International Journal of Research in Engineering, Science and Management
Volume 4, Issue 6, June 2021
https://www.ijresm.com | ISSN (Online): 2581-5792

Quantitative Analysis of the Human Sleep Cycle Using Automatic Smoothing Filters

Tian Xiang Gao[1], Yuka Nagao[2], Kazuhiko Kume[3*]
[1,2,3]*Department of Neuropharmacology, Graduate School of Pharmaceutical Sciences, Nagoya City University, Nagoya, Japan*

***Abstract*: Human sleep is divided into rapid eye movement (REM) sleep and non-REM (NREM) sleep, and sleep cycles consisting of NREM sleep followed by REM sleep are a fundamental unit of one-night sleep. Although sleep cycle characteristics may have clinical significance, even qualitative analysis, needless to say quantitative analysis is rarely performed in routine polysomnography (PSG) examination. This is partly because the actual hypnograms have many irregular transitions between stages and appear different from the typical pattern. In order to make the hypnogram simple and the visual judgement of sleep cycle easy, we designed filters to process raw hypnogram data into a simplified schematic one. One group of filters are designed for smoothing the data, converting short intervening protruding epochs into flat continuous stages. The other group of filters bind two similar stages, namely NREM stage1 and 2, and NREM stage3 and 4 into one category, respectively. With these filters, the actual hypnograms are transformed into simplified form, and it became easy for clinicians and patients to perceive the sleep cycle properties. We applied the filters to publicly available sleep stage data and confirmed their validity and efficiency.**

***Keywords*: filter, polysomnography, python, sleep cycle, sleep stage.**

1. Introduction

Human sleep consists of rapid eye movement (REM) sleep and non-REM (NREM) sleep, which was established in 1950's [1, 2]. Typical one-night sleep starts with NREM sleep followed by REM sleep, and this sleep cycle repeats 4 to 5 times during one night [3, 4]. One sleep cycle generally spans 70 - 100 min for the first and 90 - 120 min for the second and subsequent cycles [5].

The sleep cycle analysis has been regarded clinically significant, but ordinary polysomnography (PSG) report does not usually contain a sleep cycle analysis but shows only the actual hypnogram. Therefore, clinicians and patients should judge the properties of the sleep cycle only by its visual appearance and do not usually quantitate the length of each cycle.

However, the sleep cycle judgement from the hypnograms, is sometimes tricky, especially for those patients with sleep disorders such as sleep apnea syndrome. The hypnograms sometimes contain the features such as frequent transitions between different stages by short intervening epochs of different stage from surrounding epochs, and short and intermittent REM stage between the sleep cycles. Although these features are important indicators of the quality of a patient's sleep, they may hinder the visual perception of sleep cycles.

In this manuscript, we designed several filters to process the raw sleep stage data into simple stage transition data, which visually represents the underlining sleep cycle.

2. Methods

A. *Dataset*

Sleep-EDF Database Expanded [6], available online on PhysioNet [7] was used for the analysis. The staging hypnogram files data was exported by software Polyman to csv files for further study. This research used only the Sleep Cassette Study section, which contains 153 files from healthy subjects.

B. *Hardware and software environment*

Computer: Windows 10 PC (i7 9700K/GeForce RTX2070 super/64 GB) with Anaconda Python 3.7.

3. Results

A. *Smoothing filters*

As a first step to process the sleep stage transition data, we developed stepwise smoothing filters, as shown in Figure 1. These filters detect a short intervening different stage during prolonged stage. GetOnePoint filter detects 5 continuous epochs, of which only one epoch at the center is different stage from the rest of the other 4 epochs, and change it into the same stage. This produces 5 continuous epochs of the same stage. GetTwoPoint and GetThreePoint filters similarly detect, 6 and 7 epochs, of which only two and three epochs at the center are different from the other 4 epochs and change them into the same stage. Using these three filters stepwise, short stages spanning only for 1 to 3 epochs are removed, and the transition number significantly decreases.

Another two filters, GetOnePoint_f and GetTwoPoints_f, are created also remove short stages between two different stages. They detect one or two epochs different from both the preceding and following stages. The filter name "_f" refers to those

*Corresponding author: kume.kazuhiko@gmail.com epochs are changed and merged with the preceding (former) stage.

