QUANTIFICATION OF NOCTURNAL BLOOD PRESSURE OSCILLATIONS IN OBSTRUCTIVE SLEEP APNEA PATIENTS

By

Yao-Shun Chuang

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN BIOMEDICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2020

ACKNOWLEDGEMENTS

Everything in this world has the meaning from the God's arrangement. Two years ago, I can't imagine I would study abroad, and two years later, I wouldn't be able to finish this work without the help from my guide, Dr. Khosrow Behbehani. Dr. Behbehani patiently teaches me all the knowledge, leads me to think and makes space for me to practice the idea. There wouldn't exist a way to express in words how grateful I am because I can't even begin to explain how much your help meant to me.

Secondly, I want to express my gratitude to the graduate academic advisor, Julie Rockow, and the administrative assistant, Alicia Gill, in Bioengineering Department because, with their support and help, I could focus more on the research.

I would not have been able to complete this work without the help, support and love of almighty God, my parents, my best buddy, Wade Yeh, in Taiwan and my friends in UTA.

June 1, 2020

ABSTRACT

QUANTIFICATION OF NOCTURNAL BLOOD PRESSURE OSCILLATIONS
IN OBSTRUCTIVE SLEEP APNEA PATIENTS

Yao-Shun Chuang, M.S.

The University of Texas at Arlington, 2020

Supervising Professor: Khosrow Behbehani

An approach to quantifying and analysis of nocturnal blood pressure (BP) variations that are elicited by sleep-disordered breathing (SDB) is presented. A sample-by-sample aggregation of the dynamic BP variations during normal breathing and BP oscillations prompted by apnea episodes is performed. This approach facilitates the visualization and analysis of BP oscillations. Nocturnal BP oscillations reflect the cardiovascular stresses that SDB mediates and may be of clinical significance. Preliminary results from an analysis of full night study of 10 SDB subjects (8 Male 2 Female, 53±5.7 yrs., Body Mass Index 34.5±7.8 kg/m2, Apnea-Hypopnea Index 63.5±28.7) are presented. Aggregate trajectory and quantitative values for changes concomitant with obstructive apnea episodes are presented for systolic blood pressure (SBP), diastolic blood pressure (DBP), mean arterial blood pressure (MAP) and pulse pressure (PP).

Further, the effect of sleep stages is considered. The results show 17.6 mmHg (13.7%) surge in SBP and 9.5 mmHg (13.7%) surge in DBP. By computing the MAP and PP, an 11.8 mmHg (13.3%) and a 9.8 mmHg (16.4%) surge in MAP and PP, respectively, were observed. When compared to their respective values during normal breathing, SBP, DBP, and PP show significantly different mean values ($p<0.001$). Additionally, the peak of the surge in SBP, DBP, and MAP occurred about 7s post the end of apnea events. This delay is around 8s for PP. Further, the rate of surge in SBP and DBP due to apnea was estimated by computing the slopes of their rise. They were0.9 mmHg/s and 0.6

mmHg/s, respectively. . When computing the surge rate of SBP in various sleep stages, the slopes were approximately 0.95 mmHg/s, 0.93 mmHg/s, 0.89 mmHg/s and 0.7 mmHg/s for of Stages 1, 2, 3, and REM, respectively. Similar analysis for DBP showed the slopes to be approximately 0.64 mmHg/s, 0.69 mmHg/s, 0.53 mmHg/s and0.39 mmHg/s for Stages 1, 2, 3, and REM, respectively. The results of this study provide means of quantifying both the rate and the level of nocturnal blood pressure oscillations.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.1. Sleep Apnea

1.1.1. What is Sleep Apnea?

Sleep apnea is characterized with resulting either from partial or complete obstruction of the upper airway during sleep by, definition, lasting at least 10 sec during sleep, which cause patients experience repetitive episodes of apnea or reduced inspiratory airflow. These events are associated with intermittent hypoxemia and possibly hypercapnia and usually provoke an arousal from sleep. The arousal is associated with restoration of upper airway patency and ventilation [1]. One study deduced the prevalence estimates of moderate to severe sleep-disordered breathing (apnea-hypopnea index, measured as events/hour, ≥15) are 10% among 30–49-year-old men; 17% among 50–70-year-old men; 3% among 30–49-year-old women; and 9% among 50–70 year-old women. These estimated prevalence rates represent substantial increases over the last 2 decades [2].

1.1.2. Types of Sleep Apnea

There are three types of sleep disorder breathing (SDB): obstructive sleep apnea (OSA), central sleep apnea (CSA) and mixed sleep apnea (MSA). OSA is the most common type, constituting greater than 85% of all cases of sleep apnea and CSA is less common [3]. A physical blockage of the airway, during sleep, causes OSA. That is, during OSA even though the respiratory effort is still present, upper airway collapses due to lack of muscle tone during sleep and results in airway obstruction that prevents airflow to the lungs. In the CSA, although the airway is not blocked, the brain fails to signal the muscles to breathe and breathing is interrupted by a lack of respiratory effort [4]. MSA is a sleep disordered breathing that includes both central and obstructive apnea. MSA

events occur more frequently in severe sleep apnea with hypopnea patients, as well as gender in male and higher level in BMI. Also, the patients with higher Epworth Sleepiness Scales and higher triglyceride scores levels are associated with higher risk for MSA events in obstructive sleep apnea-hypopnea syndrome patients [5].

1.1.3. Health Consequences of Obstructive Sleep Apnea

OSA may have adverse effects on individuals' health, depending on the severity apnea. It is associated with cardiovascular diseases such as hypertension, heart failure and cerebrovascular diseases [6], as well as diabetes and metabolic syndrome [7]. In addition, excessive daytime sleepiness and impaired neurocognitive function are related to sleep apnea. Considering long-term adverse health outcomes of OSA, it is associated with the higher risk of motor vehicle accidents [8], reduced quality of life, increased cardiovascular morbidity, increased malignancy and other mortality [9]. One study showed that long-term cardiovascular morbidity and mortality increased in untreated severe OSA patients [10].

1.1.4. Detection and Treatment of Sleep Apnea

An overnight full polysomnography (PSG) is the standard diagnostic test for OSA. PSG consists of recording cardiorespiratory and neurophysiological signals, which are used to analyze sleep and breathing. The apnea–hypopnea index (AHI) derived from the PSG is used to diagnose the disease. The apnea hypopnea index (AHI), a count of the number of apneas and hypopneas per hour of sleep, is the key measure used for case identification, for quantifying disease severity, and for defining disease prevalence in normal and clinical populations. [11] AHI scale of 5-15 events/hour is generally classified as mild apnea, 15-30 events/hour as moderate apnea. If AHI is above 30 it is categorized as severe apnea [12].

Therapeutic approaches for treating OSA include positive airway pressure (Continuous Positive Airway Pressure, or CPAP and bi-level Positive Airway Pressure), oral appliances, and surgery [13]. The mainstay of OSA treatment is CPAP which delivers a column of pressurized air via

a nasal or facial mask interface to keep the airway open. With the positive pressure, CPAP essentially provides a pneumatic splint to keep the airway open and prevent its collapse. CPAP is highly efficacious in reducing obstructive events, but still, its health benefits are limited by patient compliance with the therapy [14]. Historically, it is shown that two thirds of the patients prescribed CPAP adhere to the therapy. [9].

1.2. Arterial Blood Pressure

1.2.1. What is Arterial Blood pressure

The pressure measurement in large arteries in the systemic circulation is considered as blood pressure in human beings. Arterial pressure directly corresponds to cardiac output, arterial elasticity, and peripheral vascular resistance [15]. The measured number can be divided into systolic blood pressure and diastolic blood pressure in Figure 1.2-1. Systolic blood pressure (SBP) refers to the maximum pressure within the large arteries when the heart muscle contracts to propel blood through the body. Diastolic blood pressure (DBP) describes the lowest pressure within the large arteries during heart muscle relaxation between beatings [16].

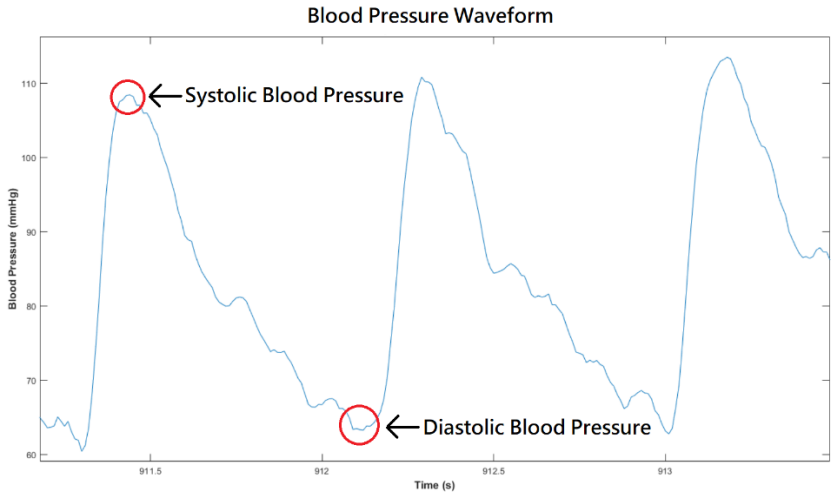

Figure 1.2-1 Blood Pressure Waveform

1.2.2. High Blood Pressure

Blood pressure will increase with age, even in apparently healthy individuals, it is

acknowledged as a feature of human aging [17]. As shown in Table 1.2-1, according to 2018 guideline from the American College of Cardiology (ACC) and the American Heart Association (AHA) for the Prevention, Detection, Evaluation and Management of High Blood Pressure in Adults, blood pressure (BP) should be categorized as normal (<120/80 mm Hg), elevated (120-129/<80 mm Hg), stage 1 hypertension (130-139/80-89 mm Hg), or stage 2 hypertension (≥140/90 mm Hg) [18].

Table 1.2-1 Stages of Hypertension

Category	Systolic (mmHg)	Diastolic (mmHg)
Normal	**< 120**	**< 80**
Elevated	**120 – 129**	**< 80**
Stage 1 Hypertension	**130-139**	**80-89**
Stage 2 Hypertension	**≥ 140**	**≥ 90**

1.2.3. Relationship between High Blood Pressure and Sleep Apnea

Sleep apnea has been associated with hypertension and cardiovascular disease (CVD) in studies [19]. It is now the most common identifiable cause of secondary hypertension [20]. In addition, apnea and hypopnea events cause instantaneous elevations in blood pressure, which are associated blood oxygen desaturation, arousal, and sympathetic activation. Some researchers propose that these episodic BP may lead to elevated blood pressure during the daytime and, ultimately, sustained hypertension [21].

1.2.4. Nocturnal Blood Pressure Variations

Blood pressure fluctuates appreciably throughout day-night with a clear decrease during sleep [22]. In healthy individuals, the blood pressure dipping regularly exceeds 10% of mean daytime values in normotensive and primary hypertensive subjects [23] [24]. Blood pressure declination during sleep causes a baroreflex threshold downward shift, which the threshold counter-regulate the modulation the activity of the sympathetic nerve to the muscle vascular. The result of this shift is a reduced sympathetic activity and that cause a blood pressure decline during sleep [22]. In addition, during the nighttime, blood flow to the skeletal muscles is decreased through local auto-regulation,

which increases total peripheral resistance and decreases cardiac output compared with the daytime [23].

1.2.5. Relationship between Nocturnal Blood Pressure and Sleep Apnea

A drop or dipping in the blood pressure during a night of sleep is important for health [25]. However, studies found a non-dipping pattern occurred in older OSA patients. It was found that patients with moderate to severe SDB have significant association with sleep-time hypertension, indicating increased BP load during sleep [25]. The non-dipping pattern is considered as a future development marker of hypertension in those who are normotensive at baseline. In patients with hypertension, non-dipping has been associated with worse cardiovascular prognosis and increased target organ damage, [26]. Moreover, the increase of variation of nocturnal blood pressure becomes one characteristic of nocturnal hypertension in OSA patients [20] . The phenomenon may be due to a temporary increase in arterial stiffness resulting from sympathetically mediated vasoconstriction, or increased cardiac stroke volume driven by isotropic effects of sympathoexcitation, or both [27].

1.3. Study Overview and Organization

1.3.1. Objective of the study

The goal of this study is to quantify the dynamic variations in nocturnal BP of OSA patients. By selecting SBP, DBP, mean arterial pressure (MAP), and pulse pressure (PP) as prime measures of BP, we quantify the oscillation level of BP during sleep apnea episodes. The proposed method for this study uses the SBP and DBP derived from continuously monitored blood pressure waveform for all apnea episode to obtain an aggregate representation of the BP due to OSA. Further, these the interaction of these variations with sleep stages are studied.

1.3.2. Thesis Organization

Chapter two of this thesis describes the usage and components of the continuous blood pressure monitoring method, and its principle of operation. Also, this chapter includes the experimental set up, protocols, subject demographics, the algorithm used in signal processing, and

statistical analysis. Chapter three presents the results obtained from data analysis and statistical analysis of the results. The conclusions and limitations of this study, and the future work are included in last chapter.

CHAPTER 2

MATERIALS AND METHODS

To obtain and monitor the whole night blood pressure, we utilized non-invasive blood pressure monitor, Finapres. The accessories of the monitor are described in the following section. In this chapter, we also narrowed the summary of our subjects, the monitors and equipment used in this research and the steps of data processing.

2.1. Non-invasive Blood Pressure Monitoring

We chose non-invasive blood pressure monitor to monitor and collect the whole night blood pressure because traditional method to monitor the blood pressure is difficult to be applied on whole night blood pressure monitor. Due to the rapid change in blood pressure in OSA, arm cuff measurement systems in traditional approach doesn't allow to record the variation from the blood pressure. Further, there are also other ways to monitor blood pressure, invasive arterial line, which is known as to be the "gold standard" of blood pressure measurement [15]. However, due to the invasive approach, the risk of monitor blood pressure surge, and that affects the wishes of participants. Thus, we prefer the non-invasive blood pressure monitor and, currently, there are two main types of non-invasive blood pressure monitors, Finapres and arterial tonometry.

Finapres (FINger Arterial PRESsure) is the first generation using finger cuff technology in continuous noninvasive measurement of BP. It is based on the volume-clamp method invented by the Czech physiologist Jan Peñáz. The diameter of peripheral arteries in a finger artery under a cuff is clamped, which keep at a constant diameter in the presence of the changes in arterial pressure during each heartbeat. Inside the finger cuff, an infrared photo-plethysmograph is built to measure the mean of the changes in diameter. The finger cuff keeps the diameter of the underlying arteries

constant by dynamically applying a counter-pressure throughout the cardiac cycle [28]. About the accuracy of Finapres, in the review study, the weighted accuracy of finger arterial pressure measurement comprising a total of 1031 subjects was −0.8±11.7 mmHg (range −48 to 30 mmHg) for systolic pressure, −1.6±8.5 mmHg (range −20.1 to 18.5 mmHg) for mean pressure and −1.6±7.7 mmHg (range −13.4 to 25 mmHg) for diastolic pressure [29].

In the principle of operation in Finapres, the basic device consists of a small finger cuff containing a photoplethysmograph—a light source on one side of the cuff and infrared receiver on the opposite side—with the ability to estimate the blood volume of the finger via the infrared light absorbance. Thus, the signal obtained from the plethysmograph is used in a feedback loop allowing for adjustment of the cuff to keep blood volume constant and the vessels in a constant state of "vascular unloading." By calibrating the Peñáz-technique values with a non-invasive cuff placed on the upper arm, the signal obtained from the plethysmograph is used in a feedback loop allowing for adjustment of artery pressures [15].

On the finger artery, the relationship between internal and external pressure on the arterial wall is given by $P_t = P_a - P_e$, where P_t, is trans-mural pressure, P_a, is arterial pressure, and P_e is external pressure, and all pressure values are referenced to atmospheric pressure. The diameter of the arterial wall varies directly with variations in P_t. As P_t, increases, the arterial wall dilates (viscoelasticity), and vice versa. When P_t, = 0, the artery is stated to be unloaded. Determining the size of the unloaded artery requires an analysis of the viscoelasticity of the finger arteries as expressed in pressure-volume characteristics [30].

However, every device has its pros and cons. The accuracy may be affected by factors such as vascular disease, cold temperature, Raynaud's disease as well as other factors have been reported to contribute to finger plethysmogram failure in about 1% of patients when looking specifically at the Finapres technology [29]. In other study, systemic vasoactive drugs may induce error in the device, which has the significant effect that intravenous phenylephrine had on the reliability of the arterial volume clamp method Additionally, cooling the finger upon which the cuff was placed may cause a higher systolic and diastolic pressure bias [29] [31].

Another continuous noninvasive blood pressure monitor is arterial tonometry in Figure 2.1-1,

which also provide beat-to-beat measurement. It also produces a wave-form similar to Peñáz technique. Tonometry are placed over a superficial artery with sufficient bony support and utilize the pressure-pulse method. The sensor is placed on the radial artery to compress it until the vessel is flattened against the bone but no occluded. On the skin surface, due to contact pressure, a pressure transducer is able to measure arterial blood pressure [15].

Figure 2.1-1 Demonstration of an arterial tonometry device (T-Line, Tensys Medical) *[15]*

For the comparison between Finapres and arterial tonometry, we chose Finapres because the process of recording nocturnal pressure is rapid, non-invasive, and has minimal impact on disrupting sleep. In the tonometry, the technicians need to manual positioning of tonometer over radial artery which can be challenging and dependent on the experience of the technician [15]. Also, during the sleeping, it might be problematic under conditions with significant patient motion [28]. As a result, it would be less desirable for monitoring patient's blood pressure throughout the night. The motion of the sensor through the night is less problematic, as both the finger probe and the wrist unit are securely fastened to the patient finger and wrist in Figure 2.4-1, respectively. In the review study,

there is another non-invasive blood pressure monitor, Pulse transit time. However, it isn't fully developed to correlate pulse transit time to blood pressure [15].

2.2. Blood Pressure Measure and Mode Monitor

In this study, the unit of the noninvasive blood pressure consists of a touch screen monitor, a wrist, module which is worn on the wrist, finger cuffs and a heart reference system that can be connected to the wrist unit [32].

In Nexfin HD monitor (BMEYE, Amsterdam, Netherlands), it is used in this study to monitor beat to beat blood pressure. The advantage of the Nexfin monitor include portable, use easily and a graphical interface with tough screen which utilize the volume-clamp methodology of Peñáz [33]. Also, on the monitor, users allow to view the hemodynamic value in each heartbeat, which includes systolic pressure, diastolic pressure, mean arterial pressure, pulse rate, stroke volume, cardiac output and systemic vascular resistance [32]. Further, Nexfin provides accurate pressure measurement with good within-subject precision. In this study, we have only used the blood pressure waveform.

The finger cuffs employs a photo electric plethysmograph for detecting the blood flow and is also equipped with inflatable thin transparent plastic bladder controlled by the wrist unit through air hose connector [32]. The photoelectric part comprises an infrared light-emitting diode (LED) of wavelength 950 nm and an infrared photodiode for detection. The cuff can be wrapped around finger such that LED and photodiode are positioned on the opposite sides of the finger to allow optimal signal [30]. The infrared radiation is able to be utilized in the electromagnetic spectrum because infrared absorption by blood is far more sensitive than its absorption by bone and other blood less tissues [34].

In the heart reference systems, to avoid the earth's gradient changes in blood pressure due to hydrostatic effect during invasive blood pressure measurement or using sphygmomanometer, the pressure monitoring sensor is kept at heart level. While using Nexfin blood pressure monitor, an integrated heart reference system (HRS) is used to allow accurate heart level blood pressure measurement with free movement of hand irrespective of its vertical height with respect to the heart [32].

2.3. Subject Demographics

The data was collected from sleep apnea patients who were undergoing 8 hour polysomnography (except one subject who was subjected to have partial Polysomnography) at Sleep Consultants, Inc. (Fort Worth, TX). These subjects were given complete instructions about the experiment and signed informed consent. The data were collected according to the protocols which will be described later in Section 2.6 of this chapter. The subject demographics for the groups with corresponding mean and standard deviation (σ) are as shown in Table 2.3-1. The average age, height, weight, BMI and AHI for subjects were 53.2± 5.7 years, 177.6 ± 9 cm, 238 ±49.1 lb., and 63.5±28.7.

Table 2.3-1 Subject Demographic

SUBJECT	GENDER	AGE	HEIGHT(CM)	WEIGHT(LB)	BMI	AHI
1	M	50	175.3	175	25.8	63.6
2	M	56	182.9	298	40.4	105.4
3	M	47	175.3	235	34.7	77.4
4	F	57	162.6	248	42.6	21.8
5	M	50	180.3	174	24.3	18.3
6	M	45	162.6	210	36.0	82.2
7	F	48	175.3	325	48.0	42.3
8	M	61	188.0	215	27.6	91.3
9	M	56	182.9	290	39.3	87.3
10	M	62	190.5	210	26.2	44.9
Summary		53.2 ± 5.7	177.6 ± 9.0	238.0 ± 49.1	34.5 ± 7.8	63.5 ± 28.7

2.4. Blood Pressure Measurement

The data analyzed in this study was collected by our lab and details of which is reported in the thesis [32] from the former graduated student, Raichel Mary Alex. In the subsections below, the unit of blood pressure measurement, the principles of operation, and the process of applying it to a subject are described [32].

2.4.1. Finger Cuff Positioning

A finger cuffs, an important part of the Nexfin monitor, are wrapped around a subject's finger to detect arterial blood pressure. The proper placement of finger cuff is critical to get reliable blood pressure measurement. Thus, it can be applied to any of the larger fingers (middle, index or ring finger) with an appropriate size Nexfin Finger cuff, has to be placed in center between the two knuckles of the middle phalanx and is secured with a small Velcro strip.

2.4.2. Applying Heart Reference System

The monitor has a sensor known as the heart reference system for avoiding the hydrostatic effect due to the distance of finger from the heart, which the subjects enable to move their hands freely and not have to keep them at the heart level. Calibration of the heart reference system should be done to zero the HRS before attaching the sensor to the subject's arm. The pressure transducer is fastened to a finger strap and is wrapped around the middle phalanx of an adjacent finger to which the finger cuff is wrapped [32]. Figure 2.4-1 shows the application of Heart Reference System.

Figure 2.4-1 Application of Heart Reference System

2.4.3. Wrist Unit Placement

The wrist unit has to be wrapped around the wrist of the subject and connects to the finger cuff and the heart reference system on one side and the other side is contacted with the Nexfin monitor. The wrist unit is connected to the Nexfin monitor using a large connector at the end of the

wrist unit cable [32]. The connection of wrist unit to the pressure monitor is as shown in Figure 2.4-2.

Figure 2.4-2 Wrist Unit Connection to Monitor and Finger Cuff

2.5. Data Acquisition

2.5.1. Computer-Based Data Acquisition Unit

Data Acquisition Unit (DAQ) was used to collect the analog outputs from all the physiological monitors. DAQ 6024E, manufactured by the National Instruments (Austin, TX), was used in data collection, which has two 12 bit analog output lines, 8 digital Input/output lines, and two 24 bits counters. CB-68 LP which interfaces with the DAQ is the output from the Nexfin with a printed circuit board. By using custom program with Lab View 9.0 software, DAQ collected the input signal at 1 kHz of sampling frequency. A custom-designed Lab View program saved the data in lvm file format which was imported into MATLAB for signal analysis [32]. The DAQ and the interface from National Instruments are shown below in Figure 2.5-1.

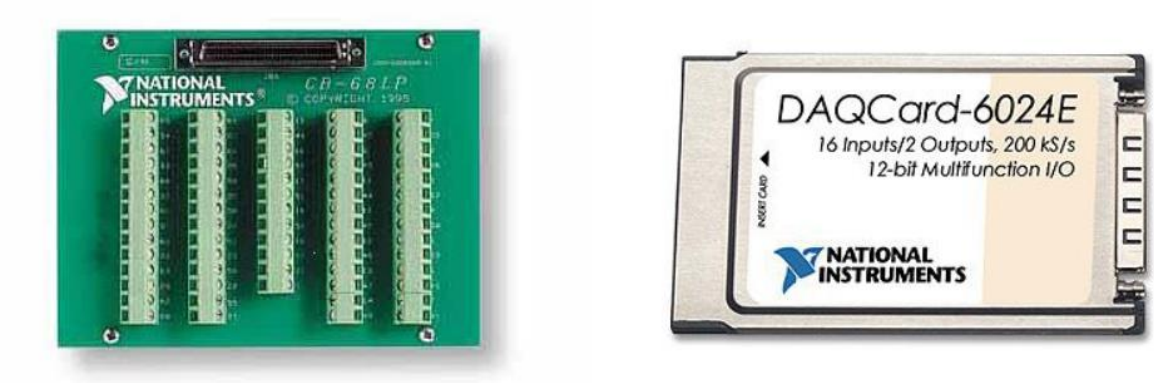

Figure 2.5-1 Connector Block and DAQ

2.6. Experimental Set Up for Sleep Apnea Subjects

2.6.1. Experimental Set Up

The experimental data for this study was collected at our collaborating accredited sleep laboratory (Sleep Consultants Inc., Fort Worth, Texas). For this purposed, full night polysomnography (PSG) was conducted on subjects who have previously been diagnosed with having obstructive sleep apnea. As part of PSG, the parameters monitored included electroencephalogram (EEG), electroculogram (EOG), electromyogram (EMG), oral and nasal airflow, chest and abdominal movement, leg movements, snoring, blood oxygen saturation and a video monitoring of the subject using Sandman sleep study system (Natus Medical Inc., Pleasanton, California). In addition to PSG data, patient nocturnal blood pressure was continuously recorded during the sleep study as described above. A synchronization signal was used for both Sandman software and the DAQ board to ensure that the alignment of the blood pressure data with the PSG data [32].

2.6.2. Apnea Scoring

A certified sleep lab technician blind to the objective of this study scored the PSG data after the completion of the PSG. The tasks of scoring included sleep stage determination and the identification of apneas and hypopneas during the sleep. The resulting PSG file also contained the start time, duration of the stages, and the length of apnea events. Once scoring was done, the relevant PSG data was imported to MATLAB software to create a graphical representation of these

stages and events.

2.7. Data Processing

The raw blood pressure data was extracted from the data recorded using the Lab VIEW program and processed in Matlab. Before the data processing, it was synchronized between polysomnography and other monitors. The processing steps and the features extracted from the waveform are narrowed in details in followings.

2.7.1. Meaning of Systolic Blood Pressure, Diastolic Blood Pressure, Mean Blood Pressure, and Pulse Pressure

The definition of systolic blood pressure is the maximum pressure experienced in the aorta when the heart contracts and ejects blood into the aorta from the left ventricle; the diastolic blood pressure is defined as the minimum pressure experienced in the aorta when the heart is relaxing before ejecting blood into the aorta from the left ventricle [35].

The definition of the mean arterial pressure (MAP) is the time-weighted integral of the instantaneous pressures derived from the area under the curve of the pressure-time waveform of one entire cardiac cycle [36], which directly corresponds to cardiac output, arterial elasticity, and peripheral vascular resistance [16]. It is a measure of tissue perfusion, essentially independent of pulse pressure, and can be used to calculate other hemodynamic variables. This is important when the blood pressure is adjusted by use of drugs to maintain an adequate perfusion pressure [36]. The MAP formula is described in Section 2.7.5 and is widely-accepted.

Pulse pressure (PP) is the consequence of intermittent ventricular ejection from the heart. PP is influenced by some cardiac and vascular factors. In addition to the pattern of left ventricular ejection, the determinants of PP (and SBP) are the cushioning capacity of arteries and the timing and intensity of wave reflections [37].

2.7.2. Detection of Systolic and Diastolic Blood Pressure

The raw data was processed with Matlab and used “findpeaks” function. Three settings in “findpeaks” function are used, which are min peak distance, min peak height, and min peak prominence. The meaning of these settings is showed in Figure 2.7-1. Since different sleeping stages, apnea severity, and physiological and anatomical different are effected the blood pressure waveform, the settings of “findpeaks” are varies not only in each individuals but in different period of data. Thus, by utilized nowadays strong computing power of computer, I created an algorithm to generate the setting automatically. The assumption of the algorithm is that the highest quantity of peaks should be found in regular blood pressure. First, in each subject, the data was dismantled into 50 pieces. I tried the settings limitation manually and generated the upper and lower bond of those setting. Second, to get the best value of each settings, each piece will go through all the possibility in each setting to include all the possibility and I collect all the quantity of peaks at the mean time. Third, the quantity result is analyzed and choose the point that numbers of peaks started to drop and find the corresponding value of setting. Last, with applying the previous best setting to the next setting, repeating the process on all the setting in each piece. Thus, after executing with all the setting steps by steps, I can obtain the best settings for this piece and rerun “findpeaks” function with the setting to get the peaks’ value and location. The different between detecting systolic and diastolic is set the data to opposite number. By flipping the data to the opposite number, the diastolic point changes from trough to peak, which is able to apply with “findpeaks” method.

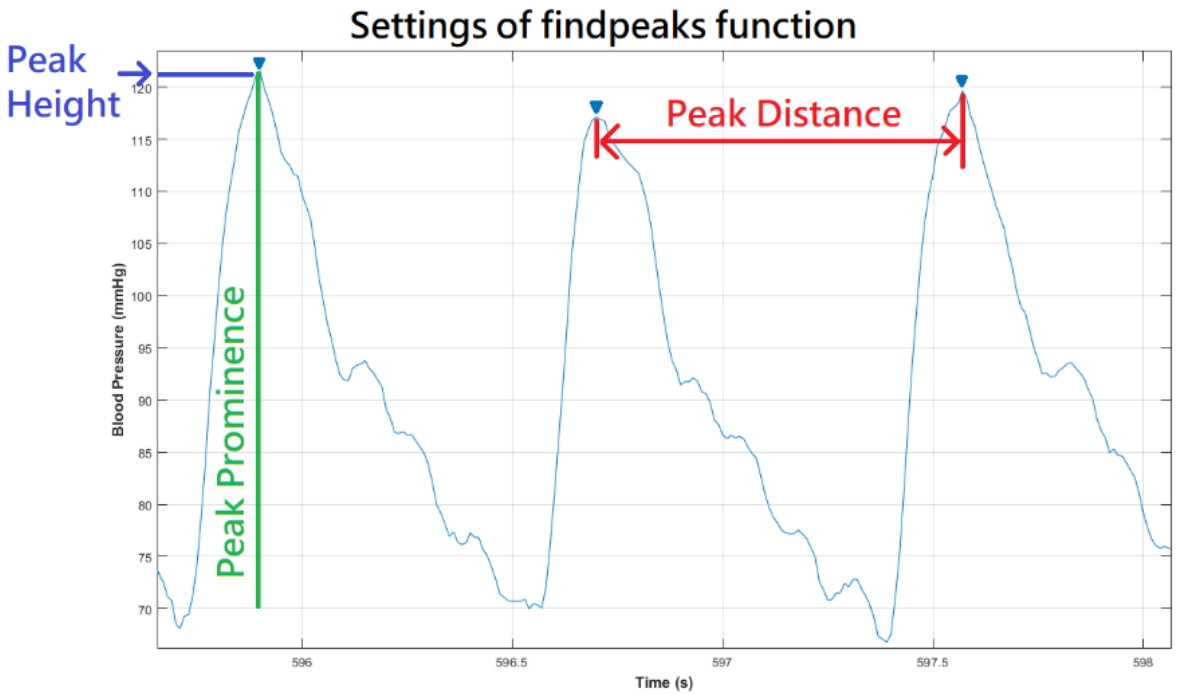

Figure 2.7-1 Findpeaks Setting Diagram

2.7.3. Removal of Noisy and Calibration from Finapres

In Finapres, the calibration is executed in a non-specific points in time, but frequently. These calibration period can last anywhere between approximately 2s to 3s, which Figure 2.7-2 shows an example of the calibration period, in blue double arrow. Since during the calibration periods no measure of BP is made, these periods need to be identified in the whole night BP data and be removed from analysis. After getting the SBP and DBP value from Section 2.7.2, the first steps is manually removed the noisy at either the beginning or the end Figure 2.7-3. For detecting the calibration period and noisy occurred between the recordings, the algorithm is based on the assumption that the majority of SBP and DBP are correct, which the distance between each SBP and DBP should fall into a high concentration distribution. Therefore, the distribution includes with the noisy and calibration will be a right-skewed graph in Figure 2.7-4. Then, I calculated the 99th percentiles in both SBP and DBP distribution and collected those period beyond the 99th percentile as calibration period. The proposed calibration detection method is able to detect around 90% of calibration periods. However, due to unexpected situation, e.g. accidental dislocation of the cuff on

the finger, one still needs to manually review the results and remove the noisy and in calibration periods.

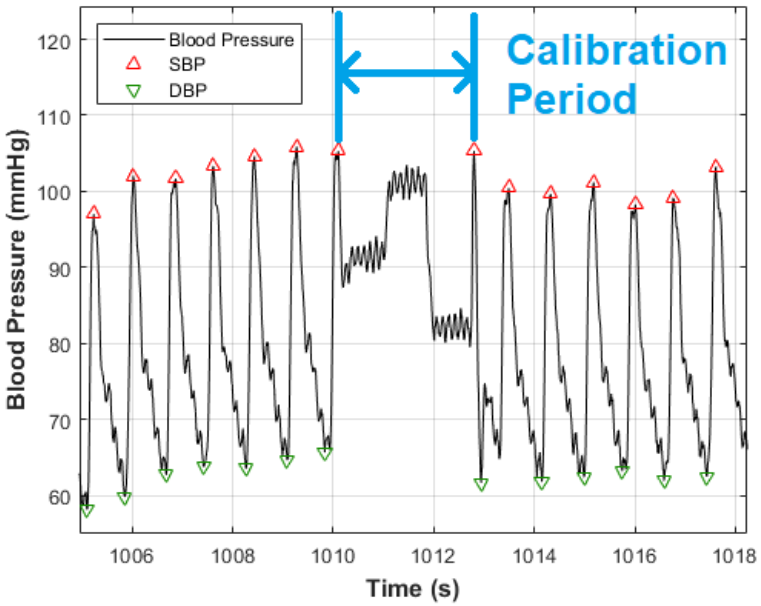

Figure 2.7-2 A example of calibration period

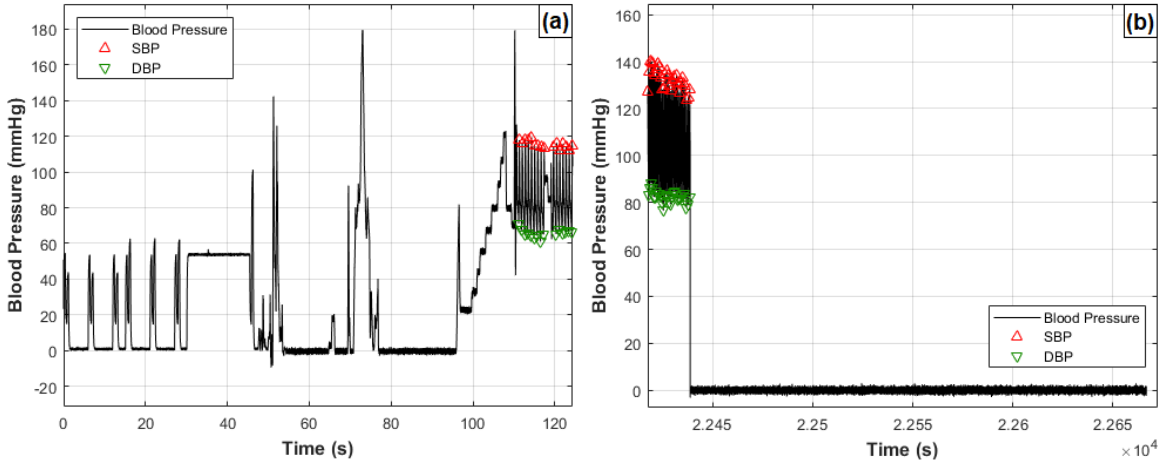

Figure 2.7-3 A example of noisy and calibration; (a) occurred at the beginning in the whole night recording; (b) (a) occurred at the end in the whole night recording

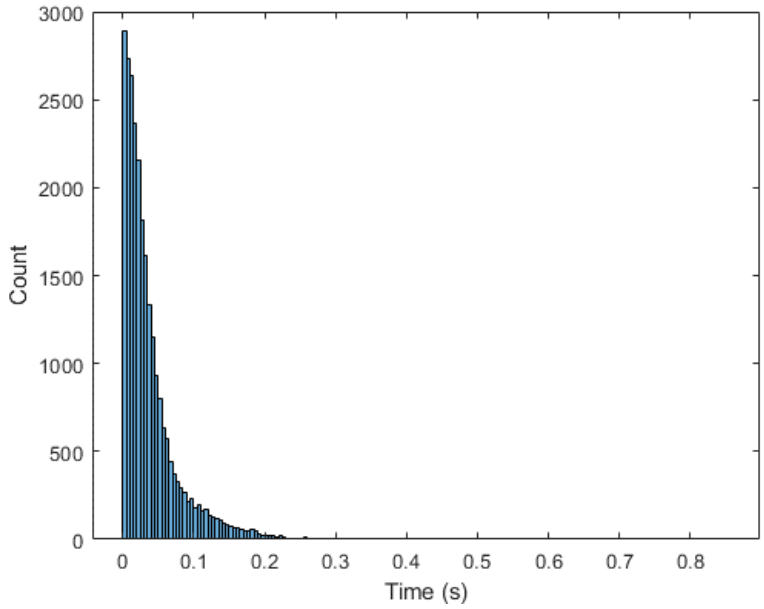

Figure 2.7-4 A example of the distance between each SDB points in a right-skewed distribution

2.7.4. Interpolation of the Blood Pressure Peaks

After obtaining all the locations and values of the systolic and diastolic pressure, regarding to the type of blood pressure, I used the “spline” function in Matlab, which is a cubic spline function. An example of the interpolation in SBP, DBP, MAP and PP are in illustrated in Figure 2.7-5.

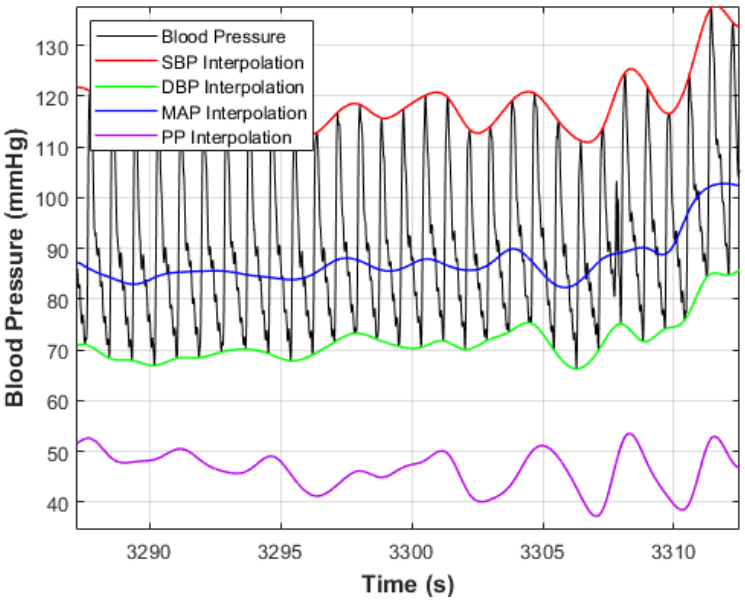

Figure 2.7-5 Interpolated Blood Pressure

2.7.5. Computation of Pulse and Mean Arterial Pressure

The MAP is estimated using the following formula:

$$\mathrm{MAP} = \left(\frac{2}{3}\right)\mathrm{DBP} + \left(\frac{1}{3}\right)\mathrm{SBP}$$

where MBP is the mean arterial blood pressure, DBP is the diastolic blood pressure, and SBP is the systolic blood pressure. The classic pulse pressure coefficient (PPC), 1/3, originated from the work of Gauer, who measured intra-arterial iliac pressure in young healthy male subjects at rest [36].

Pulse pressure (PP) is the difference between the systolic and diastolic blood pressures. The formula of PP is calculated by the following:

$$\mathrm{PP} = \mathrm{SBP} - \mathrm{DBP}$$

A proof from the study is that the systolic blood is usually approximately 120 mm Hg and the diastolic blood pressure is often approximately 80 mm Hg. Thus, normal pulse pressure is approximately 40 mm Hg. A change in pulse pressure (△ PP) is proportional to stroke volume change (△ V) but inversely proportional to arterial compliance (C):

$$\triangle\,\mathrm{PP} = \frac{\triangle\,\mathrm{V}}{\mathrm{C}}$$

Since the change in volume is due to the stroke volume of blood being ejected from the left ventricle (SV), we can approximate pulse pressure as:

$$\mathrm{PP} = \frac{\mathrm{S\,V}}{\mathrm{C}}$$

A normal young adult at rest has a stroke volume of approximately 80 mL. Arterial compliance is approximately 2 mL/mmHg, which confirms that normal pulse pressure is approximately 40 mmHg [35].

2.7.6. Baseline Calculations

For the analysis irrespective of the effects of sleep stages, we first needed to determine the Baseline as blood pressure level when no apnea is present. For each subject, we identified one

minute of normal breathing during the night at which no SDB was occurred. The beat-to-beat blood pressure pulses during this interval were processed to extract SBP and DBP trajectories; referring to these trajectories as SBP Baseline and DBP Baseline. To obtain aggregate SBP and DBP Baselines for the subject sample population, the individual subject SBP and DBP Baselines were respectively averaged using a sample-by-sample averaging.

Since the blood pressure varies regards to each sleeping stages, the Baseline from each sleep stage is defined as greater or equal to 60 sec. and no apnea event occurred in that period. In addition, either side of the window in this 60 sec. is away from the apnea event for at least 30 sec. The algorithms of extracting SBP and DBP are similarly above. Figure 2.7-6 indicates a baseline in Stage 2. The blue dashed line is the center in this period of Stage 2 and the blue solid line reveals the before and after 30s from the center, respectively. The period between two blue solid line is inside Stage 2 (green double arrow) and then, we collected this period as the Baseline in Stage 2.

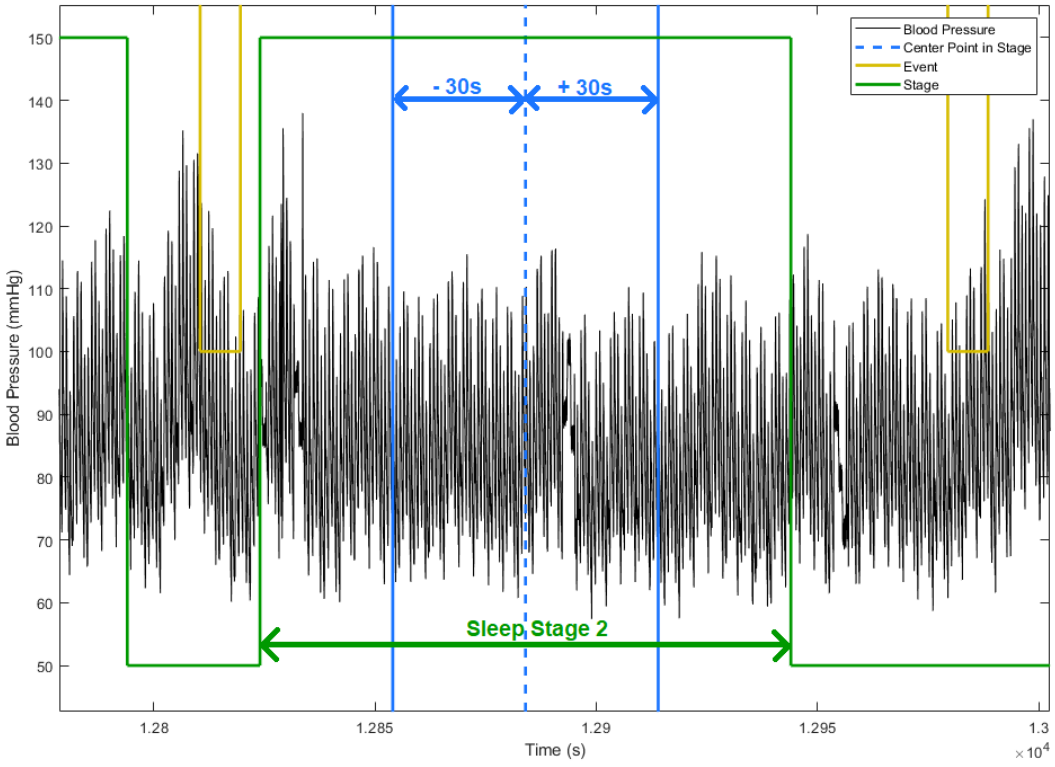

Figure 2.7-6 A example for the Baseline collection in sleep stage 2

2.7.7. Segmentation Method of Whole Night Blood Pressure

When considering the variations of BP elicited by apnea events, and irrespective of the sleep stages, we opted to consider apnea episodes that were at least 30 s apart from the next respiratory event. Figure 2.7-7 illustrates an example of an apnea episodes that was analyzed from one of the subjects. As shown, the apnea event is not succeeded by another apnea event for at least 30s. In Figure 2.7-7, the arrows labeled with -30s and +30s denote the temporal width of the interval over which we analyzed the blood pressure (BP) values to capture the impact of each apnea episode. The resulting SBP (red) and DBP (green) envelopes were used for aggregating the BP variations. By following this criteria, we aggregated all the apnea epochs by aligning the end of the apnea events, as shown in Figure 2.7-8.

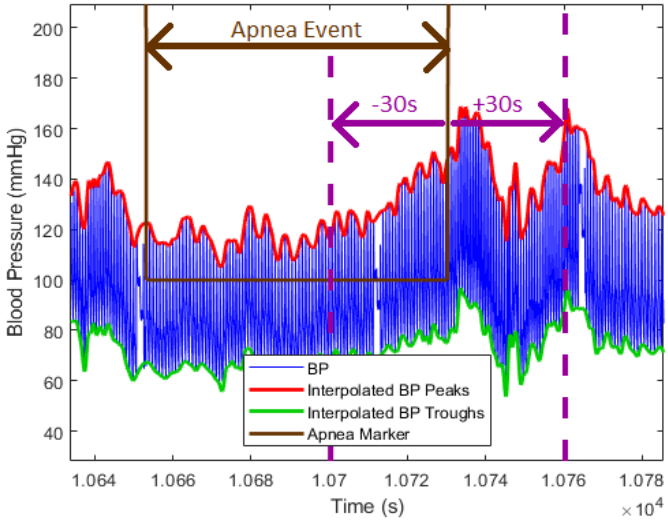

Figure 2.7-7 A selected apnea event with no any other event occurred in the next 30 s

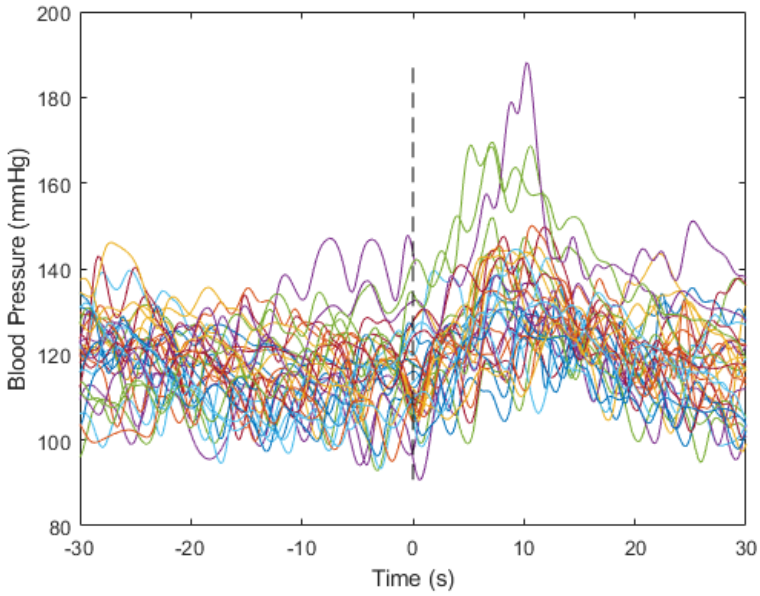

Figure 2.7-8 Aggregated all apnea epochs: the dash line represents the end of the apnea event.

In contrast, when analyzing the blood pressure variations during the sleep stages, we not only selected the events met the previous criteria, but also the whole period of each event is in a single sleeping stage. In totality of all subjects' data, there were around 20% of cross stages events. Those cross stages epochs were excluded from analysis to allow more accurate assessment of the effect of each stage on blood pressure surges elicited by apnea events during that stage. Figure 2.7-9 shows an example of selected epochs. As shown, the apnea event (marked with blue solid line) is away from the next respirator event by at least 30 sec on both side for 30s (red arrow) and the entire period of the event is within the Stage 2 (green arrow). The number of events varies from stage to stage. To generate the meaningful statistic result, if the number of events in any of the stages for a subject was less than 5, we did not consider those for analysis of that subject's data, but when analyzing all subjects' data, we included those results.

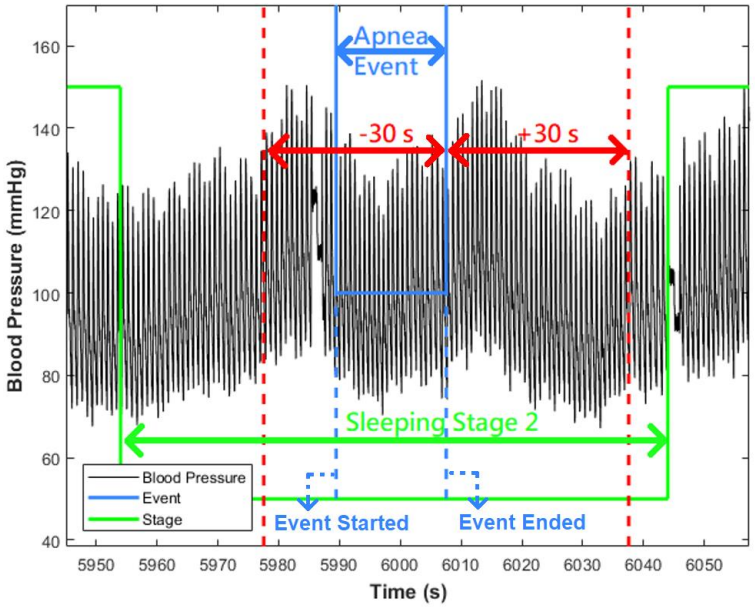

Figure 2.7-9 An apnea event in Stage 2 from beginning to the end and away from other apnea epochs for 30s on both side.

2.7.8. Calculation of Slopes from the Blood Pressure Variation

To assess the speed of surges in blood pressure that are elicited by apnea events, the slope of surges in systolic and diastolic pressure rises for each selected apnea event – those that were selected in accordance with the criteria described in Section 2.7.7 – were computed. Although we only collected the data for 30s before and after the end of apnea episodes in Section 2.7.7, the starting points may occur beyond 30s window. Figure 2.7-10a indicates the majority of the started points we analyzed and Figure 2.7-10b reveals the situation the duration of the event is over 30s during calculating the slope. In these figures, the red circle is the starting point and the green circle as the end point. The event markers were marked by the sleep technician. The start point is the time where the event started and the value is calculated as the mean value of 5 sample points before the start points to compensate imprecise marking. For the end point, we detected the highest peak which occurred after the end of the apnea event (e.g., yellow lines in Figure 2.7-10 a and b mark the end of the apnea even). Similarly, to reduce the marking inaccuracy, we got the average from the peak points and 2 sample points before and after the highest peak (i.e., average of five points with the highest point temporally centered between them).

