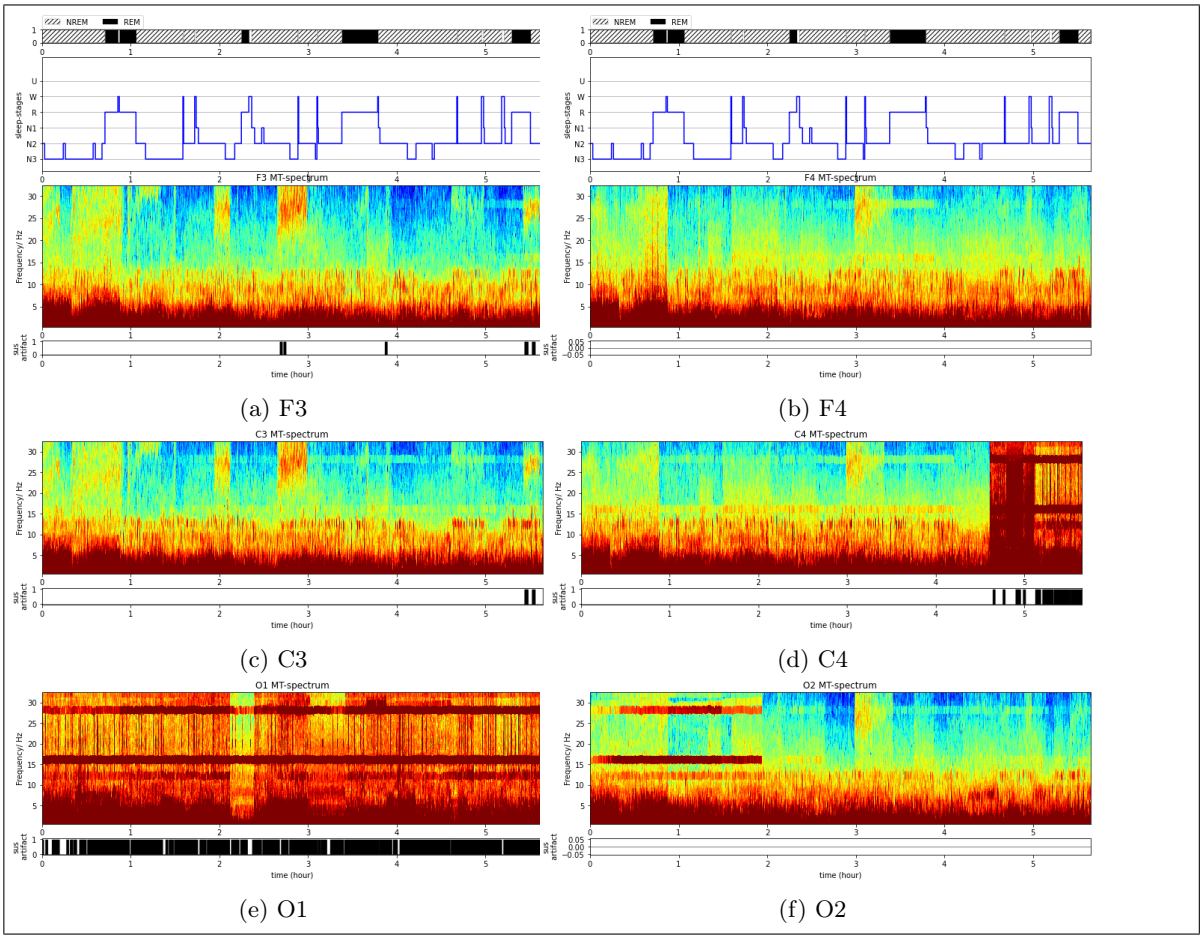


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.1: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels Z-transformed correlation coefficients based on with moving window with 60 and threshold factor 2 with global.

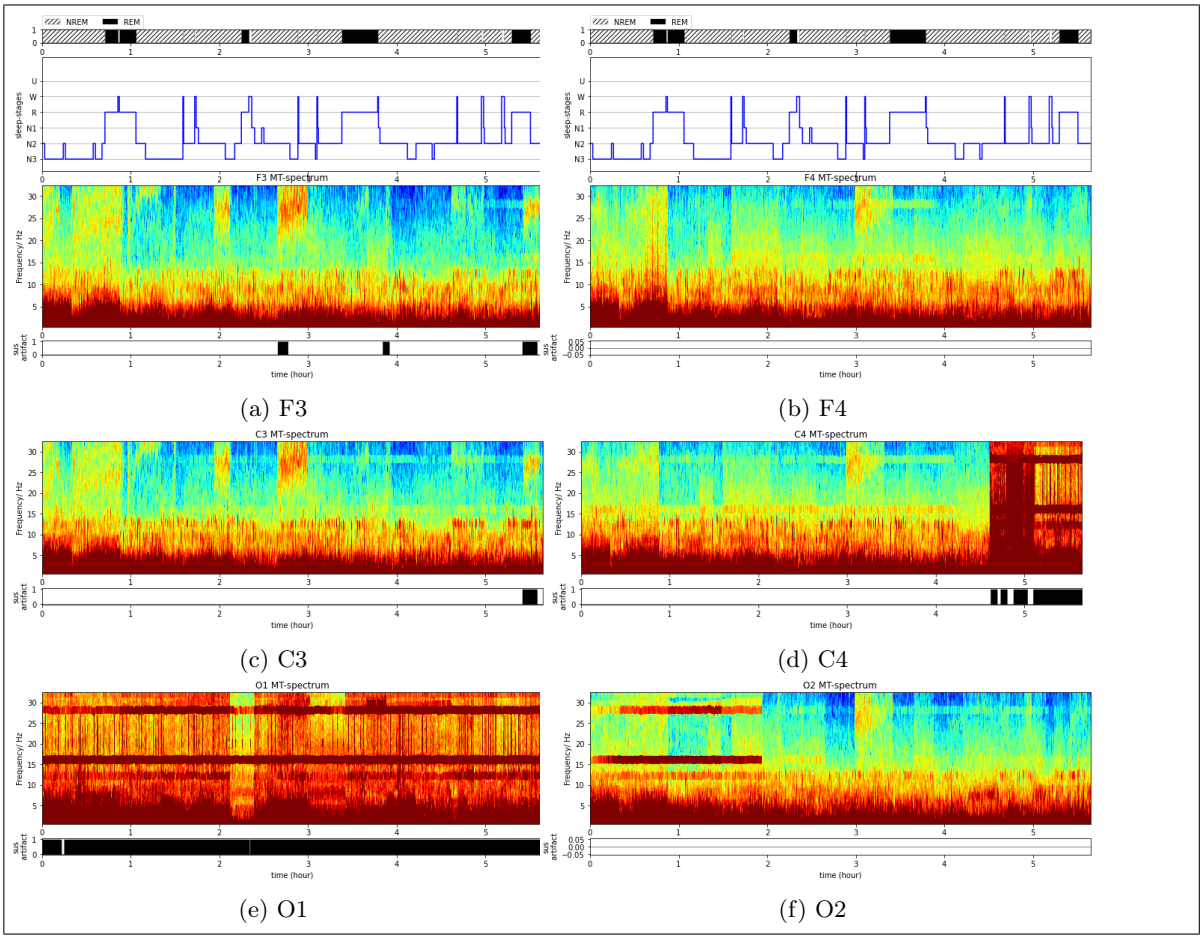


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.2: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels Z-transformed correlation coefficients based on with moving window with 60 and threshold factor 2 with global.with convolutional window ≈2 minutes