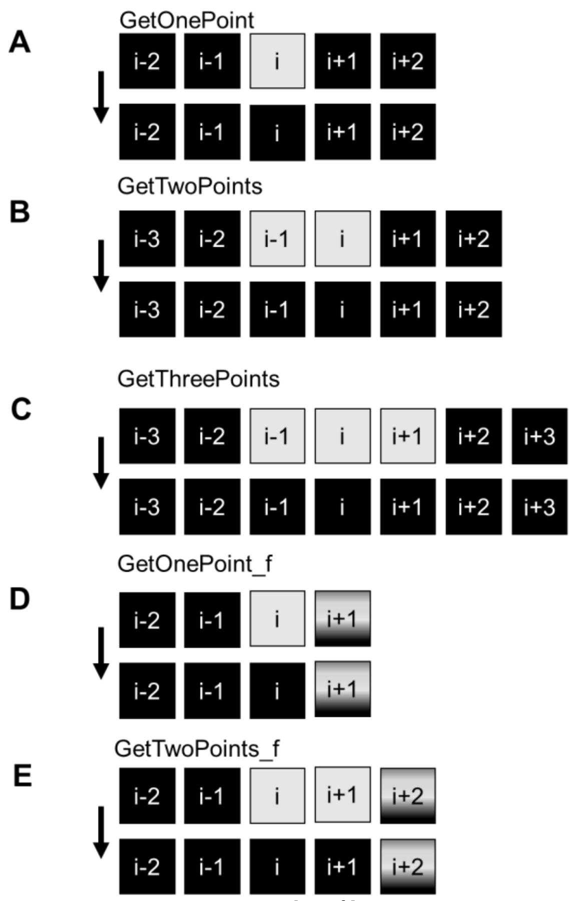


Fig. 1. Smoothing filters

Each block represents an epoch of 30 s. The different colors of the blocks represent the different sleep stages. The number inside represents the sequential order of time of the epoch.

A. GetOnePoint filter, converts one different epoch into the same stage surrounding it within 5 consecutive epochs containing four epochs of one same stage.
B. GetTwoPoints filter, converts 2 different epochs into the same stage surrounding them within 6 consecutive epochs containing four epochs of one same stage.
C. GetThreePoints filter, converts 3 different epochs into the same stage surrounding them within 7 consecutive epochs containing four epochs of one same stage.
D. GetOnePoint_f filter, converts one different epoch into the same stage preceding it within 4 consecutive epochs containing 2 epochs of one same stage.
E. GetTwoPoints_f filter, converts two different epochs into the same stage preceding them within 5 consecutive epochs containing 2 epochs of one same stage.

B. *Simplifying filters*

The second step shown in figure 2, is just for the purpose of simplifying the hypnogram. Four2Three filter converts NREM stage4 into NREM stage3, so all the stage 3 and 4 are combined into stage 3, which is regarded as deep sleep. This filter also makes the data compatible with the contemporary sleep stage classification, which abolished NREM stage 4. Two2One changes NREM stage2 into NREM stage1, so all the stage 1 and 2 are combined into stage 1, which is regarded as shallow sleep. With these filters, sleep stages are now only 4, namely wake, REM, NREM stage 1 and 3. The final filter OneInThree changes NREM stage1, which is found among NREM stage3 into NREM stage3. These filters enhance the separation of shallow sleep (N1 to N2) and deep sleep (N3 to N4).

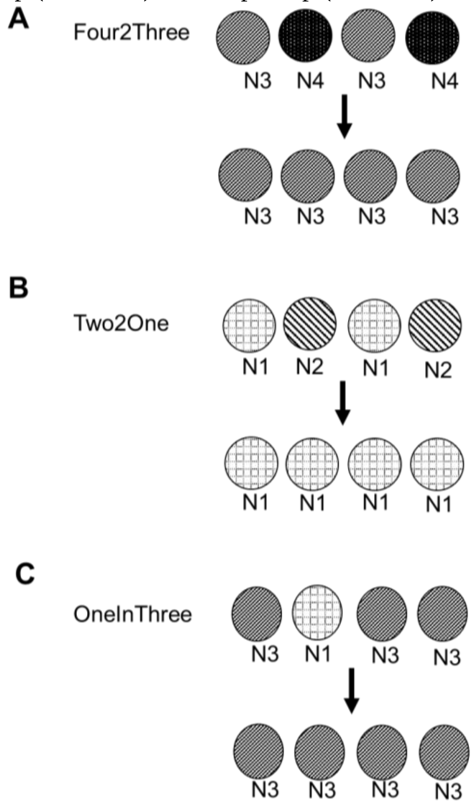


Fig. 2. Simplifying filters

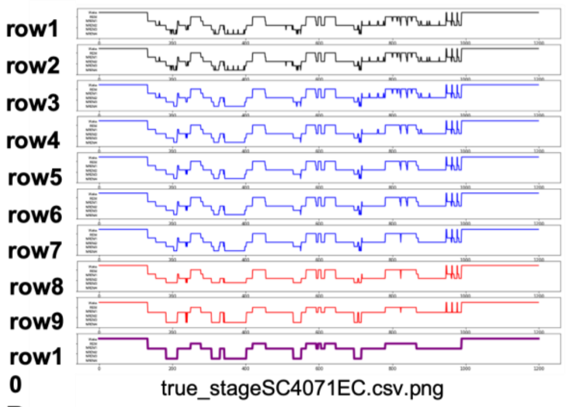
Each circle represents epochs (one or more) of the designated stage.