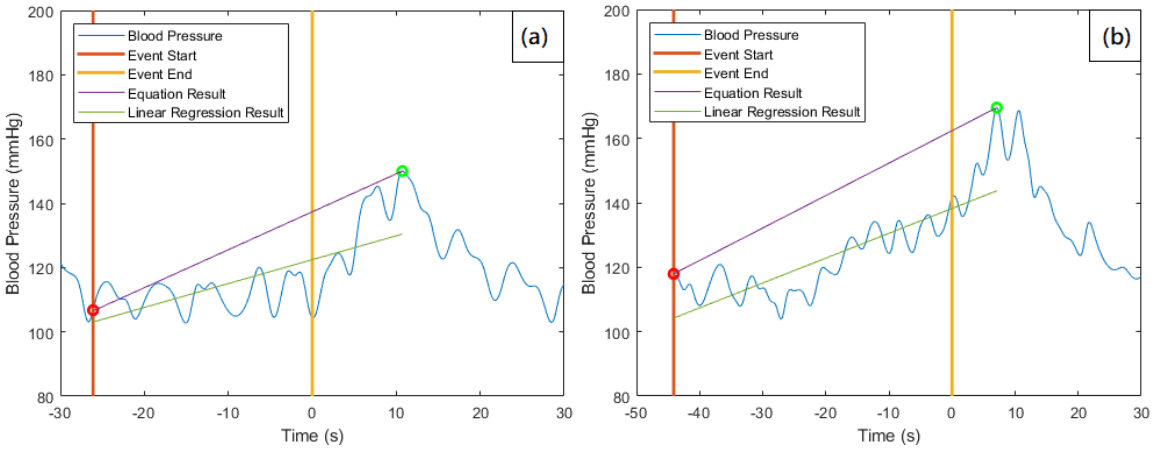

Figure 2.7-10 Slope calculation for a single event: (a) the duration of the event is less than 30s; (b) the duration of the event is beyond 30s

For slope calculation, we considered two methods. The first approach is utilizing the slope equation, which only taking the start and the endpoints to come up with the slope and interception. The other is applying with the linear equation, a least-squares fit, by using all the points from the beginning of the event to the peak point. To have better understanding, we generated both results and found out the slopes from linear regression (green line in Figure 2.7-10) are typically less than the results from slope equation (purple line in Figure 2.7-10), due to the compensation from the events duration. In Figure 2.7-10, we can clearly visualize the findings. Thus, we will apply the slope equations for the analysis.

2.7.9. Statistical Analysis

In this study, we analyzed the data from each subject and also aggregated the BP variations for both normal breathing (i.e., the Baseline) and apnea epochs. Then, we calculated the mean and standard deviation values for the blood pressure features that were considered (i.e., MBP, PP, SBP, and DBP). A two-tailed t-test was used to determine whether the mean of the peak pressures during apnea episodes were significantly different from the Baseline means for each of the BP features. Also, in the slope analysis, we utilized a t-test not only in the comparison between the slope in events and the Baseline, but between each sleep stage. Additionally, we divided the data set regarding their sleep stages to analyze the change between sleeping stages. During the extracting process, we also utilized the slope equation to calculate the slope for each apnea events to determine the effect of

blood pressure variation. In addition, the sleep study data is analyzed using two-tailed t-test, which considering p-value < 5% as significant difference. We also generated a 95 % confidence interval for each data set. To investigate the effect of sleep stages on BP variations, ANOVA test was applied to the data after examining the normality and homogeneity of the measured features (i.e., SBP, DBP, MBP, and PP). The null hypothesis (i.e., h0) tested using ANOVA test is the mean value in the stages are all the same. The alternative hypothesis (i.e., h1) is that not all the mean values are the same. If the null hypothesis was rejected, we utilized Tukey Test to determine the differences between the mean values of the BP features across the sleep stages. For any of the BP features (i.e., SBP, DBP, MBP, or PP) that the data failed the homogeneity and normal distribution tests, a rank-based nonparametric test, Wilcoxon rank sum test, was applied to find the difference in the median of the feature across sleep stages.

CHAPTER 3

RESULTS

In this section, the results are all based on the criteria described in Section 2.7. In this chapter, there are three main fields, Baseline analysis, the comparison between OSA events and the Baseline, and comparisons of BP surge during apnea episodes in sleep stages. Also, the last two fields are included with the slope analysis based on Section 2.7.8. The discussion is in Chapter 4.

3.1. Baseline Effect on Blood Pressure Features

As was described in Section 2.7.6 above, the Baseline blood pressure traces were all selected from the beginning of the night when subjects were still awake.

3.1.1. Systolic Blood Pressure (SBP)

The first row of data in

Table *3.1-1* shows the average results of blood pressure during normal breathing for all subjects, while the rows below that show the results for each subject. Further, the entry in the Col. 3 represents the BMI and the 4th column reveals the AHI to the subjects. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in

Table *3.1-1*. The average SBP in normal breathing from the accumulated results is 128.7 mmHg in Col. 5 in the first row. Figure 3.1-1 provides the visual form to summarize the blood pressure distribution from the subjects and the accumulated results.

Table 3.1-1 Subjects information and normal breathing results

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Subject	**Age (yr.)**	**BMI (m^2/kg)**	**AHI (event/hr.)**	**Average BP in Normal Breathing (mmHg)**
Accumulated results		34.5 ± 7.8	63.5 ± 28.7	128.7 ± 12.7
1	50	25.8	63.6	120.1 ± 4.2
2	56	40.4	105.4	141.2 ± 5.8
3	47	34.7	77.4	114.9 ± 4.0
4	61	27.6	91.3	138.4 ± 3.2
5	45	36.0	82.2	118.4 ± 4.3
6	57	42.6	21.8	148.2 ± 4.0
7	48	48.0	42.3	109.5 ± 3.5
8	56	39.3	87.3	131.2 ± 4.5
9	62	26.2	44.9	136.9 ± 3.3
10	50	24.3	18.3	128.0 ± 2.2

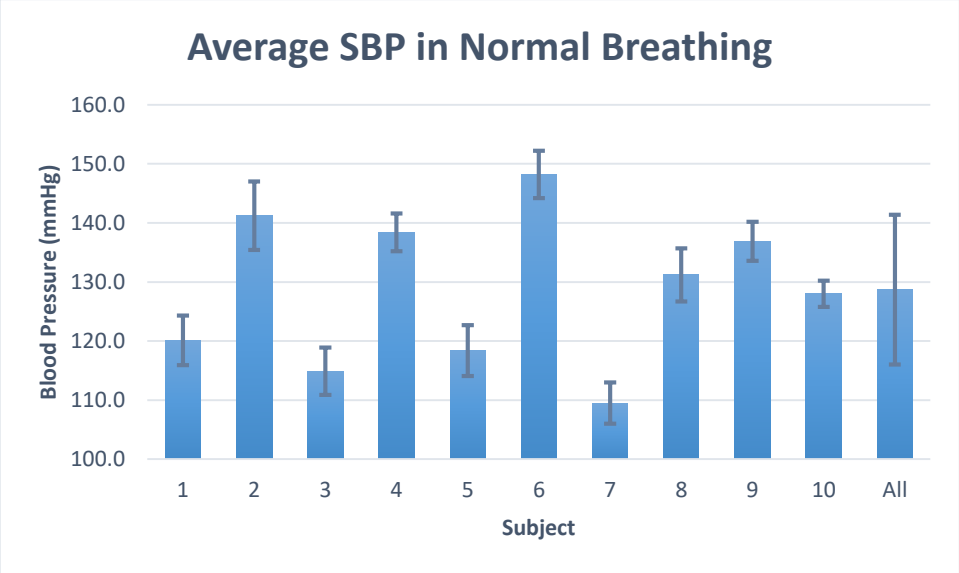

Figure 3.1-1 Average SBP distribution of all subjects in normal breathing

Figure 3.1-2a displays a plot of the SBP trajectories during the Baseline normal breathing from all of the subjects. Further, Figure 3.1-2b shows the result of aggregating these trajectories as well as the corresponding 95% confidence interval (CI), assuming Gaussian Distribution of the mean of each SBP sample.

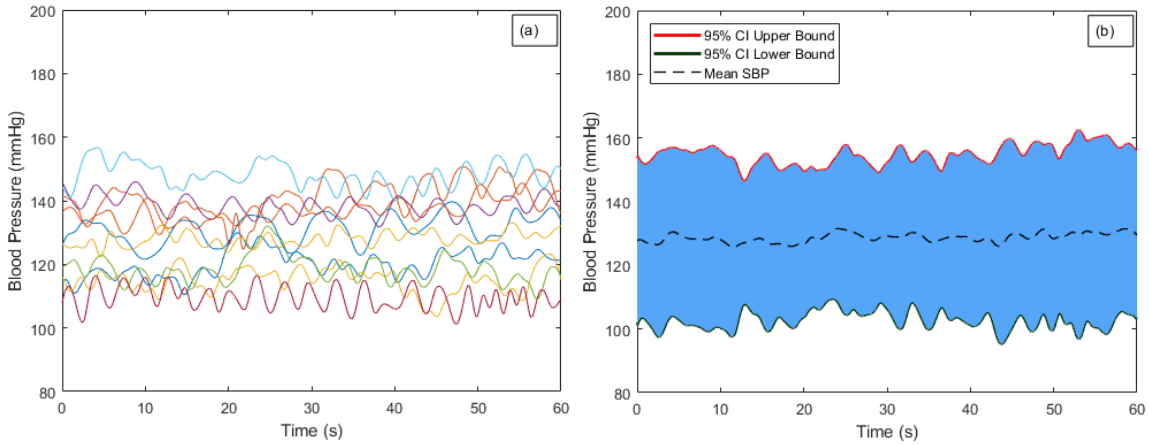

Figure 3.1-2 Aggregated Baseline of blood pressure signals from all the subjects: (a) SBP Baseline recordings for each of the 10 subjects; (b) Aggregated SBP Baseline with 95% confidence interval envelope.

3.1.2. Diastolic Pressure (DBP)

Similar to the results reported in Section 3.1.1, above, the first row of data in Table 3.1-2 shows the aggregate results of DBP during Baseline normal breathing from all subjects, while the rows below that show the results for each subject. Specifically, the entry in the 3rd column represents the BMI and the 4th column reveals the AHI to the subjects. The average DBP in normal breathing from the accumulated results is 69.1 mmHg in Col. 5 in the first row. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in Table 3.1-2. Figure 3.1-3 provides the visual form to summarize the blood pressure distribution from the subjects and the accumulated results.

Table 3.1-2 Subjects information and normal breathing results

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Subject	**Age (yr.)**	**BMI (m^2/kg)**	**AHI (event/hr.)**	**Average BP in Normal Breathing (mmHg)**
Accumulated results		34.5 ± 7.8	63.5 ± 28.7	69.1 ± 5.8
1	50	25.8	63.6	66.8 ± 2.5
2	56	40.4	105.4	67.2 ± 2.3
3	47	34.7	77.4	68.8 ± 3.0
4	61	27.6	91.3	70.4 ± 1.4
5	45	36.0	82.2	76.2 ± 2.7
6	57	42.6	21.8	76.2 ± 1.9

Col. 1	COL. 2	COL. 3	COL. 4	COL. 5
Subject	**Age (yr.)**	**BMI (m^2/kg)**	**AHI (event/hr.)**	**Average BP in Normal Breathing (mmHg)**
7	48	48.0	42.3	56.8 ± 2.6
8	56	39.3	87.3	72.6 ± 3.4
9	62	26.2	44.9	68.7 ± 1.5
10	50	24.3	18.3	67.1 ± 1.5

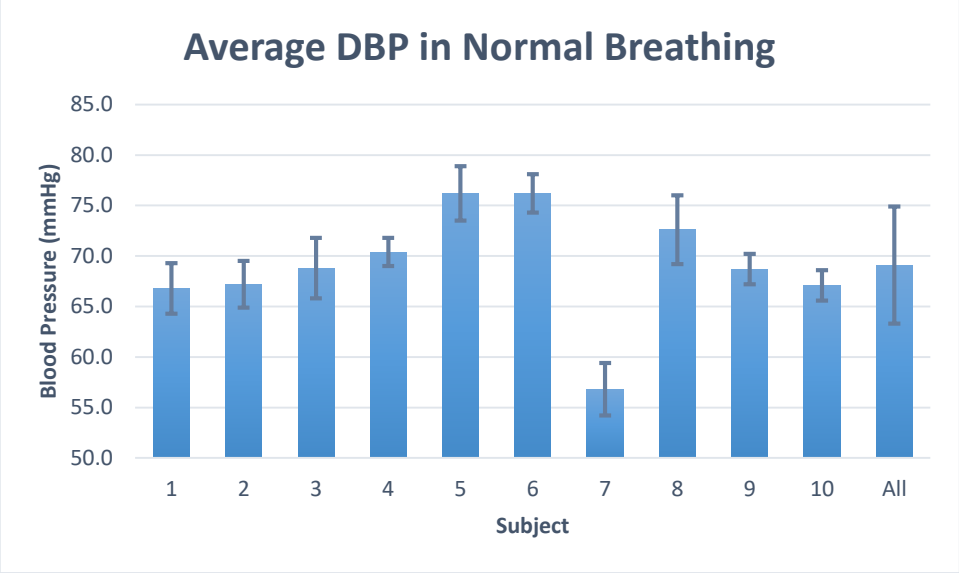

Figure 3.1-3 Average DBP distribution of all subjects in normal breathing

Figure 3.1-4a displays a plot of the DBP trajectories in the normal breathing from the all the subjects and Figure 3.1-4b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

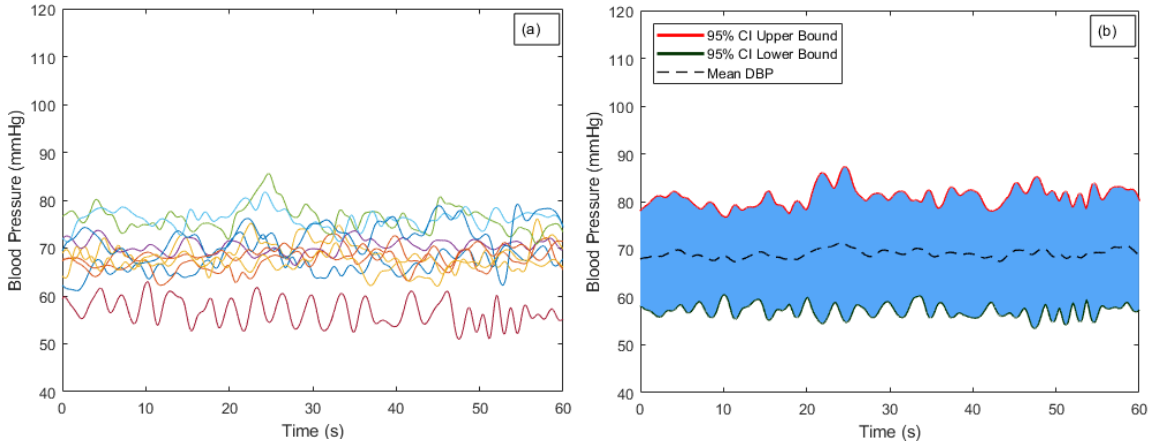

Figure 3.1-4 Aggregated Baseline of blood pressure signals from all the subjects: (a) DBP Baseline recordings for each of the 10 subjects; (b) Aggregated DBP Baseline with 95% confidence interval envelope.

3.1.3. Mean Arterial Pressure (MAP)

In Table 3.1-3, the first row of data shows the aggregate results of MAP in normal breathing from all subjects, while the rows below that show the results for each. Additionally, the entry in the 3rd column represents the BMI and the 4th column reveals the AHI to the subjects. In Col. 5 in the first row, the average MAP in normal breathing from the accumulated results is 89.0 mmHg. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in Table 3.1-3 and Table 3.1-2. Figure 3.1-5 provides the visual form to summarize the MBP distribution from the subjects and the accumulated results.

Table 3.1-3 Subjects information and normal breathing results

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Subject	**Age (yr.)**	**BMI (m^2/kg)**	**AHI (event/hr.)**	**Average BP in Normal Breathing (mmHg)**
Accumulated results		34.5 ± 7.8	63.5 ± 28.7	89.0 ± 7.0
1	50	25.8	63.6	84.7 ± 2.8
2	56	40.4	105.4	92.0 ± 3.3
3	47	34.7	77.4	84.2 ± 3.2
4	61	27.6	91.3	93.0 ± 1.7
5	45	36.0	82.2	90.3 ± 3.1
6	57	42.6	21.8	100.2 ± 2.1
7	48	48.0	42.3	74.4 ± 2.3
8	56	39.3	87.3	92.1 ± 3.4
9	62	26.2	44.9	91.5 ± 1.6
10	50	24.3	18.3	87.4 ± 1.1

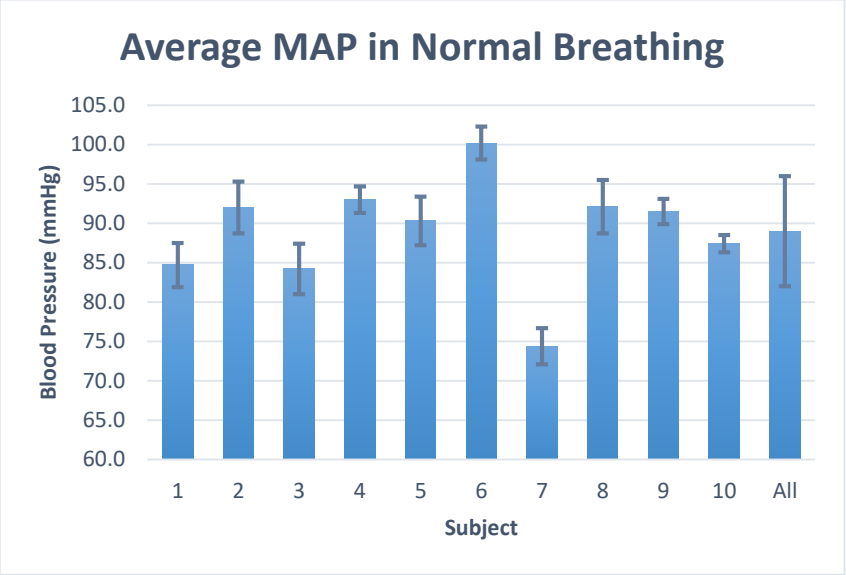

Figure 3.1-5 Average MAP distribution of all subjects in normal breathing

Figure 3.1-6a displays a plot of the DBP trajectories in the normal breathing from the all the subjects and Figure 3.1-6b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

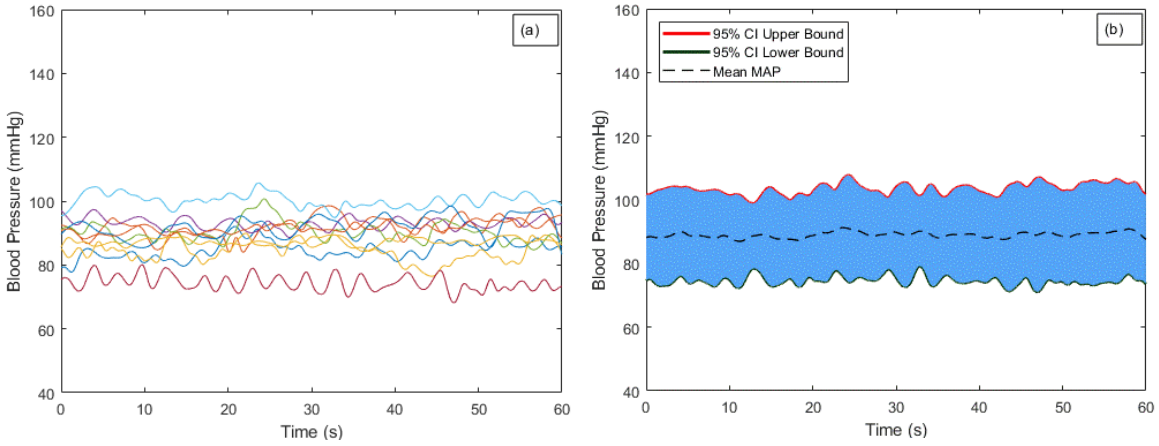

Figure 3.1-6 Aggregated Baseline of blood pressure signals from all the subjects: (a) MAP Baseline recordings for each of the 10 subjects; (b) Aggregated MAP Baseline with 95% confidence interval envelope.

3.1.4. Pulse Pressure (PP)

The first row of data in Table 3.1-4 shows the aggregate results of PP in normal breathing for all subjects, while the rows below that show the results for each. Additionally, the entry in the 3rd

column represents the BMI and the 4th column reveals the AHI to the subjects. In Col. 5 in the first row, the average PP in normal breathing from the accumulated results is 59.6 mmHg. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in Table 3.1-4Table 3.1-2. Figure 3.1-7 provides the visual form to summarize the blood pressure distribution from the subjects and the accumulated results.

Table 3.1-4 Subjects information and normal breathing results

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Subject	**Age (yr.)**	**BMI (m^2/kg)**	**AHI (event/hr.)**	**Average BP in Normal Breathing (mmHg)**
Accumulated results		34.5 ± 7.8	63.5 ± 28.7	59.6 ± 10.9
1	50	25.8	63.6	53.2 ± 2.9
2	56	40.4	105.4	73.9 ± 4.5
3	47	34.7	77.4	46.0 ± 2.8
4	61	27.6	91.3	68.1 ± 2.7
5	45	36.0	82.2	42.1 ± 2.9
6	57	42.6	21.8	72.0 ± 3.6
7	48	48.0	42.3	52.7 ± 3.7
8	56	39.3	87.3	58.7 ± 4.1
9	62	26.2	44.9	68.0 ± 3.3
10	50	24.3	18.3	60.9 ± 2.8

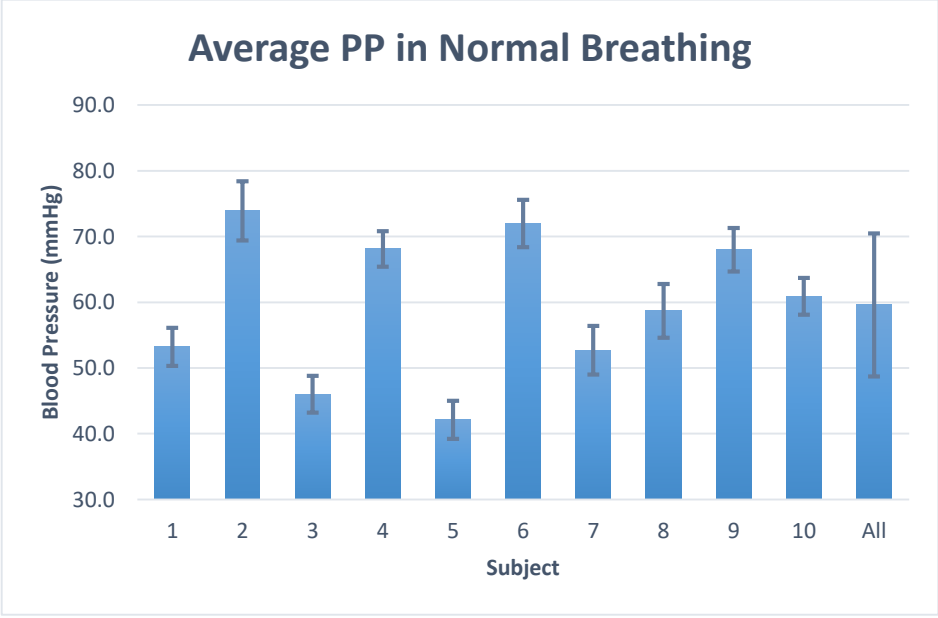

Figure 3.1-7 Average PP distribution of all subjects in normal breathing

Figure 3.1-8a displays a plot of the PP trajectories in the normal breathing from the all the subjects and Figure 3.1-8b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

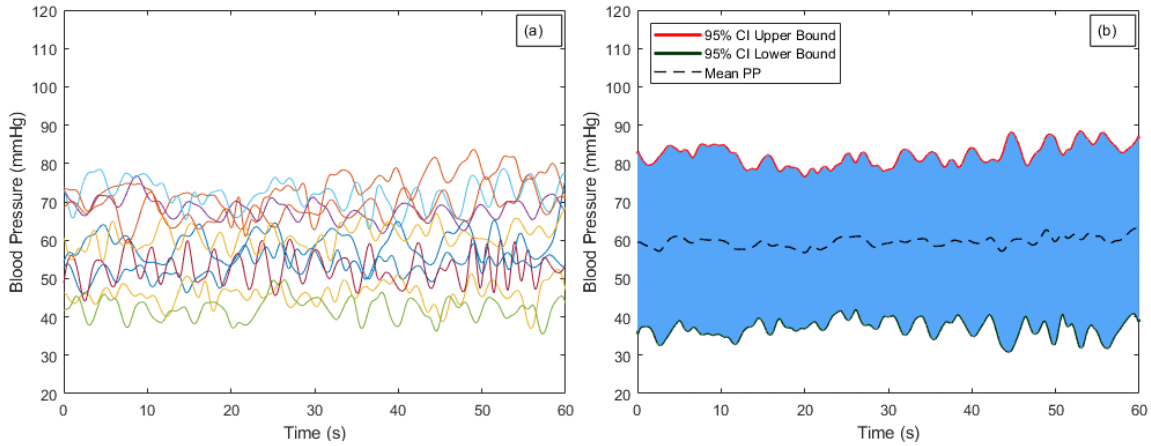

Figure 3.1-8 Aggregated Baseline of blood pressure signals from all the subjects: (a) PP Baseline recordings for each of the 10 subjects; (b) Aggregated PP Baseline with 95% confidence interval envelope.

3.2. Effect of Obstructive Sleep Apnea Episodes on Measures of Blood Pressure

In this section, the results of analyzing the BP changes between sleep apnea episodes and normal breathing are presented. The extracted data is followed by the criteria in Section 2.7.6 and Section 2.7.7, and the normal values are the results in Section 3.1. Blood pressure measurement includes SBP, DBP, MAP and PP, which the formula of Mean Arterial Pressure and Pulse Pressure calculation is in Section 2.7.5. The results include the mean Baseline value of BP with the standard deviation, mean peak BP during apnea episodes with the standard deviation, the difference between peak and Baseline BP, the percentage change of the difference, the temporal location of peak value, and quantity of apnea events used. In statistical analysis, the p-value less than 5% is considered as statistical significance.

3.2.1. Systolic Blood Pressure (SBP)

The first row of data in Table 3.2-1 shows the aggregate results from all subjects, while the rows below that show the results for each of the 10 subjects. Specifically, the entry in the 2nd column

for the first row shows the average and standard deviation (STD) of all sample-by-sample values of the SBP Baselines for all subjects. The average result is shown with the brown horizontal line in Figure 3.2-1b.The entry under Col. 3 in the first row of the table shows the average and standard deviation of the peak SBP values during all apnea events. The red circle in Figure 3.2-1b indicates the peak point of the average. The magnitude and % deviation of the mean for SBP during apnea event from the Baseline mean are shown under Col. 4 and Col. 5, respectively. The temporal location of the SBP peak is at 7s (first row Col.6). Column 7 shows the total number of apnea events that contributed to the results show in each row. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in Table 3.2-1. Further, in statistical analysis, the comparison of the Baseline BP and peak BP from all subjects (Col.2 and Col.3), p-values is less than 1% in the two-tailed T-test.

Table 3.2-1 Systolic Blood Pressure Variations

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7
Subject	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
ALL	128.7 ± 12.7	146.3 ± 19.5	17.6	13.7%	7.0	274
NO. 1	141.2 ± 5.8	158.4 ± 10.1	17.2	12.2%	5.8	49
NO. 2	114.9 ± 4.0	140.8 ± 23.1	25.9	22.5%	9.8	17
NO. 3	138.4 ± 3.2	173.5 ± 13.3	35.1	25.4%	14.5	42
NO. 4	118.4 ± 4.3	137.4 ± 13.0	19.0	16.0%	6.0	44
NO. 5	148.2 ± 4.0	160.4 ± 17.6	12.2	8.2%	15.6	16
NO. 6	109.5 ± 3.5	120.7 ± 8.4	11.2	10.2%	10.6	19
NO. 7	131.2 ± 4.5	151.6 ± 12.4	20.4	15.5%	4.8	15
NO. 8	136.9 ± 3.3	155.2 ± 11.9	18.3	13.4%	9.9	6
NO. 9	128.0 ± 2.2	145.2 ± 15.4	17.2	13.4%	5.9	32
NO. 10	120.1 ± 4.2	133.6 ± 14.3	13.5	11.2%	7.1	34

Figure 3.2-1a displays a plot of the SBP trajectories during all 274 apnea events in the sample population which were separated from the next apnea event by at least 30s. Figure 3.2-1b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

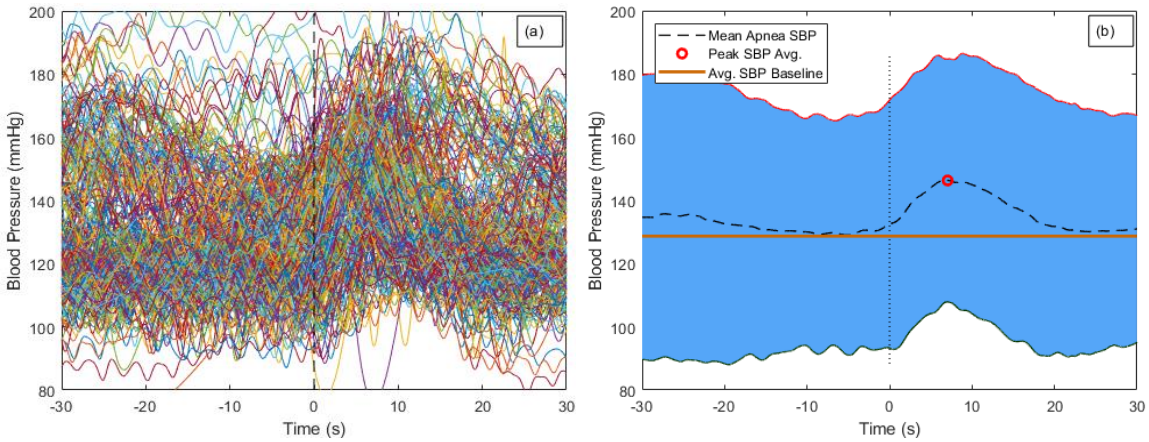

Figure 3.2-1 Aggregated blood pressure oscillations elicited by apnea events: (a) Systolic blood pressure (SBP) recordings for all analyzed apnea events for all subjects; (b) Aggregated SBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

3.2.2. Diastolic Blood Pressure (DBP)

Similar to the results shown Section 3.2.1, the first row of data in

Table *3.2-2* shows the aggregate results from all subjects, while the rows below that show the results for each of the 10 subjects. Specifically, the entry in the 2nd column for the first row shows the average and standard deviation of all sample-by-sample values of the DBP Baselines for all subjects. The average result is the brown horizontal line in Figure 3.2-2b.The entry under Col. 3 in the first row of the table shows the average and standard deviation of the peak DBP values during all apnea events. The red circle in Figure 3.2-2b indicates the peak point of the average. The magnitude and % deviation of the mean for DBP during apnea event from the Baseline mean are shown under Col. 4 and Col. 5, respectively. The temporal location of the DBP peak is at 6.4s (first row Col.6). Column 7 shows the total number of apnea events that contributed to the results show in each row. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in

Table *3.2-2*. Further, in statistical analysis, the comparison of the Baseline BP and peak BP from all subjects (Col.2 and Col.3), p-values is less than 0.1% in the two-tailed T-test.

Table 3.2-2Diastolic Blood Pressure Variations

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7
Subject	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
ALL	69.1 ± 5.8	78.6 ± 9.9	9.5	13.7%	6.4	274
NO. 1	67.2 ± 2.3	77.7 ± 4.3	10.5	15.6%	5.9	49
NO. 2	68.8 ± 3.0	85.9 ± 17.6	17.1	24.9%	9.9	17
NO. 3	70.4 ± 1.4	82.7 ± 6.6	12.3	17.5%	11.6	42
NO. 4	76.2 ± 2.7	85.7 ± 9.2	9.5	12.5%	6.5	44
NO. 5	76.2 ± 1.9	74.1 ± 5.8	-2.1	-2.8%	25.0	16
NO. 6	56.8 ± 2.6	64.2 ± 6.3	7.4	13.0%	8.2	19
NO. 7	72.6 ± 3.4	81.5 ± 10.9	8.9	12.3%	8.7	15
NO. 8	68.7 ± 1.5	86.5 ± 7.9	17.8	25.9%	7.3	6
NO. 9	67.1 ± 1.5	78.9 ± 8.6	11.8	17.6%	4.6	32
NO. 10	66.8 ± 2.5	80.5 ± 10.2	13.7	20.5%	7.2	34

Figure 3.2-2a displays a plot of the DBP trajectories during all apnea events in the sample population which were separated from the next apnea event by at least 30s. Figure 3.2-2b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

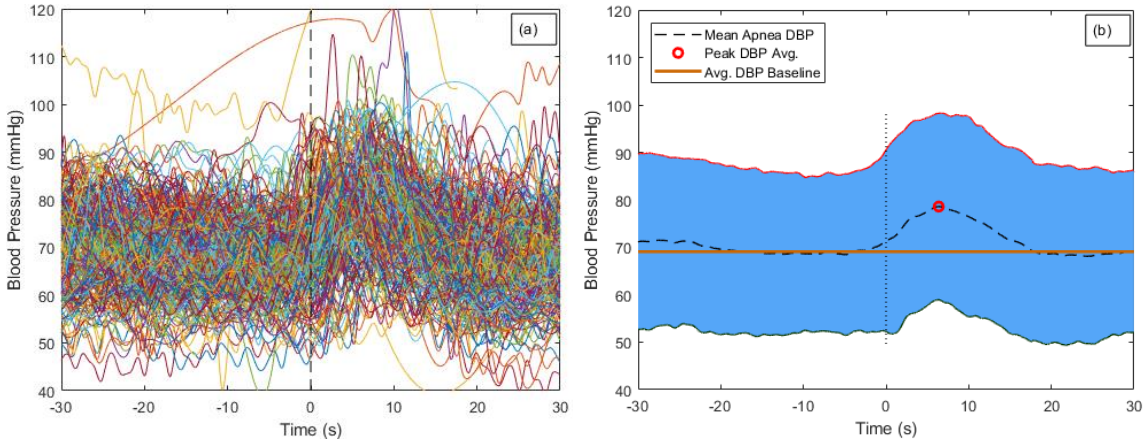

Figure 3.2-2 Aggregated blood pressure oscillations elicited by apnea events: (a) Diastolic blood pressure (DBP) recordings for all analyzed apnea events for all subjects; (b) Aggregated DBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

3.2.3. Mean Arterial Pressure (MAP)

The first row of data in Table 3.2-3 shows the aggregate results from all subjects, while the rows below that show the results for each of the 10 subjects. Specifically, the entry in the 2nd column for the first row shows the average and standard deviation of all sample-by-sample values of the MAP Baselines for all subjects. The average result is the brown horizontal line in Figure 3.2-3b.The entry

under Col. 3 in the first row of the table shows the average and standard deviation of the peak MAP values during all apnea events. The red circle in Figure 3.2-3b indicates the peak point of the average. The magnitude and % deviation of the mean for MAP during apnea event from the Baseline mean are shown under Col. 4 and Col. 5, respectively. The temporal location of the MAP peak is at 6s (first row Col.6). Column 7 shows the total number of apnea events that contributed to the results show in each row. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in Table 3.2-3. Further, in statistical analysis, the comparison of the Baseline BP and peak BP from all subjects (Col.2 and Col.3), p-values is less than 0.1% in the two-tailed T-test.

Table 3.2-3 Mean Arterial Pressure Variations

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7
Subject	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
ALL	89.0 ± 7.0	100.8 ± 11.9	11.8	13.3%	6.0	274
NO. 1	92.0 ± 3.3	104.4 ± 6.0	12.5	13.6%	5.5	49
NO. 2	84.2 ± 3.2	103.8 ± 19.1	19.6	23.3%	9.6	17
NO. 3	93.0 ± 1.7	112.6 ± 8.1	19.5	21.0%	11.2	42
NO. 4	90.3 ± 3.1	102.1 ± 11.5	11.8	13.1%	6.0	44
NO. 5	100.2 ± 2.1	102.7 ± 11.3	2.5	2.5%	5.6	16
NO. 6	74.4 ± 2.3	82.6 ± 6.4	8.2	11.0%	10.5	19
NO. 7	92.1 ± 3.4	104.7 ± 10.4	12.6	13.7%	5.1	15
NO. 8	91.5 ± 1.6	107.7 ± 8.7	16.2	17.7%	7.7	6
NO. 9	87.4 ± 1.1	100.1 ± 10.1	12.8	14.6%	4.7	32
NO. 10	84.7 ± 2.8	98.0 ± 11.3	13.3	15.7%	6.8	34

Figure 3.2-3a displays a plot of the MAP trajectories during all apnea events in the sample population which were separated from the next apnea event by at least 30s. Figure 3.2-3b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

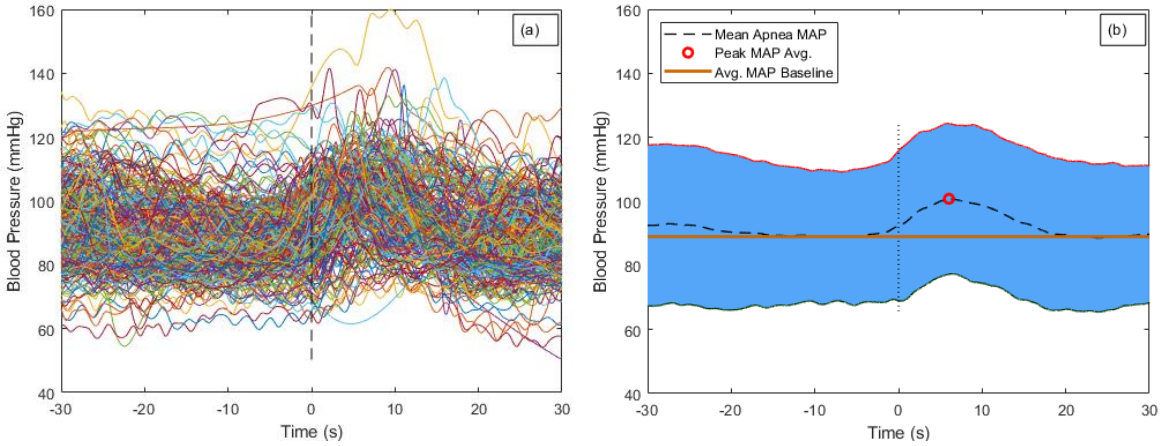

Figure 3.2-3 Aggregated blood pressure oscillations elicited by apnea events: (a) Mean arterial pressure (MAP) recordings for all analyzed apnea events for all subjects; (b) Aggregated MAP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

3.2.4. Pulse Pressure (PP)

In

Table *3.2-4*, the first row of data shows the aggregate results from all subjects, while the rows below that show the results for each of the 10 subjects. Specifically, the entry in the 2nd column for the first row shows the average and standard deviation of all sample-by-sample values of the PP Baselines for all subjects. The average result is the brown horizontal line in Figure 3.2-3b.The entry under Col. 3 in the first row of the table shows the average and standard deviation of the peak PP values during all apnea events. The red circle in Figure 3.2-3b indicates the peak point of the average. The magnitude and % deviation of the mean for PP during apnea event from the Baseline mean are shown under Col. 4 and Col. 5, respectively. The temporal location of the PP peak is at 9.1s (first row Col.6). Column 7 shows the total number of apnea events that contributed to the results show in each row. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in

Table *3.2-4*. Further, in statistical analysis, the comparison of the Baseline BP and peak BP from all subjects (Col.2 and Col.3), p-values is around 6% in the two-tailed T-test.

Table 3.2-4 Pulse Pressure Variations

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7
Subject	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
ALL	59.6 ± 10.9	69.3 ± 19.8	9.8	16.4%	9.1	274
NO. 1	73.9 ± 4.5	82.0 ± 6.8	8.2	11.1%	8.5	49
NO. 2	46.0 ± 2.8	58.2 ± 11.7	12.2	26.5%	13.5	17
NO. 3	68.1 ± 2.7	92.5 ± 11.6	24.4	35.8%	14.9	42
NO. 4	42.1 ± 2.9	52.5 ± 24.0	10.4	24.7%	9.1	44
NO. 5	72.0 ± 3.6	88.8 ± 14.8	16.9	23.5%	-5.3	16
NO. 6	52.7 ± 3.7	57.7 ± 7.4	5.0	9.5%	8.8	19
NO. 7	58.7 ± 4.1	72.8 ± 9.4	14.1	24.0%	12.3	15
NO. 8	68.0 ± 3.3	77.0 ± 10.8	9.0	13.2%	10.1	6
NO. 9	60.9 ± 2.8	71.1 ± 12.5	10.2	16.7%	6.8	32
NO. 10	53.2 ± 2.9	55.7 ± 7.7	2.5	4.7%	11.8	34

Figure 3.2-4a displays a plot of the PP trajectories during all apnea events in the sample population which were separated from the next apnea event by at least 30s. Figure 3.2-4b shows the result of aggregating these trajectories as well as the corresponding 95% CI.

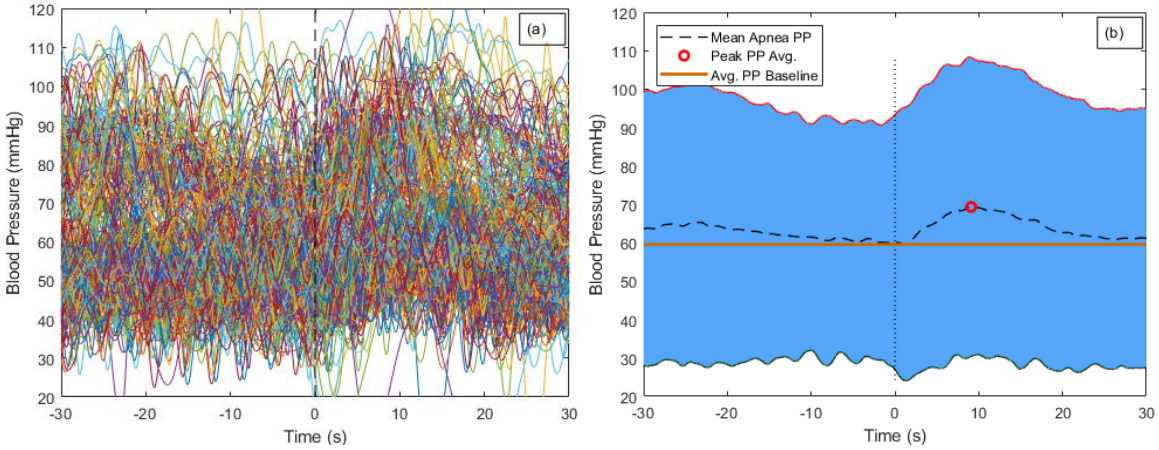

Figure 3.2-4 Aggregated blood pressure oscillations elicited by apnea events: (a) Pulse pressure (PP) recordings for all analyzed apnea events for all subjects; (b) Aggregated PP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

3.2.5. Systolic Slope

The computation of the slope of surges in SBP that are presented in this section follows the method that was described in Section 2.7.8. The resulting slope values for SBP are displayed in Table 3.2-5. In this table, the first row shows the aggregate results from all subjects in SBP, while the rows below that show the results for each of the 10 subjects. Respectively, the mean and standard deviation of slope and intercept values for Baseline are shown under Col. 2 and Col. 3. Similarly, Col 4 and Col. 5 display the mean and standard deviation of slope and intercept values for apnea events.

Column 6 shows the total number of apnea events that contributed to the results show in each row. The accumulated results from the slope equation reveals the slope is 0.9 mmHg/s and the intercept is 149.3 mmHg.

Table 3.2-5 Slope and intercept values for the surges in SBP during obstructive sleep apnea events

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6
Subject	**Slope in Normal Breathing (mmHg/s)**	**Intercept in Normal Breathing (mmHg)**	**Slope in Apnea Events (mmHg/s)**	**Intercept in Apnea Events (mmHg)**	**No. of Obstructive Apnea Events**
ALL	0.00 ± 0.0	1.28 ± 0.1	0.90 ± 0.7	149.3 ± 20.0	274
1	0.00 ± 0.0	1.15 ± 0.0	0.75 ± 0.4	133.7 ± 12.9	34
2	0.00 ± 0.0	1.42 ± 0.0	0.74 ± 0.5	159.2 ± 8.1	49
3	0.00 ± 0.0	1.16 ± 0.0	1.15 ± 1.2	136.8 ± 17.7	17
4	0.00 ± 0.0	1.46 ± 0.0	0.67 ± 0.7	172.9 ± 14.3	42
5	0.00 ± 0.0	1.19 ± 0.0	1.27 ± 0.5	137.9 ± 8.7	44
6	0.00 ± 0.0	1.44 ± 0.0	0.64 ± 0.4	169.8 ± 18.1	16
7	0.00 ± 0.0	1.08 ± 0.0	0.69 ± 0.4	119.1 ± 8.4	19
8	0.00 ± 0.0	1.26 ± 0.0	0.97 ± 0.3	151.3 ± 12.6	15
9	0.00 ± 0.0	1.36 ± 0.0	0.76 ± 0.7	157.4 ± 7.2	6
10	0.00 ± 0.0	1.26 ± 0.0	1.23 ± 0.8	147.1 ± 12.8	32

Figure 3.2-5 provides an average prediction for the blood pressure rises in the apnea events in 60s window. The orange line is calculated by using average slope and average intercept with the aggregated result from all apnea events in Table 3.2-5. The blue line in Figure 3.2-5 indicates stable BP during normal breathing. Specifically, the average slope and the intercept for the lines fitted to the Baseline SBP are close to zero in the first row of Col. 3 in Table 3.2-5. Therefore, to compensate the biased, the value in normal breathing are added with the accumulated results, 128.7 mmHg, from normal breathing in

Table *3.1-1*. At 30s, the blood pressure raises to 176.3 mmHg and the difference between the surge and Baseline is around 46 mmHg.

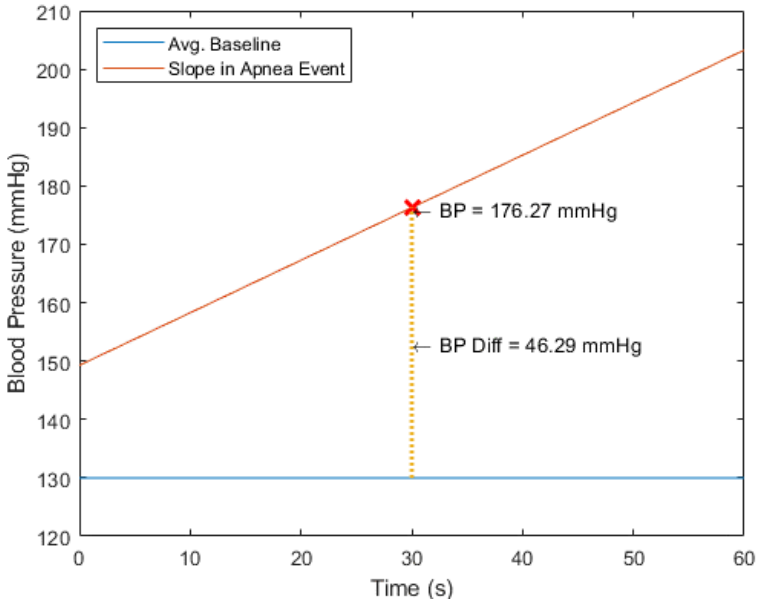

Figure 3.2-5 Slope plot in 60 in SBPs; the red cross indicates the value at 30s

3.2.6. Diastolic Slope

Similar to the computation of slopes and intercepts for SBP surges, we also computed the slope and intercept of the surges in DBP to determine whether SBP and DBP surges show any difference. The first row of data in Table 3.2-6 shows the aggregate results slope calculations from all of the subjects; results from calculations for individual subjects are also shown. . Correspondingly, the slope and intercept results in normal breathing from the slope equation with the standard deviation are shown under Col. 2 and Col. 3. Similarly, Col 4 and Col. 5 display the slope and intercept results in apnea events. Column 6 shows the total number of apnea events that contributed to the results show in each row. The accumulated results from the slope equation reveals the slope is around 0.6 mmHg/s and the intercept is 80.9 mmHg.

Table 3.2-6 Slope and intercept results from sleep apnea events in DBP

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6
Subject	**Slope in Normal Breathing (mmHg/s)**	**Intercept in Normal Breathing (mmHg)**	**Slope in Apnea Events (mmHg/s)**	**Intercept in Apnea Events (mmHg)**	**No. of Apnea Events**
ALL	0.00 ± 0.0	0.68 ± 0.1	0.62 ± 0.5	80.9 ± 9.1	274

1	0.00 ± 0.0	0.62 ± 0.0	0.57 ± 0.3	80.3 ± 9.9	34
2	0.00 ± 0.0	0.67 ± 0.0	0.47 ± 0.3	78.6 ± 4.1	49
3	0.00 ± 0.0	0.69 ± 0.0	0.54 ± 0.5	85.0 ± 12.8	17

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6
Subject	**Slope in Normal Breathing (mmHg/s)**	**Intercept in Normal Breathing (mmHg)**	**Slope in Apnea Events (mmHg/s)**	**Intercept in Apnea Events (mmHg)**	**No. of Apnea Events**
4	0.00 ± 0.0	0.72 ± 0.0	0.46 ± 0.4	83.0 ± 7.3	42
5	0.00 ± 0.0	0.77 ± 0.0	0.84 ± 0.4	85.8 ± 6.4	44
6	0.00 ± 0.0	0.72 ± 0.0	0.31 ± 0.3	79.7 ± 5.4	16
7	0.00 ± 0.0	0.59 ± 0.0	0.52 ± 0.4	64.6 ± 6.0	19
8	0.00 ± 0.0	0.69 ± 0.0	0.89 ± 0.6	82.4 ± 8.9	15
9	0.00 ± 0.0	0.68 ± 0.0	0.62 ± 0.3	88.0 ± 4.5	6
10	0.00 ± 0.0	0.64 ± 0.0	0.95 ± 0.6	81.5 ± 8.9	32

Figure 3.2-6 provides an average prediction for the blood pressure rises in the apnea events in 60s window. The orange line is calculated by using average slope and average intercept with the aggregated result from all apnea events in Table 3.2-6. The blue line in Figure 3.2-6 indicates stable BP during normal breathing. Specifically, the average slope and the intercept for the lines fitted to the Baseline SBP are close to zero in the first row of Col. 3 in Table 3.2-6. Therefore, to compensate the biased, the value in normal breathing are added with the accumulated results, 69.1 mmHg, from normal breathing in Table 3.1-2. At 30s, the blood pressure raises to 99.49 mmHg and the difference between the surge and Baseline is around 29.71 mmHg.

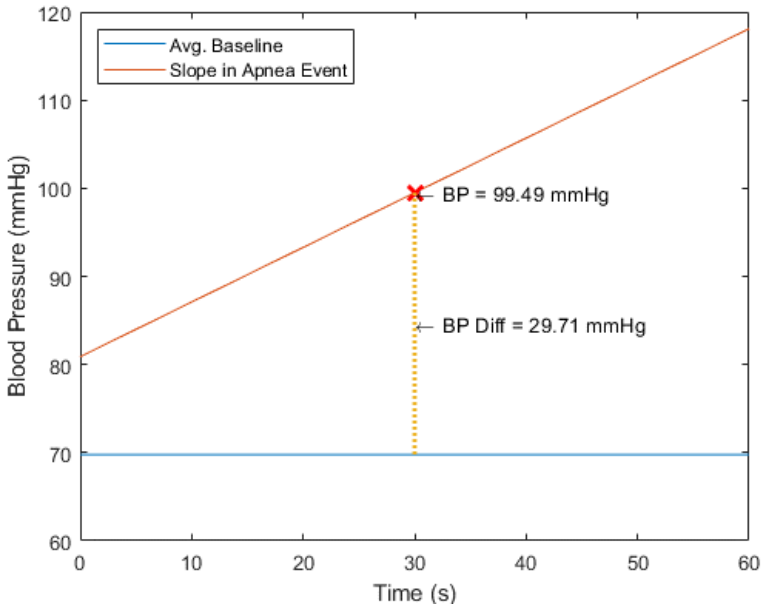

Figure 3.2-6 Slope plot in 60s in DBP; the red cross indicates the value at 30s

3.3. Comparisons of Blood Pressure Surge during Apnea Episodes in Sleep Stages

In this section, the selected apnea episodes are separated from other events by at least 30 seconds and the event in its entirety takes place within a single sleep stage (see Section 2.7.7 for details) are grouped together to determine the interaction of sleep stage and the degree of blood pressure surges elicited by apnea events. A tabulation of the number of qualified events (per criteria mentioned above) is shown in Table 3.3-1. The first row of data in Table 3.3-1 displays the aggregate results from all subjects. As can be seen from Table 3.3-1, the majority of the events occur in Stages 1, 2 and REM. The stage without events, Stage 4, will be ignored in the following subsection.