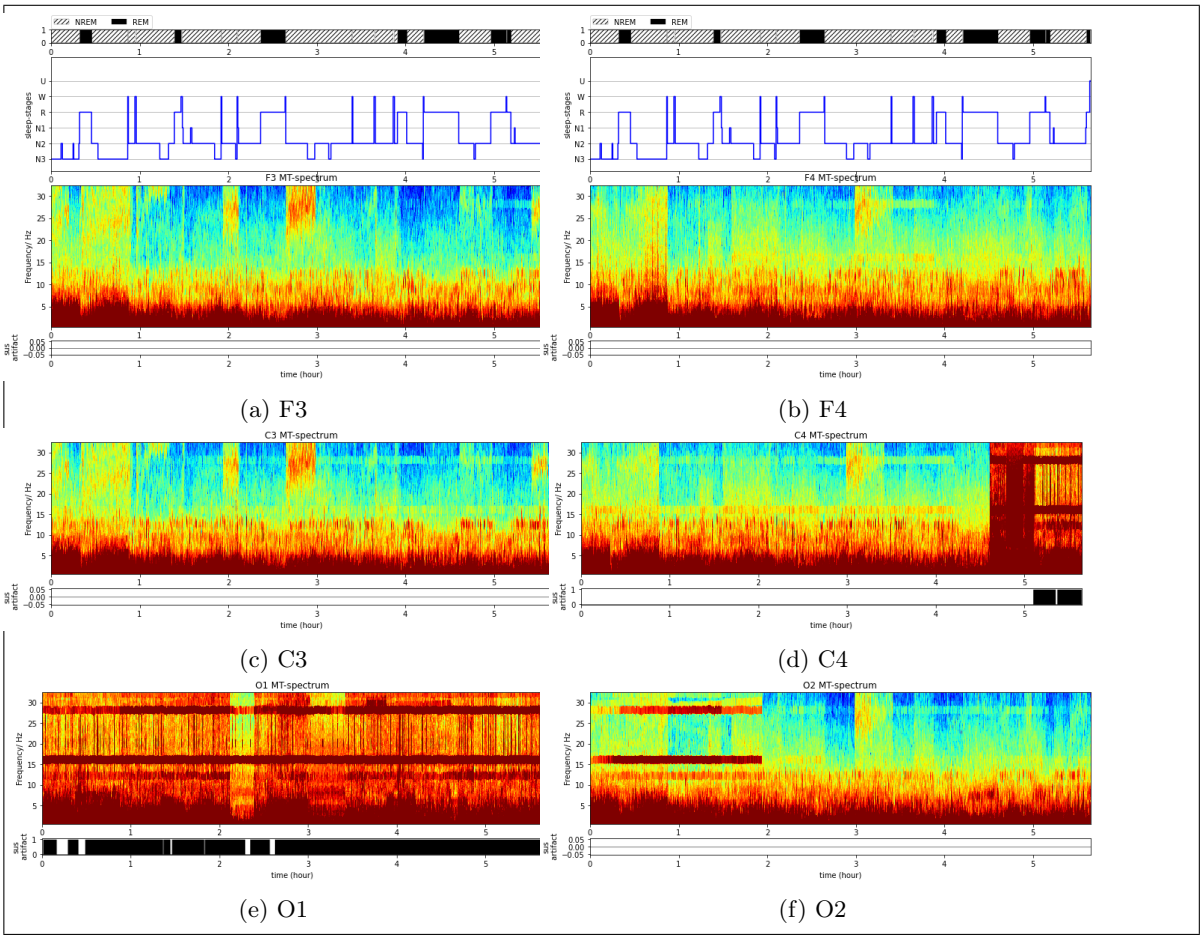


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.3: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels Z-transformed correlation coefficients based on with moving window with 60 and threshold factor 2 with global.with convolutional window ≈5 minutes



(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.4: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels raw-correlation coefficients based on convolutional window ≈5 minutes

As shown in the Figures with the Z-transformation vs the raw-correlation as in Fig. B.3 vs Fig. B.4 accordingly. The raw-correlation is enough to full fills the purpose.

B.2 Distributions with same Z-tranformationed correlation with local and global condition just changes in different sleep-stages distribution presentation by the Z-tranformation vs raw correlation

These distributions are presented to show the how the Z-transformed points (including outliers, etc.), mapped from the raw-correlation to Z-transformed correlation.

Such that all these following Figures caption contain the same part apart from **bolded choices with the "/"**.

All these on selected **Raw/ Z-mapped** correlation-coefficients groups channels with MT based spectrum **Z-mapped** correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the **NREM/ REM** full fragments (with-

out any conditions) to obtain the distribution of median; then the outliers obtained from local moving window and global conditions based on good-locals such as 75^{th} percentile(10) of pooled median of moving window on **NREM/ REM** fragments to oabtain outliers and the outliers are emphasized in red.

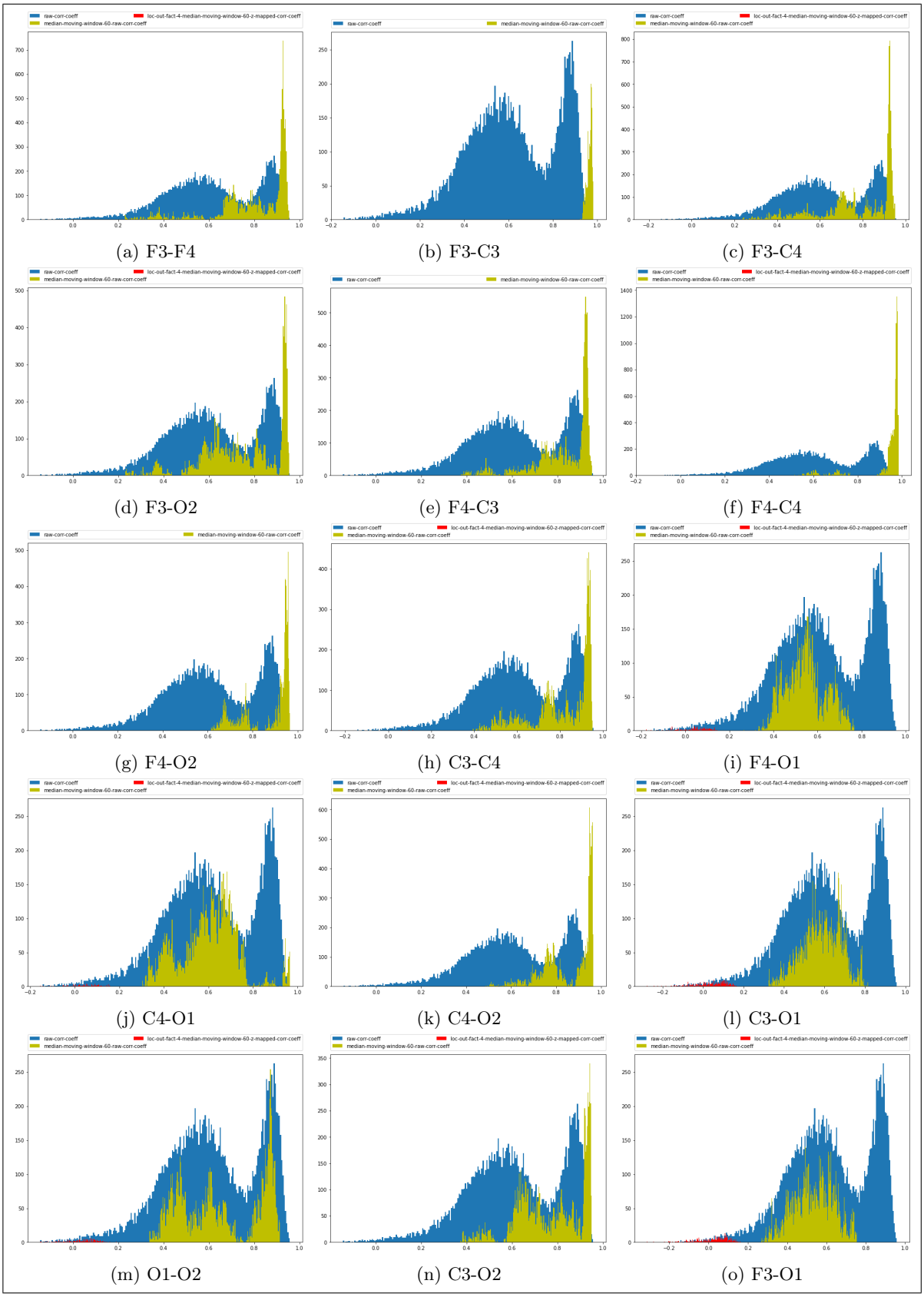
Such that the full Figure caption of,

- ✓ Fig. B.5 **Raw correlation-coefficient** on selected groups channels with MT based spectrum **Z-mapped** correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the **NREM** full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window and global conditions based on good-locals such as 75^{th} percentile(10) of pooled median of moving window on **NREM** fragments to oabtain outliers and the outliers are emphasized in red.

- ✓ Fig. B.6 **Z-mapped** correlation-coefficient on selected groups channels with MT based spectrum **Z-mapped** correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the **NREM** full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window and global conditions based on good-locals such as 75^{th} percentile(10) of pooled median of moving window on **NREM** fragments to oabtain outliers and the outliers are emphasized in red.

- ✓ Fig. B.7 **Raw** correlation-coefficient on selected groups channels with MT based spectrum **Z-mapped** correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obtain the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the **REM** full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window and global conditions based on good-locals such as 75^{th} percentile(10) of pooled median of moving window on **REM** fragments to oabtain outliers and the outliers are emphasized in red.

- ✓ Fig. B.8 **Z-mapped** correlation-coefficient on selected groups channels with MT based spectrum **Z-mapped** correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the **REM** full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window and global conditions based on good-locals such as 75^{th} percentile(10) of pooled median of moving window on **REM** fragments to oabtain outliers and the outliers are emphasized in red.