A. Four2Three filter, converts all NREM stage4 epochs into NREM stage 3.
B. Two2One filter, converts all NREM stage 2 into NREM stage 1.
C. OneInThree filter, converts NREM stage 1 epochs surrounded by NREM stage 3 epochs into NREM stage 3.

C. *Application of the filters to the actual sleep stage data*

In order to confirm whether these filters work practically, we used publicly available Sleep-EDF Database Expanded [6]. Figure 3 shows the actual transformation of the hypnogram from the original to processed one by filters. At the first step, we apply the Get_6 filter, which converts the movement times into wake times to the original data (row1), so that all the epochs are labelled as wake, NREM stage1 to 4 and REM (row2). These two rows retain original sleep stage information and shown in black lines. Then we applied the 5 smoothing filters, GetOnePoint, GetTwoPoints, GetThreePoints, GetOnePoint_f and GetTwoPoints_f, stepwisely, and the results are shown in row3 to row 7 as blue lines. We then applied 2 simplifying filters, Four2Three and Two2One filter, to completely split the NREM to deep NREM sleep and light NREM sleep. These two rows were represented in red. At the last step, we applied the OneInThree filter and the previously used filters, GetOnePoint, GetTwoPoints, GetThreePoints, GetOnePoint_f and GetTwoPoints_f again to normalize the transition pattern between deep NREM and shallow NREM sleep. This row was just be processed by a combination of all filters, so we represented it in purple. All the modified hypnogram shown by the purple line is extremely simple. With the help of modified hypnograms, we can easily figure out the number of sleep cycles visually and the length of the cycle time.

A



B

	row1	row2	row3	row4	row5	row6	row7	row8	row9	row10
Get_6		○	○	○	○	○	○	○	○	○○
GetOnePoint			○	○	○	○	○	○	○	○○
GetTwoPoints				○	○	○	○	○	○	○○
GetThreePoints					○	○	○	○	○	○○
GetOnePoint_f						○	○	○	○	○○
GetTwoPoints_f							○	○	○	○○
Four2Three								○	○	○
Two2One									○	○
OneInThree										○

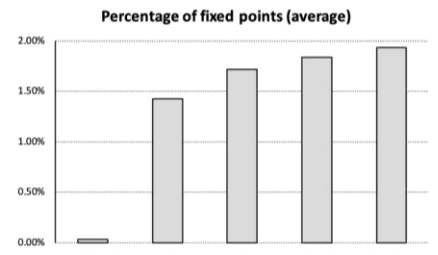
Fig. 3. Application of the filters to EDF sleep dataset

A. Representative transformation process of a raw hypnogram by the stepwise application of the filters. The data displayed here is true_stageSC4071EC. The black hypnogram (row1 〜2) is the raw data about the sleep stage. The blue hypnogram(row3〜7) are processed by smoothing filters and the blue hypnograms (row8〜9) are processed by simplifying filters. The last purple hypnogram (row10) is based on row9, but is processed by smoothing filters again.
B. A list of used filters for each row. One circle maker in one cell indicates the filter is used once, and the double circles maker means the filter was used twice.

D. *Validation of filter function*

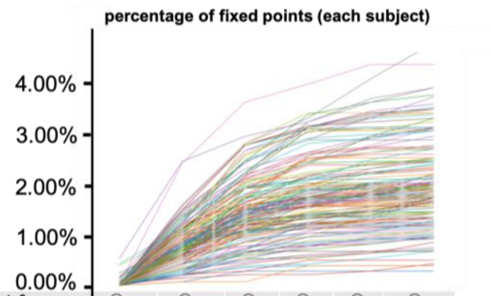
Since the filters artificially change the sleep stage data, if the degree of change is too big, they may induce erroneous information to the original data. Therefore, we measured the percentage of the changes to the original data by these filters, by adding counter programs at each filter. Because the simplifying filters are stage-specific, we excluded them from this validation. The results are shown in Figure 4. The combinations of the filter usage are the same as the table, which is shown in Figure 3, row 2 to row 7. The counter recorded how many times the filters were used. The upper panel in Figure 4A shows the average, and the bottom plot shows the individual subject data. Every line means a different subject, but they exhibit the same trend with the filters processing. The effect is most significant for the GetOnePoint filter, which changes only one epoch among 5 same stage epoch. On average, the filters changed less than 2 % of all data and thus reserves most of the important information in the original data. Although the individual differences are large, the overall trend is consistently increasing and leveling off.