Table 3.3-1 Quantity of events in each sleep stage

SUBJECT NO	STAGE 1	STAGE 2	STAGE 3	STAGE 4	REM
ALL	135	38	7	0	23
1	7	12	0	0	1
2	40	1	0	0	0
3	5	3	7	0	0
4	25	4	0	0	5
5	27	2	0	0	0
6	4	1	0	0	10

7	5	1	0	0	6
8	12	1	0	0	0
9	3	13	0	0	0
10	7	12	0	0	1

3.3.1. Analysis of Systolic Blood Pressure Surges in Various Sleep Stages

The first four row of data in

Table *3.3-2* shows the aggregate mean and standard deviation of SBP surges in each of the sleep stages considered for all subjects, while the rows below that show the results for each of the 10 subjects individually. Specifically, the entries in the 3rd column for the first four rows show the average and standard deviation of all sample-by-sample values of the SBP Baselines for all subjects in the stages 1, 2, 3 and REM sleep. It is noted that since no event in Stage 4 qualified for analysis – based on the criteria outlined in Section 2.7.7 – there are no entries for that stage. In addition, the entry under Col. 3 in the first four rows of the table shows the average and standard deviation of the peak SBP values during all apnea events respectively to each sleep stage. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in

Table *3.3-2*. In the t-test results, there is no difference (p-value > 5%) in the comparison between normal value and the peak value from each subject in

Table *3.3-2*.

Table 3.3-2 SBP variation in different sleep stages

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7	Col. 8
STAGE	**Subject**	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
1	All	137.1 ± 5.6	149.4 ± 18.8	12.3	9.0%	6.1	135
2	All	128.4 ± 5.1	141.6 ± 16.3	13.3	10.4%	6.4	38
3	All	117.6 ± 4.9	132.0 ± 9.3	14.4	12.2%	10.1	7
REM	All	149.0 ± 3.4	157.7 ± 29.6	8.7	5.8%	10.2	23
1	1	121.4 ± 4.1	130.4 ± 8.5	9.1	7.5%	10.0	7
1	8	135.7 ± 6.6	152.2 ± 9.7	16.5	12.2%	8.5	12
1	4	156.9 ± 8.9	170.5 ± 12.6	13.6	8.7%	11.7	25
1	7	109.5 ± 4.8	122.1 ± 6.2	12.6	11.5%	7.4	5
1	10	124.3 ± 6.7	142.8 ± 17.3	18.4	14.8%	5.4	7
1	5	122.1 ± 5.9	137.3 ± 13.8	15.2	12.4%	6.3	27
2	1	116.7 ± 5.4	128.8 ± 12.4	12.1	10.4%	7.1	12
2	10	128.6 ± 8.1	152.6 ± 15.0	24	18.7%	3.4	13
REM	6	160.2 ± 3.4	166.1 ± 15.5	5.9	3.7%	25.1	10

REM	7	109.9 ± 3.6	118.9 ± 8.4	9	8.2%	7.1	6

The brown horizontal lines in the Figure 3.3-1 b, Figure 3.3-2 b,

Figure 3.3-3 b, and Figure 3.3-4 b show the average result of the Baseline. In these figures, the plots on the left side displays all the SBP trajectories during the qualified apnea events within the corresponding sleep stage (i.e., stage 1, 2, 3, and REM) The plots on the right side of the figures show the result of aggregating these trajectories for each corresponding sleep stages as well as the corresponding 95% CI. Additionally, the red circles in the right-side figures mark the peak of the sample-by-sample mean apnea SBP surges corresponding to the apnea episodes.

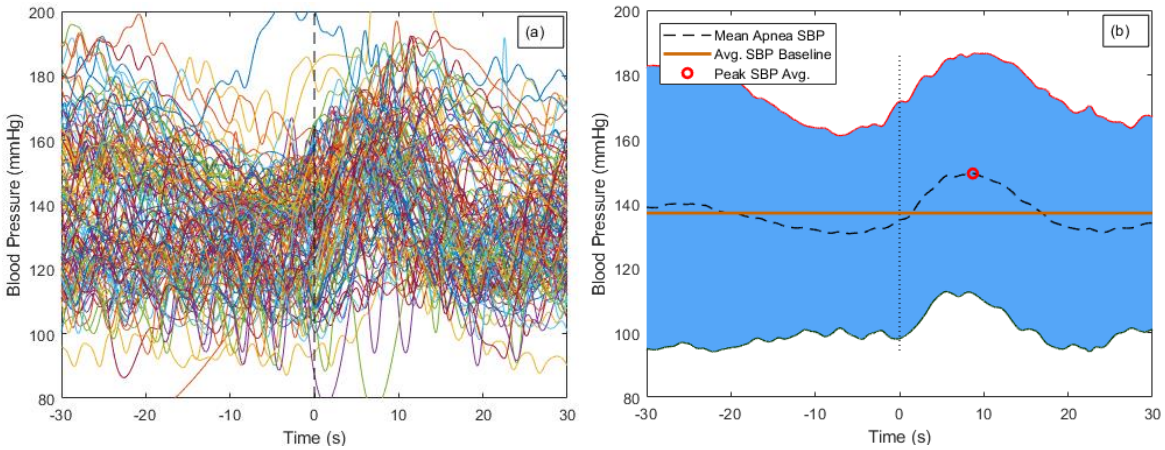

Figure 3.3-1 Aggregated blood pressure oscillations elicited by apnea events in Stage 1: (a) SBP recordings for all analyzed apnea events for all subjects; (b) Aggregated SBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

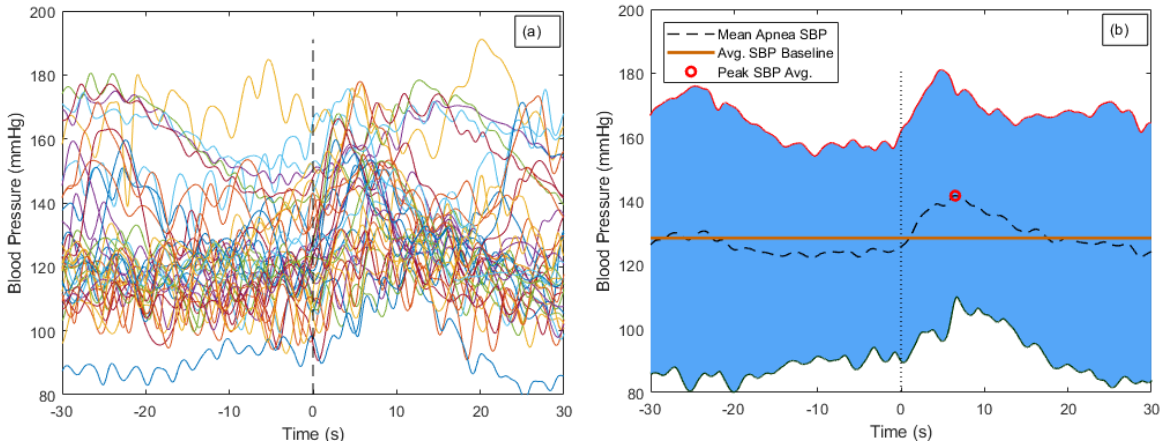

Figure 3.3-2 Aggregated blood pressure oscillations elicited by apnea events in Stage 2: (c) SBP recordings for all analyzed apnea events for all subjects; (d) Aggregated SBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

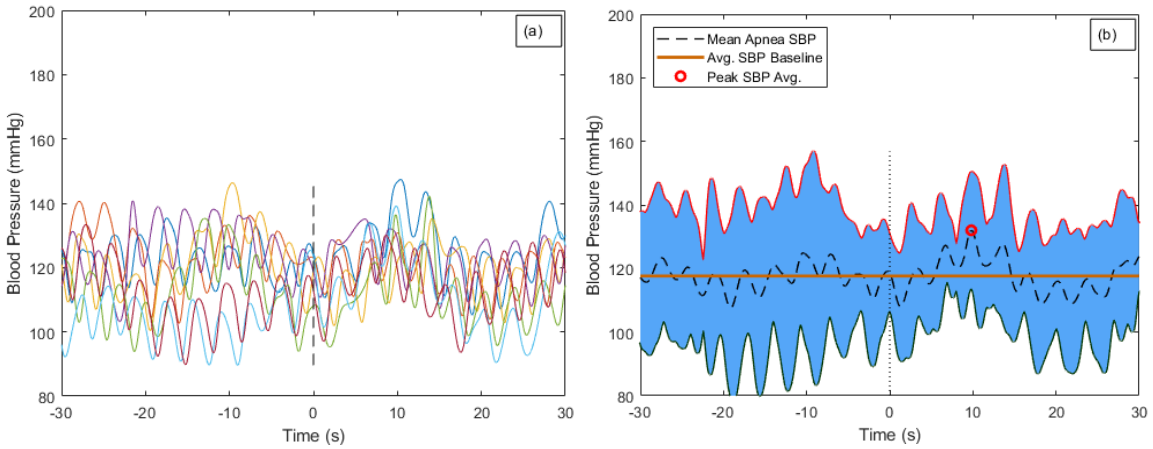

Figure 3.3-3 Aggregated blood pressure oscillations elicited by apnea events in Stage 3: (a) SBP recordings for all analyzed apnea events for all subjects; (b) Aggregated SBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

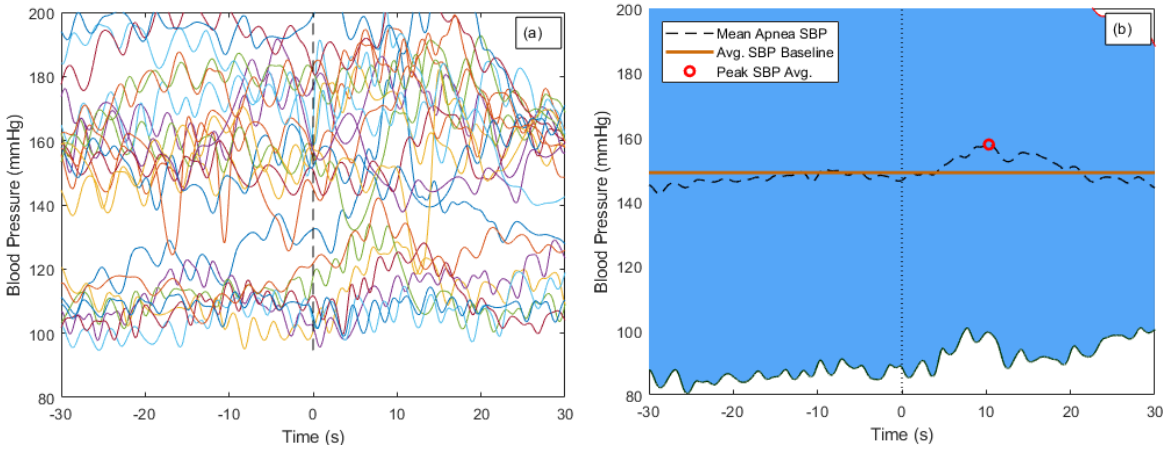

Figure 3.3-4 Aggregated blood pressure oscillations elicited by apnea events in REM: (a) SBP recordings for all analyzed apnea events for all subjects; (b) Aggregated SBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

In the statistical analysis, one-way ANOVA method was used for comparison between sleep stages. In the Baseline, the dataset is considered in the normal distribution due to the central limit theorem. Since the quantity of 1 min window in Baseline in Stage 1, 2, 3, and REM is 230, 158, 14, and 28, respectively, which each stage contains more than 60,000 values. However, by testing for the homogeneity using Bartlett's test, we reject the null hypothesis (around p=0.01) in

Table *3.3-3*. Such a result indicated that the data from stage 3 should be excluded. Figure 3.3-5 is the box-whisker plots in each sleep stage. The two horizontal solid lines indicate the minimum and maximum value in that stage and the horizontal distance of the box reveals the interquartile rage, which is from the subtraction between the upper edge of the box, third quartile and the lower edge of the box, first quartile. Also, the red line inside the box shows the median of that stage. So, the data distribution in Stage 3 clearly considers as not homogeneity in Figure 3.3-5. With only Stage 1, 2 and REM in homogeneity test, we fail to reject the null hypothesis ($p = 0.53$) in Table 3.3-4.

Table 3.3-3 The homogeneity in Stage 1, 2, 3, and REM in SBP Baseline

SLEEP STAGE	COUNT	MEAN	STANDARD DEVIATION
REM	28	64.9	7.0
3	14	55.6	5.0
2	158	66.5	9.2
1	230	71.2	7.9
POOLED	430	68.6	8.3
BARTLETT'S STATISTIC	11.0		
DEGREES OF FREEDOM	3		
P-VALUE	0.012		

Table 3.3-4 The homogeneity in Stage 1, 2, and REM in SBP Baseline

SLEEP STAGE	COUNT	MEAN	STANDARD DEVIATION
REM	28	134.7	21.9
2	158	131.0	23.5
1	230	143.2	21.6
POOLED	416	138.0	22.4
BARTLETT'S STATISTIC	1.242		
DEGREES OF FREEDOM	2		
P-VALUE	0.537		

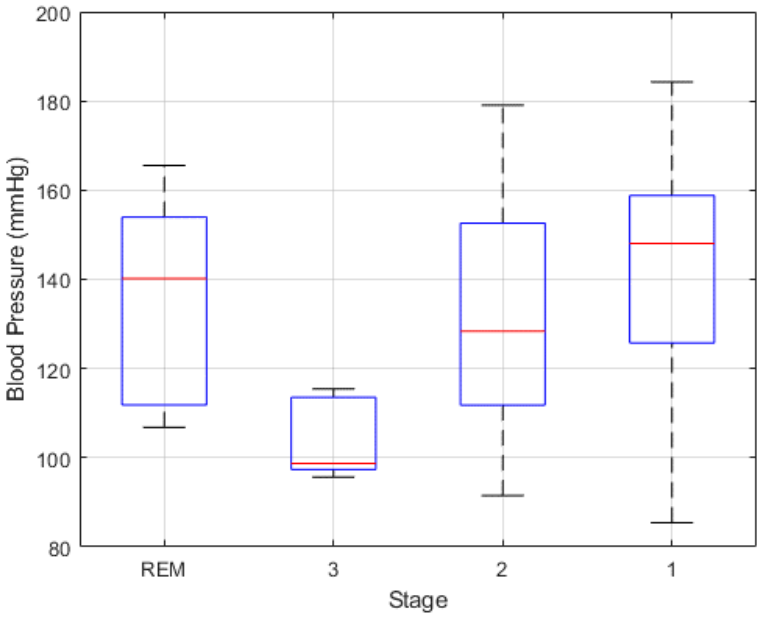

Figure 3.3-5 BP Baseline distribution in each sleep stage in SBP;

From the results of ANOVA in Table 3.3-5, we rejected the null hypothesis ($p<0.0001$), which the mean values in Stage 1, 2 and REM aren't all the same. Figure *3.3-6* is a notched box plot, which is almost the same as a box-whisker plot as well as showing the 95% confidence interval (CI) in the notched. The coefficient used in calculated the boundary of 95%CI is 1.57. From *Figure 3.3-6*, we could roughly find out that the 95%CI between Stage 1 and 2 are non-overlapping. To avoid Type II error in statistic, instead of using student t-test, we applied Tukey Test of multiple means to find the difference in Table 3.3-6 and found the mean values in Stage 1 is significantly different from Stage 2 ($p <0.0001$), but the mean values in REM has no difference in either Stage 1 ($p=0.13$) or Stage 2 ($p=0.7$). Additionally, Figure 3.3-7 is a Tukey-Kramer Confidence Interval Plot, which each horizontal line indicates the 95%CI for the corresponding stages, and there isn't overlapping between the blue line in Stage 2 and the red line in REM.

Table 3.3-5 One-way ANOVA Summary in SBP Baseline

SOURCE	SS	DF	MS	F	PROB>F
GROUPS	14207.9	2	7103.9	14.2	1.08E-06
ERROR	206573.7	413	500.2		
TOTAL	220781.5	415			

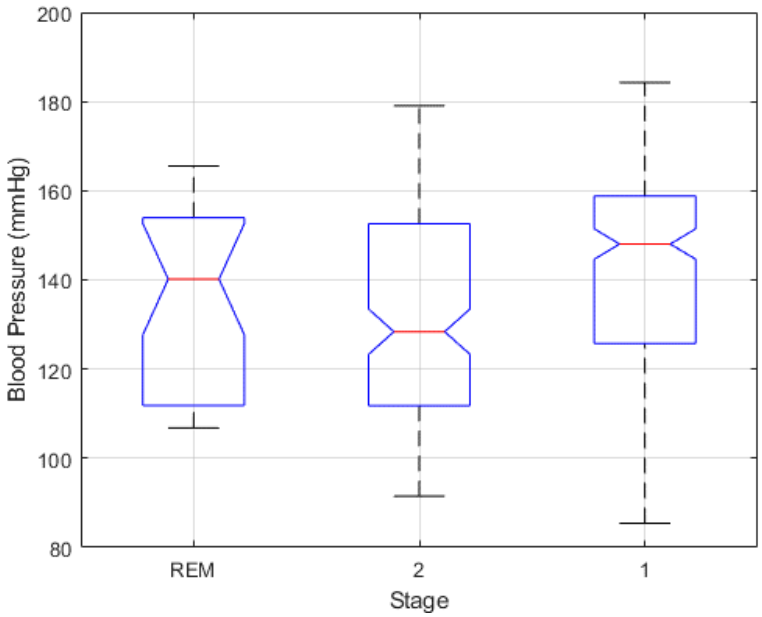

Figure 3.3-6 One-way ANOVA results in SBP Baseline

Table 3.3-6 Tukey test summary in SBP Baseline

STAGES	95% CONFIDENCE INTERVAL	ESTIMATED MEAN DIFFERENT	P-VALUE
REM vs. 2	[-7.087, 14.409]	3.661	0.7040
REM vs. 1	[-19.003, 1.980]	-8.512	0.1382
2 vs. 1	[-17.589, -6.757]	-12.173	< 0.0001

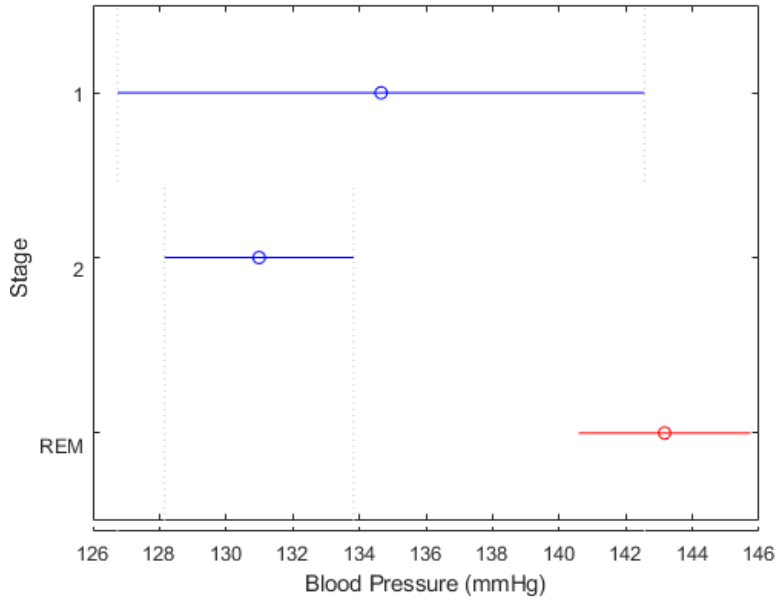

Figure 3.3-7 Tukey test result in SBP Baseline

For the statistical comparison between sleep stages with apnea episodes, we collected the blood pressure values at the temporal location of blood pressure peak in each sleep stage in Table 3.3-2 Col.4. ANOVA test is used to analyze the mean different between sleep stages elicited by apnea events. The Shapiro-Wilk parametric hypothesis test is used to check the normality. In Table 3.3-7, each stage is in normal distribution, which we fail to reject the null hypothesis (p>5%). Then, in the way to examine the homogeneity with Bartlett's test, we rejected the null hypothesis (around p=0.001) in Table 3.3-8. The standard deviation in REM in Table 3.3-8 indicated that the variance is relatively large. Figure 3.3-8 is the box-whisker plots in each sleep stage. The two horizontal solid lines indicate the minimum and maximum value in that stage and the horizontal distance of the box reveals the interquartile rage, which is from the subtraction between the upper edge of the box, third

quartile and the lower edge of the box, first quartile. Also, the red line inside the box shows the median of that stage. So, the median in REM is too close to the third quartile in Figure 3.3-8, which is the reason of not homogeneity and the data from REN should be excluded. With only Stage 1, 2 and 3 in homogeneity test, we fail to reject the null hypothesis (p = 0.1) in Table *3.3-9*.

Table 3.3-7 P-value of Shapiro-Wilk parametric hypothesis test in peak SBP elicited by apnea events

SLEEP STAGES	NUM	P-VALUE
REM	23	0.08
STAGE 3	7	0.74
STAGE 2	38	0.08
STAGE 1	135	0.08

Table 3.3-8 The homogeneity in Stage 1, 2, 3, and REM in peak SBP elicited by apnea events

SLEEP STAGE	COUNT	MEAN	STANDARD DEVIATION
REM	23	157.7	29.6
3	7	132.0	9.3
2	38	141.6	16.3
1	135	149.4	18.8
POOLED	203	148.3	19.7
BARTLETT'S STATISTIC	16.6		
DEGREES OF FREEDOM	3		
P-VALUE	0.0009		

Table 3.3-9 The homogeneity in Stage 1, 2, and REM in peak SBP elicited by apnea events

SLEEP STAGE	COUNT	MEAN	STANDARD DEVIATION
REM	7	132.0	9.3
2	38	141.6	16.3
1	135	149.4	18.8
POOLED	180	147.1	18.1
BARTLETT'S STATISTIC	4.6		
DEGREES OF FREEDOM	2.0		
P-VALUE	0.102		

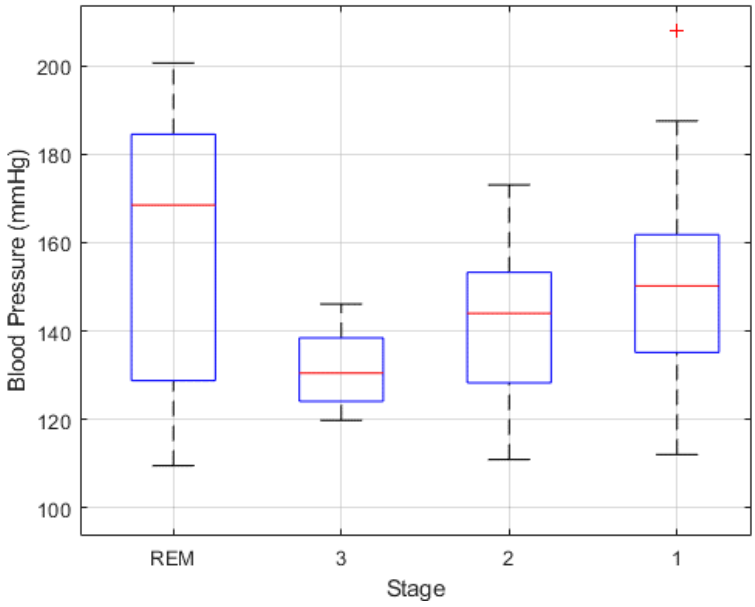

Figure 3.3-8 peak BP distribution elicited by apnea events in each sleep stage in SBP

From the results of ANOVA in Table 3.3-10, we rejected the null hypothesis ($p<0.01$), which the mean values in Stage 1, 2 and 3 aren't all the same. Figure 3.3-9 is a notched box plot, which is almost the same as a box-whisker plot as well as showing the 95% confidence interval (CI) in the notched. The coefficient used in calculated the boundary of 95%CI is 1.57. From Figure 3.3-9, we could roughly find out that the 95%CI between Stage 1 and 3 are non-overlapping. To avoid Type II error in statistic, instead of using ordinary t-test, we applied Tukey Test to find the difference in Table 3.3-11 and found the mean values in Stage 1 is significantly difference with Stage 3 ($p=0.036$), but the mean values in Stage 2 has no difference in either Stage 1 ($p=0.4$) or Stage 3 ($p=0.052$). Additionally, Figure 3.3-10 is a Tukey-Kramer Confidence Interval Plot, which each horizontal line indicates the 95%CI for the corresponding stages, and there isn't overlapping between the blue line in Stage 1 and the red line in Stage 3.

Table 3.3-10 One-way ANOVA Summary in peak SBP elicited by apnea events

SOURCE	SS	DF	MS	F	PROB>F
GROUPS	3414	2	1707.0	5.2	6.20E-03
ERROR	57794.9	177	326.5		
TOTAL	61208.9	179			

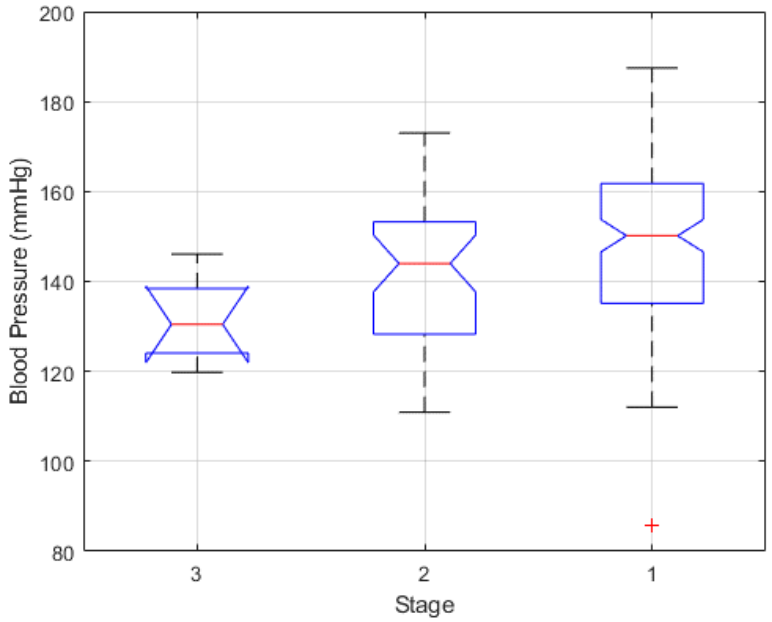

Figure 3.3-9 One-way ANOVA results in peak SBP elicited by apnea events

Table 3.3-11 Tukey test summary in peak SBP elicited by apnea events

SLEEP STAGE	95% CONFIDENCE INTERVAL	ESTIMATED MEAN DIFFERENT	P-VALUE
1 vs. 2	[-27.019, 7.819]	-9.600	0.400
1 vs. 3	[-33.744, -0.910]	-17.327	0.036
2 vs. 3	[-15.504, 0.050]	-7.727	0.052

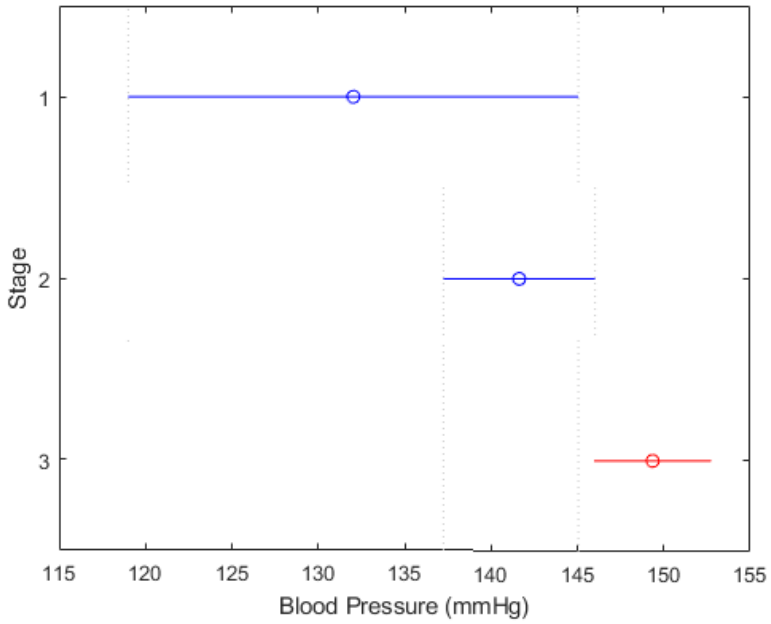

Figure 3.3-10 Tukey test result in peak SBP elicited by apnea events

3.3.2. Analysis of Diastolic Blood Pressure Surges in Various Sleep Stages

Similarly in Section 3.3.1, the first four row of data in

Table *3.3-12* shows the aggregate mean and standard deviation of SBP surges in each of the sleep stages considered for all subjects, while the rows below that show the results for each of the 10 subjects individually. Specifically, the entries in the 3rd column for the first four rows show the average and standard deviation of all sample-by-sample values of the SBP Baselines for all subjects in the stages 1, 2, 3 and REM sleep. It is noted that since no event in Stage 4 qualified for analysis – based on the criteria outlined in Section 2.7.7 – there are no entries for that stage. In addition, the entry under Col. 3 in the first four rows of the table shows the average and standard deviation of the peak SBP values during all apnea events respectively to each sleep stage. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in

Table *3.3-12*. In the t-test results, Stage 1 and 2 show significantly difference (p-value around 3%) in the comparison between normal value and the peak value from each subject in

Table *3.3-12*, but no difference in REM.

Table 3.3-12 DBP variation in different sleep stages

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7	Col. 8
STAGE	**Subject**	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
1	All	71.9 ± 3.0	79.2 ± 9.8	7.3	10.2%	6.1	135
2	All	69.2 ± 3.5	78.1 ± 8.0	8.8	12.7%	6.4	38
3	All	72.1 ± 2.9	79.2 ± 5.3	7.0	9.7%	10.1	7
REM	All	69.7 ± 1.8	75.1 ± 14.5	5.4	7.7%	10.2	23
1	1	69.3 ± 2.8	76.4 ± 7.1	7.1	10.2%	10.0	7
1	8	71.0 ± 5.2	83.0 ± 9.0	12.0	16.9%	8.5	12
1	4	75.0 ± 4.1	82.8 ± 7.1	7.8	10.4%	11.7	25
1	7	57.5 ± 3.0	65.4 ± 6.9	7.9	13.7%	7.4	5
1	10	64.1 ± 4.9	80.0 ± 9.3	15.9	24.8%	5.4	7
1	5	74.9 ± 3.5	84.2 ± 9.4	9.3	12.4%	6.3	27
2	1	68.3 ± 3.6	77.9 ± 7.8	9.6	14.1%	7.1	12
2	10	65.9 ± 5.7	82.4 ± 8.7	16.6	25.2%	3.4	13
REM	6	71.1 ± 1.6	75.3 ± 4.3	4.1	5.8%	25.1	10
REM	7	57.9 ± 2.0	63.1 ± 8.2	5.2	9.0%	7.1	6

The brown horizontal lines in the Figure 3.3-11 b, Figure 3.3-12 b, Figure 3.3-13 b, and Figure *3.3-14* b show the average result of the Baseline. In these figures, the plots on the left side displays all the DBP trajectories during the qualified apnea events within the corresponding sleep stage (i.e., stage 1, 2, 3, and REM) The plots on the right side of the figures show the result of aggregating these trajectories for each corresponding sleep stages as well as the corresponding 95% CI. Additionally, the red circles in the right-side figures mark the peak of the sample-by-sample mean apnea DBP surges corresponding to the apnea episodes.

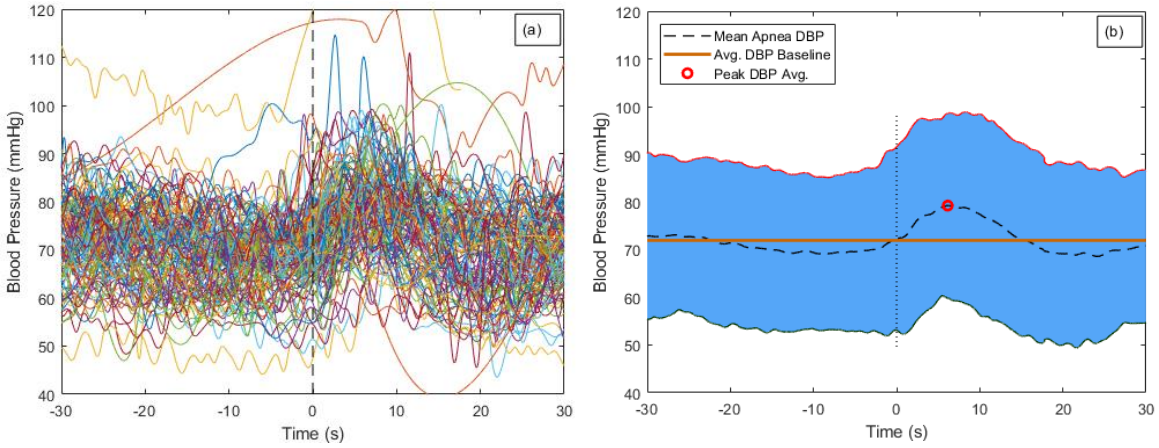

Figure 3.3-11 Aggregated blood pressure oscillations elicited by apnea events in Stage 1: (a) DBP recordings for all analyzed apnea events for all subjects; (b) Aggregated DBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

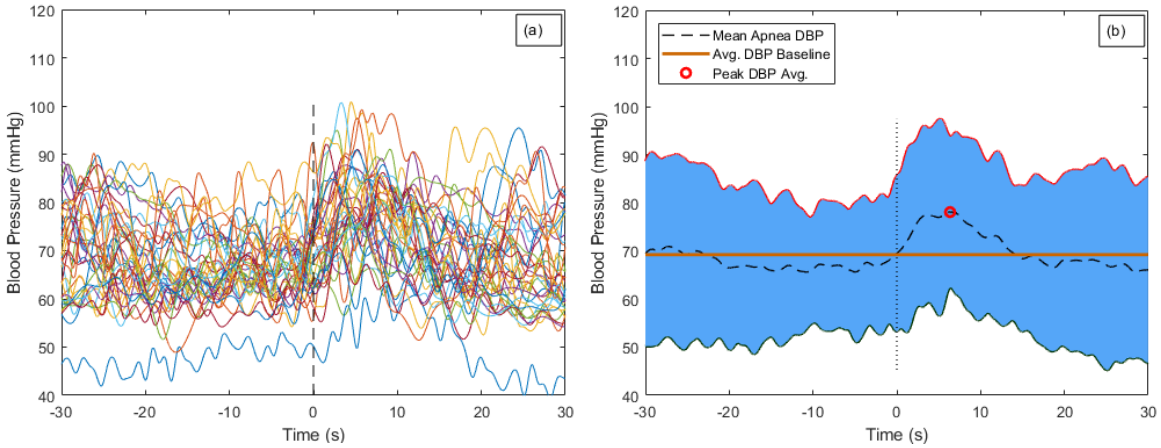

Figure 3.3-12 Aggregated blood pressure oscillations elicited by apnea events in Stage 2: (c) DBP recordings for all analyzed apnea events for all subjects; (d) Aggregated DBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

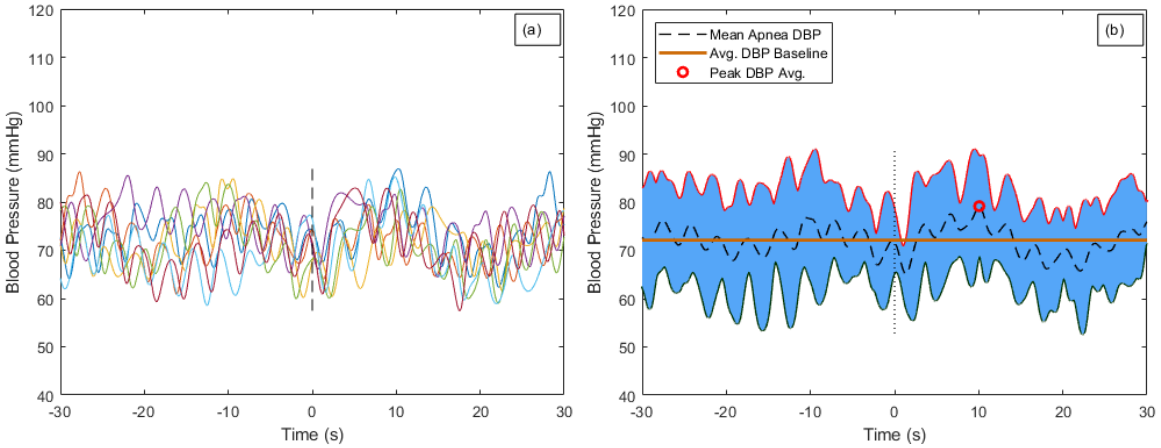

Figure 3.3-13 Aggregated blood pressure oscillations elicited by apnea events in Stage 3: (a) DBP recordings for all analyzed apnea events for all subjects; (b) Aggregated DBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

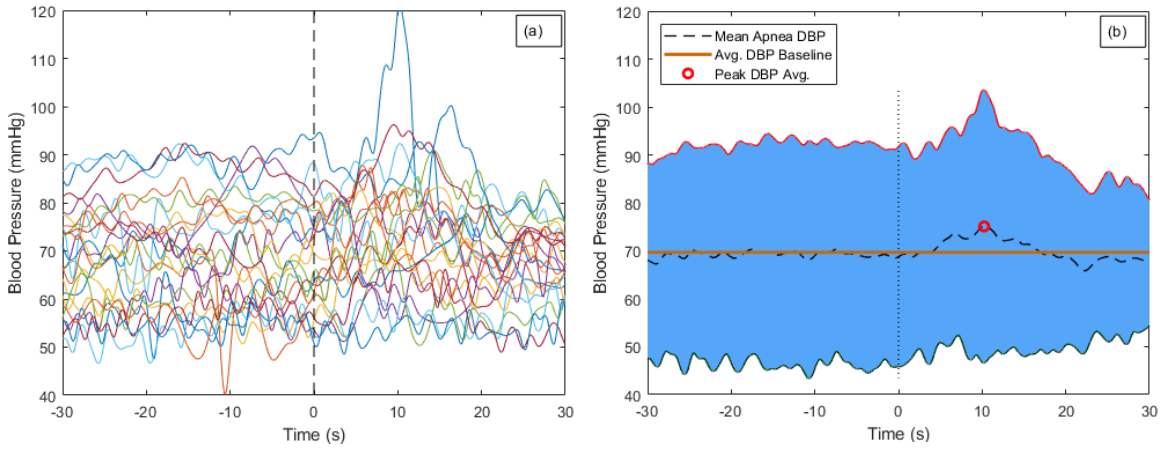

Figure 3.3-14 Aggregated blood pressure oscillations elicited by apnea events in REM: (a) DBP recordings for all analyzed apnea events for all subjects; (b) Aggregated DBP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

In the statistical analysis, one-way ANOVA method was used for comparison between sleep stages. In the Baseline, the dataset is considered in the normal distribution due to the central limit theorem. Since the quantity of 1 min window in Baseline in Stage 1, 2, 3, and REM is 230, 158, 14, and 28, respectively, which each stage contains more than 60,000 values. But, in the way to examine the homogeneity with Bartlett's test, we reject the null hypothesis (p=0.01) in Table 3.3-13. Such a result indicated that the data from stage 3 should be excluded. Figure 3.3-15 is the box-whisker plots in each sleep stage. The two horizontal solid lines indicate the minimum and maximum value in that stage and the horizontal distance of the box reveals the interquartile rage, which is from the subtraction between the upper edge of the box, third quartile and the lower edge of the box, first quartile. Also, the red line inside the box shows the median of that stage. So, the data distribution in Stage 3 clearly considers as not homogeneity in Figure 3.3-15. With only Stage 1, 2 and REM in homogeneity test, we fail to reject the null hypothesis (p = 0.05) in Table 3.3-14.

Table 3.3-13 The homogeneity in Stage 1, 2, 3, and REM in DBP Baseline

SLEEP STAGE	COUNT	MEAN	STANDARD DEVIATION
REM	28	64.9	7.0
3	14	55.6	5.0
2	158	66.5	9.2
1	230	71.2	7.9
POOLED	430	68.6	8.3
BARTLETT'S STATISTIC	11.0		
DEGREES OF FREEDOM	3		
P-VALUE	0.012		

Table 3.3-14 The homogeneity in Stage 1, 2, and REM in DBP Baseline

SLEEP STAGE	COUNT	MEAN	STANDARD DEVIATION
REM	28	64.9	7.0
2	158	66.5	9.2
1	230	71.2	7.9
POOLED	416	69.0	8.4
BARTLETT'S STATISTIC	6.0		
DEGREES OF FREEDOM	2		
P-VALUE	0.050		

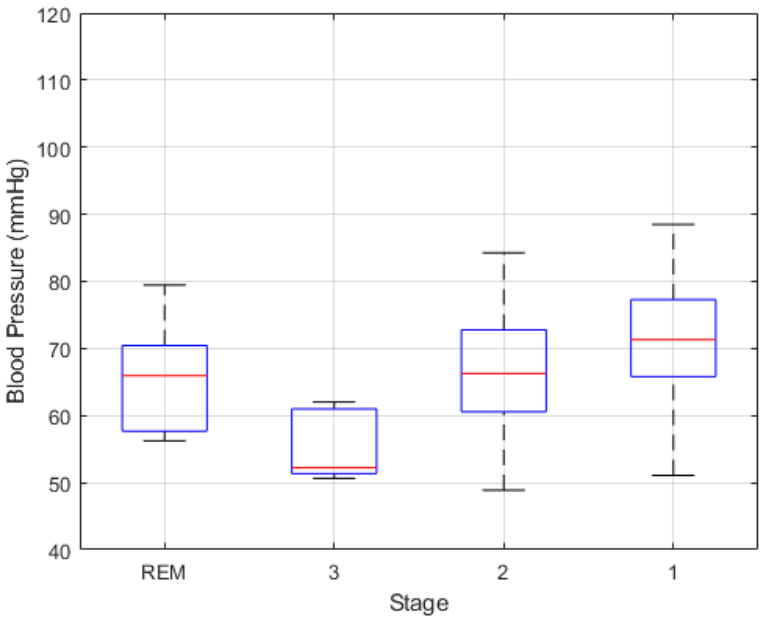

Figure 3.3-15 BP Baseline distribution in each sleep stage in DBP

From the results of ANOVA in Table 3.3-15, we rejected the null hypothesis ($p<0.0001$), which the mean values in Stage 1, 2 and REM aren't all the same. Figure 3.3-16 is a notched box plot, which is almost the same as a box-whisker plot as well as showing the 95% confidence interval (CI) in the notched. The coefficient used in calculated the boundary of 95%CI is 1.57. From Figure 3.3-16, we could roughly find out that the 95%CI between Stage 1 with either Stage 2 or REM are non-overlapping. To avoid Type II error in statistic, instead of using ordinary t-test, we applied Tukey Test to find the difference in Table 3.3-16 and found the mean value in Stage 1 is significantly difference on either Stage 2 ($p <0.0001$) or REM ($p =0.0004$). Additionally, Figure 3.3-17 is a Tukey-Kramer Confidence Interval Plot, which each horizontal line indicates the 95%CI for the corresponding stages, and there isn't overlapping between the red line in 1 with either the blue line in Stage 2 or REM.

Table 3.3-15 One-way ANOVA Summary in DBP Baseline

SOURCE	SS	DF	MS	F	PROB>F
GROUPS	2634.1	2	1317.1	18.8	1.61E-08
ERROR	29013.6	413	70.3		
TOTAL	31647.7	415			

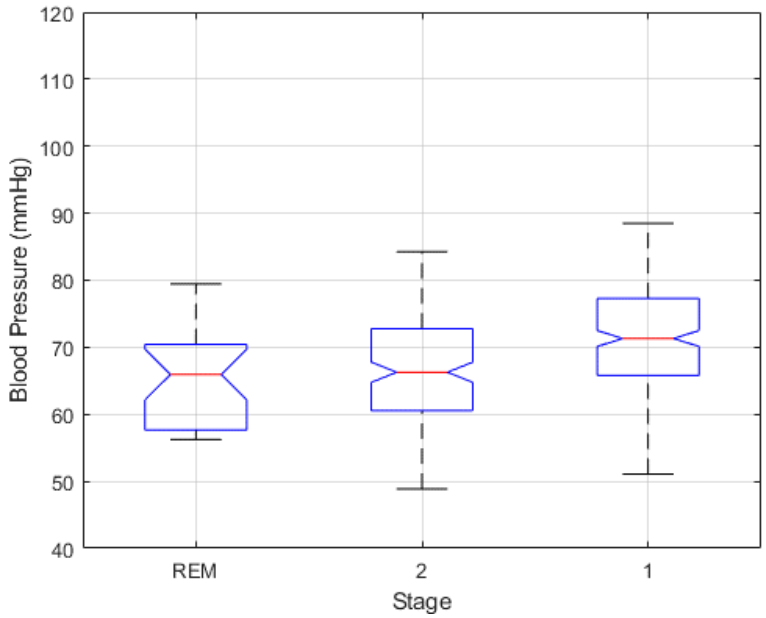

Figure 3.3-16 One-way ANOVA results in DBP Baseline

Table 3.3-16 Tukey test summary in Stage 1, 2, and REM in DBP Baseline

GROUP	95% CONFIDENCE INTERVAL	ESTIMATED MEAN DIFFERENT	P-VALUE
REM vs. 2	[-5.636, 2.420]	-1.608	0.6178
REM vs. 1	[-10.299, -2.436]	-6.367	0.0004
2 vs. 1	[-6.789, -2.730]	-4.760	< 0.0001

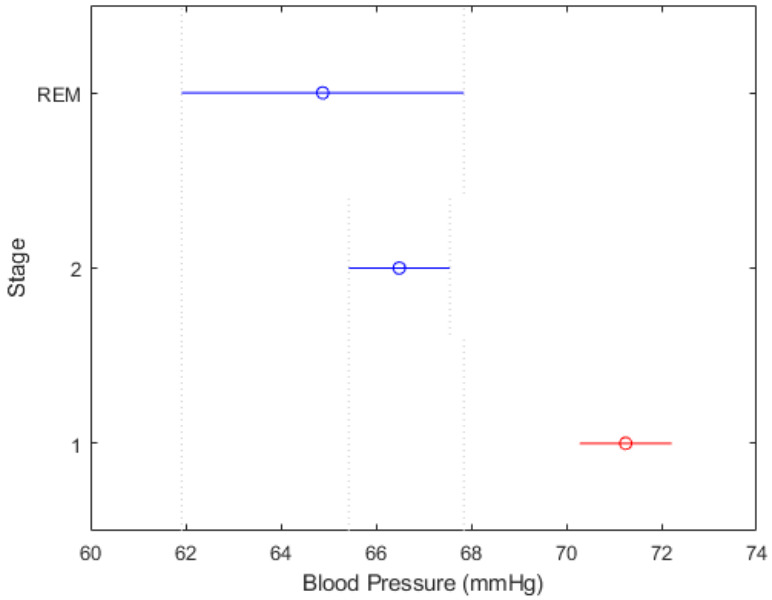

Figure 3.3-17 Tukey test result in Stage 1, 2, and REM in DBP Baseline

In the statistical comparison between sleep stages with apnea episode, we collected the blood pressure values at the temporal location of blood pressure peak in each sleep stage in

Table *3.3-12* Col.4. In DBP, ANOVA test isn't able to be used because we reject the null hypothesis in Shapiro-Wilk parametric hypothesis test, which the dataset in either sleep stages isn't in normal distribution Table 3.3-17. Therefore, we applied Wilcoxon rank sum test, a rank-based nonparametric test, to get the difference regard to the median. We fail to reject the null hypothesis, such a result that the medians in Stage 1, 2, 3, and REM are all the same.

Table 3.3-17 P-value of Shapiro-Wilk parametric hypothesis test

SLEEP STAGES	NUM	P-VALUE
REM	23	0.016
STAGE 3	7	0.001
STAGE 2	38	0.016
STAGE 1	135	0.016

Table 3.3-18 The results of Wilcoxon rank sum test

Stages	P-value	Z- value	Rank sum
REM vs. 3	0.39	-0.859	338.5
REM vs. 2	1.00	0.000	540.5
REM vs. 1	1.00	0.000	540.5
3 vs. 2	0.39	0.859	126.5
3 vs. 1	0.39	0.859	126.5
2 vs. 1	1.00	0.000	540.5

3.3.3. Analysis of Mean Arterial Blood Pressure Surges in Various Sleep Stages

In Table 3.3-19, the first four row of data shows the aggregate mean and standard deviation of SBP surges in each of the sleep stages considered for all subjects, while the rows below that show the results for each of the 10 subjects individually. Specifically, the entries in the 3rd column for the first four rows show the average and standard deviation of all sample-by-sample values of the SBP Baselines for all subjects in the stages 1, 2, 3 and REM sleep. It is noted that since no event in Stage 4 qualified for analysis – based on the criteria outlined in Section 2.7.7 – there are no entries for that stage. In addition, the entry under Col. 3 in the first four rows of the table shows the average and standard deviation of the peak SBP values during all apnea events respectively to each sleep stage. The corresponding values of the entries just described for individual subjects are presented in rows

labeled No. 1 to No.10 in Table 3.3-19. In the t-test results, there is no difference (p-value > 5%) in the comparison between normal value and the peak value from each subject in Table 3.3-19.

Table 3.3-19 MAP variation in different sleep stages

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7	Col. 8
STAGE	**Subject**	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
1	All	93.6 ± 3.7	102.2 ± 11.6	8.6	9.2%	5.8	135
2	All	88.8 ± 3.8	98.5 ± 9.2	9.7	10.9%	6.4	38
3	All	87.5 ± 3.4	96.3 ± 6.7	8.7	9.9%	9.6	7
REM	All	96.2 ± 2.2	102.3 ± 18.1	6.1	6.3%	9.6	23
1	1	86.6 ± 3.1	94.3 ± 7.2	7.6	8.8%	9.5	7
1	8	92.3 ± 5.4	104.8 ± 7.6	12.4	13.4%	4.6	12
1	4	102.4 ± 5.7	111.7 ± 8.4	9.3	9.1%	11.1	25
1	7	74.6 ± 3.4	84.0 ± 5.0	9.4	12.6%	10.7	5
1	10	84.0 ± 5.2	99.2 ± 10.8	15.2	18.1%	5.2	7
1	5	90.0 ± 3.9	100.4 ± 12.4	10.3	11.4%	5.7	27
2	1	84.3 ± 4.0	94.5 ± 8.7	10.2	12.1%	6.7	12
2	10	86.5 ± 6.2	104.7 ± 10.2	18.2	21.0%	4.3	13
REM	6	101.0 ± 2.1	105.6 ± 5.3	4.6	4.6%	24.4	10
REM	7	75.2 ± 2.5	81.5 ± 6.7	6.3	8.4%	9.7	6

The brown horizontal lines in the Figure 3.3-18 b, Figure 3.3-19 b, Figure 3.3-20 b, and Figure 3.3-21 b show the average result of the Baseline. In these figures, the plots on the left side displays all the MAP trajectories during the qualified apnea events within the corresponding sleep stage (i.e., stage 1, 2, 3, and REM). The plots on the right side of the figures show the result of aggregating these trajectories for each corresponding sleep stages as well as the corresponding 95% CI. Additionally, the red circles in the right-side figures mark the peak of the sample-by-sample mean apnea MAP surges corresponding to the apnea episodes.