(a) F3-F4 (b) F3-C3 (c) F3-C4

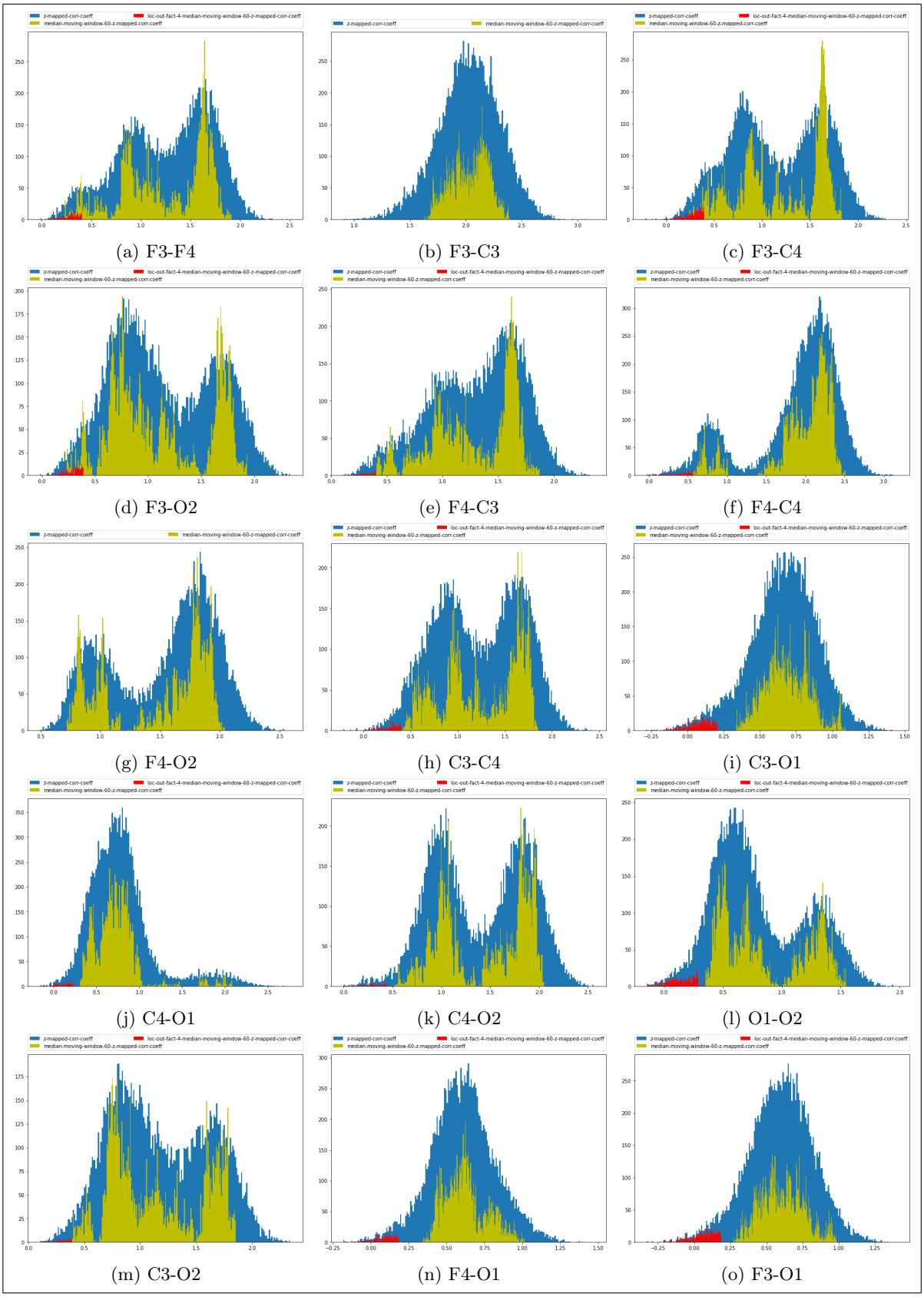
(d) F3-O2 (e) F4-C3 (f) F4-C4

(g) F4-O2 (h) C3-C4 (i) F4-O1

(j) C4-O1 (k) C4-O2 (l) C3-O1

(m) O1-O2 (n) C3-O2 (o) F3-O1

Figure B.5: Raw correlation-coefficient on selected groups channels on NREM fragments on subject 21-0995_F_9.7_1_di



(a) F3-F4 (b) F3-C3 (c) F3-C4

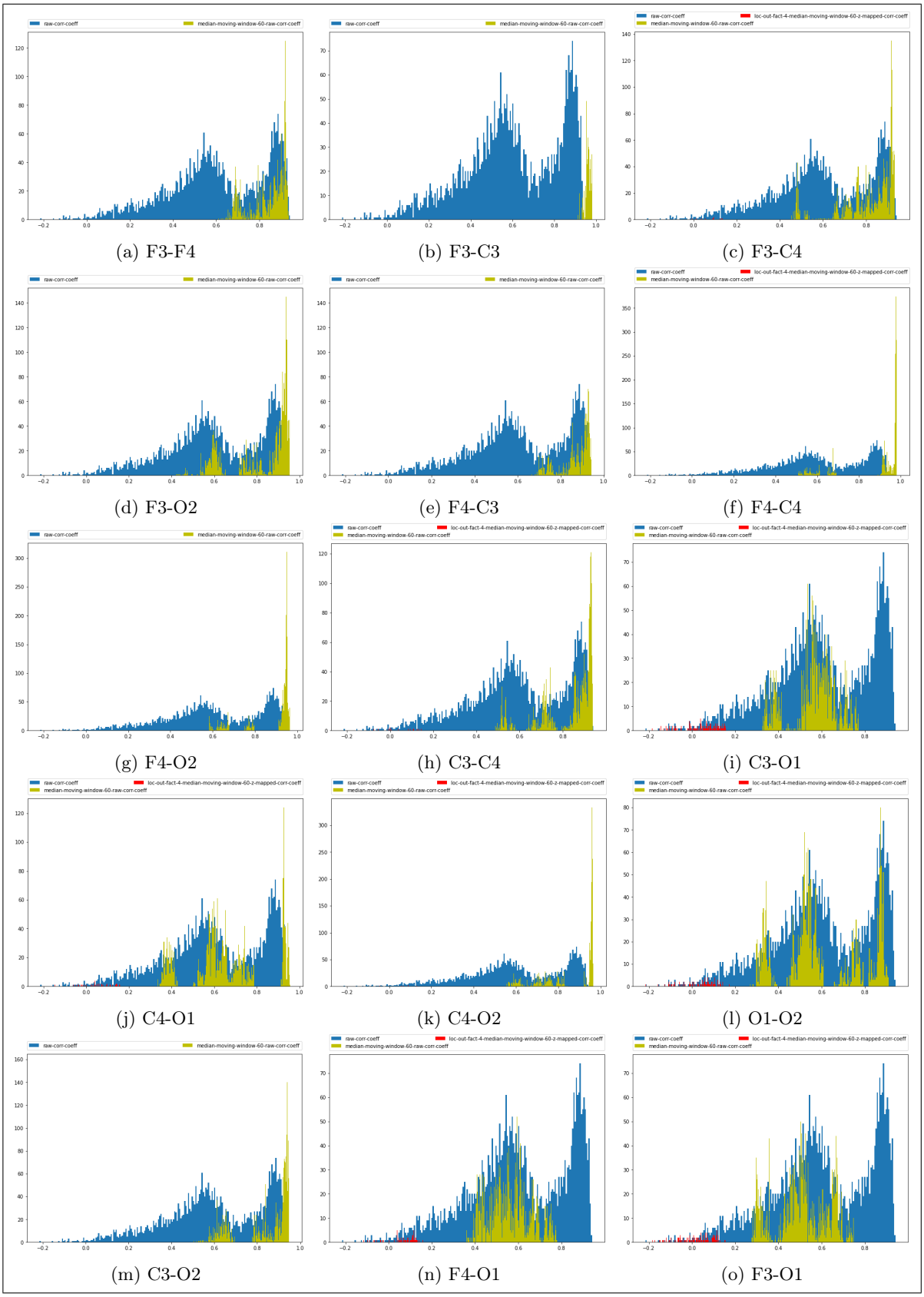
(d) F3-O2 (e) F4-C3 (f) F4-C4

(g) F4-O2 (h) C3-C4 (i) C3-O1

(j) C4-O1 (k) C4-O2 (l) O1-O2

(m) C3-O2 (n) F4-O1 (o) F3-O1

Figure B.6: Z-mapped correlation-coefficient on selected groups channels on NREM fragments on subject 21-0995_F_9.7_1_di



(a) F3-F4 (b) F3-C3 (c) F3-C4

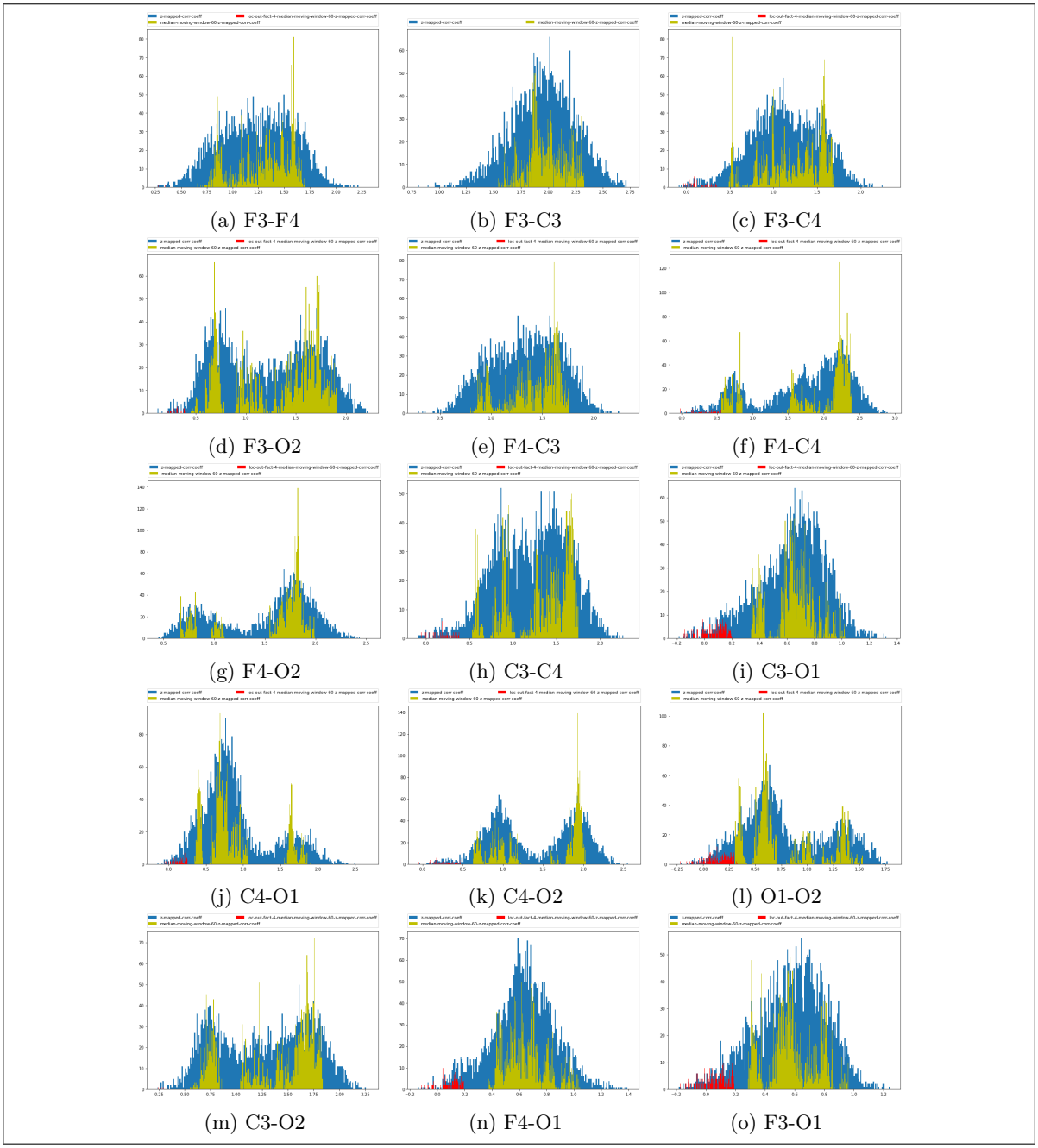
(d) F3-O2 (e) F4-C3 (f) F4-C4

(g) F4-O2 (h) C3-C4 (i) C3-O1

(j) C4-O1 (k) C4-O2 (l) O1-O2

(m) C3-O2 (n) F4-O1 (o) F3-O1

Figure B.7: Raw correlation-coefficient on selected groups channels on REM fragments on subject 21-0995_F_9.7_1_di