A



Get_6	○	○	○	○	○	○
GetOnePoint		○	○	○	○	○
GetTwoPoints			○	○	○	○
GetThreePoints				○	○	○
GetOnePoint_f					○	○
GetTwoPoints_f						○

B



Get_6	○	○	○	○	○	○
GetOnePoint		○	○	○	○	○
GetTwoPoints			○	○	○	○
GetThreePoints				○	○	○
GetOnePoint_f					○	○
GetTwoPoints_f						○

Fig. 4. Percentage of conversion by the filters

A. Average percentage of the number of epochs which were converted into the different stage from the original nomination.
B. Individual data of above. Each line represents one subject data.

E. *Sleep cycle analysis*

In order to test our filters could help analyze the sleep cycle, we developed a filter-based cycle detector program. This program detects the start time of the whole night sleep, then searches for the first REM stage and calculate the length of the first sleep cycle. Next, it continues to search the next REM stage and repeat the calculation of the length until it reaches the end of the data. Figure 5 shows the calculated results of the average lengths of each sleep cycle of all the subjects in the database.

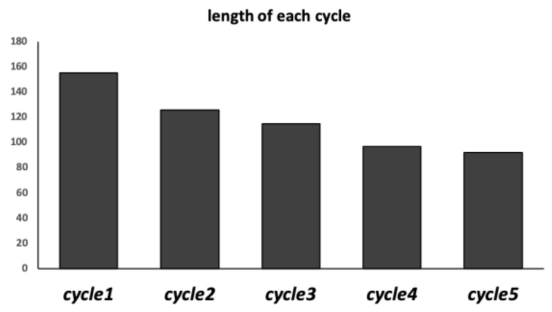


Fig. 5. Sleep cycle length analysis

The average period length from cycle 1 to cycle 5 were calculated based on filtered data. The values shown in the histogram indicate the number of subjects used for each cycle.

4. Discussion

The filters we produced makes the hypnogram very simple and make the sleep cycles easily visually perceptible. The resulting transformed hypnogram still retains the important information of sleep stages in the original data. The filters can be used separately, therefore the users can use only part of them. Since the simplifying filters remove the intermediate stage, i.e. NREM stage2, it will sometimes be better to use only smoothing filters. Actually, using only smoothing filters resulted in the simple data sufficient enough to the visual perception of sleep cycle (see row 7 of Figure 3).

We must point out that those features removed by the filters, are significantly important, and they should not be ignored. Therefore, the processed hypnograms using our filters will not substitute the original hypnograms, but they should always be used together with the originals. However, simplified version not only facilitates the perception of sleep cycle, but also induce interest in sleep cycle length itself.

Our purpose is to make the complex hypnogram as simply as possible. The simple one can be used for further cycle analysis or just make the people understand their sleep cycle pattern clearly. Although the sleep cycle is known almost from the beginning of contemporary sleep research, staging-specific analysis is rare except for the initial studies. But with the progress in the understanding of the nature of sleep through recent advances in molecular biological research, more quantitative analysis of sleep cycle may bring more insights into our understating of the sleep itself.

PSG data are usually taken at the laboratory in the hospital only for one or two nights, which limits the large-scale analysis and the information about the difference both among individual and within one individual is limited. With the growing popularity of wearable devices, especially brainwave measurement devices, it is reasonable to believe that people will have many continuous hypnograms data in the near future. So, at that time the shifting pattern in sleep cycle will be taken more seriously than the abnormalities points in one-night sleep. We believe our program provides an advantage for sleep cycle analysis and help the accumulation of knowledge.

Author Contribution and Acknowledgement

TXG, YN, KK planned the project, collected and analyzed data, TXG, KK wrote the manuscript. This work was performed according to the standard research ethical consideration. The authors declare no conflict of interest for this work. We are grateful to the members of Kume lab for their valuable comments and encouragements. This work was supported in part by KAKENHI (JP18H02481) from Japan Society for the Promotion of Science (JSPS).