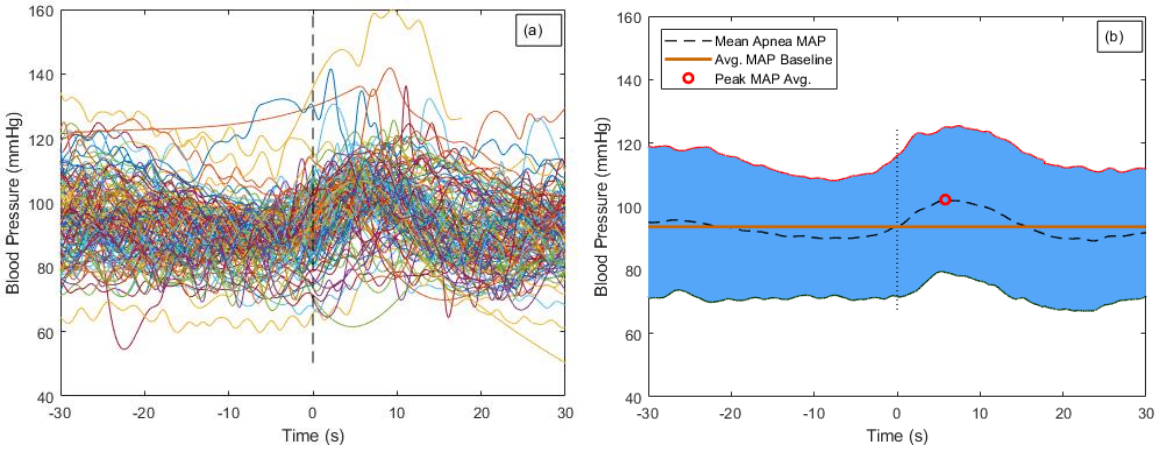

Figure 3.3-18 Aggregated blood pressure oscillations elicited by apnea events in Stage 1: (a) MAP recordings for all analyzed apnea events for all subjects; (b) Aggregated MAP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

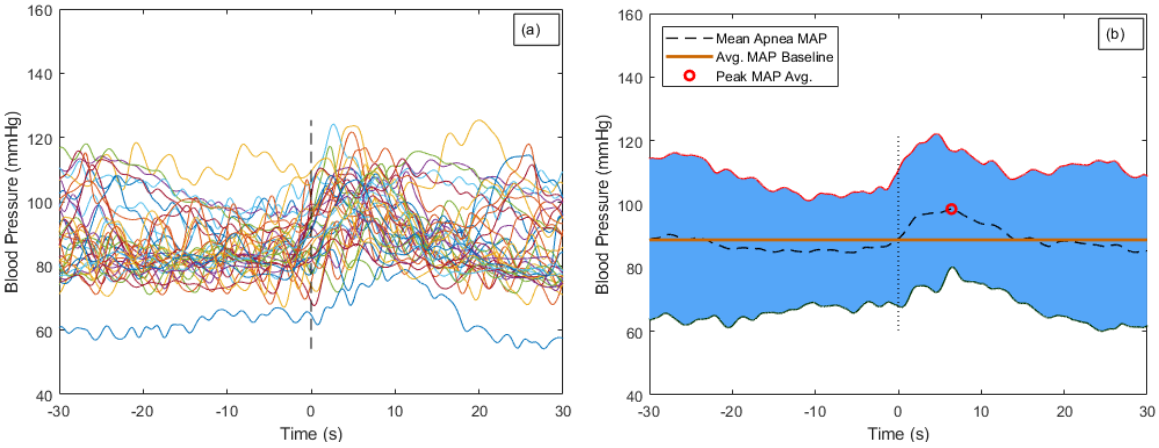

Figure 3.3-19 Aggregated blood pressure oscillations elicited by apnea events in Stage 2: (c) MAP recordings for all analyzed apnea events for all subjects; (d) Aggregated MAP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

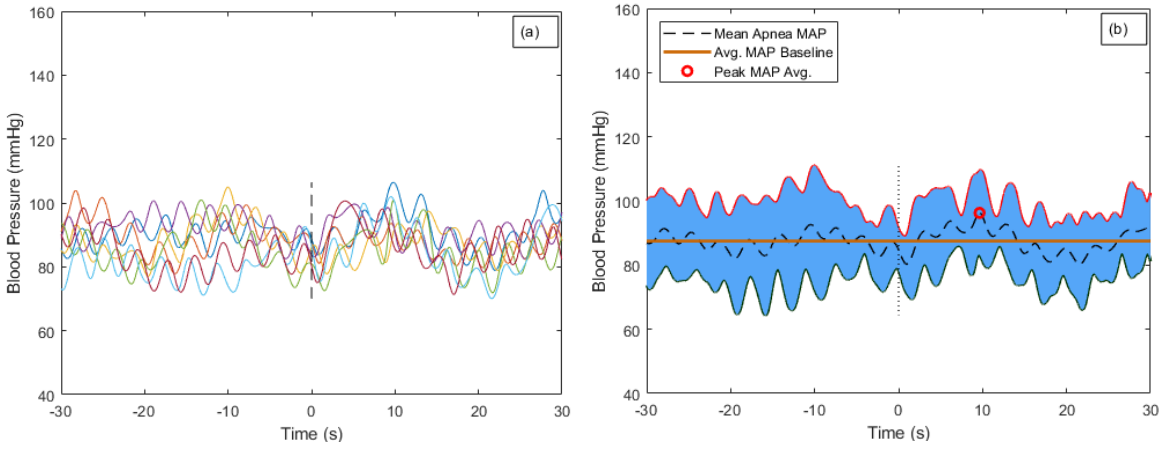

Figure 3.3-20 Aggregated blood pressure oscillations elicited by apnea events in Stage 3: (a) MAP recordings for all analyzed apnea events for all subjects; (b) Aggregated MAP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

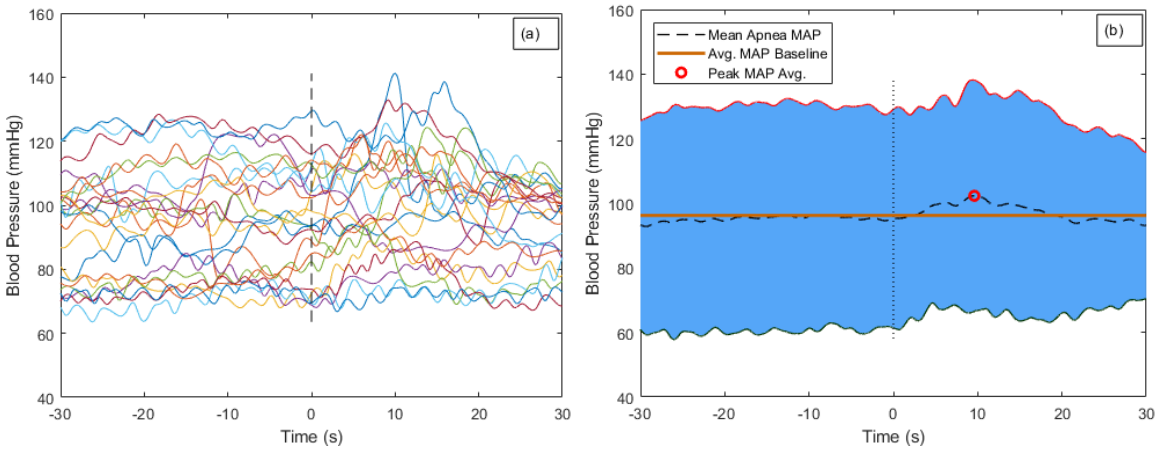

Figure 3.3-21 Aggregated blood pressure oscillations elicited by apnea events in REM: (a) MAP recordings for all analyzed apnea events for all subjects; (b) Aggregated MAP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

3.3.4. Analysis of Pulse Pressure Surges in Various Sleep Stages

The first four row of data in Table 3.3-20 shows the aggregate mean and standard deviation of SBP surges in each of the sleep stages considered for all subjects, while the rows below that show the results for each of the 10 subjects individually. Specifically, the entries in the 3rd column for the first four rows show the average and standard deviation of all sample-by-sample values of the SBP Baselines for all subjects in the stages 1, 2, 3 and REM sleep. It is noted that since no event in Stage 4 qualified for analysis – based on the criteria outlined in Section 2.7.7 – there are no entries for that stage. In addition, the entry under Col. 3 in the first four rows of the table shows the average and standard deviation of the peak SBP values during all apnea events respectively to each sleep stage. The corresponding values of the entries just described for individual subjects are presented in rows labeled No. 1 to No.10 in Table 3.3-20. In the t-test results, there is no difference (p-value > 5%) in the comparison between normal value and the peak value from each subject in Table 3.3-20.

Table 3.3-20 PP variation in different sleep stages

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7	Col. 8
STAGE	**Subject**	**Mean Baseline BP (mmHg)**	**Mean Peak BP during Apnea (mmHg)**	**Difference of Mean Peak and Baseline BP (mm Hg)**	**% Difference of Mean Peak and Baseline BP**	**Temporal Location of BP Peak (s)**	**No. of Apnea Events Averaged**
1	All	65.1 ± 3.2	71.6 ± 21.2	6.5	10.0%	9.1	135
2	All	59.3 ± 2.7	65.2 ± 15.6	5.8	9.8%	7.3	38
3	All	45.9 ± 3.6	57.7 ± 11.3	11.8	25.7%	13.7	7
REM	All	79.1 ± 2.3	84.4 ± 23.9	5.3	6.7%	15.8	23
1	1	52.1 ± 2.1	56.6 ± 3.2	4.5	8.6%	10.9	7
1	8	64.9 ± 3.5	71.8 ± 12.4	6.9	10.6%	9.6	12
1	4	81.8 ± 5.3	89.1 ± 8.9	7.2	8.8%	-23.1	25
1	7	52.2 ± 2.7	59.7 ± 3.0	7.5	14.4%	11.5	5
1	10	60.4 ± 5.6	76.1 ± 21.4	15.7	26.0%	8.4	7
1	5	45.5 ± 2.2	51.3 ± 30.8	5.8	12.7%	8.2	27
2	1	48.5 ± 2.8	54.9 ± 7.3	6.3	13.0%	12.6	12
2	10	63.1 ± 4.5	75.9 ± 11.4	12.8	20.3%	6.1	13
REM	6	88.6 ± 3.1	94.4 ± 9.5	5.8	6.5%	18.0	10
REM	7	52.1 ± 2.4	58.3 ± 5.8	6.2	11.9%	14.1	6

The brown horizontal lines in the Figure 3.3-22 b, Figure 3.3-23 b, Figure 3.3-24 b, and Figure 3.3-25 b show the average result of the Baseline. In these figures, the plots on the left side displays all the PP trajectories during the qualified apnea events within the corresponding sleep stage (i.e., stage 1, 2, 3, and REM) The plots on the right side of the figures show the result of aggregating these trajectories for each corresponding sleep stages as well as the corresponding 95% CI. Additionally, the red circles in the right-side figures mark the peak of the sample-by-sample mean apnea PP surges corresponding to the apnea episodes.

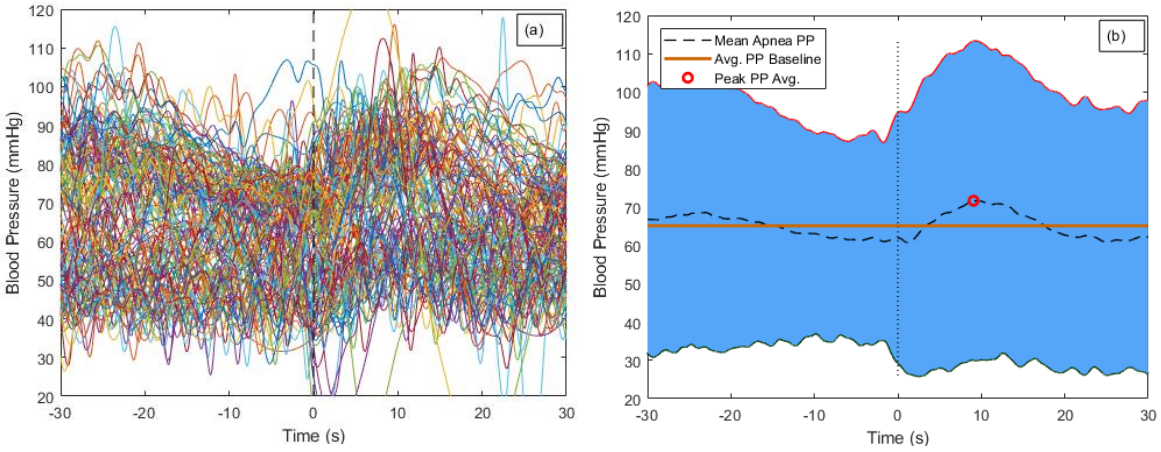

Figure 3.3-22 Aggregated blood pressure oscillations elicited by apnea events in Stage 1: (a) PP recordings for all analyzed apnea events for all subjects; (b) Aggregated PP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

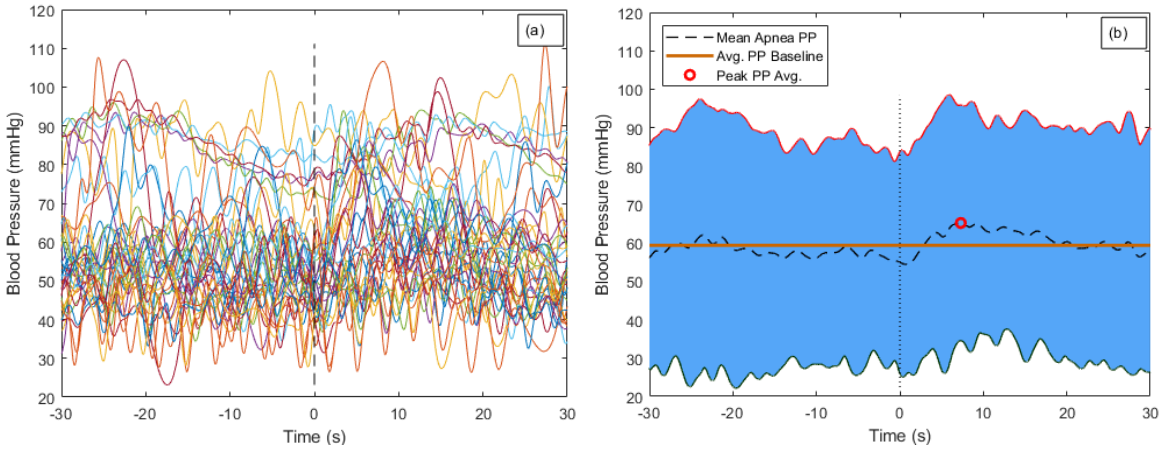

Figure 3.3-23 Aggregated blood pressure oscillations elicited by apnea events in Stage 2: (c) PP recordings for all analyzed apnea events for all subjects; (d) Aggregated PP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

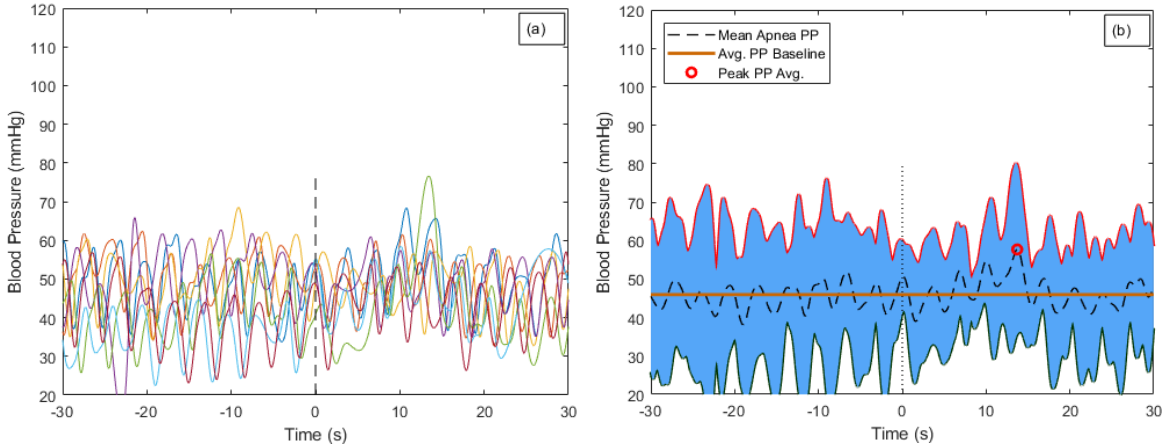

Figure 3.3-24 Aggregated blood pressure oscillations elicited by apnea events in Stage 3: (a) PP recordings for all analyzed apnea events for all subjects; (b) Aggregated PP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

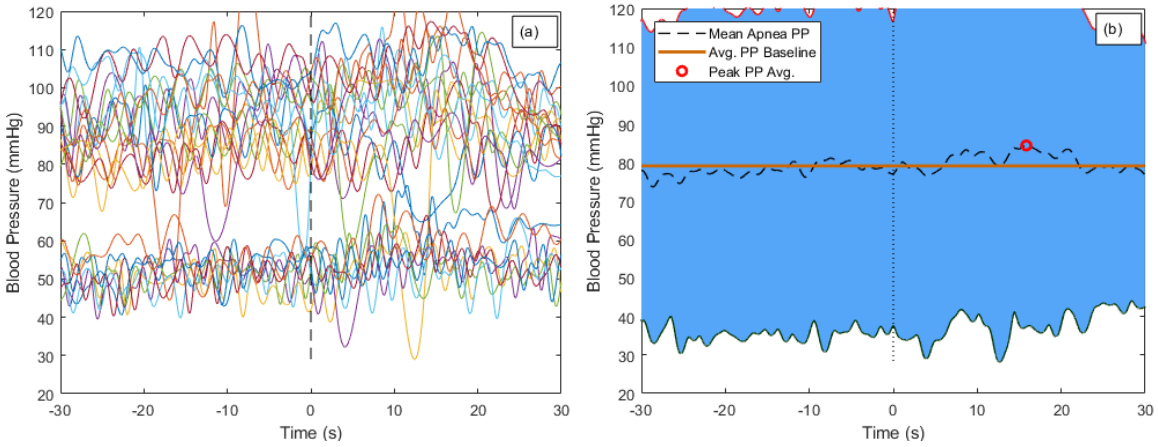

Figure 3.3-25 Aggregated blood pressure oscillations elicited by apnea events in REM: (a) PP recordings for all analyzed apnea events for all subjects; (b) Aggregated PP oscillations elicited by apnea with 95% confidence interval envelope (shaded area);

3.3.5. Analysis of Rate of Systolic Blood Pressure Surges in Various Sleep Stages

Based on the method that was detailed in Section 2.7.8., the possible impact of sleep stages on the rate of blood pressure surge due to an apnea was studied. The first row of data in

In the t-test results, Stage 1, 2 and REM show significantly difference (p-value < 0.1%) in the comparison between the Baseline slope and the slope in apnea episodes from each subject in

Table 3.3-21, but there is no difference in the comparison between the Baseline and apnea epochs intercept in any sleep stages. Also, p-value are all greater than 5% in the comparison between sleep stages in either Baseline or apnea events, which reveal that there is no difference.

Table 3.3-21 shows the aggregate results of the slop of SBC in all subjects, while the rows below that show the results for each of the 10 subjects individually. Similarly, the mean and standard deviation of the slope and intercept values for normal breathing (i.e. no apnea event present) are shown in Col. 3 and Col. 4. Further, Col. 5 and Col. 6 display the slope and intercept values associated with the apnea events. Column 7 shows the total number of apnea events that satisfied the set criteria in Section 2.7.7 and contributed to the results shown in each row of the table.

In the t-test results, Stage 1, 2 and REM show significantly difference (p-value < 0.1%) in the comparison between the Baseline slope and the slope in apnea episodes from each subject in

Table 3.3-21, but there is no difference in the comparison between the Baseline and apnea epochs intercept in any sleep stages. Also, p-value are all greater than 5% in the comparison between sleep stages in either Baseline or apnea events, which reveal that there is no difference.

Table 3.3-21 Slope and intercept results from apnea events in different sleep stages in SBP

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7
Stage	**Subject**	**Slope for Normal Breathing (mmHg/Sec)**	**Intercept for Normal Breathing (mmHg)**	**Slope for Apnea Events (mmHg/Sec)**	**Intercept for Apnea Events (mmHg)**	**No. of Apnea Events**
1	All	0.01 ± 0.2	137.9 ± 23.7	0.95 ± 0.7	152.0 ± 17.0	135
2	All	-0.02 ± 0.2	134.7 ± 24.6	0.93 ± 0.6	144.2 ± 19.4	38
3	All	0.00 ± 0.1	104.0 ± 12.6	0.89 ± 1.0	128.3 ± 9.8	7

REM	All	0.06 ± 0.2	133.3 ± 22.6	0.68 ± 0.4	163.1 ± 31.2	23
1	1	0.07 ± 0.2	113.1 ± 10.0	0.52 ± 0.3	131.5 ± 5.7	7
Col. 1	COL. 2	COL. 3	COL. 4	COL. 5	COL. 6	COL. 7
Stage	**Subject**	**Slope for Normal Breathing (mmHg/Sec)**	**Intercept for Normal Breathing (mmHg)**	**Slope for Apnea Events (mmHg/Sec)**	**Intercept for Apnea Events (mmHg)**	**No. of Apnea Events**
1	2	-0.11 ± 0.4	149.0 ± 18.8	0.79 ± 0.5	159.6 ± 8.4	40
1	3	-0.03 ± 0.1	125.7 ± 25.7	1.97 ± 1.7	148.6 ± 25.3	5
1	8	-0.01 ± 0.3	133.4 ± 11.2	0.94 ± 0.3	150.6 ± 9.8	12
1	6	0.01 ± 0.2	158.3 ± 15.6	0.14 ± 0.1	153.9 ± 30.7	4
1	4	-0.02 ± 0.4	156.3 ± 14.9	0.80 ± 0.8	169.1 ± 13.3	25
1	7	0.01 ± 0.2	106.7 ± 10.9	0.63 ± 0.3	119.4 ± 5.2	5
1	9	0.04 ± 0.1	140.6 ± 9.9	1.18 ± 0.8	156.1 ± 5.0	3
1	10	0.01 ± 0.2	129.0 ± 13.3	1.74 ± 1.2	142.6 ± 9.9	7
1	5	-0.05 ± 0.3	121.1 ± 12.8	1.19 ± 0.5	139.1 ± 7.1	27
2	1	-0.24 ± 0.2	130.6 ± 11.1	0.90 ± 0.4	128.4 ± 8.8	12
2	2	-0.20 ± 0.4	150.1 ± 17.0	0.62 ± 0.0	158.3 ± 0.0	1
2	3	-0.05 ± 0.1	115.7 ± 7.9	0.74 ± 0.5	136.7 ± 16.6	3
2	6	0.02 ± 0.2	159.9 ± 15.9	0.91 ± 0.0	172.8 ± 0.0	1
2	4	0.00 ± 0.4	153.1 ± 12.7	0.14 ± 0.1	172.1 ± 3.2	4
2	7	-0.01 ± 0.1	104.7 ± 10.0	1.23 ± 0.0	105.6 ± 0.0	1
2	9	0.01 ± 0.2	143.1 ± 11.1	0.47 ± 0.0	149.1 ± 0.0	1
2	10	-0.02 ± 0.2	128.4 ± 13.2	1.22 ± 0.8	153.4 ± 11.7	13
2	5	-0.09 ± 0.2	119.0 ± 12.1	1.41 ± 0.0	130.4 ± 23.2	2
3	3	0.12 ± 0.0	114.9 ± 0.0	0.89 ± 1.0	128.3 ± 9.8	7
REM	1	0.31 ± 0.5	117.7 ± 6.4	1.20 ± 0.0	175.8 ± 0.0	1
REM	6	0.04 ± 0.1	152.0 ± 10.3	0.78 ± 0.4	175.0 ± 9.1	10
REM	4	0.19 ± 0.1	160.7 ± 14.9	0.49 ± 0.2	196.0 ± 11.5	5
REM	7	0.06 ± 0.1	109.1 ± 5.6	0.57 ± 0.4	118.0 ± 6.2	6
REM	10	-0.05 ± 0.2	128.8 ± 19.8	0.79 ± 0.0	136.0 ± 0.0	1

Figure 3.3-26 provides an average prediction for the blood pressure rises in the apnea events in 60s window. The solid lines are calculated by using average slope and average intercept with the aggregated result from all apnea events in

Table 3.3-21. The horizontal dashed line in Figure 3.3-26 indicates stable BP during normal breathing. At 30s, the blood pressure surges to 179 mmHg, 171 mmHg, 155 mmHg and 184 mmHg in the order of Stage 1, 2, 3 and REM.

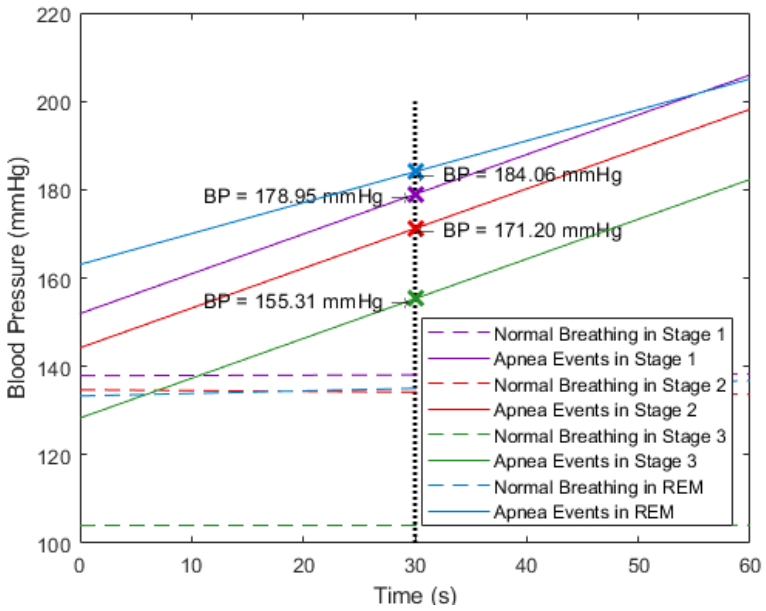

Figure 3.3-26 Slope plot in 60 in SBP in each sleep stages; the red cross indicates the value at 30s

3.3.6. Analysis of Rate of Systolic Blood Pressure Surges in Various Sleep Stages

In slope analysis, it follows the criteria in Section 2.7.8. The possible impact of sleep stages on the rate of blood pressure surge due to an apnea was studied. The first row of data in

Table *3.3-22* shows the aggregate results of the slop of SBC in all subjects, while the rows below that show the results for each of the 10 subjects individually. Similarly, the mean and standard deviation of the slope and intercept values for normal breathing (i.e. no apnea event present) are shown in Col. 3 and Col. 4. Further, Col. 5 and Col. 6 display the slope and intercept values associated with the apnea events. Column 7 shows the total number of apnea events that satisfied the set criteria in Section 2.7.7 and contributed to the results shown in each row of the table.

In the t-test results, Stage 1, 2 and REM show significantly difference (p-value < 1%) in the comparison between Baseline slope and the slope in apnea episodes from each subject in

Table *3.3-22*. In the comparison between Baseline and apnea events intercept, Stage 1 and 2 also shows significantly difference (p-value < 5%), but not in REM. In addition, p-value are all greater than 5% in the comparison in both slope and intercept between sleep stages in either Baseline or apnea events in, which reveal that there is no difference.

Table 3.3-22 Slope and intercept results from apnea events in different sleep stages in DBP

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7
Stage	**Subject**	**Slope in Normal Breathing (mmHg/Sec)**	**Intercept in Normal Breathing (mmHg)**	**Slope in Apnea Events (mmHg/Sec)**	**Intercept in Apnea Events (mmHg)**	**No. of Apnea Events**
1	All	0.00 ± 0.1	69.3 ± 9.6	0.64 ± 0.5	81.1 ± 8.7	135
2	All	-0.02 ± 0.1	68.1 ± 10.7	0.69 ± 0.5	81.4 ± 8.4	38
3	All	0.00 ± 0.1	55.8 ± 7.9	0.53 ± 0.5	78.4 ± 3.2	7
REM	All	0.03 ± 0.2	64.2 ± 9.4	0.39 ± 0.2	78.5 ± 12.7	23
1	1	0.04 ± 0.1	66.2 ± 6.5	0.46 ± 0.2	77.3 ± 5.5	7
1	2	-0.08 ± 0.1	71.9 ± 5.8	0.51 ± 0.3	78.6 ± 4.1	40
1	3	0.04 ± 0.1	73.9 ± 16.0	0.76 ± 0.6	95.0 ± 20.3	5
1	8	0.02 ± 0.2	67.1 ± 7.6	0.89 ± 0.6	83.6 ± 5.6	12
1	6	0.00 ± 0.1	73.7 ± 8.3	0.05 ± 0.1	77.0 ± 9.6	4
1	4	-0.02 ± 0.2	74.2 ± 6.2	0.56 ± 0.5	82.2 ± 8.4	25
1	7	0.01 ± 0.1	57.8 ± 6.2	0.48 ± 0.2	64.7 ± 3.8	5
1	9	0.03 ± 0.1	76.4 ± 6.1	0.85 ± 0.3	86.2 ± 3.9	3
1	10	0.01 ± 0.2	64.2 ± 6.4	1.28 ± 0.8	77.1 ± 9.8	7
1	5	-0.03 ± 0.2	74.6 ± 9.4	0.77 ± 0.4	85.4 ± 6.5	27
2	1	-0.20 ± 0.1	77.5 ± 8.1	0.62 ± 0.3	77.1 ± 5.4	12
2	2	-0.14 ± 0.2	72.3 ± 6.3	0.15 ± 0.0	76.1 ± 0.0	1
2	3	0.02 ± 0.1	68.9 ± 9.2	0.56 ± 0.2	85.8 ± 5.0	3
2	6	0.00 ± 0.1	74.2 ± 8.7	0.57 ± 0.0	86.0 ± 0.0	1
2	4	-0.02 ± 0.2	73.8 ± 6.1	0.14 ± 0.0	81.8 ± 2.4	4
2	7	-0.01 ± 0.1	56.7 ± 6.1	0.77 ± 0.0	56.0 ± 0.0	1
2	9	-0.02 ± 0.1	78.5 ± 6.8	0.35 ± 0.0	85.5 ± 0.0	1
2	10	0.00 ± 0.2	63.2 ± 7.7	0.99 ± 0.6	85.8 ± 7.8	13
2	5	-0.07 ± 0.2	74.2 ± 8.5	0.94 ± 0.1	82.7 ± 15.4	2
3	3	0.17 ± 0.0	67.1 ± 0.0	0.53 ± 0.5	78.4 ± 3.2	7
REM	1	0.25 ± 0.4	68.0 ± 6.8	0.93 ± 0.0	111.5 ± 0.0	1
REM	6	0.02 ± 0.1	68.2 ± 7.6	0.37 ± 0.3	79.6 ± 2.7	10
REM	4	-0.06 ± 0.1	75.9 ± 2.8	0.25 ± 0.1	89.6 ± 5.7	5
REM	7	0.05 ± 0.1	55.6 ± 4.5	0.40 ± 0.2	62.8 ± 5.2	6
REM	10	-0.02 ± 0.2	64.2 ± 10.8	0.58 ± 0.0	72.8 ± 0.0	1

Figure 3.3-27 provides an average prediction for the blood pressure rises in the apnea events in 60s window. The solid lines are calculated by using average slope and average intercept with the aggregated result from all apnea events in

Table *3.3-22*. The horizontal dashed line in Figure 3.3-27 indicates stable BP during normal breathing. At 30s, the blood pressure surges to 100 mmHg, 102 mmHg, 94 mmHg and 90 mmHg in the order of Stage 1, 2, 3 and REM.

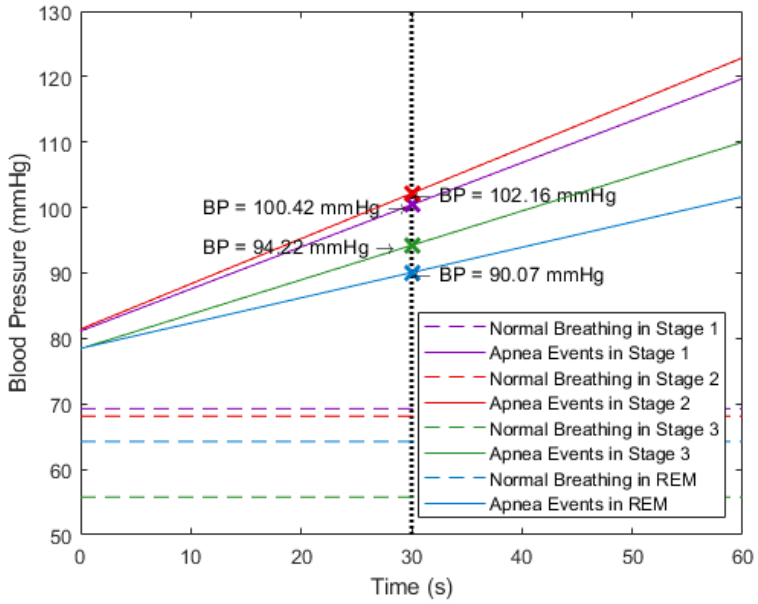

Figure 3.3-27 Slope plot in 60 in DBP in each sleep stages; the red cross indicates the value at 30s

3.4. Subject Time of Sleep Summary

One of the physiologically important parameters for determining the quality of sleep is the duration of each sleep stage. The total time in bed is an account of the elapsed time between when the lights in the bedroom of the sleep lab is turned off until when the test is terminated in the morning and lights are turned back on. Further, to obtain the actually sleep time for subjects, we totaled the duration of each of the stages. Figure 3.4-1 shows the time in bed and actual sleep duration for all 10 subjects. The average percentages of each sleep stage are shown in Figure 3.4-2. The proportion of Stage 1 is around 61%; 29% in Stage 2, around 2% in Stage 3 and around 8% in REM. Another measure of interest is the distribution of the number of events in various sleep stages. Using the count of events given in Table 3.3-2, one can obtain this distribution as shown in Figure 3.4-3.

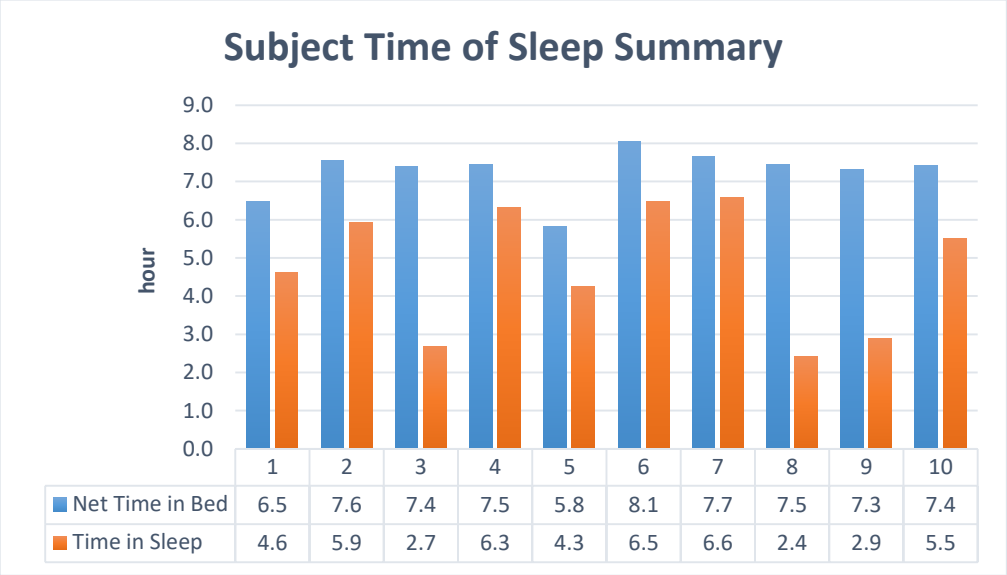

Figure 3.4-1 Subject Time of Sleep Summary n all 10 subjects

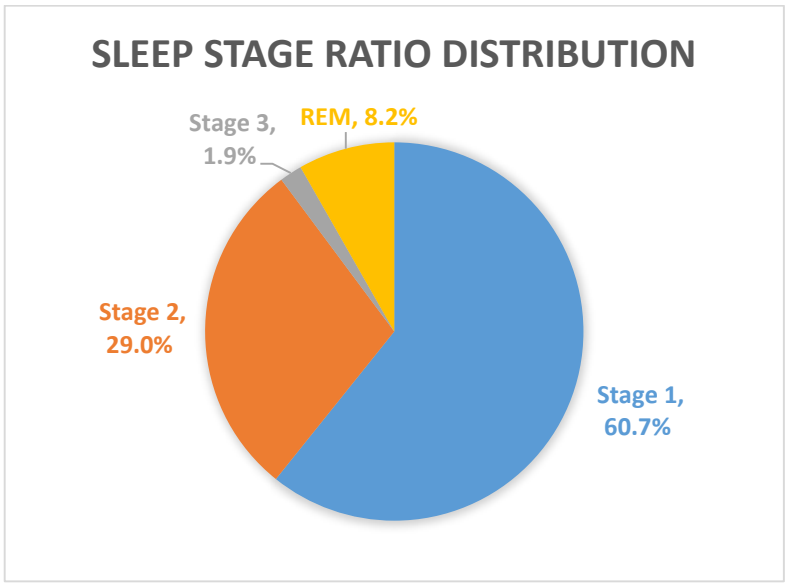

Figure 3.4-2 Aggregate results for sleep stages in all 10 subjects

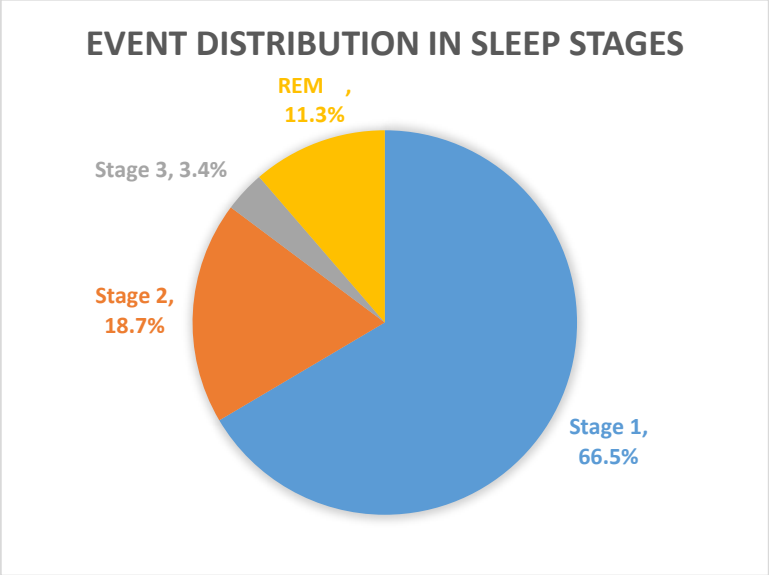

Figure 3.4-3 Aggregated result of event distribution in each sleep stage

Chapter 4

Discussion and Conclusion

4.1. Discussion

The aim of the study presented here was to provide an analysis method that yields clinically relevant information about nocturnal blood pressure variations in sleep apnea patients. Continuous recording of blood pressure, as reported in this study, is of importance, because it can reveal the severity and frequency of oscillations in the patient blood pressure throughout the night. Considering that long continuous recording (6 to 8 hours) of blood pressure produces a high volume of data, an approach to aggregate this high volume of data for clinical and scientific study is of interest. The results can help us understand more into the various physiological changes occurring because of OSA. This section deals with the interpretation and significance of the results in chapter 3.

4.1.1. Comparison of Aggregated Sleep Apnea Events with the Blood Pressure in Normal Breathing

Examining the Baseline values in aggregate averages in SBP and DBP in Section 3.1 reveals that patients in the sample population had an overall relatively stable BP during normal breathing. This is evidenced by the levels of MAP and PP being normal. Further, one can observe from the

Table *3.1-1* and Table 3.1-2 in Section 3.1, that the standard deviations of SBP, DBP, MAP, and PP during the Baseline for the subjects are small (between 1.4 to 4.5 mmHg).

This is in contrast with the distinct oscillations observed in all four quantitative measures (i.e., SBP, DBP, MBP, and PP) when an apnea event is present (Figure 3.2-1 through Figure 3.2-4). The level of the oscillations in the BP post selected apnea events is reflected in the sharp rise in the mean and standard deviation of all four measures in the tables in Section 3.2 which shows that the peak of the average of SBP surges over 17.6 mmHg (around 14%), average of DBP surges by 9.5 mmHg (around 14%), average of MAP surges by 9.5 mmHg (around 14%) and average of PP surges by 9.8 mmHg (over 15%) with respect to their respective Baseline average values in the order of Table 3.2-1 through

Table *3.2-4*. The t-test of the mean values for the Baseline BP and the peak BP elicited by apnea in Table 3.2-1,

Table *3.2-2* and Table 3.2-3 showed that they are significantly different ($p<0.001$) in SBP, DBP, and MAP. The p-value in PP is 6%, which isn't different. Since PP is calculated from the subtraction between SBP and DBP, the change is compensated by the formula. The greater variations in the blood pressure elicited by apnea events results in a relatively wide 95% confidence interval envelopes in the part b in Figure 3.2-1 and Figure 3.2-2, indicating that the expected range of oscillations can be in excess of 50 mmHg. Such large oscillations in the nocturnal BP would be of interest to clinicians as they may have health implications.

Considering part b in Figure 3.2-1 through Figure 3.2-4, it can be observed that the peak of oscillations that are induced by apnea events do not occur within the apnea interval, rather they occur after the apnea has ended. As the aggregated results in Table 3.2-1 through

Table *3.2-4* show the average delay to the peak are between from 6s to 9s. Another observation of interest in the figures is the average trajectory of the BP changes elicited by apnea events. As can be seen, the trajectories of all four measures return toward the Baseline after peaking are oscillatory, indicating underdamped overall BP restoration control to the pre-apnea level. From Figure 3.2-1 through Figure 3.2-4, the surged blood pressure take about 20s to the return to the baseline after the end of apnea. This interval can serve as a preliminary estimation of the blood pressure recovery, if no other apnea event occurs. It also shows the adequacy of selecting the 30 s interval as the minimum temporal distance between apnea events for the analysis. This interval length allowed one to observe the recovery due to separated apnea events.

In the results of analysis of the slope of the BP surges due to apnea events, one observes that on the average BP rises about 1 mmHg every second (Section 3.2.5) . By applying aggregated result of the temporal location of BP peak, 7s (in Table 3.2-1), we could get 155.6 mmHg with the intercept in 149 mmHg. However, the mean peak BP during apnea episodes in Table 3.2-1 Col. 3 is 146.3 mmHg. Such a difference reveals estimating the surge by utilizing the slope may be higher than the mean peak BP values. With respect to the variations in DBP, the slope in apnea events is around 0.6 mmHg/s (Section 3.2.6). Hence, the blood pressure raises to around 85 mmHg with temporal location of BP peak in 6.4s. The estimated surge level from the slope equation is still higher than the accumulated result in

Table *3.2-2*.

4.1.2. Effect of Sleep Stage on Blood Pressure Surges Elicited by Apnea Events

Analysis of the mean values of SBP, MAP and PP during Baseline showed that the mean values exhibit a general decreasing trend as the patient moves from stage 1 to deeper stages (Table 3.3-2, Table 3.3-19 and Table 3.3-20). Specifically, means and one standard deviations of SBP during Baseline were 137.1 ± 5.6, 128.4 ± 5.1, and 117.9 ± 4.9 mmHg in stages 1, 2, and 3 respectively in Table 3.3-2. In MAP in Table 3.3-19, means and one standard deviations during Baseline were 93.6 ± 3.7, 88.8 ± 3.8, 87.5 ± 3.4 mmHg in stages 1, 2, and 3 respectively, as well as in PP in Table 3.3-20. The means and one standard deviations during Baseline in stage 1, 2, and 3 were 65.1 ± 3.2, 59.3 ± 2.7, 45.9 ± 3.6, respectively. This phenomenon is reasonable since the body and brain waves slowdown in NREMS [38]. In addition, due to autonomic instability, blood pressure fluctuations increased in REM [39]. However, this phenomenon isn't showed in of the mean values of DBP during Baseline in

Table *3.3-12*. The standard deviations of SBP, DBP, MAP, and PP of the Baseline in Stage 1, 2, 3, and REM for the subjects are between 1.8 to 5.6 mmHg.

Contrarily, these are the distinct oscillations observed in Col. 4 in Table 3.3-2 through Table 3.3-20 associated with apnea events that can be seen in part b figures in Section 3.3, respectively. The level of the oscillations in the BP post selected apnea events is reflected in the sharp rise in the mean and standard deviation in either stages of all four measures in Table 3.3-2 through Table 3.3-20. In Table 3.3-2 through Table 3.3-20 Col. 5 and 6, the peak of the average of BP measures in stage 1, 2 and 3 surge level between 8.7 to 14.4 mmHg and the percentage change from 9% to around 12%, respectively. In REM, the peak average of SBP surges only 8.7 mmHg (around 6%). In DBP, the surge of the average peak in either stage 1, 2 and 3 are similar between 7 to 9 mmHg (around 10%). The peak average of DBP in REM is only around 5.5 mmHg (8%), which isn't likely in SBP to have a larger change. The track of the change in the peak of the average BP in MAP is alike in SBP. The surges in Stage 1, 2 and 3 are around 9 mmHg (10%) and, in REM, is around 6 mmHg (6%) which change is less. In PP, the peak of the average surge in Stage 1 and 2 are likely around 6 mmHg (10%) and around 5.3 mmHg (7%) change in REM. Specifically, in Stage 3, we found out that the peak of the average surge to around 12 mmHg (26%). In Col. 7 in Table 3.3-2 through Table 3.3-20, the temporal location of BP peak in Stage 1 and 2 in the aggregate results occur between from 6.5s to 9s, which are much earlier in Stage 3 and REM, 9.8s to 16s. Additionally, in the part a in Figure 3.3-25, the division is inconclusive. Currently, there are only 23 events in REM, fitting our criteria in Section 2.7.7, in 5 subjects, which further investigation is needed with more subjects.

In the statistical analysis in the Baseline, ANOVA method was used for comparison between sleep stages. From Table 3.3-5, the mean values of Baseline in SBP in Stage 1, 2 and REM aren't all the same. Then, in the Tukey test, we found out that the mean value in Stage 1 is significantly difference with Stage 2. Such a result that there is fluctuation in BP Baseline between Stage 1 and 2, but not in REM. Further, in the comparison between sleep stages elicited by apnea events by ANOVA test, the mean values in SBP with apnea episodes in Stage 1, 2 and 3 aren't all the same. After running the Tukey test, we discovered the mean value in Stage 1 is significantly difference with Stage 3. Thus, since the Baseline in SBP in Stage 1 and 2 are different, but not in peak BP, we could

conclude that the BP variation due to apnea events is different between Stage 1 and 2.

Similarly, ANOVA method was used for comparison between sleep stages in the DBP Baseline in the statistical analysis. From Table *3.3-15*, the mean values of Baseline in DBP in Stage 1, 2 and REM aren't all the same. Then, in the Tukey test in Table *3.3-16*, we found out that the mean value in Stage 1 is significantly difference with either Stage 2 or REM. Such a result that there is fluctuation in BP Baseline in Stage 1 2, and REM. Further, in the comparison between sleep stages elicited by apnea events, the ANOVA test is unable to be used because the dataset in either stage isn't in normal distribution. Thus, we applied Wilcoxon rank sum test to get the difference regard to the median. From *Table 3.3-18*, we failed to reject the null hypothesis, which the median in Stage 1, 2, 3, and REM are all the same. Therefore, since the Baseline in SBP in Stage 1 is different with either Stage 2 or REM, but not in peak BP, we could conclude that the BP variation due to apnea events is different between Stage 1 with either Stage 2 or REM.

Additionally, t-test is used in comparison of sleep stages in the percentage change. There isn't different ($p=0.055$) in the percentage change between Stage 1 and 2 regardless BP category. But, both Stage 1 and 2 compare with REM shows significantly different ($p<0.005$).

Furthermore, the duration in each sleep stage may provide us the extent of the time that the subjects experienced that stage and, hence, the level of the impact of apneas during the stage on the blood pressure oscillations. That is, one can assess if the average of the duration that subjects spent in a given stage, is associated with level of changes in the BP features. In Figure 3.4-1, the average time in bed for all 10 subjects is around 7.5hrs. There is a weak correlation to the time in sleep with either BMI or AHI. Also, by accumulated all the data in the subjects, in Figure 3.4-2, the proportion of Stage 1 is around 61%; 29% in Stage 2, around 2% in Stage 3 and around 8% in REM. From Table 3.3-2 Col. 8, we could get the event distribution in Figure 3.4-3. Thus, Stage 1 contains the majority of apnea events and also dominates the highest portion during the sleep, which follows by Stage 2.

The results of the analysis of the impact of the sleep stages on the rate of rise in BP (Section 3.3.5), the slope in apnea events in Stage 1, 2, and REM are 0.99 mmHg/s, 0.85 mmHg/s, 0.89 mmHg/s and 0.79 mmHg/s, which indicates the rate of surge decrease as going to a deeper sleep.

By applying aggregated result of the temporal location of BP peak in each sleep stage in

Table 3.3-2, we could get around 156 mmHg in Stage 1, 151 mmHg in Stage 2, 137 mmHg in Stage 3 and 163 mmHg in REM. However, the mean peak BP values during apnea in

Table 3.3-2 are all lower than the estimated value we calculated from the equation slope. Similarly logic in DBP, the slope in

Table *3.3-22* in apnea events in DBP in Stage 1, 2, and REM are 0.66 mmHg/s, 0.57 mmHg/s, 0.53 mmHg/s and 0.51 mmHg/s. And, applying the aggregated result of the temporal location of BP peak in each sleep stage in

Table *3.3-12*, the blood pressure surges to around 85 mmHg in Stage 1, 83.2 mmHg in Stage 2, 84 mmHg in Stage 3 and around 88 mmHg in REM. Again, the estimated results compare to

Table *3.3-12* Col. 4 are still higher than the mean peak BP values during apnea. Such a result indicates that estimating the surge of blood pressure by utilizing the slope may provide higher estimate than the measured blood pressure variation.

4.1.3. Novelty of the Study

The present study provides a convenient and effective way of quantifying the overall dynamics of blood pressure surges that occur as a result of each apnea episode. Specifically, it allows an investigator to measure some key features of the dynamic oscillations for the entire night in a rather compact and simple-to-understand fashion; addressing the challenge of how one reduces relatively large and variant volume of BP data for the entire night. One of the previous studies that had considered nocturnal BP variations with SpO_2 dropped and simply detected those desaturation period to get the BP surges (ranging from around 10 to 100mmHg) during sleep in OSA patients [20]. Another prior study determined the blood pressure surge by comparing with the average blood pressure in 60 min and got the change around 25.4 mmHg [40]. As a result, they did not precisely capture the more rapid dynamic components that we are able to capture.