(a) F3-F4 (b) F3-C3 (c) F3-C4

(d) F3-O2 (e) F4-C3 (f) F4-C4

(g) F4-O2 (h) C3-C4 (i) C3-O1

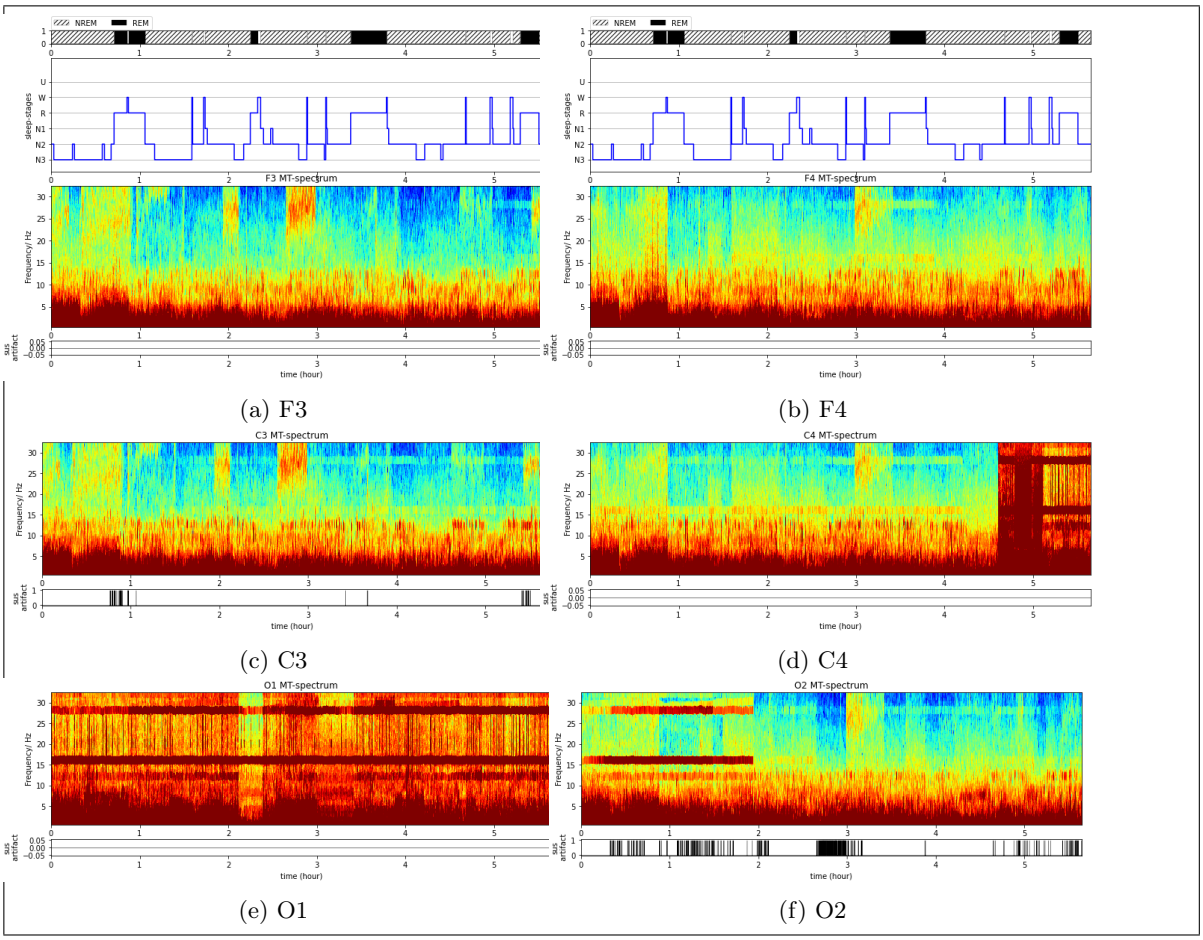
(j) C4-O1 (k) C4-O2 (l) O1-O2

(m) C3-O2 (n) F4-O1 (o) F3-O1

Figure B.8: Z-mapped correlation-coefficient on selected groups channels on REM fragments on subject 21-0995_F_9.7_1_di

B.3 Distribution based results

Here the results are obatained with the 10 minutes convolutinal window.

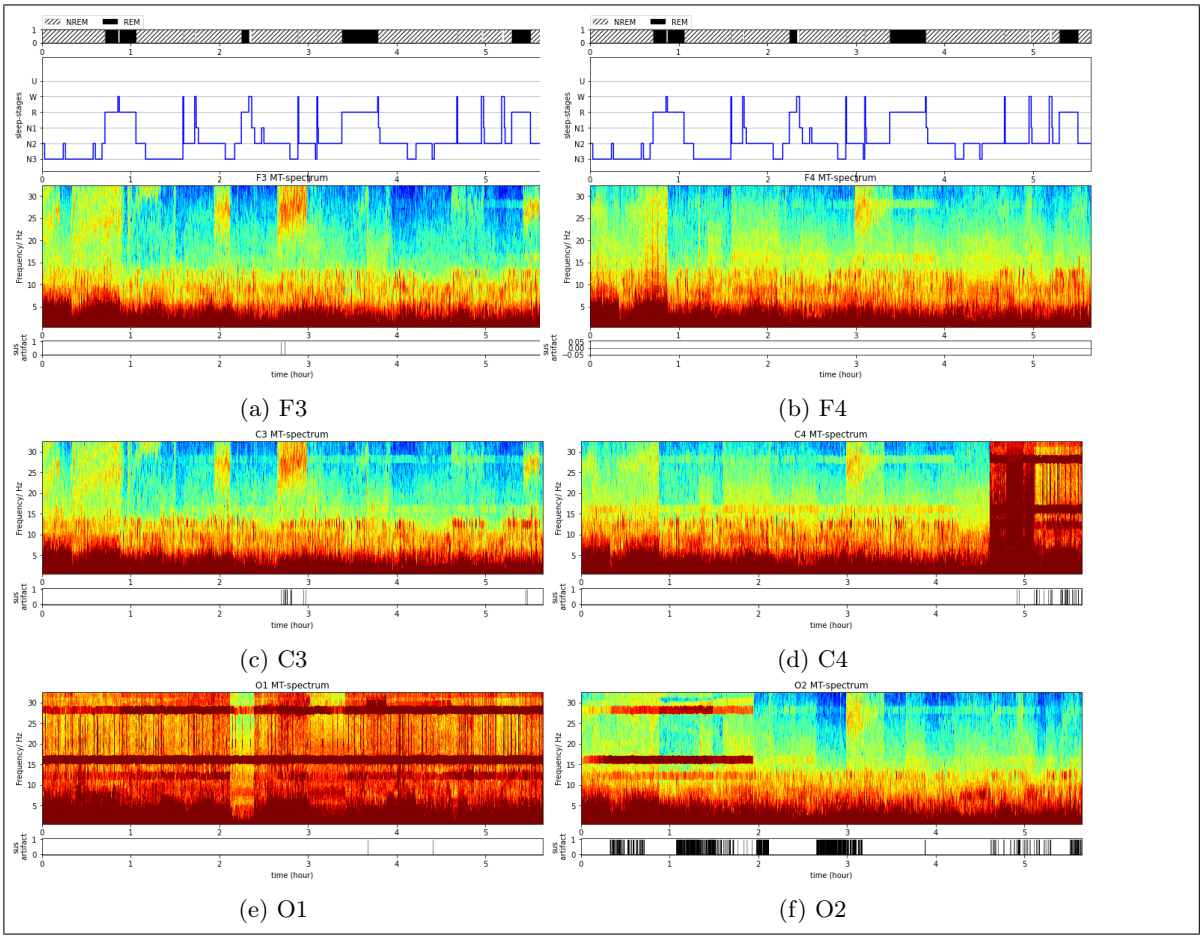


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.9: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients based on with z-transform distribution with the probability of 0.01 while combining with 10 minute convolutional window