References

[1] E. Aserinsky and N. Kleitman, “Regularly occurring periods of eye motility, and concomitant phenomena, during sleep.,” *Science.* vol. 118, no. 3062, pp. 273–4, 1953.
[2] W. Dement and N. Kleitman, “The relation of eye movements during sleep to dream activity: an objective method for the study of dreaming.,” *J. Exp. Psychol.* vol. 53, no. 5, pp. 339–46, 1957.
[3] H.P. Roffwarg, J.N. Muzio, and W.C. Dement, “Ontogenetic development of the human sleep-dream cycle.,” *Science.* vol. 152, no. 3722, pp. 604–19, 1966.
[4] E. Hartmann, “The 90-minute sleep-dream cycle.,” *Arch. Gen. Psychiatry.* vol. 18, no. 3, pp. 280–6, 1968.
[5] M.A. Carskadon and W.C. Dement, “Normal Human Sleep: An Overview.,” In: M.H. Kryger, T. Roth, and W.C. Dement, Eds. Principles and practice of sleep medicine (5th ed.), pp. 16–26, 2011.
[6] B. Kemp, A.H. Zwinderman, B. Tuk, H.A. Kamphuisen, and J.J. Oberyé, “Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the EEG.,” *IEEE Trans. Biomed. Eng.* vol. 47, no. 9, pp. 1185–94, 2000.
[7] A. L. Goldberger, L.A. Amaral, L. Glass, et al., “PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals.,” *Circulation.* vol. 101, no. 23, pp. E215-20, 2000.

Supplement: Python Program Codes of this Paper

This program file contains all the filters for the hypnogram processing and sleep cycle detection and calculation functions. All the inputs of these eleven functions are in list format of python for further development. In addition to the filters introduced in this article, we also designed a special function to draw hypnograms. In the original Sleep-EDF Database Expanded dataset, the number of 0,1,2,3,4,5 represents the Wake, NREM stage1, NREM stage2, NREM stage3, NREM stage4, REM respectively. However, in the conventional clinic style of hypnogram, the vertical coordinates from top to bottom are Wake, REM, NREM stage1, NREM stage2, NREM stage3, and NREM stage4. For plotting convenience, the function hypno_fix is created to adjust the upper and lower position of each stage. Some basic python libraries need to be installed, like NumPy, Pandas and Matplotlib. To facilitate development, all calculated result data is stored in python dictionary format.