Our study has revealed that the blood pressure surges elicited by OSA events result in pronounced peaks that is estimated to range between 11 to 35 mmHg for SBP and 7.4 to 17.8 mmHg for DBP. Further, these peaks do not occur within the boundary of detected apnea event, but rather they occur approximately 7 to 9 s post termination of the event. Moreover, the surges tend to subside, and BP returns to the baseline within approximately 20 s of the termination of the events. One of prior studies considered windows that were at least 60 min or longer to analyze the BP variations [41]. Thus, the speed and range of BP variations had not been established by previous studies. While one prior study investigated the BP surge levels at the end of the apnea events and reported similar results values (6 ± 13.8 mmHg) [42], our study not only provided the value for the surge, but also the average time course for the surge. Hence, as a result of our study, a higher temporal resolution for the BP dynamic variations resulting from apnea is now available. With future study of larger subject populations confirming the results of our study, one can possibly use the knowledge of rapid dynamic features that this study provided as a means of assessing the health of sympathetic nervous system

response to apnea challenges.

The findings in this study discussed in-depth on blood pressure variation within the sleep stages. By utilizing dynamic trajectory of the blood pressure due to apnea events in each sleep stage, we clearly defined the baseline in each stage and compare with the BP surge in that corresponding stages due to apnea events. As going to a deeper sleep in NREM, both Baseline and peak blood pressure during apnea events decrease gradually, but the percentage change gradually increase from 9% to 12.2% in SBP (around 13 mmHg). In REM, the SBP surge level only around 9 mmHg (5.8%). Additionally, in event distribution between sleep stages and the sleep stages distribution, Stage 1 contains 66% of events and covered 60% of sleep, which follows with Stage 2. The results of the previous study discovered the both BP baseline and the peak BP due to apnea events gradually increased from Stage 1 to REM, and the BP surge in NREM is around 4 mmHg and 15 mmHg in REM. The main different between our findings and theirs is due to they used either the fixed-interval or oxygen-triggered function to BP monitor. Our study analyzed each dynamic trajectory of the blood pressure due to apnea events and compared with the Baseline trajectory. Thus, our study produces more precise results between sleep stages, which is able to contribute valuable and meaningful data to the clinical and medical field.

4.2. Conclusion

Analysis of continuous nocturnal blood pressure for a sample of sleep apnea patients showed large oscillations are elicited by apnea events. These oscillations tend to have peaks that occur after an apnea event has ended, rather than within the time interval that apnea is in progress. In addition, by considering the sleep stages, the degree of rise increases as patient goes to deeper sleep stages in NREM. While the computation of the slope of the rise in BP provides a means of estimating the rate of surge in BP due to apnea, a linear line estimate of the level of the rise tend to be higher than the experimentally obtained values, Overall, the observed large magnitude of oscillations elicited by each apnea event suggest that they could have clinical significance.

From the findings, e.g. temporal location of BP peak and the difference from the baseline and mean blood pressure elicited by apnea events, it could be quite fruitful to combine the results

with other physiological data that are commonly monitored during PSG studies, such as EEG, EKG, and respiratory efforts, as well as with functional near-infrared spectroscopy to perform a time-frequency dependent analysis in order to understand more comprehensively effects of obstructive sleep apnea on cardiac and neurological systems. Additionally, the results are proved our expectation on apnea episodes inducing either sympathetically mediated vasoconstriction with temporary increase in arterial stiffness or isotropic effects of sympathoexcitation with cardiac stroke volume [27]. Our study provides methods to quantify blood pressure variations during a whole night record.

4.3. Limitation of Study

While the methods and results of our study have revealed new findings about the level, rate, and temporal course of surges in BP due to OSA, the findings can be further enhanced with inclusion of data from more subjects. In particular, analysis of the interaction of sleep stage and BP surges could benefit from more subjects as duration of some of the stages in our current subject population was relatively short. The analysis of the rate of surge in BP can be improved to achieve higher accuracy in the estimation with exploring more ways of estimating the rates, in addition to the linear equation that was done here.

APPENDIX

A. MATLAB CODE FOR PEAK AND VALLEY DETECTION

```
clc; close all;clear;
rootDir = "..\\data";
folderNum ="03";
global folderPath fileName;
folderPath = sprintf("%s\\SLEEP DATA 2013-2015 (Ann.)\\%s",
rootDir, folderNum);
fileName = sprintf('%s_DAQ_resampled', folderNum);


printLog('Message', sprintf('Start working on %s~~~~~~~',
fileName));


m = load(sprintf('%s\\%s.mat', folderPath, fileName));
[len,~] = size(m.DAQ_rsmpl);


x = 0.00: 0.01:(len/100);
x = x(1:numel(x)-1);
bp = m.DAQ_rsmpl(:,3);


clear m;
%%
m = load(sprintf("%s\\%s_stage_event.mat", folderPath, folderNum));


event = m.EVENT;
stage = m.STAGE;
event = round(event);
stage = round(stage);
clear m;
%%


% for i = 1:size(m.DAQ_rsmpl, 2)
```

```
i = 4;
figure;
stPt = 1; endPt = 2000000;
plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,3));


hold on;
% plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,2));%time signal 4
synchronized
plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,i));
% plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,5));
% plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,6));
% plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,7));
% plot(x(stPt:endPt), m.DAQ_rsmpl(stPt:endPt,8));


plot(x(stPt:endPt),event(stPt:endPt), 'LineWidth',2, 'Color',
colorConvertor('#D7BD00') );
plot(x(stPt:endPt),stage(stPt:endPt));
hold off;
title(sprintf('Channel: %d',i));
% end


%% for find period in testing mode
targetIdx = find(x < 3340, 1, 'last');
w = 1;
getTarget = false;
while w <= numel(wholeperiod) && getTarget == false
    if targetIdx <= wholeperiod(w)
        disp([w-1, wholeperiod(w-1), targetIdx]);
        getTarget = true;
    end
    w = w + 1;
end
```

```
%%
% Set up
global allPeakVal allPeakLc allTroughVal allTroughLc
allPeakVal = bp(:) .* 0;
allPeakLc = bp(:) .* 0;
allTroughVal = bp(:) .* 0;
allTroughLc = bp(:) .* 0;


%%
cut = floor(numel(x) * 0.02);
wholeperiod = 1:cut:numel(x);
totalPointPeakAdd = 0;
totalPointTroughAdd = 0;
% upperlimit = 1.5;
% lowerlimit = 1.1;
periodIdx = 1;
% while periodIdx < 8
while periodIdx < numel(wholeperiod)
%         disp([periodIdx,numel(wholeperiod),
wholeperiod(periodIdx),wholeperiod(periodIdx +1 )]);
    printLog('Message',sprintf('Now Working on %d / %d  >> IDX is
from %d to %d', ...
                    periodIdx,numel(wholeperiod),
wholeperiod(periodIdx),wholeperiod(periodIdx +1 )));
    periodStart = wholeperiod(periodIdx);
    periodEnd = wholeperiod(periodIdx +1 );


    [tmpPk, tmpLc] = findPeriodPeak(x, bp, periodStart, periodEnd,
false);
    allPeakVal(totalPointPeakAdd +1 : totalPointPeakAdd +
numel(tmpPk)) = tmpPk;
    allPeakLc( totalPointPeakAdd +1 : totalPointPeakAdd +
numel(tmpPk)) = tmpLc;
```

```
    totalPointPeakAdd = totalPointPeakAdd + numel(tmpPk);


    [tmpTroughPk, tmpTroughLc] = findPeriodTrough(x, bp,
periodStart, periodEnd, false);
    allTroughVal(totalPointTroughAdd +1 : totalPointTroughAdd +
numel(tmpTroughPk)) = tmpTroughPk;
    allTroughLc( totalPointTroughAdd +1 : totalPointTroughAdd +
numel(tmpTroughPk)) = tmpTroughLc;
    totalPointTroughAdd = totalPointTroughAdd + numel(tmpTroughPk);


    periodIdx = periodIdx + 1;
end


printLog('Message','ALMOST DONE~~~~~~~~~~~~~~~~~~~~~~~~~~~~~');
%% Finish the last part
periodStart = wholeperiod(periodIdx);
periodEnd = numel(x);
disp([periodIdx,numel(wholeperiod),
wholeperiod(periodIdx),wholeperiod(numel(wholeperiod))]);
[tmpPk, tmpLc] = findPeriodPeak(x, bp, periodStart, periodEnd,
false);
allPeakVal(totalPointPeakAdd +1 : totalPointPeakAdd + numel(tmpPk))
= tmpPk;
allPeakLc( totalPointPeakAdd +1 : totalPointPeakAdd + numel(tmpPk))
= tmpLc;


[tmpTroughPk, tmpTroughLc] = findPeriodTrough(x, bp, periodStart,
periodEnd, false);
allTroughVal(totalPointTroughAdd +1 : totalPointTroughAdd +
numel(tmpTroughPk)) = tmpTroughPk;
allTroughLc( totalPointTroughAdd +1 : totalPointTroughAdd +
numel(tmpTroughPk)) = tmpTroughLc;


printLog('Message','ALL DONE~~~~~~~~~~~~~~~~~~~~~~~~~~~~~');


%%
```

```
figure;
startPoint = 1;
plot(x(startPoint:periodEnd), bp(startPoint: periodEnd));
hold on;
stidx = 2708651; endidx = 2708651;
plot([x(stidx) x(stidx)], [0 4], '--k');
plot([x(endidx) x(endidx)], [0 4], '--k');
plot(allPeakLc, allPeakVal, 'r^');
plot(allTroughLc, allTroughVal,'v', 'Color',
colorConvertor('#54009E'));
hold off;



%%
[allPeakLc, allPeakVal, allTroughLc, allTroughVal] =
removeZeronNan(allPeakLc, allPeakVal, allTroughLc, allTroughVal);
savint2file(nan);
```

B. MATLAB CODE FOR REMOVING CALIBRATION

```
%% Init 1-1
clc;close all; clear;


folderNum = "15";
rootDir = "..\\data";
fileName = sprintf("%s_DAQ_resampled", folderNum);
folderPath = fullfile(rootDir, "SLEEP DATA 2013-2015 (Ann.)",
folderNum);


dict_fPath = fullfile(folderPath, sprintf('%s.mat', fileName));
m = load(dict_fPath);
[len,~] = size(m.DAQ_rsmpl);
x = 0.00: 0.01:(len/100);
```

```
y = m.DAQ_rsmpl(:,3);


dict_fPath = fullfile(folderPath, sprintf('%s_dict.mat',
fileName));
m = load(dict_fPath);
dict = m.dict;
dict.x = x(1:numel(x)-1);
dict.y = y;
clear m x y;


%% Check the figure 1-2
close all;
coeff = 1;
figure;
plot(dict.x, dict.y.*coeff);
hold on;
plot(dict.peakLc, dict.peakVal.*coeff, 'r^');
plot(dict.troughLc, dict.troughVal.*coeff,'v', 'Color',
colorConvertor('#158C00'));


% upbond = 1;
% plot([dict.x(1), dict.x(end)], [upbond upbond], '--k');
% lowbond = 0.4;
% plot([dict.x(1), dict.x(end)], [lowbond lowbond], '-k');
hold off;
title(folderNum);



%% Remove the value from one value
%                           pklow bond, tr low bond
dict = removeFromOneValue(dict, upbond, lowbond, true);
%%
%                           pk up bond, tr up bond
```

```
dict = removeFromOneValue(dict, 2, 1, false);

%% Remove the starting part from location
dict = removeFromOnePt(dict, 3.06, 'start');

%% Remove the end part from location
dict = removeFromOnePt(dict, 27340.46, 'end');

%% Remove selected point 1- removePt.txt
% disp([0,0,0,numel(dict.peakVal)]);
dict = removeFromRemovePtTXT(dict, folderNum);
% disp([0,0,0,numel(dict.peakVal)]);

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% Remove some trough in the calibration
% Get the troughVal different 2-1
trValDif = abs(dict.troughVal(1:end-1) - dict.troughVal(2:end));

% Remove some trough in the calibration 2-2
figure;
h = histogram(trValDif);

global status;
status = nan;
graphSetting;

%% Simply determined by the threshold 2-3-1
threshold = prctile(h.Data,99);
disp(threshold);
len_trValDif = numel(trValDif);
outLierLs = nan(len_trValDif, 1);
```

```
for i = 1:len_trValDif
    try
        if trValDif(i) > threshold && trValDif(i+1) > threshold
            outLierLs(i) = i;
        end
    catch
        fprintf('Error on %d / %d\r',i, len_trValDif);
    end
end
outLierLs = outLierLs(~isnan(outLierLs));


% Subplot 1 >> 2-3-2
figure;
subP1 = subplot(3,1,1);
plot(dict.x, dict.y);
title(numel(outLierLs));
hold on;
plot(dict.peakLc, dict.peakVal, 'r^');
% plot(dict.peakLc(caliLs), dict.peakVal(caliLs), 'kx',
'MarkerSize',15,  'LineWidth',2);
plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


% plot(dict.peakLc(caliLs(1)), dict.peakVal(caliLs(1)), 'o',
'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));
plot(dict.troughLc(outLierLs), dict.troughVal(outLierLs), 'gx',
'MarkerSize',15,  'LineWidth',2);
plot(dict.troughLc(outLierLs + 1), dict.troughVal(outLierLs + 1),
'ro', 'MarkerSize',15,  'LineWidth',2);
hold off;


% Determined by the threshold in the front and mean in the back 2-
3-2
```

```
%  the cur 2 nx > threshold and nx 2 nxx > mean diff with next 7
points

meanptNum = 7;

len_trValDif = numel(trValDif);

outLierLs = nan(len_trValDif, 1);

for i = 1:len_trValDif

    try

        endDiff = mean(abs(dict.troughVal(i+1: i+meanptNum) -
dict.troughVal(i+2:i+meanptNum+1)));


        if trValDif(i) > threshold && trValDif(i+1) > endDiff*1.05

            outLierLs(i) = i;

        end

    catch

        fprintf('Error on %d / %d\r',i, len_trValDif);

    end

end

outLierLs = outLierLs(~isnan(outLierLs));


%

subP2 = subplot(3,1,2);

plot(dict.x, dict.y);

title(numel(outLierLs));

hold on;

plot(dict.peakLc, dict.peakVal, 'r^');

plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


plot(dict.troughLc(outLierLs), dict.troughVal(outLierLs), 'gx',
'MarkerSize',15,  'LineWidth',2);

plot(dict.troughLc(outLierLs + 1), dict.troughVal(outLierLs + 1),
'ro', 'MarkerSize',15,  'LineWidth',2);

hold off;
```

```
% Use the mean diff to detect 2-3-3
%  cur 2 nx > threshold and nx 2 nxx > threshold
%  and check the front 7 points mean diff and the post 7 points
diff to see
%  whether it also greater
meanptNum = 7;
len_trValDif = numel(trValDif);
outLierLs = nan(len_trValDif, 1);
shiftlv = 1.05;
% for i = 1:3
for i = 1:len_trValDif
    try
        if trValDif(i) > threshold && trValDif(i+1) > threshold
%             disp('in');
            frontDiff = mean(abs(dict.troughVal(i-meanptNum: i-1) -
dict.troughVal(i-meanptNum - 1:i-2)));
            endDiff = mean(abs(dict.troughVal(i+1: i+meanptNum) -
dict.troughVal(i+2:i+meanptNum + 1)));
            if trValDif(i) > frontDiff * shiftlv && trValDif(i+1) >
endDiff * shiftlv
                outLierLs(i) = i;
            end
        end
    catch
        fprintf('ERROR at i = %d\r', i);
    end
end
outLierLs = outLierLs(~isnan(outLierLs));
%
% Get the figure to verify %% Remove some trough in the calibration
2-3-4
% figure;
subP3 = subplot(3,1,3);
```

```
plot(dict.x, dict.y);

title(numel(outLierLs));

hold on;

plot(dict.peakLc, dict.peakVal, 'r^');

plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


plot(dict.troughLc(outLierLs), dict.troughVal(outLierLs), 'gx',
'MarkerSize',15,  'LineWidth',2);

plot(dict.troughLc(outLierLs + 1), dict.troughVal(outLierLs + 1),
'ro', 'MarkerSize',15,  'LineWidth',2);

hold off;

linkaxes([subP1, subP2, subP3],'xy');


%% Manual Exclude the mis judge points  2-4-1 confirm the points
first

tmpls = [16295.6, 16200];

tls = nan(numel(tmpls),1);

for i = 1: numel(tmpls)


    tmp = find( troughLc == tmpls(i));

    tls(i) = tmp;

    disp(tmp);

end


% Manual Exclude the mis judge points  2-4-2 remove points in the
outliers

outLierLs(outLierLs == tmpls) = nan;

outLierLs = outLierLs(~isnan(outLierLs));


%% Remove outliers %% Remove some trough in the calibration 2-5

dict.troughVal(outLierLs+1) = nan;

dict.troughLc(outLierLs+1) = nan;

dict.troughVal = dict.troughVal(~isnan(dict.troughVal));
```

```
dict.troughLc = dict.troughLc(~isnan(dict.troughLc));
disp('done');


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Remove some peak in the calibration
% Get the troughVal different 3-1
pkValDif = abs(dict.peakVal(1:end-1) - dict.peakVal(2:end));

% Remove some trough in the calibration 3-2
figure;
h = histogram(pkValDif);

%% Simply determined by the threshold 3-3-1
threshold = prctile(h.Data,99.75);
len_pkValDif = numel(pkValDif);
outLierLs = nan(len_pkValDif, 1);
for i = 1:len_pkValDif
    try
        if pkValDif(i) > threshold && pkValDif(i+1) > threshold
            outLierLs(i) = i;
        end
    catch
        fprintf('Error on %d / %d\r',i, len_pkValDif);
    end
end
outLierLs = outLierLs(~isnan(outLierLs));

% Subplot 1 >> 3-3-2
figure;
subP1 = subplot(3,1,1);
plot(dict.x, dict.y);
```

```
title(numel(outLierLs));

hold on;

plot(dict.peakLc, dict.peakVal, 'r^');

% plot(dict.peakLc(caliLs), dict.peakVal(caliLs), 'kx',
'MarkerSize',15,  'LineWidth',2);

plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


% plot(dict.peakLc(caliLs(1)), dict.peakVal(caliLs(1)), 'o',
'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));

plot(dict.peakLc(outLierLs), dict.peakVal(outLierLs), 'gx',
'MarkerSize',15,  'LineWidth',2);

plot(dict.peakLc(outLierLs + 1), dict.peakVal(outLierLs + 1), 'ko',
'MarkerSize',15,  'LineWidth',2);

hold off;


% Determined by the threshold in the front and mean in the back 3-
3-2

%  the cur 2 nx > threshold and nx 2 nxx > mean diff with next 7
points

meanptNum = 7;

len_pkValDif = numel(pkValDif);

outLierLs = nan(len_pkValDif, 1);

for i = 1:len_pkValDif

    try

        endDiff = mean(abs(dict.peakVal(i+1: i+meanptNum) -
dict.peakVal(i+2:i+meanptNum+1)));


        if pkValDif(i) > threshold && pkValDif(i+1) > endDiff

            outLierLs(i) = i;

        end

    catch

        fprintf('Error on %d / %d\r',i, len_pkValDif);

    end

end
```

```
outLierLs = outLierLs(~isnan(outLierLs));


%
subP2 = subplot(3,1,2);
plot(dict.x, dict.y);
title(numel(outLierLs));
hold on;
plot(dict.peakLc, dict.peakVal, 'r^');
plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


plot(dict.peakLc(outLierLs), dict.peakVal(outLierLs), 'gx',
'MarkerSize',15,  'LineWidth',2);
plot(dict.peakLc(outLierLs + 1), dict.peakVal(outLierLs + 1), 'ko',
'MarkerSize',15,  'LineWidth',2);
hold off;



% Use the mean diff to detect 3-3-3
%  cur 2 nx > threshold and nx 2 nxx > threshold
%  and check the front 7 points mean diff and the post 7 points
diff to see
%  whether it also greater
meanptNum = 7;
len_pkValDif = numel(pkValDif);
outLierLs = nan(len_pkValDif, 1);


% for i = 1:3
for i = 1:len_pkValDif
    try
        if pkValDif(i) > threshold && pkValDif(i+1) > threshold
%             disp('in');
            frontDiff = mean(abs(dict.peakVal(i-meanptNum: i-1) -
dict.peakVal(i-meanptNum - 1:i-2)));
```

```
            endDiff =  mean( abs(dict.peakVal(i+1: i+meanptNum) -
dict.peakVal(i+2:i+meanptNum + 1)));
            if pkValDif(i) > frontDiff && pkValDif(i+1) > endDiff
                outLierLs(i) = i;
            end
        end
    catch
        fprintf('ERROR at i = %d\r', i);
    end
end
outLierLs = outLierLs(~isnan(outLierLs));
%
% Get the figure to verify %% Remove some trough in the calibration
3-3-4
% figure;
subP3 = subplot(3,1,3);
plot(dict.x, dict.y);
title(numel(outLierLs));
hold on;
plot(dict.peakLc, dict.peakVal, 'r^');
plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


plot(dict.peakLc(outLierLs), dict.peakVal(outLierLs), 'gx',
'MarkerSize',15,  'LineWidth',2);
plot(dict.peakLc(outLierLs + 1), dict.peakVal(outLierLs + 1), 'ko',
'MarkerSize',15,  'LineWidth',2);
hold off;
linkaxes([subP1, subP2, subP3],'xy');



%% Remove outliers %% Remove some trough in the calibration 3-5
dict.peakVal(outLierLs+1) = nan;
dict.peakLc(outLierLs+1) = nan;
```

```
dict.peakVal = dict.peakVal(~isnan(dict.peakVal));
dict.peakLc  = dict.peakLc (~isnan(dict.peakLc));
disp('done');



%% Check the figure 3-6
figure;
plot(dict.x, dict.y);
hold on;
plot(dict.peakLc, dict.peakVal, 'r^');
plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));
hold off;


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

%

% The part for checking peak location different

%

%

%% Backup For the
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%

origpeakLc = dict.peakLc(:);
origpeakVal = dict.peakVal(:);
origtroughLc = dict.troughLc(:);
origtroughVal = dict.troughVal(:);
%% Restore
dict.peakLc = origpeakLc(:);
dict.peakVal = origpeakVal(:);
dict.troughLc = origtroughLc(:);
```

```
dict.troughVal = origtroughVal(:);

%% Detect the Calibration Period 3-1
dict = removeZeronNan(dict);
peakLcDif = abs(dict.peakLc(1:end-1) - dict.peakLc(2:end));

%% histogram of peaklcdiff 3-2
figure;
h = histogram(peakLcDif, 'BinLimits', [0, 3]);

% Detect the calibration part by using the THRESHOLD 3-3
threshold = prctile(h.Data,98.5);
disp(threshold);
len_peakLcDif = numel(peakLcDif);
caliLs = nan(len_peakLcDif, 1);
for i = 1:len_peakLcDif
    if peakLcDif(i) > threshold
        caliLs(i) = i;
    end
end
caliLs = caliLs(~isnan(caliLs));



%% Check the points 3-4

figure;
plot(dict.x, dict.y);
title(numel(caliLs));
hold on;
plot(dict.peakLc, dict.peakVal, 'r^');
plot(dict.peakLc(caliLs), dict.peakVal(caliLs), 'kx',
'MarkerSize',15,  'LineWidth',2);
```

```
plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


% plot(dict.peakLc(caliLs(765:768)), dict.peakVal(caliLs(765:768)),
'bo', 'MarkerSize',20,  'LineWidth',2);

% plot(dict.peakLc(caliLs(1)), dict.peakVal(caliLs(1)), 'o',
'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));

% plot(dict.troughLc(idx), dict.troughVal(idx), 'gx',
'MarkerSize',15,  'LineWidth',2);

hold off;




%% Get the points >> in case 3-5

clc;

idx = find(dict.peakLc > 10328.46,1, 'first');

disp([dict.peakLc(idx), dict.peakVal(idx)]);

disp(dict.peakLc(idx) - dict.peakLc(idx+1));


%% Get the calibration period in to variable >> caliPeriod 3-6

len_caliLs = numel(caliLs);

caliPeriod = cell(1, len_caliLs);

for i = 1:len_caliLs

    disp(i);

    pkIdx = caliLs(i);

    [group, idx] = getEndCaliPoints("peak", pkIdx, dict.peakLc,
dict.troughLc, threshold);

    disp({group, idx});


    if group == "peak"

        caliPeriod(i) = {[dict.peakLc(pkIdx), dict.peakVal(pkIdx),
dict.peakLc(idx), dict.peakVal(idx)]};

    else

        caliPeriod(i) = {[dict.peakLc(pkIdx), dict.peakVal(pkIdx),
dict.troughLc(idx), dict.troughVal(idx)]};
```

```
    end

end

disp('done');
% %%
% caliPeriod = caliPeriod(1:numel(caliPeriod) - 4);



% Need to combine the calibration part if it is consecutive 3-10
% for i = 1:3
i = 1;
len_caliPeriod =  numel(caliPeriod);
while i < len_caliPeriod
    disp(i);
    curP = caliPeriod{i};
    ce1 = curP(3);
    ce2 = curP(4);
    [i, caliPeriod] = checkNextCaliPart(i+1, caliPeriod, ce1, ce2);

end


caliPeriod(cellfun(@(caliPeriod)
any(isnan(caliPeriod)),caliPeriod)) = [];
disp('done');
%% Check again with the calibration period 3-7
figure;
startPoint = 1;


plot(dict.x, dict.y);
title(numel(caliLs));
hold on;
```

```
plot(dict.peakLc, dict.peakVal, 'r^');

plot(dict.peakLc(caliLs), dict.peakVal(caliLs), 'kx',
'MarkerSize',15,  'LineWidth',2);

plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));


% plot(dict.peakLc(caliLs(1)), dict.peakVal(caliLs(1)), 'o',
'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));

% plot(dict.troughLc(idx), dict.troughVal(idx), 'gx',
'MarkerSize',15,  'LineWidth',2);

for i = 1:numel(caliPeriod)

    tmp = caliPeriod{i};

    s1 = tmp(1);

    s2 = tmp(2);

    e1 = tmp(3);

    e2 = tmp(4);


    plot(s1, s2, '<', 'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));
    plot(e1, e2, 'g>', 'MarkerSize',15,  'LineWidth',2);


end


hold off;




% %% Backup 3-8

%

% oriCali = caliPeriod;

%

% %% Restore 3-9

%

% caliPeriod = oriCali;
```

```
%% Also clean up the point during the calibration 3-11
for i = 1:numel(caliPeriod)
    tmp = caliPeriod{i};
    startLc = tmp(1);
    endLc = tmp(3);

    pkIdx = find( dict.peakLc > startLc & dict.peakLc < endLc );
    trIdx = find( dict.troughLc > startLc & dict.troughLc <
endLc );

    dict.peakLc(pkIdx) = nan;
    dict.peakVal(pkIdx) = nan;

    dict.troughLc(trIdx) = nan;
    dict.troughVal(trIdx) = nan;
end

dict = removeZeronNan(dict);

%% Check again with the calibration period 3-7
figure;
startPoint = 1;

plot(dict.x, dict.y);
title(numel(caliLs));
hold on;
plot(dict.peakLc, dict.peakVal, 'r^');
% plot(dict.peakLc(caliLs), dict.peakVal(caliLs), 'kx',
'MarkerSize',15,  'LineWidth',2);
plot(dict.troughLc, dict.troughVal,'v', 'Color',
colorConvertor('#158C00'));
```

```
% plot(dict.peakLc(caliLs(1)), dict.peakVal(caliLs(1)), 'o',
'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));

% plot(dict.troughLc(idx), dict.troughVal(idx), 'gx',
'MarkerSize',15,  'LineWidth',2);

for i = 1:numel(caliPeriod)

    tmp = caliPeriod{i};

    s1 = tmp(1);

    s2 = tmp(2);

    e1 = tmp(3);

    e2 = tmp(4);


    plot(s1, s2, '<', 'MarkerSize',15,  'LineWidth',2 ,'Color',
colorConvertor('#111111'));

    plot(e1, e2, 'g>', 'MarkerSize',15,  'LineWidth',2);


end


hold off;



%% Final update the result to the the file 3-12 Done

update2file(dict, caliPeriod, folderPath, fileName);


%%


function [inFront, idx] = getEndCaliPoints(inFront, idx, peakLc,
troughLc, threshold)

    disp(['in', inFront]);



    if inFront == "peak"

        endTrIdx = find(peakLc(idx) < troughLc, 1, 'first');
```

```
        endPkIdx = idx + 1;

        trIdxFromPk = find(troughLc < peakLc(endPkIdx), 1, 'last');
        if trIdxFromPk > endTrIdx
            endTrIdx = trIdxFromPk;
        end

    else
        endPkIdx = find(troughLc(idx) < peakLc, 1, 'first');
        endTrIdx = idx + 1;
    end

%         disp([endTrIdx, endPkIdx]);
    t = troughLc(endTrIdx);
    p = peakLc(endPkIdx);



    % Check who is closer to the begin points>> less value >>
closer
    if p < t
        inFront = "peak";
        if abs(peakLc(endPkIdx + 1) - p) > threshold
            [inFront, idx] = getEndCaliPoints(inFront, endPkIdx,
peakLc, troughLc, threshold);
        else
            idx = endPkIdx;
        end
    else
        inFront = "trough";
        if abs(troughLc(endTrIdx + 1) - t) > threshold
            [inFront, idx] = getEndCaliPoints(inFront, endTrIdx,
peakLc, troughLc, threshold);
```

```
        else
            idx = endTrIdx;
        end
    end
end


%% Check consecutive calibration period to merge them into the
first one
function [nxIdx, caliPeriod] = checkNextCaliPart(nxIdx, caliPeriod,
firstEndLc, firstEndVal)
    if nxIdx <= numel(caliPeriod)
        nxP = caliPeriod{nxIdx};


        nxEndLc = nxP(3);
        nxEndVal = nxP(4);
        disp([firstEndLc, nxEndLc]);
        disp([firstEndVal, nxEndVal]);
        if firstEndLc == nxEndLc && firstEndVal == nxEndVal
            disp('got');
            disp(nxIdx);
            caliPeriod{nxIdx} = nan;
            disp(caliPeriod{nxIdx});
            [nxIdx, caliPeriod ] = checkNextCaliPart(nxIdx + 1,
caliPeriod, firstEndLc, firstEndVal);
        end
    end
end


%
function dict = removeFromRemovePtTXT(dict, folderNum)
    peakLc = round(dict.peakLc, 2);
    peakVal = round(dict.peakVal, 5);
```

```
    filePath = sprintf('D:\\Dream\\UT
Arlington\\Thesis\\data\\SLEEP DATA 2013-2015
(Ann.)\\RemovePt\\%s.txt', folderNum);
    fprintf('Using file: %s to clean the points', filePath);
    fid = fopen(filePath);
    m = textscan(fid, '%f\t%f');
    fclose(fid);


    deleteLsLc = m{1};
    deleteLsVal = m{2};
    deleteLsVal = round(deleteLsVal, 5);
    total = 0;
    for i = 1:numel(deleteLsLc)
%     for i = 17:17
        delPtLc = deleteLsLc(i);
        idx = find(peakLc == delPtLc);


%         disp([idx, delPtLc, dict.peakLc(idx)]);
        if numel(idx) == 1 && peakVal(idx) == deleteLsVal(i)
            total = total + 1;
%             disp(idx);
%             dict.peakLc(idx) = -9;
%             dict.peakVal(idx) = -9;
            dict.peakLc(idx) = nan;
            dict.peakVal(idx) = nan;
        else
            disp(i);


        end


    end
    fprintf("Delete List Size: %d, Delete Num: %d\r",
numel(deleteLsLc), total);
```

```
end


function dict = removeFromOneValue(dict, pkBond, trBond,
isLowerBond)
%     pkLowerBond = 1;
    if ~isnan(pkBond)
        if isLowerBond
            idxPk = dict.peakVal < pkBond;
            fprintf('remove from Peak below %f - # %d\r',pkBond,
numel(idxPk));
        else
            idxPk = dict.peakVal > pkBond;
            fprintf('remove from Peak over %f - # %d\r',pkBond,
numel(idxPk));
        end
        dict.peakLc(idxPk) = nan;
        dict.peakVal(idxPk) = nan;


    end


    if ~isnan(trBond)
        if isLowerBond
            idxTr = dict.troughVal < trBond;
            fprintf('remove from Trough below %f - # %d\r',trBond,
numel(idxTr));
        else
            idxTr = dict.troughVal > trBond;
            fprintf('remove from Trough over %f - # %d\r',trBond,
numel(idxTr));
        end


        dict.troughLc(idxTr) = nan;
        dict.troughVal(idxTr) = nan;
```

```
    end
end


function dict = removeFromOnePt(dict, cutPt, cutDirection)
%     cutPt = 16015.91;
    if cutDirection == "end"
        rm = find(dict.peakLc > cutPt);
    elseif cutDirection == "start"
        rm = find(dict.peakLc < cutPt);
    end
    fprintf('remove from Peak - # %d\r',numel(rm));
    dict.peakLc(rm) = [];
    dict.peakVal(rm) = [];

    if cutDirection == "end"
        rm = find(dict.troughLc > cutPt);
    elseif cutDirection == "start"
        rm = find(dict.troughLc < cutPt);
    end

    dict.troughLc(rm) = [];
    dict.troughVal(rm) = [];
    fprintf('remove from trough - # %d\r',numel(rm));

end
```

```
function [inFront, idx] = OriginGetEndCaliPoints(inFront, idx,
peakLc, troughLc, threshold)
    disp(['in', inFront]);
    if inFront == "peak"
        endTrIdx = find(peakLc(idx) < troughLc, 1, 'first');
        endPkIdx = idx + 1;
    else
        endPkIdx = find(troughLc(idx) < peakLc, 1, 'first');
        endTrIdx = idx + 1;
    end


    disp([endTrIdx, endPkIdx]);
    t = troughLc(endTrIdx);
    p = peakLc(endPkIdx);


    % Check who is closer to the begin points>> less value >>
closer
    if p < t
        inFront = "peak";
        if abs(peakLc(endPkIdx + 1) - p) > threshold
            [inFront, idx] = getEndCaliPoints(inFront, endPkIdx,
peakLc, troughLc, threshold);
        else
            idx = endPkIdx;
        end
    else
        inFront = "trough";
        if abs(troughLc(endTrIdx + 1) - t) > threshold
            [inFront, idx] = getEndCaliPoints(inFront, endTrIdx,
peakLc, troughLc, threshold);
        else
            idx = endTrIdx;
        end
    end
```

```
end

%% Save Peaks, Trough and Calibration Period
function savint2file(caliPeriodLs)
    printLog('Message', 'Start the saving process...');
    global allPeakVal allPeakLc allTroughVal allTroughLc folderPath
fileName

    dict = struct('peakLc', allPeakLc, ...
                'peakVal', allPeakVal, ...
                'troughLc', allTroughLc, ...
                'troughVal', allTroughVal, ...
                'caliPeriod', caliPeriodLs);

    filePath = sprintf('%s\\%s_dict.mat', folderPath, fileName);
    save(filePath, 'dict');
    printLog('Message','The saving process is done');
end

function update2file(dict, caliPeriodLs, folderPath, fileName)
    printLog('Message', 'Start the updating process...');

    dict.caliPeriod = caliPeriodLs;

    filePath = sprintf('%s\\%s_dict.mat', folderPath, fileName);
    save(filePath, 'dict');
    printLog('Message','The updating process is done');
end
```

```
%
function removeMiddlePt(pkStartPt, pkEndPt, troStartPt, troEndPt)
    global allPeakVal allPeakLc allTroughVal allTroughLc

    for idx = pkStartPt:pkEndPt
        allPeakVal(idx) = nan; allPeakLc(idx) = nan;
    end

    for idx = troStartPt: troEndPt
        allTroughVal(idx) = nan;  allTroughLc(idx) = nan;
    end
end

% Remove the 0 and NaN
function dict = removeZeronNan( dict)

    dict.peakLc(dict.peakLc == 0) = nan;
    dict.peakVal(dict.peakVal == 0) = nan;
    dict.troughLc(dict.troughLc == 0) = nan;
    dict.troughVal(dict.troughVal == 0) = nan;

    dict.peakLc = dict.peakLc(~isnan(dict.peakLc));
    dict.peakVal = dict.peakVal(~isnan(dict.peakVal));
    dict.troughLc = dict.troughLc(~isnan(dict.troughLc));
    dict.troughVal = dict.troughVal(~isnan(dict.troughVal));
    printLog('Message','Clear the nan pt');
end
```

C. MATLAB CODE FOR COMPUTATION OF PULSE PRESSURE, MEAN PRESSURE, SLOPES, BASELINE

```
function [targetVal, targetLc] = getStatus(dict)
```

```
    % Set up which value is used SBP or DBP or MAP or PP
    global status;

    switch status
        case "SBP"
            targetVal = dict.peakVal;
            targetLc = dict.peakLc;
        case "DBP"
            targetVal = dict.troughVal;
            targetLc = dict.troughLc;
        case "MAP"
            peakLc = dict.peakLc;       peakVal = dict.peakVal;
            troughLc = dict.troughLc;   troughVal = dict.troughVal;

            targetVal = getMAP;
            targetLc = peakLc;
        case "PP"
            peakLc = dict.peakLc;       peakVal = dict.peakVal;
            troughLc = dict.troughLc;   troughVal = dict.troughVal;

            targetVal = getPP;
            targetLc = peakLc;
    end

    function valLs = getMAP
        pklen = numel(dict.peakLc);
        valLs = nan(pklen, 1);

%         for pkidx = 1: 11
        for pkidx = 1: pklen-1
            % get the first trough after current peak
            curPkLc = peakLc(pkidx);
```

```
            nxTrIdx = getTroLcIdx(curPkLc, true);
            % get the previous trough from the NEXT peak
            preTrIdx = getTroLcIdx(peakLc(pkidx+1), false);
            % it should be the same, if it is not then skip
            if nxTrIdx ~= preTrIdx
                continue;
            end
            % fprintf('IDX: %d, preTrIdx: %d, nxTrIdx: %d\r',pkidx,
preTrIdx, nxTrIdx);
            valLs(pkidx) = ( troughVal(nxTrIdx).*2 +
peakVal(pkidx))/3 ;
        end


        pkidx = pklen;
        % get the first trough after current peak
        curPkLc = peakLc(pkidx);
        nxTrIdx = getTroLcIdx(curPkLc, true);


        if numel(nxTrIdx) ~= 0
            % fprintf('IDX: %d, preTrIdx: %d, nxTrIdx: %d\r',pkidx,
preTrIdx, nxTrIdx);
            valLs(pkidx) = ( troughVal(nxTrIdx).*2 +
peakVal(pkidx))/3 ;
        end
    end


    function troIdx = getTroLcIdx(pkLc, getNx)
        if getNx
            troIdx = find(pkLc < troughLc, 1, 'first');
        else
            troIdx = find(pkLc > troughLc, 1, 'last');
        end
    end
```

```
    function valLs = getPP
        pklen = numel(dict.peakLc);
        valLs = nan(pklen, 1);

%         for pkidx = 1: 11
        for pkidx = 1: pklen-1
            % get the first trough after current peak
            curPkLc = peakLc(pkidx);
            nxTrIdx = getTroLcIdx(curPkLc, true);
            % get the previous trough from the NEXT peak
            preTrIdx = getTroLcIdx(peakLc(pkidx+1), false);
            % it should be the same, if it is not then skip
            if nxTrIdx ~= preTrIdx
                continue;
            end
            % fprintf('IDX: %d, preTrIdx: %d, nxTrIdx: %d\r',pkidx,
preTrIdx, nxTrIdx);
            valLs(pkidx) = peakVal(pkidx) - troughVal(nxTrIdx);
        end

        pkidx = pklen;
        % get the first trough after current peak
        curPkLc = peakLc(pkidx);
        nxTrIdx = getTroLcIdx(curPkLc, true);

        if numel(nxTrIdx) ~= 0
            % fprintf('IDX: %d, preTrIdx: %d, nxTrIdx: %d\r',pkidx,
preTrIdx, nxTrIdx);
            valLs(pkidx) = peakVal(pkidx) - troughVal(nxTrIdx);
        end

    end
```

```
end
```