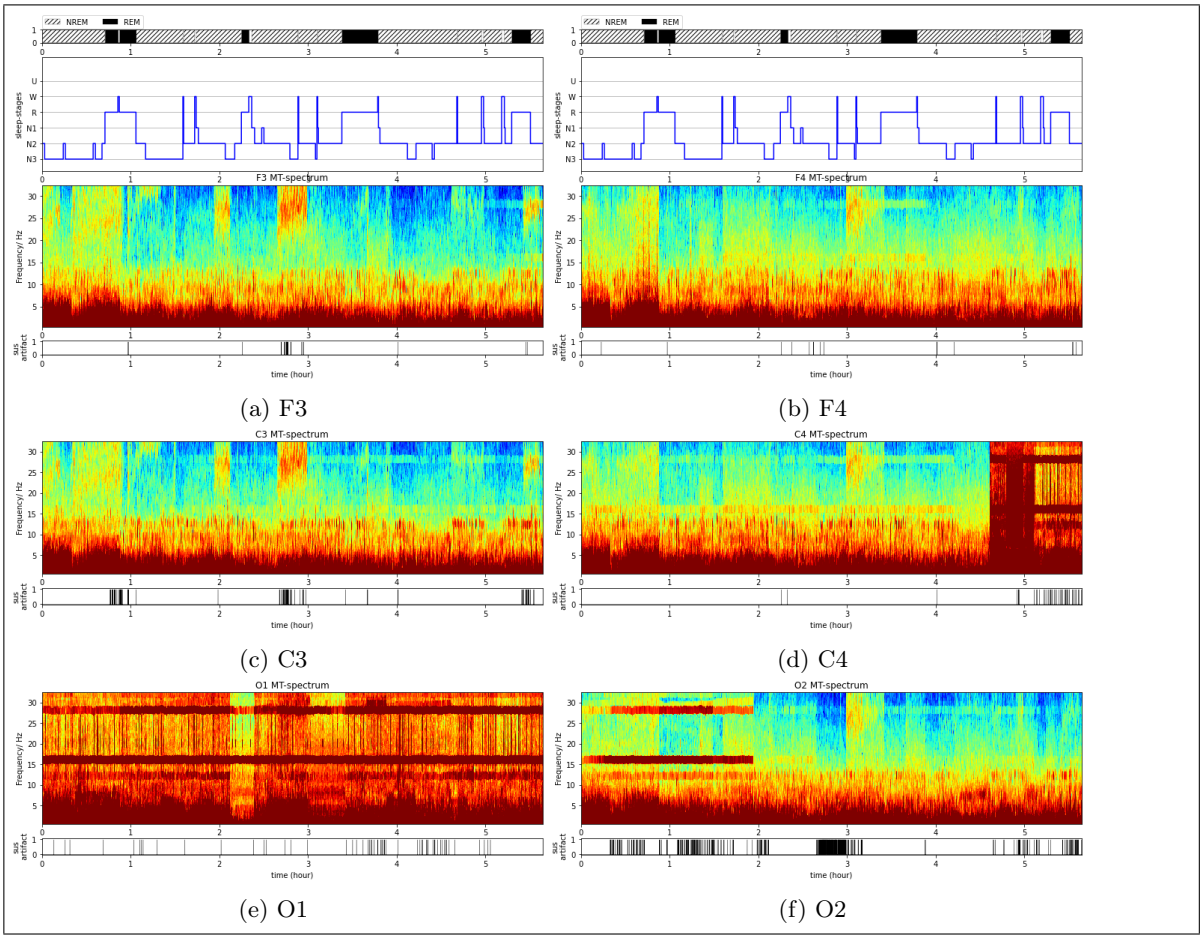


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.10: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels raw correlation coefficients based distribution with the probability of 0.01 while combining with 10 minute convolutional window

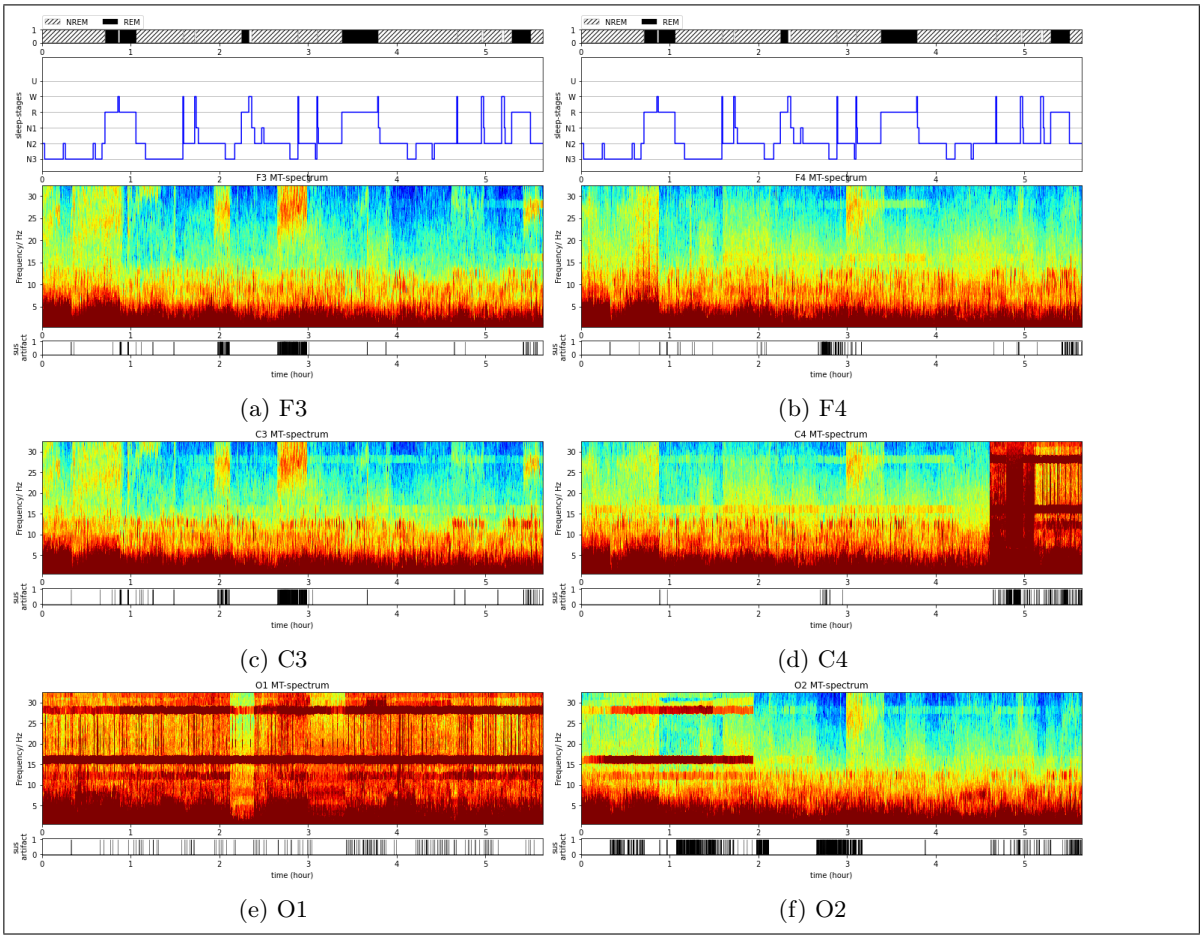


(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

Figure B.11: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients based on with z-transform distribution with 1.5



(a) F3 (b) F4

(c) C3 (d) C4

(e) O1 (f) O2

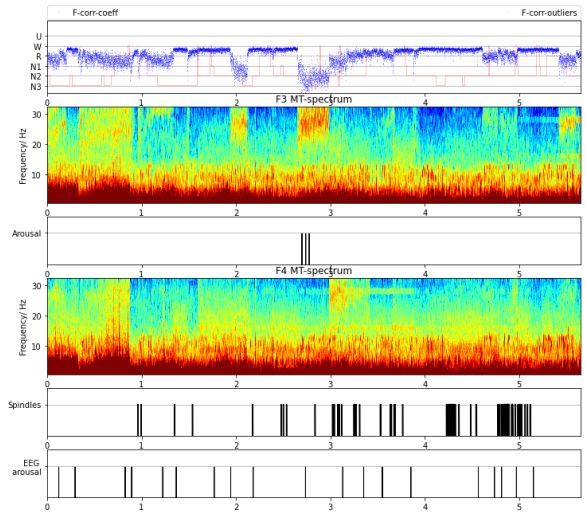
Figure B.12: $21-0995\_F\_9.7\_1\_di$ subject's final channels suspected artifacts detected via the mean of the channels correlation coefficients based on with distribution with 1.5

B.4 Local vs Global threshold on Moving window difference

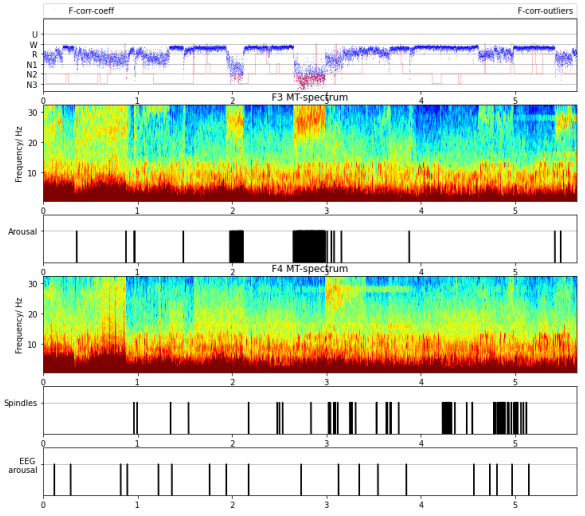
The seven correlation combinations (*corr*) are shown in Figures such as,

- ✓ F3-F4 shown in Fig. B.14
- ✓ C3-C4 shown in Fig. B.16
- ✓ F3-C3 shown in Fig. B.20
- ✓ F4-C4 shown in Fig. B.22
- ✓ C3-O1 shown in Fig. B.24
- ✓ C4-O2 shown in Fig. B.26
- ✓ O1-O2 shown in Fig. B.18

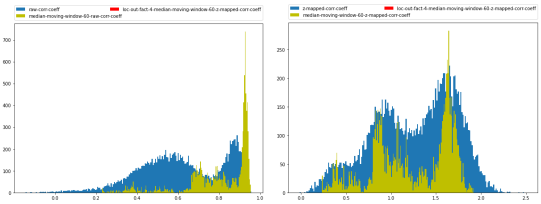
Where it clearly seen the global outlier annotation along with the local-moving window annotate more outliers; such that left-side vs right-side coloumn.