A. *Python code: filter_gtx.py*

```
  # -*- coding: utf-8 -*-

#1
def get_6(raw):
  l = len(raw)
  for k in range(1,int(l-4),1):
    if raw[k] == 6:
      raw[k]=  raw[k-1]
    return raw

#2
def GetOnePoint(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
        if raw[k-1] == raw[k+1]  and raw[k-2] == raw[k+2]:
            raw[k]=  raw[k-1]
    return raw

#3
def GetTwoPoints(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
        if raw[k-3] == raw[k+2]  and raw[k-2] == raw[k+1]:
            if  raw[k] == raw[k-1] and raw[k] != raw[k+2]:
                raw[k] = raw[k+2]
                raw[k-1] =raw[k+2]
    return raw

#4
def GetThreePoints(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
        if raw[k-3] == raw[k+2]  and raw[k-2] == raw[k+3]:
            if  raw[k-1] == raw[k] and raw[k]==raw[k+1]:
                if raw[k] != raw[k+2]:
                    raw[k] = raw[k+2]
                    raw[k-1] =raw[k+2]
                    raw[k+1] =raw[k+2]
    return raw

#5
def GetOnePoint_f(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
         if raw[k-2] == raw[k-1]  and raw[k-1] != raw[k]:
             if abs(raw[k]-raw[k-1]) > 2 and raw[k] !=  raw[k+1]:
                 raw[k] = raw[k-1]
    return raw

#6
def GetTwoPoint_f(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
         if raw[k-2] == raw[k-1]  and raw[k] == raw[k+1]:
             if abs(raw[k]-raw[k-1]) > 2 and raw[k] !=  raw[k+2]:
                 raw[k] = raw[k-1]
                 raw[k+1] = raw[k-1]
    return raw
#7
def Four2Three(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
        if raw[k] == 4:
            raw[k]=3
    return raw

#8
def Two2One(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
        if raw[k] == 2:
            raw[k]=1
    return raw

#9
def OneInThree(raw):
    l = len(raw)
    for k in range(1,int(l-4),1):
        if raw[k-1] == raw[k+1] and raw[k-1] != raw[k] :
            raw[k] =raw[k-1]
    return raw

#10
def DetectCycleByRem(raw_sim):
    cylce_list = []
    l = len(raw_sim)
    for n in range(4,int(l-5),1):
        if raw_sim[n] == 5 and raw_sim[n-1] == 5:
            #if raw[n-2] and raw[n-3]== 5:
            if raw_sim[n-2] ==5:
                if raw_sim[n+1] <= 1 :
                #if raw_sim[n+1] <= 1 and raw_sim[n+1] <=1:

                    #if raw[n+3] ==1 and raw[n+4] ==1:
                    print(n)
                    cylce_list.append(n)
    return cylce_list

#11
def RemDelta(cylce_list):
    l = len(cylce_list)
    delta = []
    for i in range(0,l-1,1):
        print(i)
        print(l)
        print(cylce_list)
        d = cylce_list[i+1] - cylce_list[i]
        delta.append(d)
    print(delta)
    return delta

#12
def CauculteRem(cycle_list):
    l = len(cycle_list)
    unique_list = []
    for m in range(1,int(l),1):
        if cycle_list[m] - cycle_list[m-1] < 40:
            cycle_list[m-1] = cycle_list[m]
    [unique_list.append(x) for x in cycle_list if x not in
unique_list]
return unique_list

#13
def CauculteOneSet(raw):
    oneset_list= []
    l = len(raw)
    for m in range(4,int(l-5),1):
        if raw[m-1] == 0 and raw[m] == 0:
```

```
      if  raw[m+1] == 1 and raw[m+2] == 1:
        #if (raw[m+3] > 0 and raw[m+4] > 0) and
(raw[m+4] >0 and raw[m+4] >0) :
        if raw[m+3] +raw[m+4] +
raw[m+5]+raw[m+6]+raw[m+7] > 3:
          print(m+1)
          return m+1
          #oneset_list.append(m+1)

#14
def Caucultewake(raw):
  wake_list=[]
  l = len(raw)
  for m in range(4,int(l-20),1):
    if raw[m] +raw[m+1] +raw[m+2] +raw[m+3] +raw[m+4]
+raw[m+5] +raw[m+6] +raw[m+7] +raw[m+8]+raw[m+9]
+raw[m+10] +raw[m+11]+raw[m+12]+ raw[m+13] +
raw[m+14] +raw[m+15] +raw[m+16] +raw[m+17]
+raw[m+18] ==0:
      if  raw[m-1] > 0:
        #if (raw[m+3] > 0 and raw[m+4] > 0) and
(raw[m+4] >0 and raw[m+4] >0) :
        if raw[m-3] +raw[m-4] + raw[m-5]+raw[m-
6]+raw[m-7] > 2:
          wake = m
          print(m)
          wake_list.append(m)
    print(wake_list)
  if m == int(l-21) and (wake_list ==[] or wake_list[-1]<500):
    wake_list.append(l)
  print(wake_list)
  return wake_list[-1]

#15
def CountsDi(list1,list2):
  l1 = len(list1)
  l2 = len(list2)
  count = 0
  if l1 != l2:
    print('tow lists have not the same length!' )
  else:

    for m in range(0,int(l1),1):
      if  list1[m] != list2[m]:
        count = count+1
        print(m)
        print(count)
  return count


#16
def hypno_fix(list):
  l = len(list)
  for g in range(0,l,1):
    if list[g] <5 and list[g] >0 :
      list[g] = -list[g]
    elif list[g]==0:
      list[g] = 1
    elif list[g]==5:
      list[g] = 0
  return list

#17
def hishi(list):
  l = len(list)
  for g in range(0,l,1):
    list[g] = list[g]/2880

  return list
```