D. MATLAB CODE FOR AGGREGATION OF DATA WITHOUT CONSIDERING SLEEP STAGES

```
clc;clear; close all;

global folderNum halfWindow_sz xx xx_ne fieldLs allEvent coeff ls
status fs halfWindow_len;

global slopeStru normalSlopeStru;

% Init some variable %


% ---------------------------------

% Decide which type of value to use-

status = "SBP";

% status = "DBP";

% status = "MAP";

% status = "PP";

% ---------------------------------

rootDir = "..\\data";

fs = 100;

halfWindow_len = 30; %sec >>> Normal Breathing Window size

xx_ne = 0 : 1/fs : 2 * halfWindow_len - 1/fs; % >>> Normal
Breathing Window size


halfWindow_sz = halfWindow_len * fs;


xx = halfWindow_len * (-1):1/fs:halfWindow_len - 1/fs; % >>> Event
Window Size

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

nonEvent = initNonEvent(rootDir);

% nonEvent = load(sprintf( '%s\\SLEEP DATA 2013-2015
(Ann.)\\nonEvent_windowsz_%d.mat', rootDir, halfWindow_len * 2) );

% nonEvent = nonEvent.noneEvent;
```

```
collectSlope = 0;

slopeStru = initEventslopeStruc(false, false, nan);

normalSlopeStru = initEventslopeStruc(false, true, nonEvent);

printLog('Message', 'Cell: 1~4 col >>> Equation slope, Equation
intercept, Regres slope, intercept');


showDiffColor = false;

enableStatistcs = true;

getPkStd = true;


% ls = ["01", "03", "05", "10", "15"];% Male Group

% ls = [ "07", "11" ];                % Female Group


% ls = ["01", "03", "05", "10", "15",  "07", "11" ];

ls = ["01", "03", "05", "10", "15",  "07", "11" , "06", "12",
"13"];

allEquIdx = 1; allEquEvent = cell(1000, 1);

isAll = true;

for i = 1: numel(ls)

folderNum = ls(i);

% for i = 1: 1

% folderNum = "03";


% folderPath = sprintf("..\\SLEEP DATA 2013-2015 (Ann.)\\%s",
folderNum);

[dict, splinePk, event, stage] = init(rootDir);


splinePk = removeNoisy(folderNum, dict, splinePk);


% collect each event stages in eventCell, same in stageLs

stageStru = getStageList(stage);

eventStru = getEventStageList(event);
```

```
% Remove the event period which have no splinePk value ( ratio <
80%)
% eventStru = checkEventLs(eventStru, splinePk);


% [eventPtLs, slopeLs] = getEventPtnSlope(dict, eventLs, false);


coeff = 100;
% Rescale the y unit %
splinePk = splinePk * coeff;
stage = stage * coeff;
event = event * coeff;



% Init.
% Include all the event with spline peak value period in each stage


allEvent = struct('sn5', [], ...
                        'sn4', [], ...
                        'sn3', [], ...
                        'sn2', [], ...
                        'sn1', [], ...
                        's0' , [], ...
                        's1' , []);
eventStr = "Event";    noneventStr = "NonEvent";
fieldLs = ["sn5",  "sn4", "sn3", "sn2", "sn1", "s0", "s1", "all"];


printLog("Message",sprintf("%s - Subject: %s >>> Initialization
done...", status, folderNum));


% getWholeNightPlot(nan, nan, dict, event, stage, splinePk, nan,
eventStru.e1)
```

```
% ~~~ GET THE EQUAL EVENTS OF EACH STAGE
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

allEvent.('all') = cell(600,1);

shortEventLs = getEqualLenEventWoutStage(splinePk, eventStru,
collectSlope);

if collectSlope

    removeEmptyinSlope;

    getSlopePlot(folderNum);

end


if isAll

    % % >>> Add to allEqualEvent

    len = numel(allEvent.all);

    disp({folderNum, len});

    allEquEvent(allEquIdx:allEquIdx + len - 1) = allEvent.all;

    allEquIdx = allEquIdx + len;


end


plotEventinLinear("all");

[allEventInv, lc, pk, pkSTD] =
newGetInverseAllEvent(enableStatistcs, getPkStd);

[normalVal, normalStd] = getNormalVal(nonEvent);

printLog('Messege', ...

        sprintf('%s - Subject: %s >>> Normal Val: %3.1f +- %2.1f,
Peak Val: %3.1f +- %5.1f, Pk vs normal: %2.1f (%2.1f %%%%), Peak
Time: %3.1f, # of Events: %3d',...

        status, folderNum, normalVal, normalStd, pk, pkSTD, pk-
normalVal, (pk-normalVal)*100/normalVal , lc,
numel(allEvent.all)));


getCenterPlot(allEventInv, "all" , shortEventLs, showDiffColor,
normalVal, [lc, pk]);


end % for loop end (for all the subjects)
```

```
%%
if collectSlope
    getSlopePlot('isAll');
end


close all;
if isAll
    allEvent.all = allEquEvent(:);
    allEvent.all = allEvent.all(~cellfun('isempty',allEvent.all));
    plotEventinLinear( "isAll" );
    enableStatistcs = true;   getPkStd = true;
    [allEventInv, lc, pk, pkSTD] =
newGetInverseAllEvent(enableStatistcs, getPkStd);


    [normalVal, normalStd] = getTotNormalVal(nonEvent);
    printLog('Messege', sprintf('%s - All in one >>> Normal Val:
%2.1f +- %2.1f, Peak Val: %2.1f +- %2.1f, Pk vs normal: %2.1f
(%2.1f %%%%), Peak Time: %2.1f, # of Events: %3d',...
                                    status, normalVal,normalStd, pk,
pkSTD, pk-normalVal, (pk-normalVal)*100/normalVal , lc,
numel(allEvent.all)));


    getCenterPlot(allEventInv, "isAll" , shortEventLs,
showDiffColor, normalVal, [lc, pk]);
end


disp('done for all');




%% Normal Breathing >> No Event \ No Event \ No Event \ No Event \
No Event >> All
close all;clc;
plotLinearNonEvent(xx_ne, nonEvent);
enableStatistcs = true;
```

```
showDiffColor = false;
allNonEventInv = getInverseNonEvent(nonEvent, enableStatistcs);
getNonEventCenterPlot(allNonEventInv, xx_ne, showDiffColor);
% % No Event \ No Event \ No Event \ No Event \ No Event \ No Event
\ No Event
%%
function getSlopePlot(typeStr)
    global slopeStru folderNum ls status normalSlopeStru;

    figure;
    maxSlope = 2;%max(linearSlopeLs);
    plot([0, maxSlope], [0, maxSlope], '--k');
    hold on;

    if strcmp(typeStr, 'isAll')
        allStIdx = 1;
        inAll = nan(1000, 2);
        for i = 1: numel(ls)
            tmpLs = slopeStru.('sub' + ls(i));
            equaSlopeLs = tmpLs(:, 1);
            equalInterCept = tmpLs(:, 2);
            linearSlopeLs = tmpLs(:, 3);
            tmpLsLen = numel(equaSlopeLs);

            endIdx = allStIdx + tmpLsLen - 1;
%             disp({ls(i), allStIdx, endIdx, endIdx - allStIdx,
tmpLsLen});
            inAll(allStIdx:endIdx, 1) = equaSlopeLs;
            inAll(allStIdx:endIdx, 2) = equalInterCept;
            allStIdx = allStIdx + tmpLsLen;
        end

        idx = ~isnan(inAll(:, 1));
```

```
        allSlopeLs = inAll(idx,1);
        allInterCept = inAll(idx, 2);


        plot(equaSlopeLs, linearSlopeLs, '*');
        printLog("Slope", sprintf("%s - All, mean equation slope,
%2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-, %2.3f",...
                 status, mean(allSlopeLs), std(allSlopeLs),
mean(allInterCept), std(allInterCept) ));


        allStIdx = 1;
        inAll = nan(1000, 2);
        for i = 1: numel(ls)
            tmpLs = normalSlopeStru.('sub' + ls(i));
            equaSlopeLs = tmpLs(:, 1);
            equalInterCept = tmpLs(:, 2);
            linearSlopeLs = tmpLs(:, 3);
            tmpLsLen = numel(equaSlopeLs);


            endIdx = allStIdx + tmpLsLen - 1;
%             disp({ls(i), allStIdx, endIdx, endIdx - allStIdx,
tmpLsLen});
            inAll(allStIdx:endIdx, 1) = equaSlopeLs;
            inAll(allStIdx:endIdx, 2) = equalInterCept;
            allStIdx = allStIdx + tmpLsLen;
        end


        idx = ~isnan(inAll(:, 1));
        allSlopeLs = inAll(idx,1);
        allInterCept = inAll(idx, 2);


        plot(equaSlopeLs, linearSlopeLs, '*');
        printLog("Slope", sprintf("%s - All,normal slope, mean
equation slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f",...
```

```
                status, mean(allSlopeLs), std(allSlopeLs),
mean(allInterCept), std(allInterCept) ));


    else
        tmpLs = slopeStru.('sub' + folderNum);
        equaSlopeLs = tmpLs(:, 1);
        linearSlopeLs = tmpLs(:, 3);
        plot(equaSlopeLs, linearSlopeLs, '*');
        printLog("Slope", sprintf("%s - Subject: %s, mean equation
slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f",...
                    status, folderNum, mean(equaSlopeLs),
std(equaSlopeLs), mean(tmpLs(:, 2)), std(tmpLs(:, 2) ) ));


        tmpLs = normalSlopeStru.('sub' + folderNum);
        equaSlopeLs = tmpLs(:, 1);
        linearSlopeLs = tmpLs(:, 3);
        plot(equaSlopeLs, linearSlopeLs, '*');
        printLog("Slope", sprintf("%s-Subject:%s,normal slope, mean
equation slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f",...
                    status, folderNum, mean(equaSlopeLs),
std(equaSlopeLs), mean(tmpLs(:, 2)), std(tmpLs(:, 2) ) ));


    end


%     for i = 1: numel(ls)
%         if strcmp(typeStr, 'isAll') || strcmp(typeStr, ls(i))
%             tmpLs = slopeStru.('sub' + ls(i));
%             equaSlopeLs = tmpLs(:, 1);
%             linearSlopeLs = tmpLs(:, 3);
%             plot(equaSlopeLs, linearSlopeLs, '*');
%             printLog("Slope", sprintf("%s - Subject: %s, mean
equation slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f",...
```

```
%                           status, typeStr, mean(equaSlopeLs),
std(equaSlopeLs), mean(tmpLs(:, 2)), std(tmpLs(:, 2) ) ));
%         end
%     end


    hold off;


    xlabel('Equation Slope') ;
    ylabel('Linear Regression Slope');
    if strcmp(typeStr, 'isAll')
        title('All the subjects');
    else
        title(folderNum);
    end
end


%%
function shortLs = getEqualLenEventWoutStage(splinePk, eventLs,
getSlope)
    global allEvent fieldLs folderNum status slopeStru;
    event1 = eventLs.e1;
    len_e1 = numel(eventLs.e1);


    shortLs = nan(50, 1);
    shortIdx = 1;


    subFolderName = 'sub' + folderNum;


    eventIdx = 1;
    for e = 1: len_e1
        %--- [WARNING 1 ] Haven't consider seperate the point by
using the
        %---              ratio of length of two events
```

```
        %--- [WARNING 2 ] Still need to consider if there is no
next pt
        %---              get the distance to the nx event

        isOverlap = checkOverlapping(event1, len_e1, e);
%         isOverlap = false;

        if isOverlap == true
            % Collecting the quantity of the event
            % Ignore this point and look on the next one.
            shortLs(shortIdx) = e;
            shortIdx = shortIdx + 1;
            % disp(i);
        else
            % This event is longer or equal to window size, SO
start collecting

            cur_e = event1{e};
            endIdx = cur_e(2);
            collectedPeriod = getNonOverlapPeriod(endIdx,
splinePk);
            if numel(collectedPeriod(~isnan(collectedPeriod))) == 0
                printLog("Warning",sprintf("Subject: %s (%s) >>> on
event idx: %d >>> without any data >>> skip in collected to
allevent", ...
                                folderNum, status, e));
                shortLs(shortIdx) = e;
                shortIdx = shortIdx + 1;
            else
                allEvent.('all'){eventIdx} = collectedPeriod;
                eventIdx = eventIdx + 1;

                if getSlope
```

```
                    stIdx = cur_e(1);
%                     try
                    [equSlope, equIntercept, lineSlope,
lineIntercept] = getLinearRegression(stIdx, endIdx, splinePk,
false);
                    slopeStru.(subFolderName)(e, :) = [equSlope,
equIntercept, lineSlope, lineIntercept];


%                     catch
%                         printLog('ERROR', sprintf('Event idx: %d
>>> fail to get the slope', e));
%                     end
                end
            end



        end
    end


    % Remove empty value in each cell
    for e = 1: numel(fieldLs)
        fieldName = fieldLs(e);
        if ~isempty(allEvent.(fieldName))
            allEvent.(fieldName) =
allEvent.(fieldName)(~cellfun('isempty',allEvent.(fieldName)));
        end
    end


    shortLs = shortLs(~isnan(shortLs));


    printLog("Message",sprintf("%s - Subject: %s >>> Get the all
same length event (without consider the stage)", status,
folderNum));
end
```

```
function removeEmptyinSlope
    global slopeStru folderNum;
    tmpLs = slopeStru.('sub' + folderNum);

    nanEmptyColIdx = ~isnan(tmpLs(:, 1));
    tmpLs = tmpLs(nanEmptyColIdx, :);
    nanEmptyColIdx = tmpLs(:,1) ~= 0;
    slopeStru.('sub' + folderNum) = tmpLs(nanEmptyColIdx, :);
end


function getNonEventCenterPlot(allNonEventInv, xx, showDiffColor )
    global status;
    stageDict = allNonEventInv.normalBre;
    avePlotSetting = {'--k', 'LineWidth', 1};
    upCILineSetting = { 'Color', 'r', 'LineWidth', 2};
    lowCILineSetting = { 'Color', colorConvertor('#003300'),
'LineWidth', 2};
    if showDiffColor
        period = getColorPeriod(stageDict.HexColor);

        figure;
%         plot([0 0], [min(stageDict.Min) max(stageDict.Max)],
'Color','k', 'LineStyle','--');
        hold on;

        for p = 1:numel(period)
            stIdx = period{p}(1);
            endIdx = period{p}(2)-1;

            curve1 = stageDict.Min(stIdx:endIdx).';
            curve2 = stageDict.Max(stIdx:endIdx).';
            x = xx(stIdx:endIdx);
```

```
            plot(x, curve2, upCILineSetting{:});
            plot(x, curve1, lowCILineSetting{:});



            x2 = [x, fliplr(x)];
            inBetween = [curve1, fliplr(curve2)];


            colorLs = colormap(jet(20));
            fill(x2, inBetween,
colorLs(stageDict.HexColor(stIdx), :), 'LineStyle','none',
'HandleVisibility','off');
            hcb = colorbar;
            set(get(hcb,'Title'),'String','A Title')
            caxis([0 100]);


            plot(x, stageDict.Ave(stIdx:endIdx),
avePlotSetting{:});
        end
        hold off;
        colorbar ;


    else
        figure;
%         plot([0 0], [min(stageDict.Min) max(stageDict.Max)],
'Color','k', 'LineStyle','--');


        hold on;
        curve1 = stageDict.Max.';
        curve2 = stageDict.Min.';


        plot(xx, curve1, upCILineSetting{:});
        plot(xx, curve2, lowCILineSetting{:});
```

```
        x2 = [xx, fliplr(xx)];
        inBetween = [curve1, fliplr(curve2)];
        fill(x2, inBetween, colorConvertor('#55a6f8'),
'LineStyle','none', 'HandleVisibility','off');

        plot(xx, stageDict.Ave, avePlotSetting{:});
        hold off;
    end
    legend({'95% CI Upper Bound','95% CI Lower Bound',...
                    'Mean ' + status },'Location','northwest');
    graphSetting;
    addAnnotation('b');
    title(sprintf("%s Normal Breathing\n", status));

end
%% Plot all linear normal breathing
function plotLinearNonEvent(xx_ne, nonEvent)
    global coeff status;
    figure;
    hold on;
    for i = 1: numel(nonEvent(1,:))
        oneNonEvent = nonEvent{2, i}.(status);
        oneNonEvent = oneNonEvent *coeff;
        plot(xx_ne, oneNonEvent);

    %     line([0 0], [min_num max_num], 'Color','k',
'LineStyle','--');
    end
    hold off;
    addAnnotation('a');
    graphSetting;
    title(sprintf("Normal Breathing, Number of Subject: %d\n",
numel(nonEvent(1,:))));
```

```
end
%%
function [totNorVal, totNorStd] = getTotNormalVal(nonEvent)
    global coeff status;
    interval = numel(nonEvent{2,1}.(status));
    totNorLs = nan(1, numel(nonEvent(2,:)) * interval);

    for i = 1:numel(nonEvent(2,:))
        stIdx = 1 + interval * (i-1);
        endIdx = interval * i;
        totNorLs(stIdx: endIdx) = nonEvent{2,i}.(status);
    end
    totNorVal = mean(totNorLs) * coeff;
    totNorStd = std(totNorLs) * coeff;

end

function [normalVal, normalStd] = getNormalVal(nonEvent)
    global folderNum coeff status;
    try
        subjLs = string(nonEvent(1,:));
        subjIdx = subjLs == folderNum;
        subjLs = nonEvent{2, subjIdx}.(status) ;
        normalVal = mean(subjLs).*coeff;
        normalStd = std(subjLs).*coeff;
    catch
        printLog('Messege', 'Cannot get the normal value');
        normalVal = nan;
        normalStd = nan;
    end
```

```
end


function getCenterPlot(allEventInv, stageName, shortEventLs,
showDiffColor, normalVal, avePk)
    global folderNum allEvent fieldLs xx halfWindow_sz ls status;


    avePlotSetting = getAvePlotSetting;
    centerLineSetting = getCenterLineSetting;
    [upCILineSetting ,lowCILineSetting] = getCILineSetting;


    for i = 1: numel(fieldLs)


        fieldName = fieldLs(i) + "Val";


        %                                       You Can specify
which field
        %                                       To view all the
graphy, use ""
        if (isfield(allEventInv,fieldName)) && (stageName == "" ||
strcmp(fieldLs(i), stageName) || strcmp(stageName, 'isAll'))
            stageDict = allEventInv.(fieldName);
            dict_len = numel(stageDict);
        else
            dict_len = 0;
        end


        if dict_len > 0
            aveLc = avePk(1);
            avePk = avePk(2);


            if showDiffColor
                period = getColorPeriod(stageDict.HexColor);
                figure;
                hold on;
```

```
                for p = 1:numel(period)
                    stIdx = period{p}(1);
                    endIdx = period{p}(2)-1;

                    curve1 = stageDict.Max(stIdx:endIdx).';
                    curve2 = stageDict.Min(stIdx:endIdx).';
                    x = xx(stIdx:endIdx);

                    plot(x, curve1, upCILineSetting{:});
                    plot(x, curve2, lowCILineSetting{:});

                    x2 = [x, fliplr(x)];
                    inBetween = [curve1, fliplr(curve2)];

                    colorLs = colormap(jet(20));
                    fill(x2, inBetween,
colorLs(stageDict.HexColor(stIdx), :), 'LineStyle','none',
'HandleVisibility','off');
                    hcb = colorbar;
                    set(get(hcb,'Title'),'String','A Title')
                    caxis([0 100]);

                    plot(x, stageDict.Ave(stIdx:endIdx),
avePlotSetting{:});
                end
                plot([0 0], [min(stageDict.Min)
max(stageDict.Max)], centerLineSetting{:});
                plotNormalVal;
                plotAvePk;
                hold off;
%                 legend({'95% CI Upper Bound','95% CI Lower
Bound', 'Mean value', 'Mean Average BP',  'Highest Average
Point'},'Location','southwest');
                colorbar ;
```

```
            else
                h = figure;
                hold on;
                curve1 = stageDict.Max.';
                curve2 = stageDict.Min.';
                plot(xx, curve1, upCILineSetting{:});
                plot(xx, curve2, lowCILineSetting{:});

                x2 = [xx, fliplr(xx)];
                inBetween = [curve1, fliplr(curve2)];
                fill(x2, inBetween, colorConvertor('#55a6f8'),
'LineStyle','none', 'HandleVisibility','off');

                plot(xx, stageDict.Ave, avePlotSetting{:});
                plotAvePk;
                plot([0 0], [min(stageDict.Min)
max(stageDict.Max)], centerLineSetting{:});
                plotNormalVal;
                hold off;

            end

            legend({'Mean Apnea ' + status,'Peak ' + status + '
Avg.', 'Avg. ' + status + ' Baseline'},'Location','northwest');
            graphSetting;

            if strcmp(stageName, "isAll") % aggrecated result
                shortNum = numel(shortEventLs);
                eventNum = numel(allEvent.(fieldLs(i)));
                totNum = shortNum + eventNum;
                title(sprintf("Subject Count: %d, # Event: %d (%d
%%) | <%d points: %d | Total: %d\n",...
```

```
                        numel(ls), eventNum, round(eventNum * 100/
totNum), halfWindow_sz, shortNum, totNum ));

            elseif strcmp(fieldLs(i), "all") % means in all
withouth considering stages

                shortNum = numel(shortEventLs);

                eventNum = numel(allEvent.(fieldLs(i)));

                totNum = shortNum + eventNum;

                title(sprintf("Subject: %s, Stage: %s, # Event: %d
(%d %%) | <%d points: %d | Total: %d\n",...

                        folderNum, getStageName(fieldLs(i)),
eventNum, round(eventNum * 100/ totNum), halfWindow_sz, shortNum,
totNum ));

            else

                title(sprintf("Subject: %s, Stage: %s\nEvent
Amount: %d", folderNum,
getStageName(fieldLs(i)),numel(allEvent.(fieldLs(i))) ));

            end


            addAnnotation('b');


%             if strcmp(stageName, "isAll")

%
savefig(h,'all_combination_regardless_overlap.fig');

%             else

%
savefig(h,sprintf('%s_combination_regardless_overlap.fig',
folderNum));

%             end

        end

    end


    function plotNormalVal

        plot([xx(1) xx(end)], [normalVal normalVal],
'Color',colorConvertor('#cc6600'), 'LineStyle','-', 'LineWidth',
2);

    end

    function plotAvePk
```

```
        plot(aveLc, avePk, 'ro', 'LineWidth', 2);
    end
end


function [upSetting, lowSetting] = getCILineSetting
    upSetting = { 'Color', 'r', 'LineWidth', 2,
'HandleVisibility','off'};
    lowSetting = { 'Color', colorConvertor('#003300'), 'LineWidth',
2, 'HandleVisibility','off'};
end


function setting = getCenterLineSetting
    setting = {'Color','k', 'LineStyle',':','LineWidth', 1,
'HandleVisibility','off'};
end


function setting = getAvePlotSetting
    setting = {'--k', 'LineWidth', 1};
end


function [allEventInv, lc, pk, pkStd] =
newGetInverseAllEvent(useStatistic, getPkStd)
    global allEvent fieldLs xx status;
    z_Val = 1.96;


    % Use as top limit, eg 10 >>> (5, 10]
    percentLs = [ 5, 10, 15, 20, ...
                  25, 30, 35, 40, ...
                  45, 50, 55, 60, ...
                  65, 70, 75, 80, ...
                  85, 90, 95,100];
%     percentLs = linspace(1, 100, colorScaleCount


    allEventInv = struct('sn5Val', [], ... %'sn5', [],...
```

```
                        'sn4Val', [], ... %'sn4', [],...
                        'sn3Val', [], ... %'sn3', [],...
                        'sn2Val', [], ... %'sn2', [],...
                        'sn1Val', [], ... %'sn1', [],...
                        's0Val' , [], ... %'s0' , [],...
                        's1Val' , []);    %'s1' , []);


    for i = 1:numel(fieldLs)
%     for i = 1:1
        % in Stage level
        eventLs = allEvent.(fieldLs(i));
%         disp(fieldLs(i));
        cell_len = numel(eventLs);
        if cell_len > 0
            colSz = numel(eventLs{1});
            %--- [WARNING] Have NOT apply inverse matrix to the
struct-> allEventInv
            %--- Create inverse matrix
            matr = nan(colSz, cell_len);
            for e = 1: cell_len
                matr(:, e) = eventLs{e};
            end


            valDict = struct('Min', nan(colSz, 1), 'Max',
nan(colSz, 1),...
                             'Ave', nan(colSz, 1), 'HexColor',
nan(colSz, 1));%strings(colSz, 1));


%             if fieldLs(i) == "all"
%                 disp('yes');
%             end
```

```
            %--- data in each row
            for r = 1: colSz
                row = matr(r, :);
                oriLen = numel(row);
                row = row(~isnan(row));
                newLen = numel(row);
                meanVal = mean(row);

                if useStatistic && newLen >= 5
                    oriLen = newLen;
                    int = z_Val * std(row);
                    CI_low = meanVal - int;
                    CI_up = meanVal + int;
%                     row = row( row >= CI_low);
%                     row = row( row <= CI_up);
                    newLen = numel(row);
                end
                percentVal = newLen * 100/ oriLen;
%                   colorTag = colorLs(find(percentLs >= percentVal,
1, 'first'));
                if numel(row) > 0 && newLen >= 5
                    % assign val from each row
                    valDict.Min(r) =  CI_low;
                    valDict.Max(r) =  CI_up;
                    valDict.Ave(r) = meanVal;
                    valDict.HexColor(r) = find(percentLs >=
percentVal, 1, 'first');
    %                 valDict.HexColor(r) = percentVal;
                end

            end

            allEventInv.(fieldLs(i) + 'Val') = valDict;
```

```
            [lc, pk] = getPeakVal(valDict.Ave);
            if getPkStd
                pkIdx = xx == lc;
                row = matr(pkIdx, :);
                row = row(~isnan(row));
                pkStd = std(row);
            else
                pkStd = nan;
            end


        end
    end


    printLog("Message",status + " - Inverse done! Get the
allEventInv ");


end


%% Plot all the event in each stage.
function plotEventinLinear(fieldType)
    global xx allEvent fieldLs folderNum;
    for i = 1: numel(fieldLs)
        eventList = allEvent.(fieldLs(i));


        cell_len = numel( allEvent.(fieldLs(i)));
        if cell_len > 0
            h = figure;
%             line([0 0], [100 130], 'Color','k', 'LineStyle','--
');
            hold on;
            max_num = 0;
            min_num = 200;
```

```
            for j = 1:  cell_len
                if max_num < max(eventList{j})
                    max_num = max(eventList{j});
                end

                if min_num > min(eventList{j})
                    min_num = min(eventList{j});
                end
%                 disp(j);
                plot(xx, eventList{j});
            end

            line([0 0], [min_num max_num], 'Color','k',
'LineStyle','--');
            hold off;
            xlabel('-30 to 30 (s)') ;
            ylabel('Blood Pressure (mmHg)');
            graphSetting;
            addAnnotation('a');

            if strcmp(fieldType, "isAll") % aggrecated result
                title(sprintf("All Subject, Stage: %s, Event
Amount: %d\n", getStageName(fieldLs(i)),cell_len));
            elseif strcmp(fieldType, "all") % means in all withouth
considering stages
                title(sprintf("Subject: %s, Stage: %s, Event
Amount: %d\n", folderNum, getStageName(fieldLs(i)),cell_len));
            end

%             if strcmp(fieldType, "isAll")
%                 savefig(h,'all_linear_regardless_overlap.fig');
%                 close(h)
%             else
```

```
%
savefig(h,sprintf('%s_linear_regardless_overlap.fig', folderNum));
%                 close(h)
%             end
        end
    end
end

function period = getColorPeriod(colorLs)
    idx = 1;
    curColor = colorLs(1);
    period = cell(numel(colorLs), 1);
    pIdx = 1;
    for c = 2: numel(colorLs)
        if colorLs(c) ~= curColor
            period{pIdx} = [idx c];
            curColor = colorLs(c);
            idx = c;
            pIdx = pIdx + 1;
        end
    end
    period{pIdx} = [idx c];
    period = period(~cellfun('isempty',period));
end

function allEventInv = getInverseNonEvent(nonEvent, useStatistic)
    global status;
    z_Val = 1.96;

    % Use as top limit, eg 10 >>> (5, 10]
    percentLs = [ 5, 10, 15, 20, ...
                 25, 30, 35, 40, ...
```

```
                45, 50, 55, 60, ...
                65, 70, 75, 80, ...
                85, 90, 95,100];
%     percentLs = linspace(1, 100, colorScaleCount

    allEventInv = struct('normalBre', []);
    colSz = numel(nonEvent{2,1}.(status));
    matr = zeros(numel(nonEvent(1,:)), colSz );




    for i = 1:numel(nonEvent(1,:))
%     for i = 1:1
        % in Stage level
        oneNonEvent = nonEvent{2, i}.(status);
        matr(i,:) = oneNonEvent.*100;
    end

    valDict = struct('Min', nan(colSz, 1), 'Max', nan(colSz, 1),...
                     'Ave', nan(colSz, 1), 'HexColor', nan(colSz,
1));%strings(colSz, 1));

    %--- data in each column
    for r = 1: colSz
        row = matr(:, r);
        oriLen = numel(row);
        row = row(~isnan(row));
        newLen = numel(row);
        meanVal = mean(row);

        CI_low = min(row);
        CI_up = max(row);
```

```
        if useStatistic && newLen >= 5
            oriLen = newLen;
            int = z_Val * std(row);
            CI_low = meanVal - int;
            CI_up = meanVal + int;
        end

        percentVal = newLen * 100/ oriLen;
    %   colorTag = colorLs(find(percentLs >= percentVal, 1,
'first'));
        if numel(row) > 0
            % assign val from each row
            valDict.Min(r) = CI_low;
            valDict.Max(r) = CI_up;
            valDict.Ave(r) = meanVal;
            valDict.HexColor(r) = find(percentLs >= percentVal, 1,
'first');
    %       valDict.HexColor(r) = percentVal;
        end

    end

    allEventInv.normalBre = valDict;
    printLog("Message",status + " - Inverse done! Get the
allNonEventInv ");

end

function stageCell = getStageList(stage)
    global folderNum status;
    sn1ls = cell(600, 1); sn2ls = cell(600, 1); sn3ls = cell(600,
1);
    sn4ls = cell(600, 1); sn5ls = cell(600, 1);  s0ls = cell(600,
1);
```

```
    s1ls  = cell(600, 1);


    sn1lsIdx = 1; sn2lsIdx = 1; sn3lsIdx = 1; sn4lsIdx = 1;
sn5lsIdx = 1;
    s1lsIdx = 1; s0lsIdx = 1;


    targetNum = stage(1);
    startPt = 1;
    for i = 2:numel(stage)
        curNum = stage(i);
    %     disp([i,curNum]);
        if curNum ~= targetNum
            switch targetNum
                case -5
                    sn5ls{sn5lsIdx} = [startPt, i-1];
                    sn5lsIdx = sn5lsIdx + 1;
                case -4
                    sn4ls{sn4lsIdx} = [startPt, i-1];
                    sn4lsIdx = sn4lsIdx + 1;
                case -3
                    sn3ls{sn3lsIdx} = [startPt, i-1];
                    sn3lsIdx = sn3lsIdx + 1;
                case -2
                    sn2ls{sn2lsIdx} = [startPt, i-1];
                    sn2lsIdx = sn2lsIdx + 1;
                case -1
                    sn1ls{sn1lsIdx} = [startPt, i-1];
                    sn1lsIdx = sn1lsIdx + 1;
                case 0
                    s0ls{s0lsIdx} = [startPt, i-1];
                    s0lsIdx = s0lsIdx + 1;
                case 1
```

```
                s1ls{s1lsIdx} = [startPt, i-1];
                s1lsIdx = s1lsIdx + 1;
        end
        targetNum = curNum;
        startPt = i;
    end
end

% catch the last one if it is still remain the same;
if curNum == targetNum
    switch targetNum
        case -5
            sn5ls{sn5lsIdx} = [startPt, i];
        case -4
            sn4ls{sn4lsIdx} = [startPt, i];
        case -3
            sn3ls{sn3lsIdx} = [startPt, i];
        case -2
            sn2ls{sn2lsIdx} = [startPt, i];
        case -1
            sn1ls{sn1lsIdx} = [startPt, i];
        case 0
            s0ls{s0lsIdx} = [startPt, i];
        case 1
            s1ls{s1lsIdx} = [startPt, i];
    end
end

sn5ls = sn5ls(~cellfun('isempty', sn5ls));
sn4ls = sn4ls(~cellfun('isempty', sn4ls));
sn3ls = sn3ls(~cellfun('isempty', sn3ls));
sn2ls = sn2ls(~cellfun('isempty', sn2ls));
```

```
    sn1ls = sn1ls(~cellfun('isempty', sn1ls));
    s0ls  = s0ls(~cellfun('isempty',  s0ls));
    s1ls  = s1ls(~cellfun('isempty',  s1ls));


    stageCell = struct('sn5', [], 'sn4', [], 'sn3', [], 'sn2',
[],...
                       'sn1', [], 's0',  [], 's1' , []);


    stageCell.sn5 = sn5ls;    stageCell.sn4 = sn4ls;
stageCell.sn3 = sn3ls;
    stageCell.sn2 = sn2ls;    stageCell.sn1 = sn1ls;
stageCell.s0 = s0ls;
    stageCell.s1  = s1ls;


    printLog("Message",sprintf("%s  - Subject: %s >>> Get the stage
list", status, folderNum));
end


%% collect each event stages in eventCell
function eventCell = getEventStageList(event)
    global folderNum status;
    e1ls = cell(600, 1); e2ls = cell(600, 1); e3ls = cell(600, 1);
e4ls = cell(600, 1); e10ls = cell(600, 1);
    e1lsIdx = 1; e2lsIdx = 1; e3lsIdx = 1; e4lsIdx = 1; e10lsIdx =
1;
    % event =
[10,10,10,10,3,3,2,2,2,1,1,1,1,4,4,4,4,3,3,3,2,2,1,1,10,10];
    %         1  2  3  4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5
6
    targetNum = event(1);
    startPt = 1;
    for i = 2:numel(event)
        curNum = event(i);
    %     disp([i,curNum]);
        if curNum ~= targetNum
```

```
        switch targetNum
            case 10
                e10ls{e10lsIdx} = [startPt, i-1];
                e10lsIdx = e10lsIdx + 1;
            case 4
                e4ls{e4lsIdx} = [startPt, i-1];
                e4lsIdx = e4lsIdx + 1;
            case 3
                e3ls{e3lsIdx} = [startPt, i-1];
                e3lsIdx = e3lsIdx + 1;
            case 2
                e2ls{e2lsIdx} = [startPt, i-1];
                e2lsIdx = e2lsIdx + 1;
            case 1
                e1ls{e1lsIdx} = [startPt, i-1];
                e1lsIdx = e1lsIdx + 1;
        end
        targetNum = curNum;
        startPt = i;
    end
end

% catch the last one if it is still remain the same;
if curNum == targetNum
    switch targetNum
        case 10
            e10ls{e10lsIdx} = [startPt, i];
        case 4
            e4ls{e4lsIdx} = [startPt, i];
        case 3
            e3ls{e3lsIdx} = [startPt, i];
        case 2
```

```
                e2ls{e2lsIdx} = [startPt, i];
            case 1
                e1ls{e1lsIdx} = [startPt, i];
        end
    end

    e10ls = e10ls(~cellfun('isempty', e10ls));
    e4ls = e4ls(~cellfun('isempty', e4ls));
    e3ls = e3ls(~cellfun('isempty', e3ls));
    e2ls = e2ls(~cellfun('isempty', e2ls));
    e1ls = e1ls(~cellfun('isempty', e1ls));

    eventCell = struct('e10', [], 'e4', [], 'e3', [], 'e2', [],
'e1', []);

    eventCell.e10 = e10ls;    eventCell.e4 = e4ls;    eventCell.e3
= e3ls;
    eventCell.e2 = e2ls;      eventCell.e1 = e1ls;

    printLog("Message",sprintf("%s - Subject: %s >>> Get the event
list", status, folderNum));
end
```