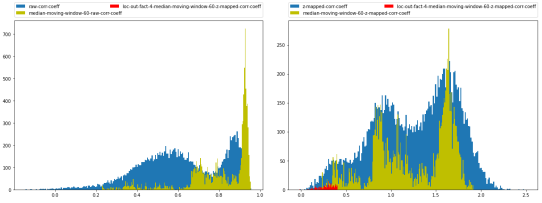
(a) Selected reref onside F3-F4 groups channels with local only outliers.



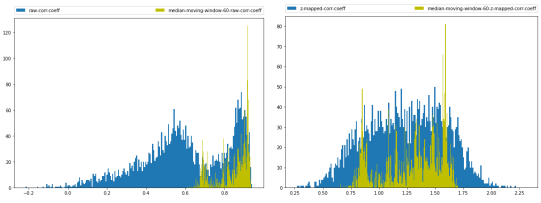
(a) Selected reref onside F3-F4 groups channels with local and global outliers.



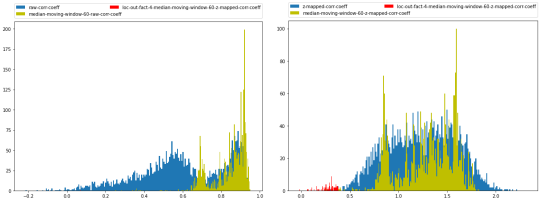
(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

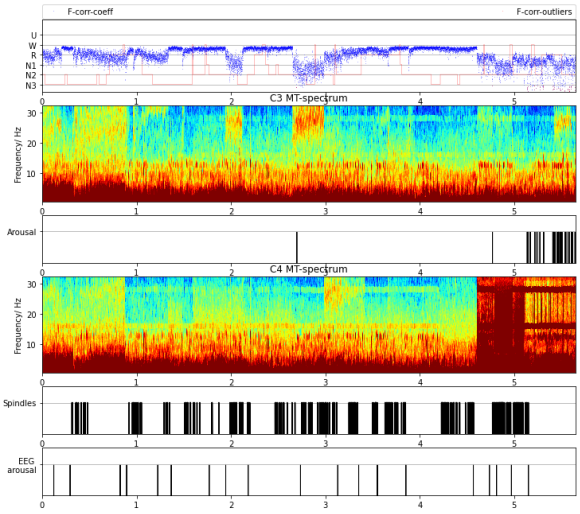


(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

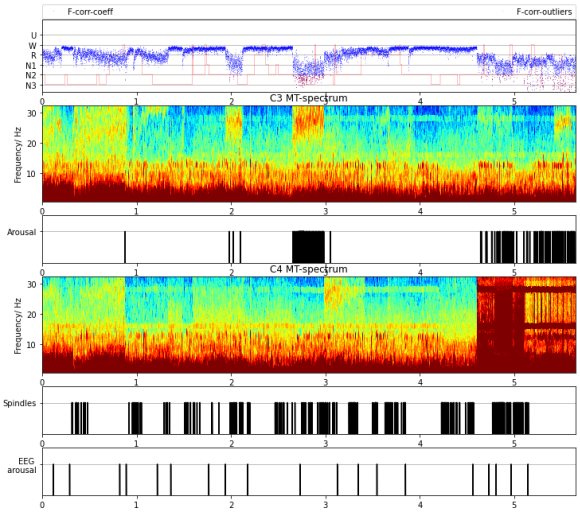
(B.14.1) with only moving window 60 based.

(B.14.2) local moving window with global based.

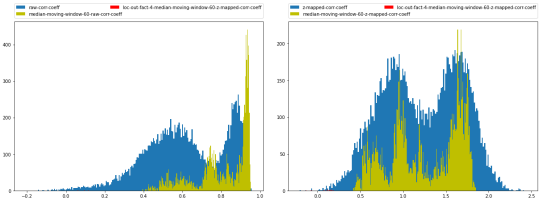
Figure B.14: F3-F4 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.



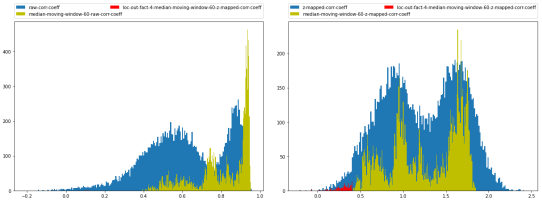
(a) Selected reref onside C3-C4 groups channels with local only outliers.



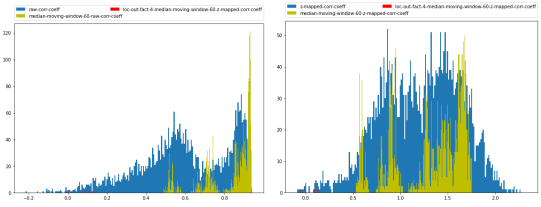
(a) Selected reref onside C3-C4 groups channels with local and global outliers.



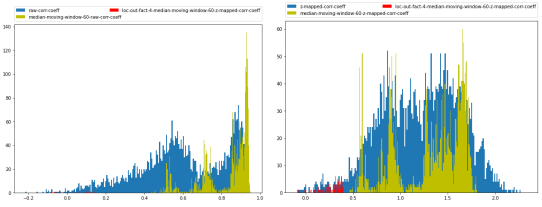
(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

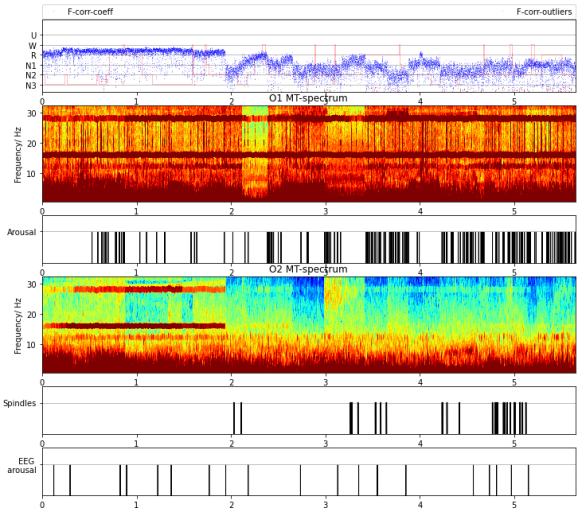


(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

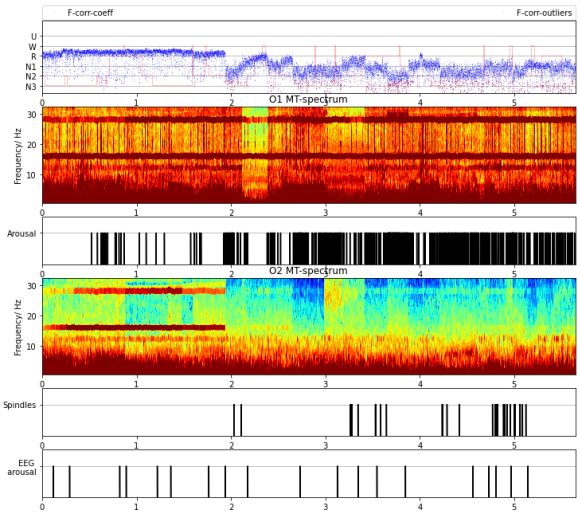
(B.16.1) with only moving window 60 based.

(B.16.2) local moving window with global based.

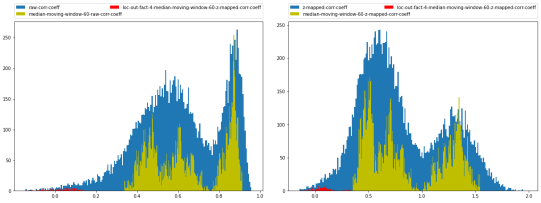
Figure B.16: C3-C4 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.



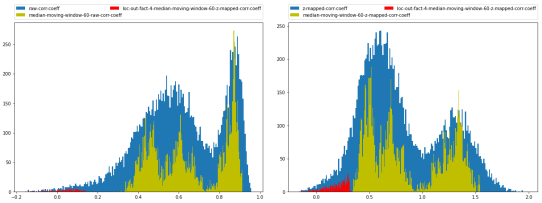
(a) Selected reref onside O1-O2 groups channels with local only outliers.



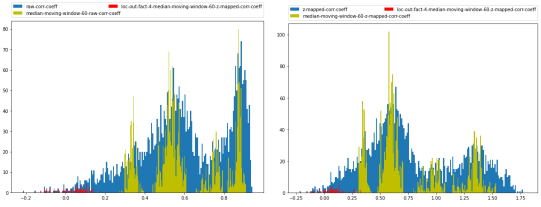
(a) Selected reref onside O1-O2 groups channels with local and global outliers.