B. Python code: analysis_sheet_edf_hypnogramfix.py

```
# -*- coding: utf-8 -*-

### Library Import

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import os
import filter_gtx


### Reading stage information from CSV
### Addition of all data to dictionary

folder = 'G:/paper_filter/true_stage/'
save_folder= 'G:/paper_filter/#/'
n = 153
list =  os.listdir(folder)

stage_dict = {}
for i in range(0,n,1):
  dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
  listi = dfi.values.tolist()
  #filter_gtx.SimplyHy_6(listi)
  stage_dict[list[i]] = listi

stage_dict_fixed = {}
counts_list = []
counts_dic= {}
stage_dictstage_dict_fixed = {}


### Process by filters
###

for i in range(0,n,1):
  dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
  listi = dfi.values.tolist()[0]
  list_row = stage_dict[list[i]]
  list1 = list_row.copy()
  list2 = list_row.copy()

  fig=plt.figure(figsize=(20,15))

  ax1=fig.add_subplot(10,1,1)
  list1 = filter_gtx.hypno_fix(list1)
```

```
    ax1.plot(list1[800:2000],linewidth = 2.0,color= 'black')
    ax1.set_ylim(-5, 2)

    ax2=fig.add_subplot(10,1,2)
    list2 = filter_gtx.get_6(list2)
    list2 =filter_gtx.get_big(list2)
    list2_h = list2.copy()
    list2_h = filter_gtx.hypno_fix(list2_h)
    ax2.plot(list2_h[800:2000],linewidth = 2.0,color= 'black')
    ax2.set_ylim(-5, 2)

    ax3=fig.add_subplot(10,1,3)
    list3 = list2.copy()
    list3 =filter_gtx.GetOnePoint(list3)
    list3_h = list3.copy()
    list3_h = filter_gtx.hypno_fix(list3_h)
    ax3.plot(list3_h[800:2000],linewidth = 2.0,color= 'blue')
    ax3.set_ylim(-5, 2)

    ax4=fig.add_subplot(10,1,4)
    list4 = list3.copy()
    list4 =filter_gtx.GetTwoPoints(list4)
    list4_h = list4.copy()
    list4_h = filter_gtx.hypno_fix(list4_h)
    ax4.plot(list4_h[800:2000],linewidth = 2.0,color= 'blue')
    ax4.set_ylim(-5, 2)

    ax5=fig.add_subplot(10,1,5)
    list5 = list4.copy()
    list5 =filter_gtx.GetThreePoints(list5)
    list5_h = list5.copy()
    list5_h = filter_gtx.hypno_fix(list5_h)
    ax5.plot(list5_h[800:2000],linewidth = 2.0,color= 'blue')
    ax5.set_ylim(-5, 2)

    ax6=fig.add_subplot(10,1,6)
    list6 = list5.copy()
    list6 =filter_gtx.GetOnePoint_f(list6)
    list6_h = list6.copy()
    list6_h = filter_gtx.hypno_fix(list6_h)
    ax6.plot(list6_h[800:2000],linewidth = 2.0,color= 'blue')
    ax6.set_ylim(-5, 2)

    ax7=fig.add_subplot(10,1,7)
    list7 = list6.copy()
    list7 =filter_gtx.GetTwoPoint_f(list7)
    list7_h = list7.copy()
    list7_h = filter_gtx.hypno_fix(list7_h)
    ax7.plot(list7_h[800:2000],linewidth = 2.0,color= 'blue')
    ax7.set_ylim(-5, 2)

    ax8=fig.add_subplot(10,1,8)
    list8 = list7.copy()
    list8 =filter_gtx.Four2Three(list8)
    list8_h = list8.copy()
    list8_h = filter_gtx.hypno_fix(list8_h)
    ax8.plot(list8_h[800:2000],linewidth = 2.0,color= 'red')
    ax8.set_ylim(-5, 2)

    ax9=fig.add_subplot(10,1,9)
    list9 = list8.copy()
    list9 =filter_gtx.Two2One(list9)
    list9_h = list9.copy()
    list9_h = filter_gtx.hypno_fix(list9_h)
    ax9.plot(list9_h[800:2000],linewidth = 2.0,color= 'red')
    ax9.set_ylim(-4, 2)

    ax10=fig.add_subplot(10,1,10)
    list10 = list9.copy()
    list10 =filter_gtx.Two2One(list10)
    list10 =filter_gtx.GetOnePoint(list10)
    list10 =filter_gtx.GetTwoPoints(list10)
    list10 =filter_gtx.GetThreePoints(list10)
    list10 =filter_gtx.GetOnePoint_f(list10)
    list10 =filter_gtx.GetTwoPoint_f(list10)
    list10 =filter_gtx.OneInThree(list10)
    list10_h = list10.copy()
    list10_h = filter_gtx.hypno_fix(list10_h)
    ax10.plot(list10_h[800:2000],linewidth = 5.0,color=
'purple')
    ax10.set_ylim(-4, 2)

    plt.savefig(save_folder + list[i]+'.png')
    stage_dict_fixed[list[i]] = list3


### Sleep cycle analysis
###

import filter_gtx
min_val = 30

cycle_dict = {}
rem_dict   = {}
oneset_dict= {}
wake_dict  = {}

for i in range(0,n,1):
    #stage = stage_dict[list[i]]
    #print(list[i])
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]

    cylce_list =
filter_gtx.DetectCycleByRem(stage_dict_fixed[list[i]])
    print(cylce_list)

    cylce_list1=filter_gtx.RemDelta(cylce_list)