E. MATLAB CODE FOR AGGREGATION OF DATA CONSIDERING SLEEP STAGES

```
clc;clear; close all;
global folderNum halfWindow_sz xx xx_ne fieldLs coeff ls status
subEvent subEventInv;
global crxSequenceStru eventStr nonEventStr allEventStage
allEventStageInv isAll;
global subEventStr allEventStageStr subEventInvStr
allEventStageInvStr fs eventTypeLs;
global slopeStru normalSlopeStru;
```

```
% Init some variable ----------------------------------------------
------ %

% ---------------------------------

% Decide which type of value to use-

% status = "SBP";

status = "DBP";

% status = "MAP";

% status = "PP";

% ---------------------------------

eventStr = "Event";    nonEventStr = "NonEvent";

subEventStr = 'subEvent';    allEventStageStr = 'allEventStage';

subEventInvStr = 'subEventInv';    allEventStageInvStr =
'allEventStageInv';

eventTypeLs = [eventStr, nonEventStr];

isAll = true;


enableStatistcs = true;

showDiffColor = false;

getSTD = true;


rootDir = "..\\data";


fs = 100;

halfWindow_len = 30; %sec >>> Normal Breathing Window size

% xx = -60:0.01:60 - 0.01;

% xx_ne = halfWindow_len * (-1) : 1/fs : halfWindow_len - 1/fs; %
>>> Normal Breathing Window size

xx_ne = 0 : 1/fs : 2 * halfWindow_len - 1/fs; % >>> Normal
Breathing Window size


halfWindow_sz = halfWindow_len * fs;

% >>> Event Window Size

xx = halfWindow_len * (-1):1/fs:halfWindow_len - 1/fs; % xx = -
30:0.01:30 - 0.01;
```

```
% Slope Calculation ------------------------------------------------
------ %
collectSlope = 0;
slopeStru = initEventslopeStruc('stage');
normalSlopeStru = initEventslopeStruc('stage');
printLog('Message', 'Cell: 1~4 col >>> Equation slope, Equation
intercept, Regres slope, intercept');


% Cross Event Analysis ----------------------------------------------
------ %
crxSequenceStru = struct('sub01', [], 'sub03', [], 'sub05', [], ...
                         'sub06', [], 'sub07', [], 'sub10', [], ...
                         'sub11', [], 'sub12', [], 'sub13', [], ...
                         'sub15', [], 'subAll',[]);
fields = fieldnames(crxSequenceStru);
for fidx = 1: numel(fields)
    crxSequenceStru.(fields{fidx}) = cell(50,2);
end


fieldLs = ["sn5",  "sn4", "sn3", "sn2", "sn1", "s0", "s1", "all"];


ls = ["01", "03", "05", "06", "07", "10", "11", "12", "13", "15"];
% ls = ["01", "03", "05", "10", "15",  "07", "11", "06", "12",
"13" ];
% ls = ["01", "03", "05", "10", "15",  "07", "11" ];
if isAll
    initAllEventStageNInv;
end


 for fNum = 1: numel(ls)
% for fNum = 1:1
% folderNum = "10";
```

```
folderNum = ls(fNum);


[dict, splinePk, event, stage] = init(rootDir);
splinePk = removeNoisy(folderNum, dict, splinePk);


% collect each event stages in eventCell, same in stageLs
% include the idx start & end point of each stage period
stageStru = getStageList(stage);
eventStru = getEventStageList(event);


% Remove the event period which have no splinePk value ( ratio <
80%)
% eventStru = checkEventLs(eventStru, splinePk, 0.8);


% [eventPtLs, slopeLs] = getEventPtnSlope(dict, eventLs, false);


coeff = 100;
% Rescale the y unit %
splinePk = splinePk * coeff;
stage = stage * coeff;
event = event * coeff;


% Init.
initSubEventStruct;
printLog("Message",sprintf("%s - Subject: %s >>> Initialization
done...", status, folderNum));


% Include all the event with spline peak value period in each stage
getNonEventWtStage(stageStru, event, splinePk, collectSlope);
removeEmptyinStru(subEventStr, nonEventStr);
getInverseStruc(subEventStr, subEventInvStr, nonEventStr,
enableStatistcs, getSTD);
plotLinearData(subEventStr, nonEventStr);
```

```
% getCenterPlot(subEventStr, subEventInvStr, nonEventStr,
showDiffColor);



getCrxResult(splinePk, eventStru, stage, false, collectSlope);

removeEmptyinStru(subEventStr, eventStr);

getSummary4Event(subEventStr);

getInverseStruc(subEventStr, subEventInvStr, eventStr,
enableStatistcs, getSTD);

plotLinearData(subEventStr, eventStr);

% getCenterPlot(subEventStr, subEventInvStr, eventStr,
showDiffColor);


if collectSlope

    removeEmptyinSlope;

    getSlopePlot(folderNum);

end


% clear subEvent;

 end



close all;

removeEmptyinStru(allEventStageStr, nonEventStr);

getInverseStruc(allEventStageStr, allEventStageInvStr, nonEventStr,
enableStatistcs, getSTD);

plotLinearData(allEventStageStr, nonEventStr);

getCenterPlot(allEventStageStr, allEventStageInvStr, nonEventStr,
showDiffColor);


close all;

removeEmptyinStru(allEventStageStr, eventStr);

getSummary4Event(allEventStageStr);
```

```
getInverseStruc(allEventStageStr, allEventStageInvStr, eventStr,
enableStatistcs, getSTD);
getCenterPlot(allEventStageStr, allEventStageInvStr, eventStr,
showDiffColor);
plotLinearData(allEventStageStr, eventStr);



if collectSlope
    getSlopePlot('isAll');
end

disp('done');



%% ANOVA on sleep stages ~~~ NON EVENT ( BASELINE )
clc; close all;
sleepField = fieldLs(1:5);
% normal
sleepValLs = getSleepValLs(nonEventStr, sleepField);

sleepValLs(:, 3) = nan;
% variances
p = vartestn(sleepValLs);
set(gca,'xticklabel',{"REM", "2", "1"});
graphSetting;
xlabel('Stage');
grid on;
disp(p);

%%

[p,tbl,stats] = anova1(sleepValLs);
stats.gnames = {'REM';'2';'1'};
```

```
set(gca,'xticklabel',{"REM", "2", "1"});
graphSetting;
xlabel('Stage');
grid on;
%%
[c,m,h,nms] = multcompare(stats);
ylabel('Stage');
xlabel('Blood Pressure (mmHg)');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%% ANOVA on sleep stages ~~~ EVENT
clc; close all;
sleepField = fieldLs(1:5);
sleepValLs = getSleepValLs(eventStr, sleepField);

% Check normality
for s = 1:numel(sleepField)
    targetLs = sleepValLs(~isnan(sleepValLs(:,s)));
    if numel(targetLs)> 1
        % Check normality
        [H, pValue, W] = swtest(targetLs);
        resultStrNormal = isRejectNull(H);
        fprintf('%-7s,#,%3d,p-value,%2.5f, normality, %s\n',
getStageName( sleepField(s)), numel(targetLs),
pValue,resultStrNormal );
    end
end

%% homogeneity
% sleepValLs(:, 1) = nan;
p = vartestn(sleepValLs);
```

```
resultStr = isRejectNull(p);
disp({'homogeneity p-val',p,resultStr});
set(gca,'xticklabel',{ "3", "2", "1"});
% set(gca,'xticklabel',{"REM", "3", "2", "1"});
graphSetting;
xlabel('Stage');
grid on;

%%
clc;
% [p,tbl,stats] = anova1(sleepValLs,
'group',{'REM','4','3','2','1'});
% [p,tbl,stats] = anova1(sleepValLs, 'group',{"3","2","1"});
[p,tbl,stats] = anova1(sleepValLs);
stats.gnames = {'3';'2';'1'};
resultStr = isRejectNull(p);
disp({'ANOVA p-val',p,resultStr});
set(gca,'xticklabel',{"3", "2", "1"});
graphSetting;
xlabel('Stage');
grid on;
%%
[c,m,h,nms] = multcompare(stats);
set(gca,'yticklabel',{"3", "2", "1"});
box on;
set(gcf,'color','w');
ylabel('Stage');
xlabel('Blood Pressure (mmHg)');

%%
for s = 1:numel(sleepField)
    targetLs = sleepValLs(~isnan(sleepValLs(:,s)));
```

```
    if numel(targetLs)> 1
        for ss = s+1:numel(sleepField)
            compareLs = sleepValLs(~isnan(sleepValLs(:,ss)));
            if numel(compareLs)> 1
                [p,h,stats] = ranksum( targetLs , compareLs);
                resultStr = isRejectNull(h);
                fprintf('%-7s vs %-7s,  p-val, %3.5f, zval, %2.5f,
ranksum, %.3f >>> %s\n', getStageName( sleepField(s)),
getStageName( sleepField(ss)), p, stats.zval, stats.ranksum,
resultStr);
            end
        end
    end

end

% p = kruskalwallis(sleepValLs);

% Percentage



disp('ANOVA Sect. Done');

%%

function sleepValLs = getSleepValLs(typeStr, sleepField)
    global allEventStage allEventStageInv nonEventStr xx;
    sleepValLs = nan(500, 5);

    if strcmp(typeStr, nonEventStr)
        for s = 1:numel(sleepField)
            stageKey = sleepField(s);
            resultCell = allEventStage.(typeStr).(stageKey);
```

```
            if size(resultCell, 1)> 1
                for r = 1:size(resultCell, 1)
                    tmp = resultCell{r};
                    tmp = tmp(~isnan(tmp));
                    sleepValLs(r, s)= mean(tmp);
                end
            end
        end

    else
        for s = 1:numel(sleepField)
            stageKey = sleepField(s);
            resultCell = allEventStage.(typeStr).(stageKey);
            resultStru = allEventStageInv.(typeStr).(stageKey);
            if isstruct(resultStru)
                pkLc = resultStru.AveLc;
                pkIdx = xx == pkLc;
                for r = 1:size(resultCell, 1)
                    tmp = resultCell{r};
                    sleepValLs(r, s)= tmp(pkIdx);
                end
            end
        end
    end

end

function resultStr = isRejectNull(val)
    if val == 0
        resultStr = 'fail to reject the null hypothesis';
    elseif val == 1
        resultStr = 'rejection of the null hypothesis';
```

```
    elseif floor(val) ~= val % if it is float
        if val >= 0.05
            resultStr = 'fail to reject the null hypothesis';
        else
            resultStr = 'rejection of the null hypothesis';
        end
    end
end
%%
function getCrxResult(targeValLs, eventStru, stage,
getOnlyStartnEnd, getSlope)
    global folderNum crxSequenceStru subEvent allEventStage
eventStr isAll status;
    global slopeStru;


    event1 = eventStru.e1;
    len_e1 = numel(event1);
    oneStageNum = 0;
    shortLs = nan(50, 1);
    shortIdx = 1;
    crxNum = 0;
    totalNonOverlap = 0;
    sequFolderNum = "sub" + folderNum;
    sequAll = "subAll";
    slopeIdx = 1;
    for e = 1: len_e1
        isOverlap = checkOverlapping(event1, len_e1, e);

        if isOverlap == true
            % Ignore this point and look on the next one.
            shortLs(shortIdx) = e;
            shortIdx = shortIdx + 1;
        else
```

```
            % The non-overlapping event on either side
            totalNonOverlap = totalNonOverlap + 1;
            cur_e = event1{e};
            cur_stIdx = cur_e(1);
            cur_endIdx = cur_e(2);
            stagePeriod = stage(cur_stIdx:cur_endIdx); % catch the
period of the stage


            uniItems = unique(stagePeriod); % Check how many stage
there is


            if numel(uniItems) == 1 % Event in single stage
                % get event endpoint >>> endIdx
                oneStageNum = oneStageNum + 1;
                stageName = getStageKey(uniItems);
                idxNum = getNxIdx(subEvent.(eventStr),stageName );
                targetPeriod = getNonOverlapPeriod(cur_endIdx,
targeValLs);
                if numel(targetPeriod(~isnan(targetPeriod))) == 0
                    printLog("Warning",sprintf("Subject: %s (%s)
>>> on event idx: %d >>> without any data >>> skip in collected to
allevent", ...
                            folderNum, status, e));
                    continue
                end
                subEvent.(eventStr).(stageName){idxNum} =
targetPeriod;


                if getSlope
                    [equSlope, equIntercept, lineSlope,
lineIntercept] = getLinearRegression(cur_stIdx, cur_endIdx,
targeValLs, false);

slopeStru.(sequFolderNum).(stageName)(slopeIdx, :) = [equSlope,
equIntercept, lineSlope, lineIntercept];
                    slopeIdx = slopeIdx + 1;
```

```
                end

                if isAll
                    idxNum =
getNxIdx(allEventStage.(eventStr),stageName );
                    allEventStage.(eventStr).(stageName){idxNum} =
targetPeriod;
                end

                if uniItems == -500
                    printLog('REM Count', sprintf('#,%s, %3.2f',
folderNum, mean(targetPeriod)));
                end

            elseif numel(uniItems) > 1 % >>> crx stage event
                crxNum = crxNum + 1;
                % collect which stage change to sequenceLs
                sequenceLs = nan(1, 10);
                lsIdx = 1;
                sequenceLs(lsIdx) = stagePeriod(1);

                for stageNum = 2: numel(stagePeriod)
                    if stagePeriod(stageNum) ~= sequenceLs(lsIdx)
                        lsIdx = lsIdx + 1;
                        sequenceLs(lsIdx) = stagePeriod(stageNum);
                    end
                end



                sequenceLs = sequenceLs(~isnan(sequenceLs));

                if getOnlyStartnEnd
                    sequ = [sequenceLs(1), sequenceLs(end)];
```

```
                else
                    sequ = sequenceLs;
                end

                % check whether it is in the list.
                idxNum = getNxIdx(crxSequenceStru, sequFolderNum);
                [inLsResult, idxInList] = isInList(sequFolderNum);

                % in list then add the value
                if inLsResult
                    crxSequenceStru.(sequFolderNum){idxInList, 2} =
crxSequenceStru.(sequFolderNum){idxInList, 2} + 1;
                else
                    % not in the list add new row
                    crxSequenceStru.(sequFolderNum){idxNum, 1} =
sequ;
                    crxSequenceStru.(sequFolderNum){idxNum, 2} = 1;
                end

                % check whether it is in the list. >> For all
                idxNum = getNxIdx(crxSequenceStru, sequAll);
                [inLsResult, idxInList] = isInList(sequAll);

                % in list then add the value
                if inLsResult
                    crxSequenceStru.(sequAll){idxInList, 2} =
crxSequenceStru.(sequAll){idxInList, 2} + 1;
                else
                    % not in the list add new row
                    crxSequenceStru.(sequAll){idxNum, 1} = sequ;
                    crxSequenceStru.(sequAll){idxNum, 2} = 1;
                end
```

```
            else
                disp('else');
            end
        end

    end

    printLog("Count", sprintf('Subject: %s >>> single stage evnet#: %3d, crx stage event #: %2d/%2d (%2.2f %%%%)',...
                              folderNum, oneStageNum, crxNum, totalNonOverlap, crxNum*100/ totalNonOverlap));

    function idxNum = getNxIdx(parentStru, childField)
        tmpLs = ~cellfun('isempty', parentStru.(childField));
        idxNum = find(tmpLs == 0, 1, 'first');
    end

    function [inList, idxInList] = isInList(sequName)
        inList = false;
        curIdx = 1;
        idxInList = nan;
        while inList == false && curIdx <= idxNum
            if isequal(crxSequenceStru.(sequName){curIdx, 1}, sequ)
                inList = true;
                idxInList = curIdx;
            end
            curIdx = curIdx + 1;
        end
    end
end

function getSlopePlot(typeStr)
    global slopeStru folderNum ls status fieldLs normalSlopeStru;
```

```
    figure;
    maxSlope = 2;%max(linearSlopeLs);
    plot([0, maxSlope], [0, maxSlope], '--k');
    hold on;

    if strcmp(typeStr, 'isAll')

        for fieldIdx = 1: numel(fieldLs)-1
            allStIdx = 1;
            inAll = nan(1000, 2);
            fieldName = fieldLs(fieldIdx);

            for i = 1: numel(ls)
                tmpLs = slopeStru.('sub' + ls(i)).(fieldName);
                equaSlopeLs = tmpLs(:, 1);
                equalInterCept = tmpLs(:, 2);
                linearSlopeLs = tmpLs(:, 3);
                tmpLsLen = numel(equaSlopeLs);
    %             disp([allStIdx,tmpLsLen]);
                inAll(allStIdx:allStIdx + tmpLsLen - 1, 1) =
equaSlopeLs;
                inAll(allStIdx:allStIdx + tmpLsLen - 1, 2) =
equalInterCept;
                allStIdx = allStIdx + tmpLsLen;
            end

            idx = ~isnan(inAll(:, 1));
            allSlopeLs = inAll(idx,1);
            allInterCept = inAll(idx, 2);

            plot(equaSlopeLs, linearSlopeLs, '*');
```

```
            printLog("Slope", sprintf("%s - All, %-7s, mean
equation slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f",...
                      status, getStageName(fieldName),
mean(allSlopeLs), std(allSlopeLs), mean(allInterCept),
std(allInterCept) ));


            % -------- Normal Slope Part -------------------------
--------
            allStIdx = 1;
            inAll = nan(1000, 2);


            for i = 1: numel(ls)
                tmpLs = normalSlopeStru.('sub' +
ls(i)).(fieldName);
                equaSlopeLs = tmpLs(:, 1);
                equalInterCept = tmpLs(:, 2);
                linearSlopeLs = tmpLs(:, 3);
                tmpLsLen = numel(equaSlopeLs);
    %             disp([allStIdx,tmpLsLen]);
                inAll(allStIdx:allStIdx + tmpLsLen - 1, 1) =
equaSlopeLs;
                inAll(allStIdx:allStIdx + tmpLsLen - 1, 2) =
equalInterCept;
                allStIdx = allStIdx + tmpLsLen;
            end


            idx = ~isnan(inAll(:, 1));
            allSlopeLs = inAll(idx,1);
            allInterCept = inAll(idx, 2);


            plot(equaSlopeLs, linearSlopeLs, '*');
            printLog("Slope", sprintf("%s - All, %-7s,normal slope,
mean equation slope, %2.3f,+-, %2.3f, mean equation intercept,
%2.3f,+-, %2.3f",...
```

```
                    status, getStageName(fieldName),
mean(allSlopeLs), std(allSlopeLs), mean(allInterCept),
std(allInterCept) ));
        end
    else
        for fieldIdx = 1: numel(fieldLs)-1
            fieldName = fieldLs(fieldIdx);
            tmpLs = slopeStru.('sub' + folderNum).(fieldName);
            equaSlopeLs = tmpLs(:, 1);
            linearSlopeLs = tmpLs(:, 3);
            plot(equaSlopeLs, linearSlopeLs, '*');
            printLog("Slope", ...
            sprintf("%s - Subject: %s, %-7s, mean equation slope,
%2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-, %2.3f,#,%d",...
                status, typeStr, getStageName(fieldName), ...
                mean(equaSlopeLs), std(equaSlopeLs), mean(tmpLs(:,
2)), std(tmpLs(:, 2)), numel(equaSlopeLs) ));


            fieldName = fieldLs(fieldIdx);
            tmpLs = normalSlopeStru.('sub' +
folderNum).(fieldName);
            equaSlopeLs = tmpLs(:, 1);
            linearSlopeLs = tmpLs(:, 3);
            plot(equaSlopeLs, linearSlopeLs, '*');
            printLog("Slope", ...
            sprintf("%s - Subject: %s, %-7s,normal slope, mean
equation slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f,#,%d",...
                status, typeStr, getStageName(fieldName), ...
                mean(equaSlopeLs), std(equaSlopeLs), mean(tmpLs(:,
2)), std(tmpLs(:, 2)), numel(equaSlopeLs) ));
        end
    end


%     for i = 1: numel(ls)
```

```
%         if strcmp(typeStr, 'isAll') || strcmp(typeStr, ls(i))
%             for fieldIdx = 1: numel(fieldLs)-1
%                 fieldName = fieldLs(fieldIdx);
%
%                 tmpLs = slopeStru.('sub' + ls(i)).(fieldName);
%                 equaSlopeLs = tmpLs(:, 1);
%                 linearSlopeLs = tmpLs(:, 3);
%                 plot(equaSlopeLs, linearSlopeLs, '*');
%                 printLog("Slope", ...
%                 sprintf("%s - Subject: %s, %-7s, mean equation
slope, %2.3f,+-, %2.3f, mean equation intercept, %2.3f,+-,
%2.3f,#,%d",...
%                     status, typeStr, getStageName(fieldName), ...
%                     mean(equaSlopeLs), std(equaSlopeLs),
mean(tmpLs(:, 2)), std(tmpLs(:, 2)), numel(equaSlopeLs) ));
%             end
%         end
%     end

    hold off;

    xlabel('Equation Slope') ;
    ylabel('Linear Regression Slope');
    if strcmp(typeStr, 'isAll')
        title('All the subjects');
    else
        title(folderNum);
    end
end

function removeEmptyinSlope
    global slopeStru folderNum fieldLs normalSlopeStru;
```

```
    for fieldIdx = 1: numel(fieldLs)-1
        fieldName = fieldLs(fieldIdx);
        tmpLs = slopeStru.('sub' + folderNum).(fieldName);
        emptyColIdx = ~isnan(tmpLs(:, 1));
        tmpLs = tmpLs(emptyColIdx, :);
        nanEmptyColIdx = tmpLs(:,1) ~= 0;
        slopeStru.('sub' + folderNum).(fieldName) =
tmpLs(nanEmptyColIdx, :);


        tmpLs = normalSlopeStru.('sub' + folderNum).(fieldName);
        emptyColIdx = ~isnan(tmpLs(:, 1));
        tmpLs = tmpLs(emptyColIdx, :);
        nanEmptyColIdx = tmpLs(:,1) ~= 0;
        normalSlopeStru.('sub' + folderNum).(fieldName) =
tmpLs(nanEmptyColIdx, :);
    end


end


function getSummary4Event(struName)
    global status fieldLs eventStr;


    tmpStru = getStruByStr(struName);


    for stgIdx = 1:numel(fieldLs)
        stageName = fieldLs(stgIdx);
        if ~strcmp(stageName, "s0") && ~strcmp(stageName, "all")
            fieldCell = tmpStru.(eventStr).(stageName);
            fieldCell_len = numel(fieldCell);


            detail = sprintf('%s - %s, Stage: %-7s, NonOverlapping
Event #: %3d',...
                        status, getTitleStr(struName),
getStageName(stageName), fieldCell_len);
```

```
            printLog('Message', detail);
        end
    end
end


%%
function getCenterPlot(struStr, struInvStr, eventType,
showDiffColor)
    global fieldLs status xx xx_ne eventStr nonEventStr;
    tmpStru = getStruByStr(struStr);
    tmpInvStru = getStruByStr(struInvStr);
    isEvent = true;


    xx_axis = xx;
    % in event plot >> the legend not include ave line and 2
confidence
    % internval line
    if strcmp(eventType, eventStr)
        avePlotSetting = getAvePlotSetting;
        [upCILineSetting ,lowCILineSetting] =
getCILineSetting(isEvent);
        centerLineSetting = getCenterLineSetting;
        nonEventInvStru = tmpInvStru.(nonEventStr);
    else
        xx_axis = xx_ne;
        isEvent = false;
        avePlotSetting = getAvePlotSetting;
        [upCILineSetting ,lowCILineSetting] =
getCILineSetting(isEvent);
    end


    if isfield(tmpInvStru, eventType)
        tmpInvStru = tmpInvStru.(eventType);
    end
```

```
    for fieldidx = 1:numel(fieldLs)
%     for i = 1:1
        % in Stage level
        fieldName = fieldLs(fieldidx);
        if ~isfield(tmpInvStru, fieldName)
            continue
        end
        stageDict = tmpInvStru.(fieldName);

        if isempty(stageDict) || ~isstruct(stageDict)
            continue
        end

        if isEvent && ~isstruct(nonEventInvStru.(fieldName))
            continue
        end

        if showDiffColor
            period = getColorPeriod(stageDict.HexColor);

            figure;
    %         plot([0 0], [min(stageDict.Min) max(stageDict.Max)],
'Color','k', 'LineStyle','--');
            hold on;

            for p = 1:numel(period)
                stIdx = period{p}(1);
                endIdx = period{p}(2)-1;

                curve1 = stageDict.Min(stIdx:endIdx).';
                curve2 = stageDict.Max(stIdx:endIdx).';
```

```
                x = xx(stIdx:endIdx);

                plot(x, curve2, upCILineSetting{:});
                plot(x, curve1, lowCILineSetting{:});



                x2 = [x, fliplr(x)];
                inBetween = [curve1, fliplr(curve2)];

                colorLs = colormap(jet(20));
                fill(x2, inBetween,
colorLs(stageDict.HexColor(stIdx), :), 'LineStyle','none',
'HandleVisibility','off');
                hcb = colorbar;
                set(get(hcb,'Title'),'String','A Title')
                caxis([0 100]);

                plot(x, stageDict.Ave(stIdx:endIdx),
avePlotSetting{:});
            end
            hold off;
            colorbar ;

        else
            figure;

            hold on;
            curve1 = stageDict.Max.';
            curve2 = stageDict.Min.';

            plot(xx_axis, curve1, upCILineSetting{:});
            plot(xx_axis, curve2, lowCILineSetting{:});
```

```
            x2 = [xx_axis, fliplr(xx_axis)];
            inBetween = [curve1, fliplr(curve2)];
            fill(x2, inBetween, colorConvertor('#55a6f8'),
'LineStyle','none', 'HandleVisibility','off');

            plot(xx_axis, stageDict.Ave, avePlotSetting{:});

            if isEvent
                plot([0 0], [min(stageDict.Min)
max(stageDict.Max)], centerLineSetting{:});

                [normalVal, normalStd] =
getNormalVal(stageDict.Ave);
                plotNormalVal;
                plotAvePk;

            end

            hold off;

            if isEvent
                legend({'Mean Apnea ' + status, 'Avg. ' + status +
' Baseline', 'Peak ' + status + ' Avg.'},'Location','northwest');
            else
                legend({'95% CI Upper Bound','95% CI Lower
Bound',...
                        'Mean Event '+
status },'Location','northwest');
            end

        end

        % for different color, it is combined with multiple period,
so
        % that's the reason it shows error.
```

```
        graphSetting;
        eventNum =  numel(tmpStru.(eventType).(fieldName));
        title(sprintf("%s-%s %s Stage: %s Event Amount: %d\n", ...
                status, getTitleStr(struInvStr), eventType,
getStageName(fieldName), eventNum ));
        addAnnotation('b');
        if isEvent
            dispSummary;
        end
    end


    function dispSummary
        pk = stageDict.AvePk;
        lc = stageDict.AveLc;
        pkSTD = stageDict.AveSTD;
        printLog('Summary', sprintf('%s - %s Stage: %-7s >>>
NormalVal: %2.1f +- %2.1f, PeakVal: %2.1f +- %2.1f, Diff: %2.1f
(%2.1f %%%%), Peak Time: %2.1f, # of Events: %3d',...
                                    status, getTitleStr(struInvStr),
getStageName(fieldName), normalVal,normalStd, pk, pkSTD, pk-
normalVal, (pk-normalVal)*100/normalVal , lc, eventNum ));
    end


    function plotNormalVal
        plot([xx_axis(1) xx_axis(end)], [normalVal normalVal],
'Color',colorConvertor('#cc6600'), 'LineStyle','-', 'LineWidth',
2);
    end
    function plotAvePk
        plot(stageDict.AveLc, stageDict.AvePk, 'ro', 'LineWidth',
2);
    end
```

```
end


function titleStr = getTitleStr(str)
    global subEventStr subEventInvStr allEventStageStr
allEventStageInvStr folderNum;
    if strcmp(str, subEventInvStr) || strcmp(str, subEventStr)
        titleStr = sprintf('Subject - %s',folderNum);
    elseif strcmp(str, allEventStageInvStr) || strcmp(str,
allEventStageStr)
        titleStr = "All";
    end
end


function getInverseStruc(oriStruStr, targetInvStruStr, eventType,
useStatistic, getPkStd)
    global fieldLs status halfWindow_sz xx;
    z_Val = 1.96;


    oriStru = getStruByStr(oriStruStr);
    oriInvStru = getStruByStr(targetInvStruStr);



    % Use as top limit, eg 10 >>> (5, 10]
    percentLs = [ 5, 10, 15, 20, ...
                  25, 30, 35, 40, ...
                  45, 50, 55, 60, ...
                  65, 70, 75, 80, ...
                  85, 90, 95,100];
    fullWindowsz = halfWindow_sz * 2;


    for fieldidx = 1:numel(fieldLs)
        % in Stage level
        fieldName = fieldLs(fieldidx);
```

```
        if ~isfield(oriStru.(eventType), fieldName)
            continue
        end
%             disp(fieldName);

        eventLs = oriStru.(eventType).(fieldName);

        cell_len = numel(eventLs);
        if cell_len > 0
            % --- Create inverse matrix
            matr = nan(fullWindowsz, cell_len);
            for e = 1: cell_len
                matr(:, e) = eventLs{e};
            end

            valDict = struct('Min', nan(fullWindowsz, 1), 'Max',
nan(fullWindowsz, 1),...
                             'Ave', nan(fullWindowsz, 1),
'HexColor', nan(fullWindowsz, 1),...
                             'AveLc', [], 'AvePk', [], 'AveSTD',
[]);%strings(colSz, 1));

            skipNum = 0;
            %--- data in each row >>> analyze sample point by
sample point
            for r = 1: fullWindowsz
                row = matr(r, :);
                oriLen = numel(row);
                row = row(~isnan(row));
                newLen = numel(row);
                meanVal = mean(row);

                if useStatistic && newLen >= 5
                    int = z_Val * std(row);
```

```
                    CI_low = meanVal - int;
                    CI_up = meanVal + int;
                end

                percentVal = newLen * 100/ oriLen;
                % colorTag = colorLs(find(percentLs >= percentVal,
1, 'first'));
                if numel(row) > 0 && newLen >= 5
                    % assign val from each row
                    valDict.Min(r) =  CI_low;
                    valDict.Max(r) =  CI_up;
                    valDict.Ave(r) = meanVal;
                    valDict.HexColor(r) = find(percentLs >=
percentVal, 1, 'first');
                else
                    skipNum = skipNum + 1;
                end

            end % for r = 1: fullWindowsz

            if skipNum ~= fullWindowsz
                [lc, pk] = getPeakVal(valDict.Ave);
                if getPkStd
                    pkIdx = xx == lc;
                    row = matr(pkIdx, :);
                    row = row(~isnan(row));
                    pkStd = std(row);
                else
                    pkStd = nan;
                end

                valDict.AveLc = lc;
                valDict.AvePk = pk;
```

```
                valDict.AveSTD = pkStd;
                oriInvStru.(eventType).(fieldName) = valDict;
            end

        end % end of if event ls contains value
    end % end of fieldLs loop

    update2StrutureByStr(targetInvStruStr, oriInvStru, eventType);

    printLog("Message", sprintf("%s - Inverse done in %s!\tFrom %s
Get the %s ",status, eventType, oriStruStr, targetInvStruStr) );

end

%% Plot all the event in each stage.
function plotLinearData(strucName, typeEvent)
    global xx fieldLs folderNum;

    [tmpStru, inAll] = getStruByStr(strucName);

    for i = 1: numel(fieldLs)
%         disp(fieldLs(i));
        if ~isfield(tmpStru.(typeEvent), fieldLs(i))
            continue
        end
        eventList = tmpStru.(typeEvent).(fieldLs(i));

        cell_len = numel(eventList);
        if cell_len > 0
            figure;
%             line([0 0], [100 130], 'Color','k', 'LineStyle','--
');
            hold on;
```

```
            max_num = 0;
            min_num = 200;
            for j = 1:  cell_len
                if max_num < max(eventList{j})
                    max_num = max(eventList{j});
                end

                if min_num > min(eventList{j})
                    min_num = min(eventList{j});
                end
                plot(xx, eventList{j});
            end

            line([0 0], [min_num max_num], 'Color','k',
'LineStyle','--');
            hold off;

            if inAll
                title(sprintf("%s - Subject:ALL, Stage: %s, %s
Amount: %d\n", typeEvent,  getStageName(fieldLs(i)),typeEvent,
cell_len));
            else
                title(sprintf("%s - Subject: %s, Stage: %s, %s
Amount: %d\n", typeEvent, folderNum,
getStageName(fieldLs(i)),typeEvent, cell_len));
            end
            xlabel('-30 to 30 (s)') ;
            ylabel('Blood Pressure (mmHg)');
            graphSetting;
            addAnnotation('a');
        end
    end
end
```

```
function getNonEventWtStage(stageStruct, event, splinePk, getSlope)
    global status folderNum fieldLs subEvent coeff nonEventStr
halfWindow_sz isAll allEventStage normalSlopeStru;
    eventLen = numel(event);
    sequFolderNum = "sub" + folderNum;
    for stgIdx = 1:numel(fieldLs)
        stageName = fieldLs(stgIdx);
        if ~strcmp(stageName, "s0") && ~strcmp(stageName, "all")
            stagePeriodLs = stageStruct.(stageName);
            stagePeriodLsLen = numel(stagePeriodLs);
            % nonNum = 0;
            slopeIdx = 1;
            stageFieldIdx = 0;
            % Check on each period in this stage
            for idx = 1:stagePeriodLsLen
                [stIdx, endIdx] = getPeriodIdx(stagePeriodLs{idx});

                % Check whether there is a apnea event( 1 x coeff)
in it
                if inList(event(stIdx:endIdx), 1 * coeff)
                    % include apnea event.
                    % The apnea event collected in
                    continue

                else
                    % It is a non event period

                    % Check this period is >= window sz
                    if endIdx - stIdx +1 >= halfWindow_sz*2

                        % Check the edge of window whether overlap
with other events
                        % Get the center point of this stage period
```

```
                        centerIdx = round(( endIdx - stIdx )/2) +
stIdx;
                        f3000Idx = centerIdx - 3000;
                        b3000Idx = centerIdx + 2999;

                        % less than total len
                        if eventLen >= b3000Idx + 3000
                            % Check forward and backward for 3000
sample to see
                            % whether overlap (contain 100)
                            if ~isGotcha( flip(event(f3000Idx -
3000: f3000Idx)), 1 * coeff) && ...
                               ~isGotcha( event(b3000Idx: b3000Idx
+ 3000), 1 * coeff)
                            % Start collecting
                            %
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
                            % (1.)
                            % How do I know whether I should move
the center
                            % point when there is an event close
by?
                            %
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ

                                stageFieldIdx = stageFieldIdx + 1;
                                targetPeriod = splinePk(f3000Idx:
b3000Idx);

subEvent.(nonEventStr).(stageName){stageFieldIdx} = targetPeriod;

                                if getSlope
                                    [equSlope, equIntercept,
lineSlope, lineIntercept] = getLinearRegression(1, 2 *
halfWindow_sz, targetPeriod, true);

normalSlopeStru.(sequFolderNum).(stageName)(slopeIdx, :) =
[equSlope, equIntercept, lineSlope, lineIntercept];
```

```
                                    slopeIdx = slopeIdx + 1;
                                end


                                if isAll

allEventStage.(nonEventStr).(stageName){getIdx(allEventStage,
stageName, nonEventStr )} = targetPeriod;
                                    end
                                end


                            end


                        end
                    end


            end


            detail = sprintf('%s - Subject: %s, Stage: %-7s, Non
Event Collect: %2d / %3d >>> %2.1f %%%%',...
                  status, folderNum,  getStageName(stageName),
stageFieldIdx, stagePeriodLsLen, stageFieldIdx *
100/stagePeriodLsLen);
            printLog('Message', detail);
        end
    end


end


function gotcha = isGotcha(targetLs, matchOne)
    gotcha = false;
    uniqLs = unique(targetLs);
    if ismember(uniqLs, matchOne)
        gotcha = true;
    end
```

```
%     matchOne = 1 * coeff;
%     targetLs = event(f3000Idx - 3000: f3000Idx);
%     targetIdx = 1;
%     targetLen = numel(targetLs);
%     gotcha = false;
%     while targetIdx <= targetLen && ~gotcha == true
%         if targetLs(targetIdx) == matchOne
%             gotcha = true;
%         end
%         targetIdx = targetIdx + 1;
%     end
end


function removeEmptyinStru(struName, typeStr)
    global fieldLs;

    tmpStru = getStruByStr(struName);

    for e = 1: numel(fieldLs)
        fieldName = fieldLs(e);
        if isfield(tmpStru.(typeStr), fieldName )
            if ~isempty(tmpStru.(typeStr).(fieldName))
                tmpStru.(typeStr).(fieldName) =
tmpStru.(typeStr).(fieldName)(~cellfun('isempty',tmpStru.(typeStr).
(fieldName)));
            end
        end
    end

    insert2StrutureByStr(struName, tmpStru)
end
```

```
function [tmpStru, isAll] = getStruByStr(str)
    global subEvent subEventInv allEventStage allEventStageInv;
    global subEventStr allEventStageStr subEventInvStr
allEventStageInvStr;
    isAll = false;
    if strcmp(str, subEventStr)
        tmpStru = subEvent;
    elseif strcmp(str, subEventInvStr)
        tmpStru = subEventInv;
    elseif strcmp(str, allEventStageStr)
        tmpStru = allEventStage;
        isAll = true;
    elseif strcmp(str, allEventStageInvStr)
        tmpStru = allEventStageInv;
        isAll = true;
    end
end


function update2StrutureByStr(str, tmpStru, eventType)
    global subEvent subEventInv allEventStage allEventStageInv;
    global subEventStr allEventStageStr subEventInvStr
allEventStageInvStr;


    tmpStru = tmpStru.(eventType);


    if strcmp(str, subEventStr)
        subEvent.(eventType) = tmpStru;
    elseif strcmp(str, subEventInvStr)
        subEventInv.(eventType) = tmpStru;
    elseif strcmp(str, allEventStageStr)
        allEventStage.(eventType) = tmpStru;
    elseif strcmp(str, allEventStageInvStr)
```

```
        allEventStageInv.(eventType) = tmpStru;
    end
end

function insert2StrutureByStr(str, tmpStru)
    global subEvent subEventInv allEventStage allEventStageInv;
    global subEventStr allEventStageStr subEventInvStr
allEventStageInvStr;

    if strcmp(str, subEventStr)
        subEvent = tmpStru;
    elseif strcmp(str, subEventInvStr)
        subEventInv = tmpStru;
    elseif strcmp(str, allEventStageStr)
        allEventStage = tmpStru;
    elseif strcmp(str, allEventStageInvStr)
        allEventStageInv = tmpStru;
    end
end

function [stIdx, endIdx] = getPeriodIdx(twoXoneVector)
    if numel(twoXoneVector) == 2
        stIdx = twoXoneVector(1);
        endIdx = twoXoneVector(2);
    else
        stIdx = nan;
        endIdx = nan;
        disp('Fail to get the stIdx & endIdx');
    end

end
```

```
function shortLs = getEqualLenEventWtStage(splinePk, stage,
eventLs)
    global halfWindow_sz subEvent folderNum eventStr;
    event1 = eventLs.e1;
    len_e1 = numel(eventLs.e1);


    shortLs = nan(50, 1);
    shortIdx = 1;


    eventIdx = 1;
    for e = 1: len_e1
        cur_e = event1{e};
        endPt = cur_e(2);
        %--- [WARNING 1 ] Haven't consider seperate the point by
using the
        %---              ratio of length of two events
        %
        %--- [WARNING 2 ] Still need to consider if there is no
next pt
        %---              get the distance to the nx event


        %--- get the distance to the nx event
        if e+1 <= len_e1
            nx_e = event1{ e + 1 };
            nx_stPt = nx_e(1);
            diffbtw_nx_e = nx_stPt - endPt;
            if diffbtw_nx_e < halfWindow_sz
                % The distance between two event are shorter than
halfWindow_sz
                longEnough = false;
            else
                longEnough = true;
            end
```

```
        end

        %--- get the distance to the pre event
        if longEnough == true && e > 1
            pre_e = event1{ e - 1 };
            pre_endPt = pre_e(2);

            if endPt - pre_endPt < halfWindow_sz
                longEnough = false;
            end
        end

        if longEnough == false
            % Collecting the quantity of the event
            % Ignore this point and look on the next one.
            shortLs(shortIdx) = e;
            shortIdx = shortIdx + 1;
            % disp(i);
        else
            % This event is longer or equal to window size, SO
start collecting

            % Check crx stage
            stageNum = nan; isCrx = false;
            checkCrxStage;



            x_stPt  = endPt - halfWindow_sz;
            x_endPt = endPt + halfWindow_sz - 1;

            oneEvent = nan(halfWindow_sz * 2 ,1);
            stIdx = 1;
```

```
            oneEvent(stIdx:halfWindow_sz) = splinePk(x_stPt:endPt-
1);

            oneEvent(halfWindow_sz + 1 :halfWindow_sz*2 - stIdx +
1) = splinePk(endPt:x_endPt);


            subEvent.('all').(eventStr){eventIdx} = oneEvent;

            eventIdx = eventIdx + 1;


            if ~isCrx

                subEvent.('sn1').(eventStr){getIdx(subEvent,
stageNum, eventStr)} = oneEvent;

            end

        end


    end


    % Remove empty value in each cell

%     for e = 1: numel(fieldLs)

%         fieldName = fieldLs(e);

%         if ~isempty(subEvent.(fieldName).(eventStr))

%             subEvent.(fieldName).(eventStr) =
subEvent.(fieldName).(eventStr)(~cellfun('isempty',subEvent.(fieldN
ame).(eventStr)));

%         end

%     end

%

    removeEmptyinStru(eventStr)


    shortLs = shortLs(~isnan(shortLs));


    printLog("Message",sprintf("Subject: %s >>> Get the all same
length event (without consider the stage)", folderNum));
```

```
    function checkCrxStage
        stIdx = cur_e(1);
        endIdx = cur_e(2);
        stagePeriod = stage(stIdx:endIdx); % catch the period of
the stage

        uniItems = unique(stagePeriod); % Check how many stage
there is

        if numel(uniItems) > 1 % >>> crx stage event
            isCrx = true;
        else
            isCrx = false;
            stageNum = round(uniItems(1));

        end
    end

end

function idxNum = getIdx(struName, fieldType, eventType )
    global fieldLs;
    if ~ismember(fieldLs, fieldType)
        fieldType = getStageKey(fieldType);
    end

    zeroLs = ~cellfun('isempty',struName.(eventType).(fieldType));
    idxNum = find(zeroLs == 0, 1, 'first');
end

function initSubEventStruct
    global fieldLs subEvent subEventInv eventTypeLs;
```

```
    % The structure is - Event - fieldStruct
    %                   - NonEvent - fieldStruct
    %
    % fieldStruct = struct('sn5',[], 'sn4',[], 'sn3',[], ...
    %                      'sn2',[], 'sn1',[], 's0' ,[], 's1' ,[]);

    subEvent = struct();
    subEventInv = struct();
    fieldStruct = struct();

    for fieldIdx = 1: numel(fieldLs)-1
        fieldName = fieldLs(fieldIdx);
        fieldStruct.(fieldName) = cell(600,1);
    end

    for idx = 1:numel(eventTypeLs)
        eventType = eventTypeLs(idx);
        subEvent.(eventType) = fieldStruct;
        subEventInv.(eventType) = fieldStruct;
    end

end

function initAllEventStageNInv
    global fieldLs allEventStage allEventStageInv eventTypeLs;
    % The structure is - Event - fieldStruct
    %                   - NonEvent - fieldStruct
    %
    % fieldStruct = struct('sn5',[], 'sn4',[], 'sn3',[], ...
    %                      'sn2',[], 'sn1',[], 's0' ,[], 's1' ,[]);

    allEventStage =  struct();
```

```
    allEventStageInv =  struct();
    fieldStruct = struct();

    for fieldIdx = 1: numel(fieldLs)-1
        fieldName = fieldLs(fieldIdx);
        fieldStruct.(fieldName) = cell(1000,1);
    end

    for idx = 1:numel(eventTypeLs)
        eventType = eventTypeLs(idx);
        allEventStage.(eventType) = fieldStruct;
        allEventStageInv.(eventType) = fieldStruct;
    end

end

%%
function [totNorVal, totNorStd] = getTotNormalVal(nonEvent)
    global coeff;
    interval = numel(nonEvent{2,1});
    totNorLs = nan(1, numel(nonEvent(2,:)) * interval);

    for i = 1:numel(nonEvent(2,:))
        stIdx = 1 + interval * (i-1);
        endIdx = interval * i;
        totNorLs(stIdx: endIdx) = cell2mat(nonEvent{2,i});
    end
    totNorVal = mean(totNorLs) * coeff;
    totNorStd = std(totNorLs) * coeff;

end
```

```
function [normalVal, normalStd] = getNormalVal(aveLs)
    try
        normalVal = mean(aveLs);
        normalStd = std(aveLs);
    catch
        printLog('Messege', 'Cannot get the normal value');
        normalVal = nan;
        normalStd = nan;
    end
end


function getOverallCenterPlot(subEventInv, stageName, shortEventLs,
showDiffColor, normalVal, avePk)
    global folderNum subEvent fieldLs xx halfWindow_sz ls status;


    for i = 1: numel(fieldLs)


        fieldName = fieldLs(i) + "Val";


        %                                       You Can specify
which field
        %                                       To view all the
graphy, use ""
        if (isfield(subEventInv,fieldName)) && (stageName == "" ||
strcmp(fieldLs(i), stageName) || strcmp(stageName, 'isAll'))
            stageDict = subEventInv.(fieldName);
            dict_len = numel(stageDict);
        else
            dict_len = 0;
        end


        if dict_len > 0
            aveLc = avePk(1);
            avePk = avePk(2);
```

```
            avePlotSetting = getAvePlotSetting;
            centerLineSetting = getCenterLineSetting;
            [upCILineSetting ,lowCILineSetting] = getCILineSetting;

            if showDiffColor
                period = getColorPeriod(stageDict.HexColor);
                figure;
                hold on;
                for p = 1:numel(period)
                    stIdx = period{p}(1);
                    endIdx = period{p}(2)-1;

                    curve1 = stageDict.Max(stIdx:endIdx).';
                    curve2 = stageDict.Min(stIdx:endIdx).';
                    x = xx(stIdx:endIdx);

                    plot(x, curve1, upCILineSetting{:});
                    plot(x, curve2, lowCILineSetting{:});

                    x2 = [x, fliplr(x)];
                    inBetween = [curve1, fliplr(curve2)];

                    colorLs = colormap(jet(20));
                    fill(x2, inBetween,
colorLs(stageDict.HexColor(stIdx), :), 'LineStyle','none',
'HandleVisibility','off');
                    hcb = colorbar;
                    set(get(hcb,'Title'),'String','A Title')
                    caxis([0 100]);

                    plot(x, stageDict.Ave(stIdx:endIdx),
avePlotSetting{:});
```

```
                end
                plot([0 0], [min(stageDict.Min)
max(stageDict.Max)], centerLineSetting{:});
                plotNormalVal;
                plotAvePk;
                hold off;
%                 legend({'95% CI Upper Bound','95% CI Lower
Bound', 'Mean value', 'Mean Average BP',  'Highest Average
Point'},'Location','southwest');
                colorbar ;


            else
                figure;
                hold on;
                curve1 = stageDict.Max.';
                curve2 = stageDict.Min.';
                plot(xx, curve1, upCILineSetting{:});
                plot(xx, curve2, lowCILineSetting{:});


                x2 = [xx, fliplr(xx)];
                inBetween = [curve1, fliplr(curve2)];
                fill(x2, inBetween, colorConvertor('#55a6f8'),
'LineStyle','none', 'HandleVisibility','off');


                plot(xx, stageDict.Ave, avePlotSetting{:});
                plotAvePk;
                plot([0 0], [min(stageDict.Min)
max(stageDict.Max)], centerLineSetting{:});
                plotNormalVal;
                hold off;


            end
```

```
            legend({'Mean Apnea ' + status,'Peak ' + status + '
Avg.', 'Avg. ' + status + ' Baseline'},'Location','northwest');
            graphSetting;


            if strcmp(stageName, "isAll")
                shortNum = numel(shortEventLs);
                eventNum = numel(subEvent.(fieldLs(i)));
                totNum = shortNum + eventNum;
                title(sprintf("Subject Count: %d\nEvent Amount: %d
(%d %%) | <%d points: %d | Total: %d ",...
                        numel(ls), eventNum, round(eventNum * 100/
totNum), halfWindow_sz, shortNum, totNum ));
            elseif strcmp(fieldLs(i), "all")
                shortNum = numel(shortEventLs);
                eventNum = numel(subEvent.(fieldLs(i)));
                totNum = shortNum + eventNum;
                title(sprintf("Subject: %s, Stage: %s\nEvent
Amount: %d (%d %%) | <%d points: %d | Total: %d ",...
                        folderNum, getStageName(fieldLs(i)),
eventNum, round(eventNum * 100/ totNum), halfWindow_sz, shortNum,
totNum ));
            else
                title(sprintf("Subject: %s, Stage: %s\nEvent
Amount: %d", folderNum,
getStageName(fieldLs(i)),numel(subEvent.(fieldLs(i))) ));
            end
        end
    end


    function plotNormalVal
        plot([xx(1) xx(end)], [normalVal normalVal],
'Color',colorConvertor('#cc6600'), 'LineStyle','-', 'LineWidth',
2);
    end
    function plotAvePk
        plot(aveLc, avePk, 'ro', 'LineWidth', 2);
```

```
    end
end

%%
function [upSetting, lowSetting] = getCILineSetting(isEvent)
    if isEvent
        upSetting = { 'Color', 'r', 'LineWidth', 2,
'HandleVisibility','off'};
        lowSetting = { 'Color', colorConvertor('#003300'),
'LineWidth', 2, 'HandleVisibility','off'};
    else
        upSetting = { 'Color', 'r', 'LineWidth', 2,
'HandleVisibility','on'};
        lowSetting = { 'Color', colorConvertor('#003300'),
'LineWidth', 2, 'HandleVisibility','on'};
    end

end

function setting = getCenterLineSetting
    setting = {'Color','k', 'LineStyle',':','LineWidth', 1,
'HandleVisibility','off'};
end

function setting = getAvePlotSetting
    setting = {'--k', 'LineWidth', 1, 'HandleVisibility','on'};
end

%% Plot all linear normal breathing
function plotLinearNonEvent(xx_ne, nonEvent)
    global coeff;
    figure;
    hold on;
    for i = 1: numel(nonEvent(1,:))
```

```
        oneNonEvent = nonEvent{2, i};
        oneNonEvent = cell2mat(oneNonEvent).*coeff;
        plot(xx_ne, oneNonEvent, 'k');


    %     line([0 0], [min_num max_num], 'Color','k',
'LineStyle','--');
    end
    hold off;
    graphSetting;
    title(sprintf("Normal Breathing\nNumber of Subject: %d",
numel(nonEvent(1,:))));


end


function period = getColorPeriod(colorLs)
    idx = 1;
    curColor = colorLs(1);
    period = cell(numel(colorLs), 1);
    pIdx = 1;
    for c = 2: numel(colorLs)
        if colorLs(c) ~= curColor
            period{pIdx} = [idx c];
            curColor = colorLs(c);
            idx = c;
            pIdx = pIdx + 1;
        end
    end
    period{pIdx} = [idx c];
    period = period(~cellfun('isempty',period));
end


function targetStageName = getStageName(fieldKey)
    switch fieldKey
```

```
        case "sn5"
            targetStageName = "REM";
        case "sn4"
            targetStageName = "Stage 4";
        case "sn3"
            targetStageName = "Stage 3";
        case "sn2"
            targetStageName = "Stage 2";
        case "sn1"
            targetStageName = "Stage 1";
        case "s0"
            targetStageName = "Stage 0";
        case "s1"
            targetStageName = "Awake";
        case "all"
            targetStageName = "All Events in all stage";
    end
end

function targetStage = getStageKey(num)
    switch num
        case -500
            targetStage = "sn5";
        case -400
            targetStage = "sn4";
        case -300
            targetStage = "sn3";
        case -200
            targetStage = "sn2";
        case -100
            targetStage = "sn1";
        case 0
```

```
            targetStage = "s0";
        case 100
            targetStage = "s1";
    end
end


function stageCell = getStageList(stage)
    global folderNum;
    sn1ls = cell(600, 1); sn2ls = cell(600, 1); sn3ls = cell(600,
1);
    sn4ls = cell(600, 1); sn5ls = cell(600, 1);  s0ls = cell(600,
1);
    s1ls  = cell(600, 1);


    sn1lsIdx = 1; sn2lsIdx = 1; sn3lsIdx = 1; sn4lsIdx = 1;
sn5lsIdx = 1;
    s1lsIdx = 1; s0lsIdx = 1;


    targetNum = stage(1);
    startPt = 1;
    for i = 2:numel(stage)
        curNum = stage(i);
    %     disp([i,curNum]);
        if curNum ~= targetNum
            switch targetNum
                case -5
                    sn5ls{sn5lsIdx} = [startPt, i-1];
                    sn5lsIdx = sn5lsIdx + 1;
                case -4
                    sn4ls{sn4lsIdx} = [startPt, i-1];
                    sn4lsIdx = sn4lsIdx + 1;
                case -3
                    sn3ls{sn3lsIdx} = [startPt, i-1];
```

```
                sn3lsIdx = sn3lsIdx + 1;
            case -2
                sn2ls{sn2lsIdx} = [startPt, i-1];
                sn2lsIdx = sn2lsIdx + 1;
            case -1
                sn1ls{sn1lsIdx} = [startPt, i-1];
                sn1lsIdx = sn1lsIdx + 1;
            case 0
                s0ls{s0lsIdx} = [startPt, i-1];
                s0lsIdx = s0lsIdx + 1;
            case 1
                s1ls{s1lsIdx} = [startPt, i-1];
                s1lsIdx = s1lsIdx + 1;
        end
        targetNum = curNum;
        startPt = i;
    end
end

% catch the last one if it is still remain the same;
if curNum == targetNum
    switch targetNum
        case -5
            sn5ls{sn5lsIdx} = [startPt, i];
        case -4
            sn4ls{sn4lsIdx} = [startPt, i];
        case -3
            sn3ls{sn3lsIdx} = [startPt, i];
        case -2
            sn2ls{sn2lsIdx} = [startPt, i];
        case -1
            sn1ls{sn1lsIdx} = [startPt, i];
```

```
            case 0
                s0ls{s0lsIdx} = [startPt, i];
            case 1
                s1ls{s1lsIdx} = [startPt, i];
        end
    end

    sn5ls = sn5ls(~cellfun('isempty', sn5ls));
    sn4ls = sn4ls(~cellfun('isempty', sn4ls));
    sn3ls = sn3ls(~cellfun('isempty', sn3ls));
    sn2ls = sn2ls(~cellfun('isempty', sn2ls));
    sn1ls = sn1ls(~cellfun('isempty', sn1ls));
    s0ls  = s0ls(~cellfun('isempty',  s0ls));
    s1ls  = s1ls(~cellfun('isempty',  s1ls));

    stageCell = struct('sn5', [], 'sn4', [], 'sn3', [], 'sn2',
[],...
                       'sn1', [], 's0',  [], 's1' , []);

    stageCell.sn5 = sn5ls;    stageCell.sn4 = sn4ls;
stageCell.sn3 = sn3ls;
    stageCell.sn2 = sn2ls;    stageCell.sn1 = sn1ls;
stageCell.s0 = s0ls;
    stageCell.s1  = s1ls;

    printLog("Message",sprintf("Subject: %s >>> Get the stage
list", folderNum));
end

%% collect each event stages in eventCell
function eventCell = getEventStageList(event)
    global folderNum;
    e1ls = cell(600, 1); e2ls = cell(600, 1); e3ls = cell(600, 1);
e4ls = cell(600, 1); e10ls = cell(600, 1);
```

```
    e1lsIdx = 1; e2lsIdx = 1; e3lsIdx = 1; e4lsIdx = 1; e10lsIdx =
1;
    % event =
[10,10,10,10,3,3,2,2,2,1,1,1,1,4,4,4,4,3,3,3,2,2,1,1,10,10];
    %         1  2  3  4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5
6
    targetNum = event(1);
    startPt = 1;
    for i = 2:numel(event)
        curNum = event(i);
    %     disp([i,curNum]);
        if curNum ~= targetNum
            switch targetNum
                case 10
                    e10ls{e10lsIdx} = [startPt, i-1];
                    e10lsIdx = e10lsIdx + 1;
                case 4
                    e4ls{e4lsIdx} = [startPt, i-1];
                    e4lsIdx = e4lsIdx + 1;
                case 3
                    e3ls{e3lsIdx} = [startPt, i-1];
                    e3lsIdx = e3lsIdx + 1;
                case 2
                    e2ls{e2lsIdx} = [startPt, i-1];
                    e2lsIdx = e2lsIdx + 1;
                case 1
                    e1ls{e1lsIdx} = [startPt, i-1];
                    e1lsIdx = e1lsIdx + 1;
            end
            targetNum = curNum;
            startPt = i;
        end
    end
```

```
    % catch the last one if it is still remain the same;
    if curNum == targetNum
        switch targetNum
            case 10
                e10ls{e10lsIdx} = [startPt, i];
            case 4
                e4ls{e4lsIdx} = [startPt, i];
            case 3
                e3ls{e3lsIdx} = [startPt, i];
            case 2
                e2ls{e2lsIdx} = [startPt, i];
            case 1
                e1ls{e1lsIdx} = [startPt, i];
        end
    end

    e10ls = e10ls(~cellfun('isempty', e10ls));
    e4ls = e4ls(~cellfun('isempty', e4ls));
    e3ls = e3ls(~cellfun('isempty', e3ls));
    e2ls = e2ls(~cellfun('isempty', e2ls));
    e1ls = e1ls(~cellfun('isempty', e1ls));


    eventCell = struct('e10', [], 'e4', [], 'e3', [], 'e2', [],
'e1', []);


    eventCell.e10 = e10ls;    eventCell.e4 = e4ls;    eventCell.e3
= e3ls;
    eventCell.e2 = e2ls;      eventCell.e1 = e1ls;


    printLog("Message",sprintf("Subject: %s >>> Get the event
list", folderNum));
end
```

F. ADDITIONAL FUNCTIONS

addAnnotation.m

```
function addAnnotation(str)
    annotation('textbox', [0.83, 0.79, 0.1, 0.1], 'String',
sprintf("(%s)",str), 'BackgroundColor', 'white');
end
```

---

checkOverlapping.m

```
function isOverlap = checkOverlapping(eventCell, eventCellLen,
curIdx)
    global halfWindow_sz;
    cur_e = eventCell{curIdx};
    cur_stIdx  = cur_e(1);
    cur_endIdx = cur_e(2);
%     disp(curIdx);
    isOverlap = false;
    if curIdx +1 <= eventCellLen
        % Check the distance between next event start point and the
endPoint of
        % current event
        nx_e = eventCell{ curIdx + 1 };
        nx_stIdx  = nx_e(1);
%         nx_endIdx = nx_e(2);
        if abs(nx_stIdx - cur_endIdx) < halfWindow_sz
            % The distance between two event are shorter than
halfWindow_sz *2
            isOverlap = true;

        end
    end
```

```
    %--- get the distance to the pre event
    if isOverlap == false && curIdx > 1
        pre_e = eventCell{ curIdx - 1 };
%         pre_stIdx  = pre_e(1);
        pre_endIdx = pre_e(2);

        if abs(cur_stIdx - pre_endIdx) < halfWindow_sz
            isOverlap = true;
        end
    end
end
```

---

colorConvertor.m

```
function colorCode = colorConvertor(HexadecimalColor)
    % Convert color code to 1-by-3 RGB array (0~1 each)
    % str = '#FF0000'; >> not " "
    colorCode = sscanf(HexadecimalColor(2:end),'%2x%2x%2x',[1
3])/255;
return
```

---

excludeOutlier.m

```
function result = excludeOutlier(value, limit, resultType,
direction)
    IQR = iqr(value);
    Q3 = quantile(value,0.75);
    Q1 = quantile(value,0.25);
    if resultType == "data"
        if direction == "up"
```

```
            value(value >= Q3+(IQR*limit)) = nan;
        elseif direction == "low"
            value(value <= Q1-(IQR*limit)) = nan;
        elseif direction == "all"
            value(value >= Q3+(IQR*limit)) = nan;
            value(value <= Q1-(IQR*limit)) = nan;
        end
        result = value;
    elseif resultType == "outlierIdx"
        if direction == "up"
            result = find(value >= Q3+(IQR*limit));
        elseif direction == "low"
            result = find(value <= Q1-(IQR*limit));
        elseif direction == "all"
            result = [find(value >= Q3+(IQR*limit))' find(value <=
Q1-(IQR*limit))'];
        end
    end
end
```

---

findDistribution.m

```
function [distX, distY] = findDistribution(pointnRange, x, y,
varPairSet, goDownward, threshold , showFigure )


% Warning >> varPairSet: only the last one is unparied. eg:
{'MinPeakDistance',0.5, 'MinPeakHeight'};
% toSmaller => true = descending; false = ascending;
% distX = nan;
% distY = nan;
```

```
% eg: findRange = -0.4:-0.01:-1.4;
findRange = pointnRange;
% disp(findRange);
meanNum = NaN(numel(findRange),1);
noPoints = false;
lenfindRange = numel(findRange);
i = 1;

while i <= lenfindRange && noPoints == false
    if goDownward%descending
        val = findRange(lenfindRange + 1 - i) .* -1;
    else
        val = findRange(i);
    end

    lc = findpeaks(y, x, varPairSet{:}, val);
%     disp(lc);
%     lc = findpeaks(findPeaksDataset);
    troughNum = numel(lc);
%     disp([lenFindM + 1 - i,minPkVal, round(minPkVal .* (-1)),
troughNum]);
%     disp(troughNum);
    meanNum(i) = troughNum;
    if (troughNum < threshold)
        noPoints = true;
    end
%     disp(findM(i));
    i = i + 1;
end

distX = findRange(1:numel(meanNum));
distY = meanNum;
```

```
if showFigure
   figure;plot(distX, distY, '-bo');
   title('From findDistribution');
end

return
```

---

findPeriodPeak.m

```
function [val, lc] = findPeriodPeak(x, y, startPoint, endPoint,
showFigure)

% startPoint = 1;
% endPoint = 55535;
% showFigure = true;
% nBP = bp(startPoint:endPoint);
% nX = x(startPoint:endPoint);

%% Break the BP to several parts
nBP = y(startPoint:endPoint);
nX = x(startPoint:endPoint);
% figure; plot(nX,nBP);

%% find peaks to get the max and min value
% [tmpPk, ~] = findpeaks(nBP, nX);
% maxVal = max(tmpPk);
maxVal = 2.5;
minVal = 0.6;
% minVal = min(tmpPk);
```

```
% disp([maxVal, minVal]);


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%
%

%------------------------------Peak Distance-------------------------
-------%

%
%

% Get the best Min Peak Distance

[tmpX, tmpY] = findDistribution(0.4:0.01:1.5,nX, nBP, ...

                {'MinPeakDistance'}, false, 0, false);

minPeakDistance = getSetVal(tmpX, tmpY, 1, false);


% % TEST MIN PEAK DISTANCE

% [maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance);

% figure;

% plot(nX, nBP, maxLc, maxPk, 'r^');

% fprintf('Number of Peak: %d\n\n', numel(maxLc));


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%
%

%------------------------------Peak Height--------------------------
-------%

%
%

% Get the best Min Peak Height

% clc; close all;

[tmpX, tmpY] = findDistribution(minVal:0.01:maxVal,nX, nBP, ...

                 {'MinPeakDistance',minPeakDistance,
'MinPeakHeight'}, false, 0, false);
```

```
minPeakHeight = getSetVal(tmpX, tmpY, 1, false);


% TEST MIN PEAK HEIGHT

% figure;

% [maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight', minPeakHeight);

% plot(nX, nBP, maxLc, maxPk, 'r^');

% title(sprintf('Number of Peak: %d', numel(maxLc)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% %%


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%
%

%----------------------------Peak Prominence-----------------------
-------%

%
%

[tmpX, tmpY] = findDistribution(0.1:0.01:0.5,nX, nBP, ...

                                  {'MinPeakDistance',minPeakDistance,
'MinPeakHeight', minPeakHeight, 'MinPeakProminence'}, false, 0,
false);

minPeakProminence = getSetVal(tmpX, tmpY, 30, false);

%
%

%% TEST MIN PEAK Prominence

% figure;

% [maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight',
minPeakHeight,'MinPeakProminence', minPeakProminence);

% plot(nX, nBP, maxLc, maxPk, 'r^');

% title(sprintf('Number of Peak: %d', numel(maxLc)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% %%

%%
```

```
[maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight', minPeakHeight,
'MinPeakProminence', minPeakProminence);%,
'MinPeakProminence',minPeakProminence);%,'Annotate','extents');
% figure;
% plot(nX, nBP, maxLc, maxPk, 'r^');


fprintf('Best Peak Distance = %f\n', minPeakDistance);
fprintf('Best Peak Height = %f\n', minPeakHeight);
fprintf('Best Peak Prominence = %f\n\n', minPeakProminence);


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%-------------------------------------------------------------------
------%%
%%---------+END OF FINDING PEAKS+-----------+END OF FINDING PEAKS+---
------%%
%%-------------------------------------------------------------------
------%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% %%


%%
if showFigure
    figure;
    plot(nX, nBP, 'Color', [0.4660 0.6740 0.1880]);
    title('From findPeroidPeak');
    hold on
    plot(maxLc, maxPk,'r^');
    hold off;
    title(sprintf("%d - %d",startPoint, endPoint));
end


val = maxPk;
lc = maxLc;
```

```
return
```

---

findPeriodTrough.m

```
function [val, lc] = findPeriodTrough(x, y, startPoint, endPoint,
showFigure)



% startPoint = 933589;
% endPoint = 1011388;
% nBP = bp(startPoint:endPoint);
% nX = x(startPoint:endPoint);
% negBP = nBP * -1;
% showFigure = true;
% testMode = true;
%%
nBP = y(startPoint:endPoint);
nX = x(startPoint:endPoint);
% figure;plot(nX, nBP);
negBP = nBP * -1;
testMode = false;
minPeakDistance = nan;
minPeakHeight = nan;
minPeakProminence = nan;
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
%
```

```
%------------------------------Peak Distance-------------------------
-------%

%
%

%

[tmpX, tmpY] = findDistribution(0.5:0.01:1.5,nX, negBP,
{'MinPeakDistance'}, false, 0, false);

minPeakDistance = getSetVal(tmpX, tmpY, 1, false);

%
%

%
%

if testMode

    [minPk,minLc] = findpeaks(negBP, nX,
'MinPeakDistance',minPeakDistance);

    figure('units','normalized','outerposition',[0 0 1 1]);plot(nX,
negBP, minLc, minPk, 'r^');

    title(sprintf('With MinPeakDistance\nNumber of Peak: %d',
numel(minPk)));

end

%
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% %%




%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

%
%

%------------------------------Peak Height--------------------------
-------%

%
%

%

% Get the best Min Peak Height

% clc;% close all;
```

```
[tmpPk, ~] = findpeaks(negBP, nX);
negMax = -0.4;
negMin = min(tmpPk);
% negMax = minVal .* -1;
% negMin = maxVal .* -1;
disp([negMin, negMax]);
[tmpX, tmpY] = findDistribution(negMin:0.01:negMax,nX, nBP, ...
                        {'MinPeakDistance',minPeakDistance,
'MinPeakHeight'}, true, 10, false);
minPeakHeight = getSetVal(tmpX, tmpY, 1, false);


%
if testMode
    [minPk,minLc] = findpeaks(negBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight', minPeakHeight);
    figure;plot(nX, negBP, minLc, minPk, 'r^');
    title(sprintf('With MinPeakDistance + MinPeakHeight\nNumber of
Peak: %d', numel(minPk)));
end
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% %%


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
%
%----------------------------Peak Prominence-----------------------
-------%
%
%
%
%
```

```
[tmpX, tmpY] = findDistribution(0.1:0.01:0.5,nX, negBP, ...
                    {'MinPeakDistance',minPeakDistance,
'MinPeakHeight', minPeakHeight, 'MinPeakProminence'}, false, 0,
false);
minPeakProminence = getSetVal(tmpX, tmpY, 30, false);



%
if testMode
    [minPk,minLc] = findpeaks(negBP, nX,
'MinPeakDistance',minPeakDistance, ...
                                          'MinPeakHeight',
minPeakHeight, ...

'MinPeakProminence',minPeakProminence);
    disp(numel(minPk));
    figure;plot(nX, negBP);
    hold on; plot(minLc, minPk, '^', 'Color',
colorConvertor('#54009E'));hold off;
    title(sprintf('With MinPeakDistance + MinPeakHeight +
MinPeakProminence: %f\n -- Number of Peak: %d',minPeakProminence,
numel(minPk)));
end
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% %%


%%
[minPk,minLc] = findpeaks(negBP, nX,
'MinPeakDistance',minPeakDistance,...
                                       'MinPeakHeight',
minPeakHeight, ...
                                       'MinPeakProminence',
minPeakProminence);
if showFigure
```

```
    figure;plot(nX, negBP, minLc, minPk, 'r^');
    title(sprintf('It is trough With
MinPeakDistance:%f\nMinPeakHeight:%f\nMinPeakProminence:%f\n --
Number of Peak: %d'...
        ,minPeakDistance,minPeakHeight,minPeakProminence,
numel(minPk)));
end


minPk = minPk .* -1;


val = minPk;
lc = minLc;


fprintf('Best Trough Distance = %f\n', minPeakDistance);
fprintf('Best Trough Height = %f\n', minPeakHeight);
fprintf('Best Trough Prominence = %f\n\n', minPeakProminence);


return
```

---

function.txt

---

addAnnotation.m

```
function addAnnotation(str)
    annotation('textbox', [0.83, 0.79, 0.1, 0.1], 'String',
sprintf("(%s)",str), 'BackgroundColor', 'white');
end
```

---

checkOverlapping.m

```
function isOverlap = checkOverlapping(eventCell, eventCellLen,
curIdx)
    global halfWindow_sz;
    cur_e = eventCell{curIdx};
    cur_stIdx  = cur_e(1);
    cur_endIdx = cur_e(2);
%     disp(curIdx);
    isOverlap = false;
    if curIdx +1 <= eventCellLen
        % Check the distance between next event start point and the
endPoint of
        % current event
        nx_e = eventCell{ curIdx + 1 };
        nx_stIdx  = nx_e(1);
%         nx_endIdx = nx_e(2);
        if abs(nx_stIdx - cur_endIdx) < halfWindow_sz
            % The distance between two event are shorter than
halfWindow_sz *2
            isOverlap = true;

        end
    end

    %--- get the distance to the pre event
    if isOverlap == false && curIdx > 1
        pre_e = eventCell{ curIdx - 1 };
%         pre_stIdx  = pre_e(1);
        pre_endIdx = pre_e(2);

        if abs(cur_stIdx - pre_endIdx) < halfWindow_sz
            isOverlap = true;
        end
```

```
    end
end
```

---

colorConvertor.m

```
function colorCode = colorConvertor(HexadecimalColor)
    % Convert color code to 1-by-3 RGB array (0~1 each)
    % str = '#FF0000'; >> not " "
    colorCode = sscanf(HexadecimalColor(2:end),'%2x%2x%2x',[1
3])/255;
return
```

---

excludeOutlier.m

```
function result = excludeOutlier(value, limit, resultType,
direction)
    IQR = iqr(value);
    Q3 = quantile(value,0.75);
    Q1 = quantile(value,0.25);
    if resultType == "data"
        if direction == "up"
            value(value >= Q3+(IQR*limit)) = nan;
        elseif direction == "low"
            value(value <= Q1-(IQR*limit)) = nan;
        elseif direction == "all"
            value(value >= Q3+(IQR*limit)) = nan;
            value(value <= Q1-(IQR*limit)) = nan;
        end
        result = value;
    elseif resultType == "outlierIdx"
        if direction == "up"
```

```
            result = find(value >= Q3+(IQR*limit));
        elseif direction == "low"
            result = find(value <= Q1-(IQR*limit));
        elseif direction == "all"
            result = [find(value >= Q3+(IQR*limit))' find(value <=
Q1-(IQR*limit))'];
        end



    end

end
```

---

findDistribution.m

```
function [distX, distY] = findDistribution(pointnRange, x, y,
varPairSet, goDownward, threshold , showFigure )


% Warning >> varPairSet: only the last one is unparied. eg:
{'MinPeakDistance',0.5, 'MinPeakHeight'};
% toSmaller => true = descending; false = ascending;
% distX = nan;
% distY = nan;


% eg: findRange = -0.4:-0.01:-1.4;
findRange = pointnRange;
% disp(findRange);
meanNum = NaN(numel(findRange),1);
noPoints = false;
lenfindRange = numel(findRange);
```

```
i = 1;

while i <= lenfindRange && noPoints == false
    if goDownward%descending
        val = findRange(lenfindRange + 1 - i) .* -1;
    else
        val = findRange(i);
    end

    lc = findpeaks(y, x, varPairSet{:}, val);
%     disp(lc);
%     lc = findpeaks(findPeaksDataset);
    troughNum = numel(lc);
%     disp([lenFindM + 1 - i,minPkVal, round(minPkVal .* (-1)),
troughNum]);
%     disp(troughNum);
    meanNum(i) = troughNum;
    if (troughNum < threshold)
        noPoints = true;
    end
%     disp(findM(i));
    i = i + 1;
end

distX = findRange(1:numel(meanNum));
distY = meanNum;

if showFigure
   figure;plot(distX, distY, '-bo');
   title('From findDistribution');
end
```

```
return
```

---

findPeriodPeak.m

```
function [val, lc] = findPeriodPeak(x, y, startPoint, endPoint,
showFigure)

% startPoint = 1;
% endPoint = 55535;
% showFigure = true;
% nBP = bp(startPoint:endPoint);
% nX = x(startPoint:endPoint);

%% Break the BP to several parts
nBP = y(startPoint:endPoint);
nX = x(startPoint:endPoint);
% figure; plot(nX,nBP);

%% find peaks to get the max and min value
% [tmpPk, ~] = findpeaks(nBP, nX);
% maxVal = max(tmpPk);
maxVal = 2.5;
minVal = 0.6;
% minVal = min(tmpPk);
% disp([maxVal, minVal]);

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%
%
```

```
%-----------------------------Peak Distance-----------------------
-------%

%
%

% Get the best Min Peak Distance

[tmpX, tmpY] = findDistribution(0.4:0.01:1.5,nX, nBP, ...

                {'MinPeakDistance'}, false, 0, false);

minPeakDistance = getSetVal(tmpX, tmpY, 1, false);


% % TEST MIN PEAK DISTANCE

% [maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance);

% figure;

% plot(nX, nBP, maxLc, maxPk, 'r^');

% fprintf('Number of Peak: %d\n\n', numel(maxLc));


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%
%

%-----------------------------Peak Height-------------------------
-------%

%
%

% Get the best Min Peak Height

% clc; close all;

[tmpX, tmpY] = findDistribution(minVal:0.01:maxVal,nX, nBP, ...

                 {'MinPeakDistance',minPeakDistance,
'MinPeakHeight'}, false, 0, false);

minPeakHeight = getSetVal(tmpX, tmpY, 1, false);


% TEST MIN PEAK HEIGHT

% figure;

% [maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight', minPeakHeight);
```

```
% plot(nX, nBP, maxLc, maxPk, 'r^');

% title(sprintf('Number of Peak: %d', numel(maxLc)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% %%


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

%
%

%----------------------------Peak Prominence-----------------------
-------%

%
%

[tmpX, tmpY] = findDistribution(0.1:0.01:0.5,nX, nBP, ...

                                {'MinPeakDistance',minPeakDistance,
'MinPeakHeight', minPeakHeight, 'MinPeakProminence'}, false, 0,
false);

minPeakProminence = getSetVal(tmpX, tmpY, 30, false);

%
%

%% TEST MIN PEAK Prominence

% figure;

% [maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight',
minPeakHeight,'MinPeakProminence', minPeakProminence);

% plot(nX, nBP, maxLc, maxPk, 'r^');

% title(sprintf('Number of Peak: %d', numel(maxLc)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% %%

%%

[maxPk,maxLc] = findpeaks(nBP, nX,
'MinPeakDistance',minPeakDistance, 'MinPeakHeight', minPeakHeight,
'MinPeakProminence', minPeakProminence);%,
'MinPeakProminence',minPeakProminence);%,'Annotate','extents');

% figure;

% plot(nX, nBP, maxLc, maxPk, 'r^');
```

```
fprintf('Best Peak Distance = %f\n', minPeakDistance);
fprintf('Best Peak Height = %f\n', minPeakHeight);
fprintf('Best Peak Prominence = %f\n\n', minPeakProminence);


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%%-----------------------------------------------------------------------
------%%

%%---------+END OF FINDING PEAKS+-----------+END OF FINDING PEAKS+-----
------%%

%%-----------------------------------------------------------------------
------%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% %%


%%
if showFigure
    figure;
    plot(nX, nBP, 'Color', [0.4660 0.6740 0.1880]);
    title('From findPeroidPeak');
    hold on
    plot(maxLc, maxPk,'r^');
    hold off;
    title(sprintf("%d - %d",startPoint, endPoint));
end

val = maxPk;
lc = maxLc;

return
```

---

getCalibr4NAN.m

```
function nanCali = getCalibr4NAN(dict, percentage)
    % nanCali >>> get the ls containing all the period will set to
nan in
    %             spline ls.
    caliperiod = dict.caliPeriod;
    caliLen = numel(caliperiod);
    caliperiodDiff = nan(caliLen, 1);

    for c = 1: caliLen
        tmp = caliperiod{c};
        stLc = tmp(1);
        endLc = tmp(3);
        calidiff = abs(stLc - endLc);
        caliperiodDiff(c) = calidiff;

    end
    % get the percentage from different list
    threshold = prctile(caliperiodDiff, percentage);

    nanCali = caliperiod(caliperiodDiff >= threshold);

end
```