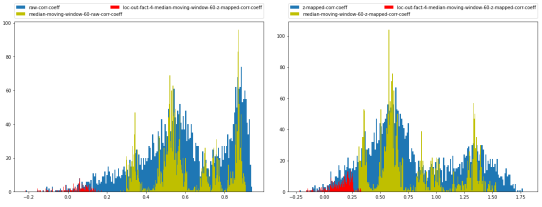
(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

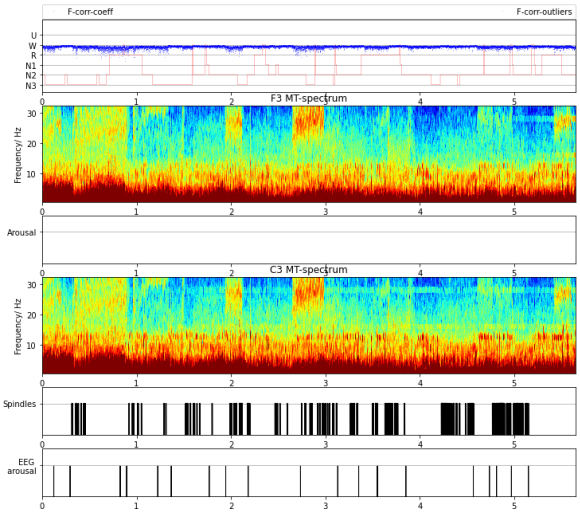


(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

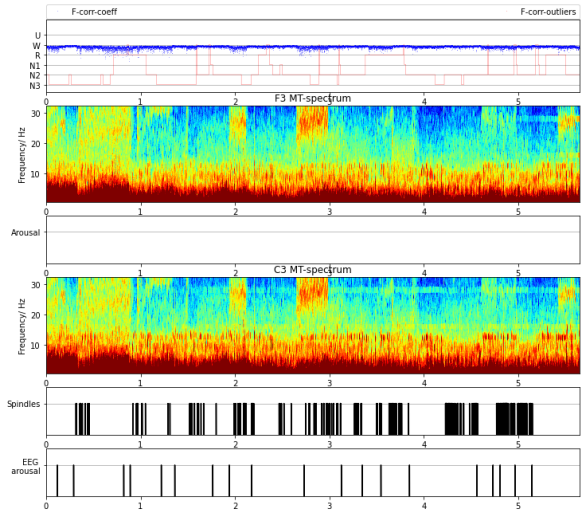
(B.18.1) with only moving window 60 based.

(B.18.2) local moving window with global based.

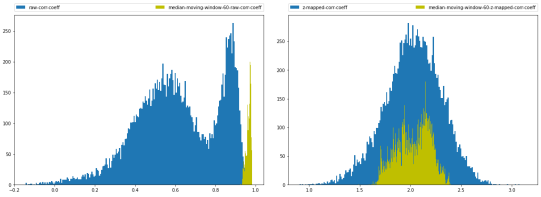
Figure B.18: O1-O2 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.



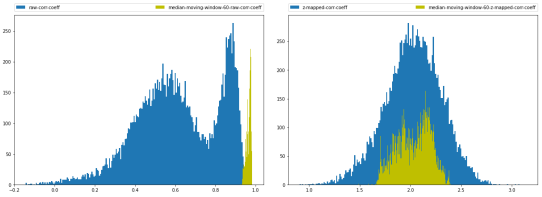
(a) Selected reref onside F3-C3 groups channels with local only outliers.



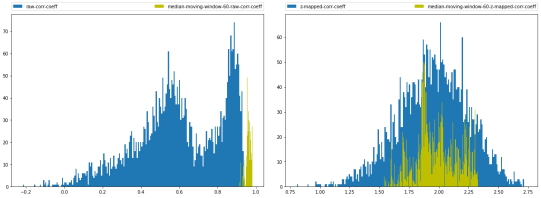
(a) Selected reref onside F3-C3 groups channels with local and global outliers.



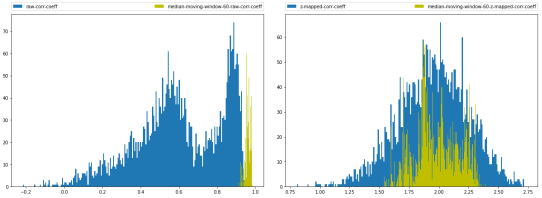
(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM



(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

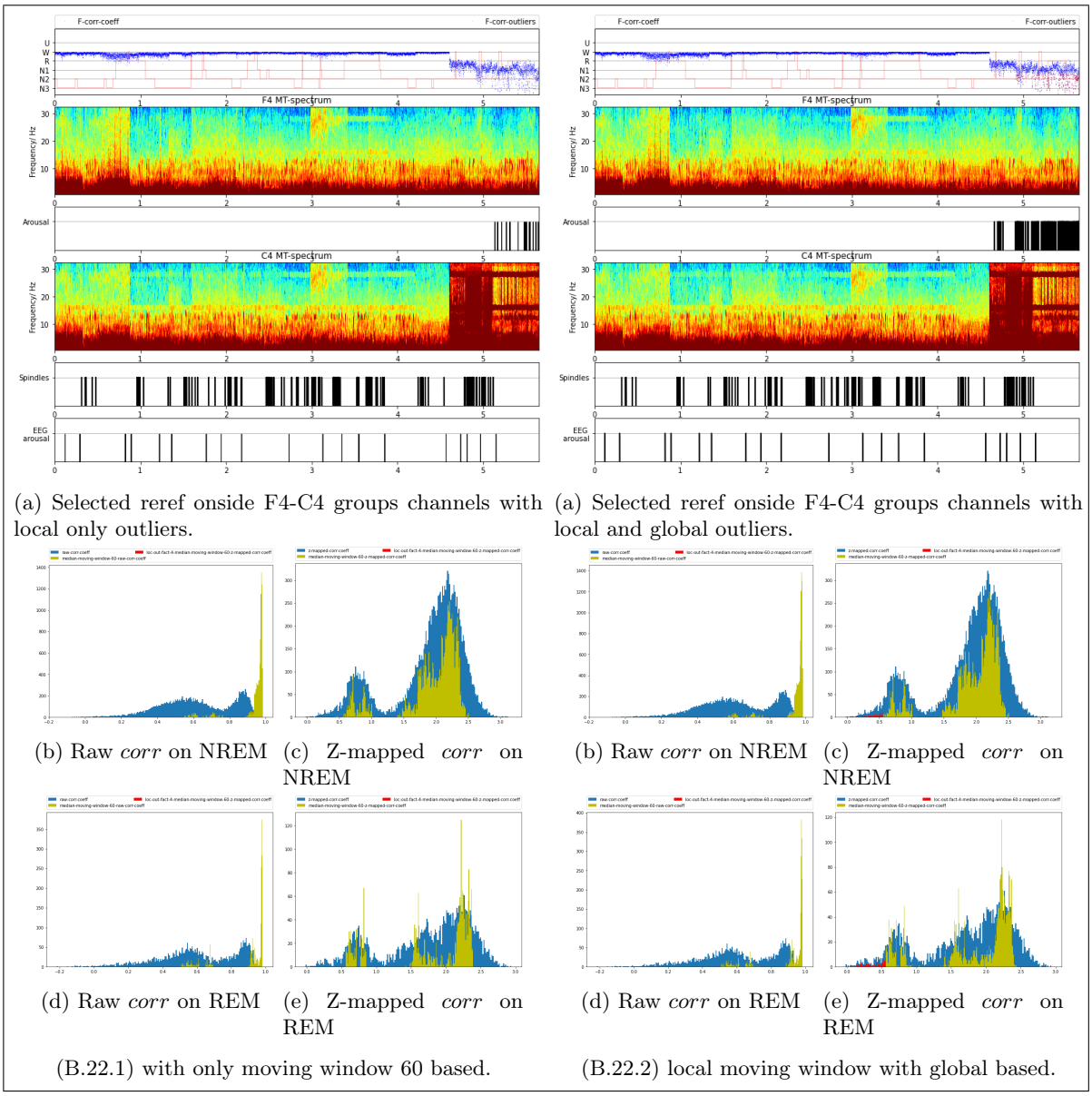


(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

(B.20.1) with only moving window 60 based.

(B.20.2) local moving window with global based.

Figure B.20: F3-C3 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.



(a) Selected reref onside F4-C4 groups channels with local only outliers.

(a) Selected reref onside F4-C4 groups channels with local and global outliers.

(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM

(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM

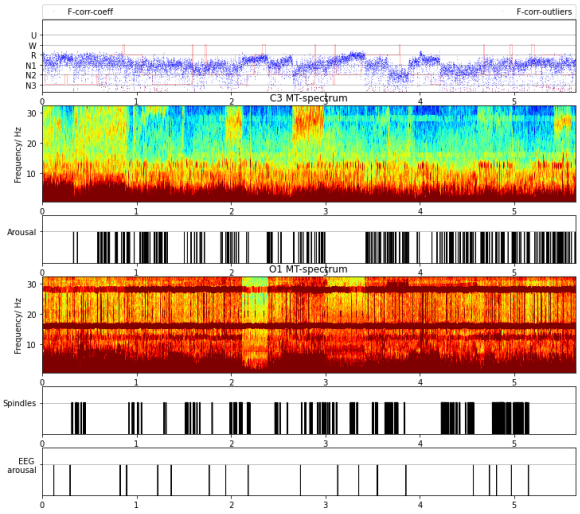
(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

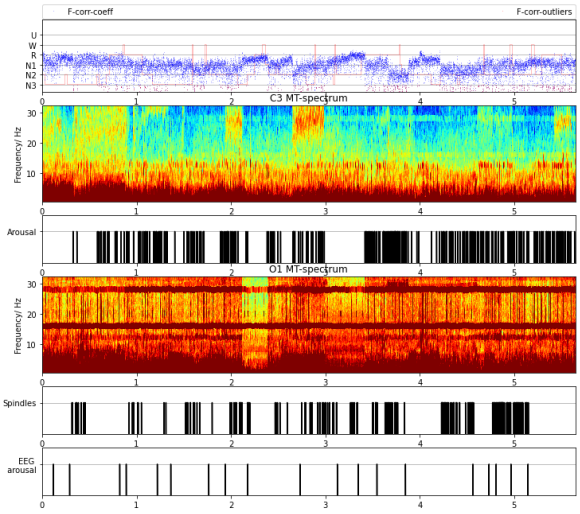
(B.22.1) with only moving window 60 based.

(B.22.2) local moving window with global based.