    cycle_dict[list[i]] = cylce_list1

    rem_list= filter_gtx.CauculteRem(cylce_list)
    rem_dict[list[i]] = rem_list

    oneset = filter_gtx.CauculteOneSet(stage_dict_fixed[list[i]])
    oneset_dict[list[i]] = oneset

    wake = filter_gtx.Caucultewake(stage_dict_fixed[list[i]])
    wake_dict[list[i]] = wake

Each_cycle = {}
```

```
Each_cycle_all={}

for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]

    print(rem_dict[list[i]])
    print(wake_dict[list[i]])
    print(oneset_dict[list[i]])

    list_sub1 = rem_dict[list[i]].copy()
    list_sub1.insert(0,oneset_dict[list[i]])
    list_sub2 = rem_dict[list[i]].copy()
    list_sub2.append(wake_dict[list[i]])
    list_sub3 = rem_dict[list[i]].copy()
    list_sub3.insert(0,oneset_dict[list[i]])
    list_sub3.append(wake_dict[list[i]])

    print(list_sub1)
    print(list_sub2)
    print(list_sub3)

    np1 = np.array(list_sub1)
    np2 = np.array(list_sub2)

    np_cycle = np2-np1

    print(np_cycle)
    np_cycle = np_cycle.tolist()

    Each_cycle[list[i]] = np_cycle
    Each_cycle_all[list[i]] = list_sub3


Each_cycle_fix = {}

import statistics
from statistics import mean

for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]
    print(Each_cycle[list[i]])
    cycle_fix = Each_cycle[list[i]].copy()

    length_cycle = len(cycle_fix)

    unique_list = []
    for j in range(0,int(length_cycle),1):
        print(j)
        if cycle_fix[j] < min_val:
            print(j)
            cycle_fix[j] = cycle_fix[j-1]
    [unique_list.append(x) for x in cycle_fix if x not in
unique_list]
    print(unique_list)
    Each_cycle_fix[list[i]] = unique_list


### Sleep cycle calculation
###

cycle_1th = []
for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]

    cycle_1_element = Each_cycle_fix[list[i]][0]
    cycle_1th.append(cycle_1_element)


cycle_1th_mean = mean(cycle_1th)

cycle_2th = []
for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]
    each_list = Each_cycle_fix[list[i]]
    l = len(each_list)
    if l<=1:
        pass
    else:
        cycle_2_element = Each_cycle_fix[list[i]][1]
        cycle_2th.append(cycle_2_element)

cycle_2th_mean = mean(cycle_2th)

cycle_3th = []
for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]
    each_list = Each_cycle_fix[list[i]]
    l = len(each_list)
    if l<=2:
        pass
    else:
        cycle_3_element = Each_cycle_fix[list[i]][2]
        cycle_3th.append(cycle_3_element)

cycle_3th_mean = mean(cycle_3th)

cycle_4th = []
for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]
    each_list = Each_cycle_fix[list[i]]
    l = len(each_list)
    if l<=3:
        pass
    else:
        cycle_4_element = Each_cycle_fix[list[i]][3]
        cycle_4th.append(cycle_4_element)

cycle_4th_mean = mean(cycle_4th)

cycle_5th = []
for i in range(0,n,1):
    dfi = pd.read_csv(folder + list[i],header = None).iloc[1:,1]
    listi = dfi.values.tolist()[0]
    each_list = Each_cycle_fix[list[i]]
    l = len(each_list)
    if l<=4:
        pass
    else:
        cycle_5_element = Each_cycle_fix[list[i]][4]
        cycle_5th.append(cycle_5_element)

cycle_5th_mean = mean(cycle_5th)

plt.plot([cycle_1th_mean,cycle_2th_mean ,cycle_3th_mean
,cycle_4th_mean ,cycle_5th_mean ,cycle_6th_mean
,cycle_7th_mean] )

print([cycle_1th_mean/2,cycle_2th_mean/2 ,cycle_3th_mean
/2,cycle_4th_mean/2 ,cycle_5th_mean /2,cycle_6th_mean/2
,cycle_7th_mean/2])

cycle_1th_mean= mean(cycle_1th)
cycle_2th_mean= mean(cycle_2th)
cycle_3th_mean= mean(cycle_3th)
cycle_4th_mean= mean(cycle_4th)
cycle_5th_mean= mean(cycle_5th)

cycle_1th_stdev = statistics.stdev(cycle_1th)
cycle_2th_stdev = statistics.stdev(cycle_2th)
cycle_3th_stdev = statistics.stdev(cycle_3th)
cycle_4th_stdev = statistics.stdev(cycle_4th)
cycle_5th_stdev = statistics.stdev(cycle_5th)

len_cycle_1th = len(cycle_1th)
len_cycle_2th = len(cycle_2th)
len_cycle_3th = len(cycle_3th)
len_cycle_4th = len(cycle_4th)
len_cycle_5th = len(cycle_5th)

print([cycle_1th_mean/2,cycle_2th_mean/2 ,cycle_3th_mean
/2,cycle_4th_mean/2 ,cycle_5th_mean /2])

print([cycle_1th_stdev/2,cycle_2th_stdev/2 ,cycle_3th_stdev
/2,cycle_4th_stdev/2 ,cycle_5th_stdev /2])
```