---

getLinearRegression.m

```
function [equSlope, equIntercept, lineSlope, lineIntercept] =
getLinearRegression(stIdx, endIdx, splinePk, isNormalVal)
    global halfWindow_sz xx fs xx_ne;
    showFigure = false;
```

```
    if ~isNormalVal
    eventLen = endIdx - stIdx;
    if eventLen > halfWindow_sz
        xaxis = eventLen * (-1/fs)-4/fs:1/fs:halfWindow_sz/fs -
1/fs; % >>> Event Window Size
        xaxis = round(xaxis, 2);
    else
        xaxis = round(xx, 2);
    end

    period = getTargetPeriod(stIdx, endIdx, splinePk);
    periodLen = numel(period);

    % The idx below is refer to the new length of the period
    stPtIdx = (periodLen - halfWindow_sz +1)  - eventLen;
%     stPtIdx = halfWindow_sz + 1 - eventLen;
    stPtLc  = xaxis(stPtIdx);
    stPtVal = mean(period(stPtIdx - 4: stPtIdx));

    [endPtLc, endPtVal, endPtIdx] = getPkVal;
    else
        xaxis = xx_ne;
        period = splinePk';
        stPtIdx = stIdx;
        endPtIdx = endIdx;
        stPtLc = xaxis(stPtIdx);
        stPtVal = period(stPtIdx);
        endPtLc = xaxis(endPtIdx);
        endPtVal = period(endPtIdx);
    end

    % Equation
```

```
    equSlope = (stPtVal - endPtVal)/(stPtLc - endPtLc);
    equIntercept = stPtVal - equSlope * stPtLc;

    % Linear Regression
    targetPeriod = period(stPtIdx: endPtIdx);
    polyxx = xaxis(stPtIdx: endPtIdx);

    a = polyfit(polyxx ,targetPeriod',1);
    lineSlope = a(1);
    lineIntercept = a(2);

    if showFigure
        figure;
        plot(xaxis, period);
        hold on;
        plot([xaxis(stPtIdx) ,xaxis(stPtIdx)], [0, 200], '-',
'LineWidth', 2);
        endEventIdx = find(xaxis == 0);
        plot([xaxis(endEventIdx),xaxis(endEventIdx)], [0, 200], '-
', 'LineWidth', 2);
        plot(xaxis(stPtIdx), period(stPtIdx),'ro', 'LineWidth', 2,
'HandleVisibility','off');
        plot(xaxis(endPtIdx), period(endPtIdx), 'go', 'LineWidth',
2, 'HandleVisibility','off');
        plot([xaxis(stPtIdx), xaxis(endPtIdx)],
[getY(xaxis(stPtIdx), equSlope, equIntercept) getY(xaxis(endPtIdx),
equSlope, equIntercept)]);
        plot([xaxis(stPtIdx), xaxis(endPtIdx)],
[getY(xaxis(stPtIdx), lineSlope, lineIntercept)
getY(xaxis(endPtIdx), lineSlope, lineIntercept)]);
        hold off;
        legend({'Blood Pressure', 'Event Start', 'Event End',
'Equation Result', 'Linear Regression Result'},
'Location','northwest');
        title(sprintf('Equation: a = %4.5f, b = %4.5f\nLinear: a =
%4.5f, b= %4.5f', equSlope, equIntercept, lineSlope,
lineIntercept));
```

```
        graphSetting;
    end


    function [lc, pk, xxidx] = getPkVal
%         figure;
%         plot(xaxis, period, '--r');

        x = xx(halfWindow_sz+1: end);
        targetLs = period(periodLen - halfWindow_sz+1: end);

%         hold on;
%         plot(x, targetLs, '-xk');
%         hold off;
        [pk, lc] = findpeaks(targetLs, x);

        idx = pk == max(pk);

        lc = round(lc(idx), 2);
        % pk = pk(idx);

        xxidx = find(xaxis == lc);

        % In the discussion with the Dr. Behbehani,
        % he wanted to take the mean of the 5 points which 2 points
before
        % and after with peak point. Total 5 points
        pk = mean(period(xxidx -2: xxidx+2));

%         if showFigure
%             figure;
%             plot(x, targetLs);
```

```
%             hold on;
%             plot(lc, pk,'go', 'LineWidth', 2);
% %             plot(lc(idx), pk(idx), 'go', 'LineWidth', 2);
%             centerSettig = {'Color','k',
'LineStyle',':','LineWidth', 1, 'HandleVisibility','off'};
%             plot([0 0], [min(pk) max(pk)], centerSettig{:} );
%             hold off;
%         end
    end


    function oneEvent = getTargetPeriod(eventStIdx, eventEndIdx,
targetLs)
        x_endPt = eventEndIdx + halfWindow_sz - 1;


        if eventLen > 3000
            x_stPt = eventStIdx - 4;
            oneEvent = nan(x_endPt - x_stPt + 1 ,1);
            firstHalfWindow = eventLen+4;
        else
            x_stPt  = eventEndIdx - halfWindow_sz;
            firstHalfWindow = halfWindow_sz;
            oneEvent = nan(2 * halfWindow_sz ,1);
        end
        oneEvent(1:firstHalfWindow) = targetLs(x_stPt:eventEndIdx-
1);
        oneEvent(firstHalfWindow + 1 : firstHalfWindow +
halfWindow_sz) = targetLs(eventEndIdx:x_endPt);


%         figure;
%         plot(xaxis, splinePk(stIdx-4: endIdx +  halfWindow_sz-1),
'--k');
%         hold on;
%         plot(xaxis, splinePk(x_stPt: x_endPt), '-xr');
%
```

```
%         plot(xaxis(1:firstHalfWindow),
oneEvent(1:firstHalfWindow), '^g');
%
%         disp((firstHalfWindow + 1) - (firstHalfWindow +
halfWindow_sz ));
%         disp(eventEndIdx - x_endPt);
%
%         plot(xaxis(firstHalfWindow + 1 : firstHalfWindow +
halfWindow_sz),...
%           oneEvent(firstHalfWindow + 1 : firstHalfWindow +
halfWindow_sz), '^g');
%         hold off;
    end


    function yVal = getY(xVal, slope, intercept)
        yVal = slope * xVal + intercept;
    end



end
```

---

getNonOverlapPeriod.m

```
function oneEvent = getNonOverlapPeriod(eventEndIdx, targetLs)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
    % return a list with total len equals to 2x halfWindow_sz
%
    % eventEndIdx at halfWindow_sz + 1
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
    global halfWindow_sz
    x_stPt  = eventEndIdx - halfWindow_sz;
    x_endPt = eventEndIdx + halfWindow_sz - 1;


    oneEvent = nan(halfWindow_sz * 2 ,1);
    stIdx = 1;
    oneEvent(stIdx:halfWindow_sz) = targetLs(x_stPt:eventEndIdx-1);
    oneEvent(halfWindow_sz + 1 :halfWindow_sz*2 - stIdx + 1) =
targetLs(eventEndIdx:x_endPt);


end
```

---

getPeakVal.m

```
function [lc, pk] = getPeakVal(ls)
    global xx;
    [pk, lc] = findpeaks(ls, xx);
%     figure;
%     plot(xx, ls);
%     hold on;
%     plot(lc, pk, '-rx', 'MarkerSize',20 );
    idx = pk == max(pk);
%     plot(lc(idx), pk(idx), 'go', 'LineWidth', 2);
%     centerSettig = getCenterLineSetting;
%     plot([0 0], [min(pk) max(pk)], centerSettig{:} );
%     hold off;
    lc = lc(idx);
    pk = pk(idx);
end
```

---

getSetVal.m

```
function val = getSetVal(x, y, slopeLimit, showFigure)
    % Calculate the slope and the difference between each slope

    slopeLs = NaN(numel(y),1);
    slopeCal = NaN(numel(y),1);
    slopeCalX = NaN(numel(y),1);

    stopCollect = false;
    startCollect = false;

    for i = 2: numel(y)
        slopeLs(i) = (y(i) - y(i-1));
    end

    % figure;plot(slopeLs);
    % title('Slope Figure');

    i = 1;
    countZero = 1;
    % indexCount = 1;
    while i <= numel(slopeLs) && stopCollect == false
        slopeVal = slopeLs(i);
    %     disp([slopeVal, countZero, i]);
        if (slopeVal == 0 || isnan(slopeVal) )  && countZero == i
            startAtZero = true;
            countZero = countZero + 1;
    %         fprintf('start at zero No: %02d\n\n',countZero);
        else
            startAtZero = false;
```

```
        end

        if startAtZero == false
        %check slope value if  it is between +-50 collect; >>> Need
more improvement
            if abs(slopeVal)<= slopeLimit %&& indexCount <= 7

                slopeCal(i) = x(i);
                slopeCalX(i) = 1;
    %             slopeCal(indexCount) = x(i);
    %             indexCount = indexCount +1;
                startCollect = true;
            elseif startCollect == true
                stopCollect = true;
            end

        end

        i = i + 1;
    end



    % calculate the average

    %%%%%%%% PLOT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if showFigure

        figure;
        plot(x, y, '-go');

        title('Combination result From getSetVal');
        yyaxis left;
```

```
        hold on;
        plot(x, slopeLs,'-k^', slopeCal,slopeCalX , 'rx');
        yyaxis right;
        legend('Distribution Curve','Slope Curve', 'Slope count 
point');
        hold off;

        figure;
        plot(x, slopeLs, '-o');
        title('Sloep Ls From getSetVal');
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

    slopeCal = slopeCal(~isnan(slopeCal));

    val = mean(slopeCal);

    if showFigure
    %     disp(slopeCal);
        disp({'Best Setting Value :',val});
    end



end
```

---

getSpline.m

```
function [splineLs, splinePk_stPt, splinePk_endPt] =
getSpline(targetLc, targetVal, sampleXX)
    % targetLc and targetVal should be the from the peaks lc and
val


    % Get the spline value from the peak value
    splineLs = spline(targetLc, targetVal, sampleXX);
    splinePk_stPt = find(sampleXX <= targetLc(1), 1, 'last');
    splineLs(1: splinePk_stPt-1) = nan;


    splinePk_endPt = find(sampleXX >= targetLc(end), 1, 'first');
    splineLs(splinePk_endPt+1: end) = nan;


end
```

---

getStageName.m

```
function targetStageName = getStageName(fieldKey)
    switch fieldKey
        case "sn5"
            targetStageName = "REM";
        case "sn4"
            targetStageName = "Stage 4";
        case "sn3"
            targetStageName = "Stage 3";
        case "sn2"
            targetStageName = "Stage 2";
        case "sn1"
            targetStageName = "Stage 1";
        case "s0"
            targetStageName = "Stage 0";
        case "s1"
```

```
            targetStageName = "Awake";
        case "all"
            targetStageName = "All Events in all stage";
    end
end
```

---

getWholeNightPlot.m

```
function getWholeNightPlot(startPoint, endPoint, dict, event,
stage, splinePk, crxStageEventLs, event1)
global folderNum;
% % The Whole night plot
figure;
if isnan(startPoint )
    startPoint =1;
end
if isnan(endPoint)
    endPoint =  numel(splinePk);
end



coeff = 100;


% endPoint =  500000;
% spStartPoint = find(dict.x(startPoint) <= targetLc, 1, 'first');
% spEndPoint = find(dict.x(endPoint) >= targetLc, 1, 'last');
% trStartPoint = find(dict.x(startPoint) <= dict.troughLc, 1,
'first');
% trEndPoint = find(dict.x(endPoint) >= dict.troughLc, 1, 'last');
plot(dict.x(startPoint: endPoint), dict.y(startPoint:
endPoint).*coeff);
```

```
hold on;

% plot(dict.troughLc(trStartPoint: trEndPoint),
dict.troughVal(trStartPoint: trEndPoint),'-', 'Color',
colorConvertor('#158C00'));

plot(dict.x(startPoint: endPoint),event(startPoint: endPoint),
'LineWidth',2, 'Color', colorConvertor('#D7BD00') );


% tmpStageIdx = find(stage == -1);

% plot(dict.x(tmpStageIdx),stage(tmpStageIdx), 'x', 'LineWidth',2);

plot(dict.x(startPoint: endPoint),stage(startPoint: endPoint) ,
'Color', colorConvertor('#009900') );




plot(dict.x(stage == 0), stage(stage == 0), 'xr', 'LineWidth',2);




splinePk_endPt = endPoint;%numel(splinePk);


splinePk_stPt = 1;

plot(dict.x(splinePk_stPt:splinePk_endPt),
splinePk(splinePk_stPt:splinePk_endPt), 'k','LineWidth',1);

% plot(dict.x(tmp), dict.y(tmp), 'o',
'LineWidth',2,'MarkerEdgeColor','k', 'MarkerFaceColor',[.49
1 .63]);


crxStageEventLs = crxStageEventLs( ~isnan(crxStageEventLs));

if ~isnan(crxStageEventLs)

    for i = 1: numel(crxStageEventLs)

        e = event1{crxStageEventLs(i)};

        plot([dict.x(e(1)) dict.x(e(1))], [0, 200], '<',
'LineWidth',2,'MarkerEdgeColor','k', 'MarkerFaceColor',[.49
1 .63]);

        plot([dict.x(e(2)) dict.x(e(2))], [0, 200], '>',
'LineWidth',2,'MarkerEdgeColor','k', 'MarkerFaceColor',[.49
1 .63]);
```

```
    end
end


hold off;
title(sprintf("Subject: %s,   Total Event: %d", folderNum,
numel(event1)));
box on;
set(gcf,'color','w');
labelSetting = {'FontSize',12,'FontWeight','bold'};
xlabel('Time (s)', labelSetting{:}) ;
ylabel('Blood Pressure (mmHg)', labelSetting{:});



end
```

---

graphSetting.m

```
function graphSetting
    global status
    box on;
    set(gcf,'color','w');
    xlabel('Time (s)') ;
%     ylabel('Count');
    ylabel('Blood Pressure (mmHg)');
    if strcmp(status, "SBP")
        ylim([80 200]);

    elseif strcmp(status, "DBP")
        ylim([40 120]);
```

```
    elseif strcmp(status, "MAP")
        ylim([40 160]);

    elseif strcmp(status, "PP")
        ylim([20 120]);

    end
end
```

---

init.m

```
function [dict, splinePk, eventLs, stageLs] = init(rootFolder)
    global folderNum fs;
    folderPath = sprintf("%s\\SLEEP DATA 2013-2015
(Ann.)\\%s",rootFolder, folderNum);
    fileName = sprintf("%s_DAQ_resampled", folderNum);
    filePath = sprintf('%s\\%s_dict.mat', folderPath, fileName);
    m = load(filePath);
    dict = m.dict;

    % assign which data should be use >>> SBP, DBP, MAP or PP -----
--------
    [targetVal, targetLc] = getStatus(dict);
%-
    %-------------------------------------------------------------
--------

    % get the blood pressure waveform
    m = load(sprintf('%s\\%s.mat', folderPath, fileName));
    % startVal = m.DAQ_rsmpl(:,1)(1);
    [len, ~] = size(m.DAQ_rsmpl);
    x = 0.00: (1/fs):(len/fs);
    dict.x = round(x(1:numel(x)-1), 4);
```

```
    dict.y = m.DAQ_rsmpl(:,3);


    [splinePk, splineStIdx, splineEndIdx] = getSpline(targetLc,
targetVal, dict.x);


    % get the event and stage
    m = load(sprintf("%s\\%s_stage_event.mat", folderPath,
folderNum));


    eventLs = m.EVENT;
    stageLs = m.STAGE;
    eventLs = round(eventLs);
    stageLs = round(stageLs);


    eventLs = eventLs(1:numel(dict.x));
    stageLs = stageLs(1:numel(dict.x));


    % if the spline start later and end earlier, the event and
state should
    % not have the value, thus set it to nan
    eventLs(1: splineStIdx-1) = nan;
    eventLs(splineEndIdx+1: end) = nan;


    stageLs(1: splineStIdx-1) = nan;
    stageLs(splineEndIdx+1: end) = nan;


    % Get Calibration length
    nanCalibr = getCalibr4NAN(dict, 99);
    splinePk = setNANtoSplineLs(nanCalibr, dict, splinePk);


    printLog('Message',sprintf('Subject: %s >>> init done',
folderNum));


end
```

initEventslopeStruc.m

```
function slopeStru = initEventslopeStruc(isStage, isNormalVal,
normalLs)
    global fieldLs status halfWindow_sz;

    slopeStru = struct('sub01', [], 'sub03', [], 'sub05', [], ...
                       'sub06', [], 'sub07', [], 'sub10', [], ...
                       'sub11', [], 'sub12', [], 'sub13', [], ...
                       'sub15', []);
    if isStage
        stageStru = struct();
        for fieldIdx = 1: numel(fieldLs)-1
            fieldName = fieldLs(fieldIdx);
            stageStru.(fieldName) = nan(300,4);
        end

        fields = fieldnames(slopeStru);
        for fidx = 1: numel(fields)
            slopeStru.(fields{fidx}) = stageStru;
        end

    else
        if isNormalVal
%             folderNumLs = normalLs(1,:);
            for i = 1:size(normalLs,2)
                folderNum = normalLs{1,i};
                subFolderName = sprintf('sub%s',folderNum{1});
                splineLs = normalLs{2,i}.(status);
                [equSlope, equIntercept, lineSlope, lineIntercept]
= getLinearRegression(1, 2 * halfWindow_sz, splineLs, true);
```

```
                slopeStru.(subFolderName)(1, :) = [equSlope,
equIntercept, lineSlope, lineIntercept];
            end
        else

            fields = fieldnames(slopeStru);
            for fidx = 1: numel(fields)
                slopeStru.(fields{fidx}) = nan(300,4);
            end
        end
    end
end
```

---

initNonEvent.m

```
function nonEvent = initNonEvent(rootDir)
    global halfWindow_len;
    nonEvent = load(sprintf( '%s\\SLEEP DATA 2013-2015
(Ann.)\\nonEvent_windowsz_%d.mat', rootDir, halfWindow_len * 2) );
    nonEvent = nonEvent.noneEvent;
end
```

---

inList.m

```
function result = inList(ls, value)
    % it is only available for 1 x n or n x 1
    result = false;
    lsLen = numel(ls);
    idx = 1;
    while result ~= true && idx <= lsLen
```

```
        if ls(idx) == value
            result = true;
        end
        idx = idx + 1;

    end

end
```

---

printLog.m

```
function printLog(type, input)
    msg = sprintf('%d/%d/%d %02d:%02d:%02.f [%s] %s \r',clock,
type, input);

    logPath = 'D:\\Dream\\UT Arlington\\Thesis\\data\\log.txt';
    fid = fopen(logPath,'a+');
    fprintf(fid, msg);
    fclose(fid);

    fprintf(msg);
end
```

---

removeNoisy.m

```
function splinePk = removeNoisy(folderNum, dict, splinePk)
    if strcmp(folderNum, '01')
        % 5247.5 1.41509951768494
        % 5307.51     1.16048637552839
```

```
        removePt(5247.5, 5307.51);

        % 17561.11    1.27284830466048
        % 18402.33    1.02926285577489
        removePt(17561.11, 18402.33);

    elseif strcmp(folderNum, '02')
        %     6601.13 1.19122935866885
        %     6925.58 1.1177336541943
        removePt(6601.13, 6925.58);

%         8447.52     1.17919077389192
%         8658.25     1.04413370082015
        removePt(8447.52, 8658.25);

        % 15070.71    1.01207758238626
        % 15150.52    1.41885783551752
        removePt(15070.71, 15150.52);

        splinePk(splinePk > 1.8) = nan;
        splinePk(splinePk < 0.50) = nan;

    elseif strcmp(folderNum, '03')
        % 4760.17     1.2867703359585
        % 4774.43     1.2877982883856
        removePt(4760.17, 4774.43);

    elseif strcmp(folderNum, '05')
        %     4669.54 1.30632813352569
        %     4752.35 1.27895726888601
        removePt(4669.54, 4752.35);
```

```
        % 8034.57     1.52207256334442
        % 8376.9 1.60877143966121
        removePt(8034.57, 8376.9);

        % 9061.6 1.55450511481939
        % 9090.49     1.94327999651919
        removePt(9061.6, 9090.49);

        % 9379.24     1.74926777561938
        % 10083.46    1.31865155032019
        removePt(9379.24, 10083.46);

        % 1238.92     1.05518573426437
        % 1799.15     1.0607154054546
        removePt(1238.92, 1799.15);

        % 1815.15     0.985600865041068
        % 1835.01     1.59142443673898
        removePt(1815.15, 1835.01);

        % 969.32 1.34536304629181
        % 1003.61     1.13553188507893
        removePt(969.32, 1003.61);

        % 11788.93    1.16419944774793
        % 11935.7     1.49600265622529
        removePt(11788.93, 11935.7);

%         splinePk(splinePk < 0.8) = nan;

        endIdx = find(dict.x <= 13346.19, 1, 'last');
        % 13346.19    1.19867314777888 last peak points
```

```
        splinePk(endIdx+1: end) = nan;
        %     event = event(1:numel(dict.x));
        %     stage = stage(1:numel(dict.x));
        %
        %     % Re collect the event
        %     stageLs = getStageList(stage);
        %     eventLs = getEventStageList(event);

    elseif strcmp(folderNum, '06')
%         removePt(4669.54, 4752.35);


    elseif strcmp(folderNum, '09')
% 12531.44 1.52655938575631
% 13305.21 1.40350326478065
        removePt(12531.44, 13305.21);

    elseif strcmp(folderNum, '10')
        %     8732.14 1.7272935987688
        %     10054.89     1.22412195640995
        removePt(8732.14, 10054.89);

    elseif strcmp(folderNum, '11')
% 8777.41  1.48944750058368
% 8794.28  1.21625136313327
        removePt(8777.41, 8794.28);

    elseif strcmp(folderNum, '12')
%         820.7  1.57661972573783
%         842.46 1.46571628009148
%
%         1054.7 1.3232515365196
```

```
%         1156.29     1.40960336601451
%
%         1875.54     1.4711422954288
%         2204.55     1.49741926674487
%
%         4305.92     1.6607204338789
%         4986.55     1.33760204513593

        endIdx = find(dict.x >= 52.67, 1, 'first');
        % 52.67  1.34141491678299 First stable peak val
        splinePk(1:endIdx-1) = nan;

    elseif strcmp(folderNum, '15')
%         559.04 1.24780534206627
%         914.61 1.1947090300395
        removePt(559.04, 914.61);

%         9368.82     1.54339149464206
%         9643.96     1.04145764667209
        removePt(9368.82, 9643.96);

    end

    function removePt(stPt, endPt)
        sIdx = find(dict.x >= stPt, 1, 'first');
        eIdx = find(dict.x <= endPt, 1, 'last');
        %     8732.14 1.7272935987688
        %     10054.89     1.22412195640995
        splinePk(sIdx+1: eIdx-1) = nan;
    end

end
```

removeNoisyDSP.m

```
function splinePk = removeNoisyDSP(folderNum, dict, splinePk)
    if strcmp(folderNum, '01')
        % 5247.5 1.41509951768494
        % 5307.51     1.16048637552839
        removePt(5247.5, 5307.51);

        % 17561.11    1.27284830466048
        % 18402.33    1.02926285577489
        removePt(17561.11, 18402.33);

    elseif strcmp(folderNum, '02')
        %     6601.13 1.19122935866885
        %     6925.58 1.1177336541943
        removePt(6601.13, 6925.58);

%         8447.52     1.17919077389192
%         8658.25     1.04413370082015
        removePt(8447.52, 8658.25);

        % 15070.71    1.01207758238626
        % 15150.52    1.41885783551752
        removePt(15070.71, 15150.52);

        splinePk(splinePk > 1.8) = nan;
        splinePk(splinePk < 0.50) = nan;

    elseif strcmp(folderNum, '03')
        % 4760.17     1.2867703359585
```

```
    % 4774.43     1.2877982883856
    removePt(4760.17, 4774.43);

elseif strcmp(folderNum, '05')

    % 965.310.766292294264087
    % 1003.44     0.723228708741105
    %
    % 1236.31     0.657815557760036
    % 1935.95     0.634068101840189
    %
    % 4668.66     0.761099297772447
    % 4753.01     0.734065278371569
    %
    % 7669.98     0.901611413113894
    % 7722.27     1.17401599752182
    %
    % 8007   0.973152439182646
    % 8377.56     1.01242870393025

    % 9379.06     1.03299584519092
    % 10085.9     0.932435906862825
    %
    % 11787.32    0.78285671887977
    % 11936.35    1.01503087148992

    removePt(965.31,1003.44);
    removePt(1236.31,1935.95);
    removePt(4668.66,4753.01);
    removePt(7669.98,7722.27);
    removePt(8007, 8377.56);
    removePt(9379.06,10085.9);
```

```
        removePt(11787.32,11936.35);

%         eIdx = find(dict.x <= 13346.19, 1, 'last');
%         % 13346.19  1.19867314777888 last peak points
%         splinePk(eIdx+1: end) = nan;
        %     event = event(1:numel(dict.x));
        %     stage = stage(1:numel(dict.x));
        %
        %     % Re collect the event
        %     stageLs = getStageList(stage);
        %     eventLs = getEventStageList(event);

    elseif strcmp(folderNum, '09')
% 12531.44 1.52655938575631
% 13305.21 1.40350326478065
        removePt(12531.44, 13305.21);

    elseif strcmp(folderNum, '10')
        %     8732.14 1.7272935987688
        %     10054.89     1.22412195640995
        removePt(8732.14, 10054.89);

    elseif strcmp(folderNum, '11')
% 8777.41  1.48944750058368
% 8794.28  1.21625136313327
        removePt(8777.41, 8794.28);

    elseif strcmp(folderNum, '12')
%         820.7  1.57661972573783
%         842.46 1.46571628009148
%
%         1054.7 1.3232515365196
```

```
%         1156.29     1.40960336601451
%
%         1875.54     1.4711422954288
%         2204.55     1.49741926674487
%
%         4305.92     1.6607204338789
%         4986.55     1.33760204513593

        eIdx = find(dict.x >= 52.67, 1, 'first');
        % 52.67  1.34141491678299 First stable peak val
        splinePk(1:eIdx-1) = nan;

    elseif strcmp(folderNum, '15')
        removePt(7280.91, 7381.02);

%         559.04 1.24780534206627
%         914.61 1.1947090300395
        removePt(559.04, 914.61);

%         9368.82     1.54339149464206
%         9643.96     1.04145764667209
        removePt(9368.82, 9643.96);

        removePt(19416.36, 19513.55);

    end

    function removePt(stPt, endPt)
        sIdx = find(dict.x >= stPt, 1, 'first');
        eIdx = find(dict.x <= endPt, 1, 'last');
        %     8732.14 1.7272935987688
        %     10054.89     1.22412195640995
```

```
        splinePk(sIdx+1: eIdx-1) = nan;
    end

end
```

---

removePoints.m

```
function dict = removePoints(folderNum, dict)
    count = 0;
    if strcmp(folderNum, '01')


    elseif strcmp(folderNum, '02')


    elseif strcmp(folderNum, '03')


    elseif strcmp(folderNum, '05')
        removePt(1221.51, 0.00551659358452486);
        removePt(1230.38, 0.19681970465736);
        removePt(1231.1, 0.538771171703811);
        removePt(1238.72, 0.485388615173904);
        removePt(7831.25, 0.69595904466782);
        removePt(8639.2, 0.807724780487715);
        removePt(8728.6, 0.769758830270064);
        removePt(9138.19, 1.17524063069134);
        removePt(9206.37, 0.890861418353335);
        removePt(10411.03, 0.335682380367277);
        removePt(10801, 0.475844521480214);
        removePt(11747.38, 0.414300885830584);
```

```
    removePt(12601, 0.442744885712318);

elseif strcmp(folderNum, '07')
    removePt(13701.54, 0.79700221048562);
    removePt(13721.28, 0.811586213393857);

elseif strcmp(folderNum, '09')


elseif strcmp(folderNum, '10')


elseif strcmp(folderNum, '11')


elseif strcmp(folderNum, '12')


elseif strcmp(folderNum, '15')

end
fprintf("Successful delete # points >>> %d\r", count);

function removePt(lc, val)
    troughLc = round(dict.troughLc, 2);
    troughVal = round(dict.troughVal, 5);

    val = round(val, 5);

    idx = find(troughLc == lc);

    if numel(idx) == 1 && troughVal(idx) == val
```

```
            dict.troughLc(idx) = nan;
            dict.troughVal(idx) = nan;
            fprintf("Delete Subject: %s (%f, %f)\r", folderNum, lc,
val);
            count = count + 1;
        else
            fprintf("Fail to delete Subject: %s (%f,
%f)\r",folderNum,  lc, val);
        end
    end

end
```

---

setNANtoSplineLs.m

```
function splineLs = setNANtoSplineLs(nanCali, dict, splineLs)
    % from getCalibr4NAN to apply to splineLs
    % set those period in splineLs to nan

    for caliIdx = 1:numel(nanCali)
        tmp = nanCali{caliIdx};
        stLc = round(tmp(1), 4);
        endLc = round(tmp(3), 4);
        stIdx = find(dict.x == stLc);
        endIdx = find(dict.x == endLc);
        splineLs(stIdx:endIdx) = nan;
    end
end
```

---

swtest.m

```
function [H, pValue, W] = swtest(x, alpha)

%SWTEST Shapiro-Wilk parametric hypothesis test of composite
normality.

%   [H, pValue, SWstatistic] = SWTEST(X, ALPHA) performs the

%   Shapiro-Wilk test to determine if the null hypothesis of

%   composite normality is a reasonable assumption regarding the

%   population distribution of a random sample X. The desired
significance

%   level, ALPHA, is an optional scalar input (default = 0.05).

%

%   The Shapiro-Wilk and Shapiro-Francia null hypothesis is:

%   "X is normal with unspecified mean and variance."

%

%   This is an omnibus test, and is generally considered relatively

%   powerful against a variety of alternatives.

%   Shapiro-Wilk test is better than the Shapiro-Francia test for

%   Platykurtic sample. Conversely, Shapiro-Francia test is better
than the

%   Shapiro-Wilk test for Leptokurtic samples.

%

%   When the series 'X' is Leptokurtic, SWTEST performs the
Shapiro-Francia

%   test, else (series 'X' is Platykurtic) SWTEST performs the

%   Shapiro-Wilk test.

%

%    [H, pValue, SWstatistic] = SWTEST(X, ALPHA)

%

% Inputs:

%   X - a vector of deviates from an unknown distribution. The
observation

%     number must exceed 3 and less than 5000.

%

% Optional inputs:

%   ALPHA - The significance level for the test (default = 0.05).
```

```
%
% Outputs:
%  SWstatistic - The test statistic (non normalized).
%
%   pValue - is the p-value, or the probability of observing the
given
%     result by chance given that the null hypothesis is true.
Small values
%     of pValue cast doubt on the validity of the null hypothesis.
%
%     H = 0 => Do not reject the null hypothesis at significance
level ALPHA.
%     H = 1 => Reject the null hypothesis at significance level
ALPHA.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%                  Copyright (c) 17 March 2009 by Ahmed Ben Saïda
%
%                   Department of Finance, IHEC Sousse - Tunisia
%
%                         Email: ahmedbensaida@yahoo.com
%
%                      $ Revision 3.0 $ Date: 18 Juin 2014 $
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%
% References:
%
% - Royston P. "Remark AS R94", Applied Statistics (1995), Vol. 44,
%   No. 4, pp. 547-551.
%   AS R94 -- calculates Shapiro-Wilk normality test and P-value
%   for sample sizes 3 <= n <= 5000. Handles censored or uncensored
data.
%   Corrects AS 181, which was found to be inaccurate for n > 50.
```

```
%   Subroutine can be found at: http://lib.stat.cmu.edu/apstat/R94
%
% - Royston P. "A pocket-calculator algorithm for the Shapiro-
Francia test
%   for non-normality: An application to medicine", Statistics in
Medecine
%   (1993a), Vol. 12, pp. 181-184.
%
% - Royston P. "A Toolkit for Testing Non-Normality in Complete and
%   Censored Samples", Journal of the Royal Statistical Society
Series D
%   (1993b), Vol. 42, No. 1, pp. 37-43.
%
% - Royston P. "Approximating the Shapiro-Wilk W-test for non-
normality",
%   Statistics and Computing (1992), Vol. 2, pp. 117-119.
%
% - Royston P. "An Extension of Shapiro and Wilk's W Test for
Normality
%   to Large Samples", Journal of the Royal Statistical Society
Series C
%   (1982a), Vol. 31, No. 2, pp. 115-124.
%
%
% Ensure the sample data is a VECTOR.
%
if numel(x) == length(x)
    x  =  x(:);               % Ensure a column vector.
else
    error(' Input sample ''X'' must be a vector.');
end
%
% Remove missing observations indicated by NaN's and check sample
size.
%
```

```
x  =  x(~isnan(x));
if length(x) < 3
   error(' Sample vector ''X'' must have at least 3 valid
observations.');
end
if length(x) > 5000
    warning('Shapiro-Wilk test might be inaccurate due to large
sample size ( > 5000).');
end
%
% Ensure the significance level, ALPHA, is a
% scalar, and set default if necessary.
%
if (nargin >= 2) && ~isempty(alpha)
   if ~isscalar(alpha)
      error(' Significance level ''Alpha'' must be a scalar.');
   end
   if (alpha <= 0 || alpha >= 1)
      error(' Significance level ''Alpha'' must be between 0 and
1.');
   end
else
   alpha  =  0.05;
end
% First, calculate the a's for weights as a function of the m's
% See Royston (1992, p. 117) and Royston (1993b, p. 38) for details
% in the approximation.
x       =   sort(x); % Sort the vector X in ascending order.
n       =   length(x);
mtilde  =   norminv(((1:n)' - 3/8) / (n + 1/4));
weights =   zeros(n,1); % Preallocate the weights.
if kurtosis(x) > 3
```

```
    % The Shapiro-Francia test is better for leptokurtic samples.


    weights =   1/sqrt(mtilde'*mtilde) * mtilde;
    %
    % The Shapiro-Francia statistic W' is calculated to avoid
excessive
    % rounding errors for W' close to 1 (a potential problem in
very
    % large samples).
    %
    W   =   (weights' * x)^2 / ((x - mean(x))' * (x - mean(x)));
    % Royston (1993a, p. 183):
    nu      =   log(n);
    u1      =   log(nu) - nu;
    u2      =   log(nu) + 2/nu;
    mu      =   -1.2725 + (1.0521 * u1);
    sigma   =   1.0308 - (0.26758 * u2);
    newSFstatistic  =   log(1 - W);
    %
    % Compute the normalized Shapiro-Francia statistic and its p-
value.
    %
    NormalSFstatistic =   (newSFstatistic - mu) / sigma;


    % Computes the p-value, Royston (1993a, p. 183).
    pValue   =   1 - normcdf(NormalSFstatistic, 0, 1);


else


    % The Shapiro-Wilk test is better for platykurtic samples.
    c    =   1/sqrt(mtilde'*mtilde) * mtilde;
    u    =   1/sqrt(n);
    % Royston (1992, p. 117) and Royston (1993b, p. 38):
```

```
    PolyCoef_1   =   [-2.706056 , 4.434685 , -2.071190 , -
0.147981 , 0.221157 , c(n)];
    PolyCoef_2   =   [-3.582633 , 5.682633 , -1.752461 , -
0.293762 , 0.042981 , c(n-1)];
    % Royston (1992, p. 118) and Royston (1993b, p. 40, Table 1)
    PolyCoef_3   =   [-0.0006714 , 0.0250540 , -0.39978 , 0.54400];
    PolyCoef_4   =   [-0.0020322 , 0.0627670 , -0.77857 , 1.38220];
    PolyCoef_5   =   [0.00389150 , -0.083751 , -0.31082 , -1.5861];
    PolyCoef_6   =   [0.00303020 , -0.082676 , -0.48030];
    PolyCoef_7   =   [0.459 , -2.273];
    weights(n)   =   polyval(PolyCoef_1 , u);
    weights(1)   =   -weights(n);


    if n > 5
        weights(n-1) =   polyval(PolyCoef_2 , u);
        weights(2)   =   -weights(n-1);


        count  =   3;
        phi    =   (mtilde'*mtilde - 2 * mtilde(n)^2 - 2 *
mtilde(n-1)^2) / ...
                (1 - 2 * weights(n)^2 - 2 * weights(n-1)^2);
    else
        count  =   2;
        phi    =   (mtilde'*mtilde - 2 * mtilde(n)^2) / ...
                (1 - 2 * weights(n)^2);
    end


    % Special attention when n = 3 (this is a special case).
    if n == 3
        % Royston (1992, p. 117)
        weights(1)  =   1/sqrt(2);
        weights(n)  =   -weights(1);
        phi = 1;
```

```
    end
    %
    % The vector 'WEIGHTS' obtained next corresponds to the same
coefficients
    % listed by Shapiro-Wilk in their original test for small
samples.
    %
    weights(count : n-count+1)  =  mtilde(count : n-count+1) /
sqrt(phi);
    %
    % The Shapiro-Wilk statistic W is calculated to avoid excessive
rounding
    % errors for W close to 1 (a potential problem in very large
samples).
    %
    W   =   (weights' * x) ^2 / ((x - mean(x))' * (x - mean(x)));
    %
    % Calculate the normalized W and its significance level (exact
for
    % n = 3). Royston (1992, p. 118) and Royston (1993b, p. 40,
Table 1).
    %
    newn    =   log(n);
    if (n >= 4) && (n <= 11)

        mu      =   polyval(PolyCoef_3 , n);
        sigma   =   exp(polyval(PolyCoef_4 , n));
        gam     =   polyval(PolyCoef_7 , n);

        newSWstatistic  =   -log(gam-log(1-W));

    elseif n > 11

        mu      =   polyval(PolyCoef_5 , newn);
        sigma   =   exp(polyval(PolyCoef_6 , newn));
```

```
        newSWstatistic  =   log(1 - W);


    elseif n == 3
        mu      =   0;
        sigma   =   1;
        newSWstatistic  =   0;
    end
    %
    % Compute the normalized Shapiro-Wilk statistic and its p-
value.
    %
    NormalSWstatistic   =   (newSWstatistic - mu) / sigma;


    % NormalSWstatistic is referred to the upper tail of N(0,1),
    % Royston (1992, p. 119).
    pValue       =   1 - normcdf(NormalSWstatistic, 0, 1);


    % Special attention when n = 3 (this is a special case).
    if n == 3
        pValue  =   6/pi * (asin(sqrt(W)) - asin(sqrt(3/4)));
        % Royston (1982a, p. 121)
    end


end
%
% To maintain consistency with existing Statistics Toolbox
hypothesis
% tests, returning 'H = 0' implies that we 'Do not reject the null
% hypothesis at the significance level of alpha' and 'H = 1'
implies
% that we 'Reject the null hypothesis at significance level of
alpha.'
%
```

```
H  = (alpha >= pValue);
```

REFERENCES

[1] D. J. Eckert and A. Malhotra, "Pathophysiology of adult obstructive sleep apnea," *Proceedings of the American thoracic society,* vol. 5, no. 2, pp. 144-153, 2008.

[2] P. E. Peppard, T. Young, J. H. Barnet, M. Palta, E. W. Hagen and K. M. Hla, "Increased prevalence of sleep-disordered breathing in adults," *American journal of epidemiology,* vol. 177, no. 9, pp. 1006-1014, 2013.

[3] T. I. Morgenthaler, V. Kagramanov, V. Hanak and P. A. Decker, "Complex sleep apnea syndrome: is it a unique clinical syndrome?," *Sleep,* vol. 29, no. 9, pp. 1203-1209, 2006.

[4] D. P. White, "Pathogenesis of obstructive and central sleep apnea," *American journal of respiratory and critical care medicine,* vol. 172, no. 11, pp. 1363-1370, 2005.

[5] X. Yang, Y. Xiao, B. Han, K. Lin, X. Niu and X. Chen, "Implication of mixed sleep apnea events in adult patients with obstructive sleep apnea-hypopnea syndrome," *Sleep and Breathing,* vol. 23, no. 2, pp. 559-565, 2019.

[6] W. McNicholas, M. Bonsignore and . Management Committee of EU Cost Action B26, "Sleep apnoea as an independent risk factor for cardiovascular disease: current evidence, basic mechanisms and research priorities," *European Respiratory Journal,* vol. 29, no. 1, pp. 156-178, 2007.

[7] F. J. Nieto, P. E. Peppard and T. B. Young, "Sleep disordered breathing and metabolic syndrome," *WMJ: official publication of the State Medical Society of Wisconsin,* vol. 108, no. 5, p. 263, 2009.

[8] S. Tregear, J. Reston, K. Schoelles and B. Phillips, "Obstructive sleep apnea and risk of motor vehicle crash: systematic review and meta-analysis," *Journal of clinical sleep medicine,* vol. 5, no. 06, pp. 573-581, 2009.

[9] K. Sutherland and P. A. Cistulli, "Recent advances in obstructive sleep apnea pathophysiology and treatment," *Sleep and Biological Rhythms,* vol. 13, no. 1, pp. 26-40, 2015.

[10] J. M. Marin, S. J. Carrizo, E. Vicente and A. G. Agusti, "Long-term cardiovascular outcomes in men with obstructive sleep apnoea-hypopnoea with or without treatment with continuous positive airway pressure: an observational study," *The Lancet,* vol. 365, no. 9464, pp. 1046-1053, 2005.

[11] W. R. Ruehland, P. D. Rochford, F. J. O'Donoghue, R. J. Pierce, P. Singh and A. T. Thornton, "The new AASM criteria for scoring hypopneas: impact on the apnea hypopnea index," *Sleep,* vol. 32, no. 2, pp. 150-157, 2009.

[12] T. Young, P. E. Peppard and D. J. Gottlieb, "Epidemiology of obstructive sleep apnea: a population health perspective," *American journal of respiratory and critical care medicine,* vol. 165, no. 9, pp. 1217-1239, 2002.

[13] T. Gharibeh and R. Mehra, "Obstructive sleep apnea syndrome: natural history, diagnosis, and emerging treatment options," *Nature and science of sleep,* vol. 2, no. -1, p. 233, 2010.

[14] T. E. Weaver, G. Maislin, D. F. Dinges, T. Bloxham, C. F. George, H. Greenberg, G. Kader, M. Mahowald, J. Younger and A. I. Pack, "Relationship between hours of CPAP use and achieving normal levels of sleepiness and daily functioning," *Sleep,* vol. 30, no. 6, pp. 711-719, 2007.

[15] E. Chung, G. Chen, B. Alexander and M. Cannesson, "Non-invasive continuous blood pressure monitoring: a review of current applications," *Frontiers of medicine,* vol. 7, no. 1, pp. 91-101, 2013.

[16] J. S. Shahoud and N. R. Aeddula, "Physiology, Arterial Pressure Regulation," Vols. -1, FL, StatPearls, 2019.

[17] M. Gurven, A. D. Blackwell, D. E. Rodríguez, J. Stieglitz and H. Kaplan, "Does blood pressure inevitably rise with age? Longitudinal evidence among forager-horticulturalists," *Hypertension,* vol. 60, no. 1, pp. 25-33, 2012.

[18] C. Armstrong, "High blood pressure: ACC/AHA releases updated guideline," *American family physician,* vol. 97, no. 6, pp. 413-415, 2018.

[19] R. Kanagala, N. S. Murali, P. A. Friedman, N. M. Ammash, B. J. Gersh, K. V. Ballman, A. S. M. Shamsuzzaman and V. K. Somers, "Obstructive sleep apnea and the recurrence of atrial fibrillation," *Circulation,* vol. 107, no. 20, pp. 2589-2594, 2003.

[20] K. Kario, "Obstructive sleep apnea syndrome and hypertension: mechanism of the linkage and 24-h blood pressure control.," *Hypertension research,* vol. 32, no. 7, p. 537, 2009.

[21] E. C. Fletcher, "The relationship between systemic hypertension and obstructive sleep apnea: facts and theory," *The American journal of medicine,* vol. 98, no. 2, pp. 118-128, 1995.

[22] F. Sayk, C. Becker, C. Teckentrup, H.-L. Fehm, J. Struck, J. P. Wellhoener and C. Dodt, "To dip or not to dip: on the physiology of blood pressure decrease during nocturnal sleep in healthy humans," *Hypertension,* vol. 49, no. 5, pp. 1070-1076, 2007.

[23] D. P. Veerman, B. P. Imholz, W. Wieling, K. H. Wesseling and G. A. van Montfrans, "Circadian profile of systemic hemodynamics," *Hypertension,* vol. 26, no. 1, pp. 55-59, 1995.

[24] F. Lombardi and G. Parati, "An update on: cardiovascular and respiratory changes during sleep in normal and hypertensive subjects," *Cardiovascular research,* vol. 45, no. 1, pp. 200-211, 2000.

[25] Y. W. Endeshaw, W. B. White, M. Kutner, J. G. Ouslander and D. L. Bliwise, "Sleep-disordered breathing and 24-hour blood pressure pattern among older adults.," *Journals of Gerontology Series A: Biomedical Sciences and Medical Sciences,* vol. 64, no. 2, pp. 280-285, 2009.

[26] K. M. Hla, T. Young, L. Finn, P. E. Peppard, M. Szklo-Coxe and M. Stubbs, "Longitudinal association of sleep-disordered breathing and nondipping of nocturnal blood pressure in the Wisconsin Sleep Cohort Study," *Sleep,* vol. 31, no. 6, pp. 795-800, 2008.

[27] R. M. Alex, H. W. Chun, S. Sun-Mitchell, D. E. Watenpaugh and K. Behbehani, "Quantitative Assessment of Apnea-Induced Dynamic Blood Pressure Variations.," *J Sleep Med Disord,* vol. 3, no. 3, p. 1050, 2016.

[28] J. Truijen, J. J. van Lieshout, W. A. Wesselink and B. E. Westerhof, "Noninvasive continuous hemodynamic monitoring," *Journal of clinical monitoring and computing,* vol. 26, no. 4, pp. 267-278, 2012.

[29] B. P. Imholz, W. Wieling, G. A. van Montfrans and K. H. Wesseling, "Fifteen years experience with finger arterial pressure monitoring: assessment of the technology," *Cardiovascular research,* vol. 38, no. 3, pp. 605-616, 1998.

[30] R. D. Boehmer, "Continuous, real-time, noninvasive monitor of blood pressure: Peňaz methodology applied to the finger," *Journal of clinical monitoring,* vol. 3, no. 4, pp. 282-287, 1987.

[31] K. H. WESSELING, J. J. SETTELS, G. M. V. D. HOEVEN, J. A. NIJBOER, M. W. BUTIJN and J. C. DORLAS, "Effects of peripheral vasoconstriction on the measurement of blood pressure in a finger," *Cardiovascular research,* vol. 19, no. 3, pp. 139-145, 1985.

[32] R. M. Alex, *Quantitative variation of blood pressure dynamics during sleep apnea,* Texas: The University of Texas at Arlington, 2010.

[33] D. W. Eeftinck Schattenkerk, J. J. Van Lieshout, A. H. Van Den Meiracker, K. R. Wesseling, S. Blanc, W. Wieling, G. A. Van Montfrans, J. J. Settels, K. H. Wesseling and B. E. Westerhof, "Nexfin noninvasive continuous blood pressure validated against Riva-Rocci/Korotkoff," *American journal of hypertension,* vol. 22, no. 4, pp. 378-383, 2009.

[34] K. Wesseling, "Physiocal, calibrating finger vascular physiology for Finapres," *Homeostasis,* vol. 36, no. -1, pp. 67-82, 1995.

[35] T. D. Homan and E. Cichowski, "Physiology, pulse pressure," Vols. -1, StatPearls , 2019.

[36] M. Razminia, A. Trivedi, J. Molnar, M. Elbzour, M. Guerrero, Y. Salem, A. Ahmed, S. Khosla and D. L. Lubell, "Validation of a new formula for mean arterial pressure calculation: the new formula is superior to the standard formula," *Catheterization and cardiovascular interventions,* vol. 63, no. 4, pp. 419-425, 2004.

[37] M. E. Safar, B. I. Levy and H. Struijker-Boudier, "Current perspectives on arterial stiffness and pulse pressure in hypertension and cardiovascular diseases," *Circulation,* vol. 107, no. 22, pp. 2864-2869, 2003.

[38] A. Silvani, "Physiological sleep-dependent changes in arterial blood pressure: central autonomic commands and baroreflex control," *Clinical and Experimental Pharmacology and Physiology,* vol. 35, no. 9, pp. 987-994, 2008.

[39] B. Jafari, "Sleep architecture and blood pressure," *Sleep medicine clinics,* vol. 12, no. 2, pp. 161-166, 2017.

[40] M. Kuwabara, H. Hamasaki, N. Tomitani, T. Shiga and K. Kario, "Novel triggered nocturnal blood pressure monitoring for sleep apnea syndrome: distribution and reproducibility of hypoxia triggered nocturnal blood pressure measurements," *The Journal of Clinical Hypertension,* vol. 19, no. 1, pp. 30-37, 2017.

[41] M. Kuwabara, N. Tomitani, T. Shiga and K. Kario, "Polysomnography derived sleep parameters as a determinant of nocturnal blood pressure profile in patients with obstructive sleep apnea," *The Journal of Clinical Hypertension,* vol. 20, no. 6, pp. 1039-1048, 2018.

[42] N. Sasaki, M. Nagai, H. Mizuno, M. Kuwabara, S. Hoshide and K. Kario, "Associations between characteristics of obstructive sleep apnea and nocturnal blood pressure surge," *Hypertension,* vol. 72, no. 5, pp. 1133-1140, 2018.

[43] L. J. Epstein and G. R. Dorlac, "Cost-effectiveness analysis of nocturnal oximetry as a method of screening for sleep apnea-hypopnea syndrome," *Chest,* vol. 113, no. 1, pp. 97-103, 1998.

[44] D. Alvarez, R. Hornero, M. Garcia, F. del Campo and C. Zamarron, "Improving diagnostic ability of blood oxygen saturation from overnight pulse oximetry in obstructive sleep apnea detection by means of central tendency measure," *Artificial intelligence in medicine,* vol. 41, no. 1, pp. 13-24, 2007.

[45] T. Konecny, T. Kara and V. K. Somers, "Obstructive sleep apnea and hypertension: an update.," *Hypertension,* vol. 63, no. 2, pp. 203-209, 2014.

[46] A. S. Cifu and A. M. Davis, "Prevention, detection, evaluation, and management of high blood pressure in adults," *Jama,* vol. 318, no. 21, pp. 2132-2134, 2017.

[47] B. Darne, X. Girerd, M. Safar, F. Cambien and L. Guize, "Pulsatile versus steady component of blood pressure: a cross-sectional analysis and a prospective analysis on cardiovascular mortality.," *Hypertension,* vol. 13, no. 4, pp. 392-400, 1989.

[48] K. Kario, "Vascular Consequences of Sleep Disordered Breathing-Sleep apnea syndrome and hypertensive target organ damage in Japan," in *The American Society of Hypertension 21st Annual Scientific Meeting and Exposition*, New York, 2006.

[49] S. Lattanzi, F. Brigo and M. Silvestrini, "Obstructive sleep apnea syndrome and the nocturnal blood pressure profile," *The Journal of Clinical Hypertension,* vol. 20, no. 6, pp. 1036-1038, 2018.

[50] R. M. Alex, *An investigation of the effect of obstructive sleep apnea on cerebral hemodynamics in relation with systemic hemodynamics,* Arlington Texas: The University of Texas at Arlington, 2015.

BIOGRAPHICAL INFORMATION

Yao-Shun Chuang was born in Taipei, Taiwan, a small island next to Mainland China. His major in undergraduate is respiratory therapy. During his studies in medical school, he acquired professional knowledge in critical respiratory therapy and long-term respiratory therapy to realize the purpose of each treatment and practiced in critical and advanced comprehensive respiratory therapy to utilize medical knowledge in real-world situations. During the practice of critical respiratory therapy, he not only understood how respiratory therapists helped patients but also how hard it was to face work-related difficulties and heavy workloads. He wondered how he could improve the procedure of caring services to create better care. Thus, he joined a Data Science Class at Institute for Information Industry to acquire certification to acquire knowledge of data after which he received his national licensure as a Respiratory Therapist. After finished the certification course, he applied as a software engineer at RiskVal Financial Solutions, one of the top market risk solution companies in New York, to sharpen his coding skills. To accomplish his original purpose, he enrolled for a Master's Degree in Bioengineering at the University of Texas at Arlington in August 2018.

His motivation was triggered by my practical experience at the hospital when I saw for the first time that combining medicine and engineering was the key to improve the quality of medical care and relieve the severe pressure of medical personnel. In the upcoming 40 years, the elderly population will be in the majority,

which will affect the labor market. With increased workloads and inadequate medical personnel, not only would medical care services be difficult to maintain, but fewer people are also likely to work in medical fields. Therefore, to improve the quality of care for patients and reduce the working pressure of medical specialists, he is planning to secure a position as an R&D engineer in medical data analysis in biomedical and biotechnology industry.