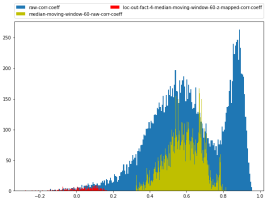
Figure B.22: F4-C4 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.



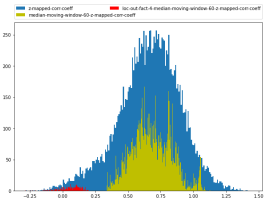
(a) Selected reref onside C3-O1 groups channels with local only outliers.



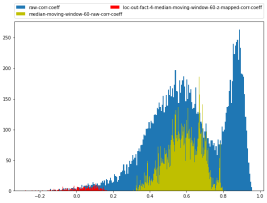
(a) Selected reref onside C3-O1 groups channels with local and global outliers.



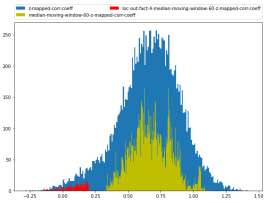
(b) Raw *corr* on NREM



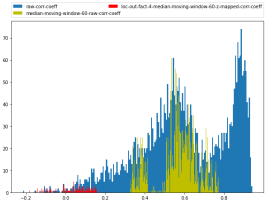
(c) Z-mapped *corr* on NREM



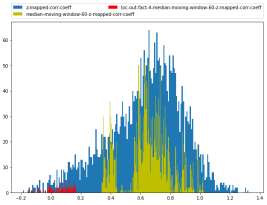
(b) Raw *corr* on NREM



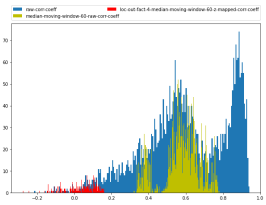
(c) Z-mapped *corr* on NREM



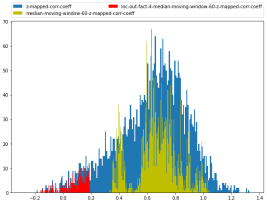
(d) Raw *corr* on REM



(e) Z-mapped *corr* on REM



(d) Raw *corr* on REM

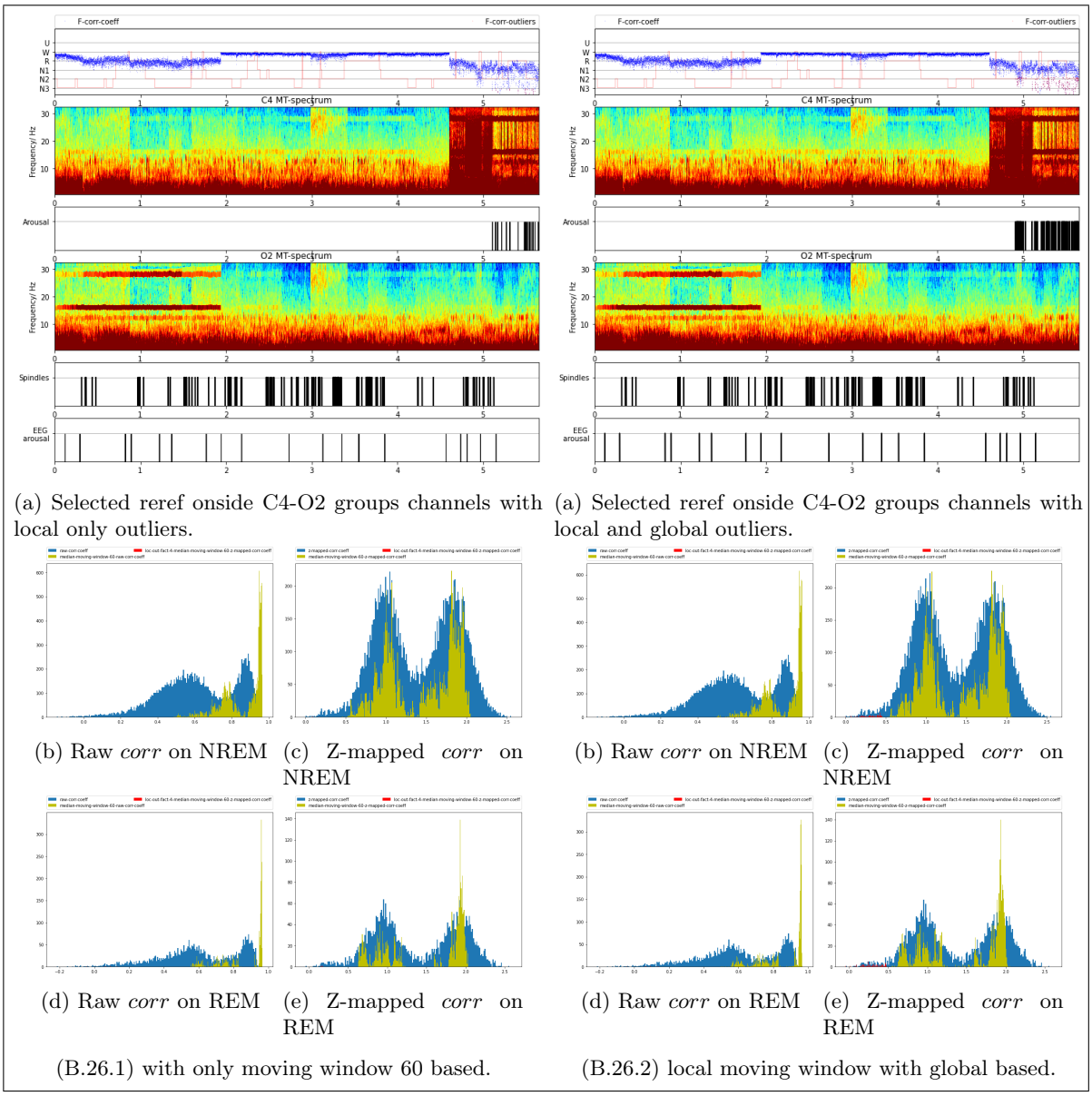


(e) Z-mapped *corr* on REM

(B.24.1) with only moving window 60 based.

(B.24.2) local moving window with global based.

Figure B.24: C3-O1 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.



(a) Selected reref onside C4-O2 groups channels with local only outliers.

(a) Selected reref onside C4-O2 groups channels with local and global outliers.

(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM

(b) Raw *corr* on NREM (c) Z-mapped *corr* on NREM

(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

(d) Raw *corr* on REM (e) Z-mapped *corr* on REM

(B.26.1) with only moving window 60 based.

(B.26.2) local moving window with global based.

Figure B.26: C4-O2 groups channels with MT based spectrum Z-mapped correlation coefficient on subject 21-0995_F_9.7_1_di while avoiding the spindle location in thrholds analysis these conditions are only used to obatin the intial distribution purpose. Then local threhold(4) with moving window 60 checked on the REM/ NREM full fragments (without any conditions) to obtain the distribution of median; then the outliers obtained from local moving window **vs with/ without** global conditions such as 75^{th} percentile (10) of pooled median of moving window on NREM/ REM fragments distribution.

Bibliography

[1] R. Vallat and M. P. Walker, "An open-source, high-performance tool for automated sleep staging," *eLife*, vol. 10, Oct. 2021. [Online]. Available: https://doi.org/10.7554/elife.70092

[2] M. J. Prerau, R. E. Brown, M. T. Bianchi, J. M. Ellenbogen, and P. L. Purdon, "Sleep neurophysiological dynamics through the lens of multitaper spectral analysis," *Physiology*, vol. 32, no. 1, pp. 60–92, Jan. 2017. [Online]. Available: https://doi.org/10.1152/physiol.00062.2015

[3] B. Babadi and E. N. Brown, "A review of multitaper spectral analysis," *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 5, pp. 1555–1564, May 2014. [Online]. Available: https://doi.org/10.1109/tbme.2014.2311996

[4] P. Anderer, S. Roberts, A. Schlögl, G. Gruber, G. Klösch, W. Herrmann, P. Rappelsberger, O. Filz, M. J. Barbanoj, G. Dorffner, and B. Saletu, "Artifact processing in computerized analysis of sleep EEG – a review," *Neuropsychobiology*, vol. 40, no. 3, pp. 150–157, 1999. [Online]. Available: https://doi.org/10.1159/000026613

[5] D. BRUNNER, R. VASKO, C. DETKA, J. MONAHAN, C. R. III, and D. KUPFER, "Muscle artifacts in the sleep EEG: Automated detection and effect on all-night EEG power spectra," *Journal of Sleep Research*, vol. 5, no. 3, pp. 155–164, Sep. 1996. [Online]. Available: https://doi.org/10.1046/j.1365-2869.1996.00009.x

[6] H. Sun, J. Jia, B. Goparaju, G.-B. Huang, O. Sourina, M. T. Bianchi, and M. B. Westover, "Large-scale automated sleep staging," *Sleep*, vol. 40, no. 10, Sep. 2017. [Online]. Available: https://doi.org/10.1093/sleep/zsx139

[7] M. Perslev, S. Darkner, L. Kempfner, M. Nikolic, P. J. Jennum, and C. Igel, "U-sleep: resilient high-frequency sleep staging," *npj Digital Medicine*, vol. 4, no. 1, Apr. 2021. [Online]. Available: https://doi.org/10.1038/s41746-021-00440-5

[8] D. Thomson, "Spectrum estimation and harmonic analysis," *Proceedings of the IEEE*, vol. 70, no. 9, pp. 1055–1096, 1982. [Online]. Available: https://doi.org/10.1109/proc.1982.12433

[9] C. L. Haley and M. Anitescu, "Optimal bandwidth for multitaper spectrum estimation," *IEEE Signal Processing Letters*, vol. 24, no. 11, pp. 1696–1700, Nov. 2017. [Online]. Available: https://doi.org/10.1109/lsp.2017.2719943