Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations | Graduate Studies

12-2012

Visual Data Mining Techniques for Functional Actigraphy Data: An Object-Oriented Approach in R

Abbass Sharif
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

 Part of the Statistics and Probability Commons

Recommended Citation

Sharif, Abbass, "Visual Data Mining Techniques for Functional Actigraphy Data: An Object-Oriented Approach in R" (2012). *All Graduate Theses and Dissertations*. 1394.
https://digitalcommons.usu.edu/etd/1394

This Dissertation is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

VISUAL DATA MINING TECHNIQUES FOR FUNCTIONAL ACTIGRAPHY DATA: AN OBJECT-ORIENTED APPROACH IN R

by

Abbass Sharif

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Mathematical Sciences

Approved:

Dr. Jürgen Symanzik Major Professor	Dr. Piotr S. Kokoszka Committee Member
Dr. Daniel C. Coster Committee Member	Dr. Christopher D. Corcoran Committee Member
Dr. Yanghee Kim Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2012

Copyright © Abbass Sharif 2012

All Rights Reserved

ABSTRACT

Visual Data Mining Techniques for Functional Actigraphy Data: An Object-Oriented Approach in R

by

Abbass Sharif, Doctor of Philosophy

Utah State University, 2012

Major Professor: Dr. Jürgen Symanzik
Department: Mathematics and Statistics

Actigraphy, a technology for measuring a subject's overall activity level almost continuously over time, has gained a lot of momentum over the last few years. An actigraph, a watch-like device that can be attached to the wrist or ankle of a subject, uses an accelerometer to measure human movement every minute or even every 15 seconds. Actigraphy data is often treated as functional data. In this dissertation, we discuss what has been done regarding the visualization of actigraphy data, and then we explain the three main goals we achieved: (i) develop new multivariate visualization techniques for actigraphy data; (ii) integrate the new and current visualization tools into an R package using object-oriented model design; and (iii) develop an adaptive user-friendly web interface for actigraphy software.

(190 pages)

PUBLIC ABSTRACT

Actigraphy, a technology for measuring a subject's overall activity level almost continuously over time, has gained a lot of momentum over the last few years. An actigraph, a watch-like device that can be attached to the wrist or ankle of a subject, uses an accelerometer to measure human movement every minute or even every 15 seconds. Actigraphy data is often treated as functional data. In this dissertation, we discuss what has been done regarding the visualization of actigraphy data, and then we explain the three main goals we achieved: (i) develop new multivariate visualization techniques for actigraphy data; (ii) integrate the new and current visualization tools into an R package using object-oriented model design; and (iii) develop an adaptive user-friendly web interface for actigraphy software.

Abbass Sharif

This work is dedicated to my wonderful mother, Donia; and my father, Ismail.

ACKNOWLEDGMENTS

My top thanks go to my mother, Donia; my dad, Ismail; my brother, Ali; and my sister, Alia. They are my earlier heaven on this earth.

Millions of thanks to my major professor, Dr. Jürgen Symanzik, for his sincere support during my doctoral years at USU. Your continuous support helped me build an attitude of excitement and devotion for my research.

Many thanks go to the Mathematics and Statistics Department at Utah State University, to all professors, students, and staff members (especially Ms. Cindy Moulton).

My sincere appreciation to my committee members, Dr. Daniel C. Coster, Dr. Piotr S. Kokoszka, Dr. Christopher D. Corcoran, and Dr. Yanghee Kim, for their agreement to work with me and sit on my committee.

To the department of Mathematics and Computer Science at the Lebanese American University in Beirut: thank you for the solid foundation I obtained while pursuing my undergraduate and master's degrees. Dr. May Abboud, Dr. Ramzi Haraty, and Dr. Samer Habre, thank you for your comprehensive support and encouragement.

I would also like to thank my friends for the moral support, positive vibes, and encouragement. A person who seeks a Ph.D., he/she should make sure to have friends of your calibre!

United States friends: Ani Aghababyan, Ani Mirzakhanyan, Armen Armaghanyan, Allyn Bernkof, Dr. Bobbe Allen, Brian Abel, Carlos Calbimonte, Chris Lewis, Dr. Daniel Useche, Danilo Lemos, Jean Carlos Guzman, Dr. James Odei, Janitha Nandalochana, Jonathan Koch, Jordan Glissmeyer, Karli Salisbury, Kayla Harris, Lauren Ayne, Lia Inoa, Dr. Magathi Jayaram, Manal El Arab, Marco Antonio Leite

Ribeiro Bodini, Mohamed El Hamoui, Mercedes Roman, Nadishan Pitigala, Nare Hayrapetyan, Nayda Gonzalez, Dr. Nicoleta Fuca, Randa Yassine, Dr. Robertas Gabrys, Rob Gentillon, Rouchelle Brockman, Dr. Roula Bachour, Dr. Ryan Hill, Satenik Sargsyan, Stella Henry, Tony Kusbach, Vance Almquist, and Wonhee Hong. **Lebanon friends:** Batoul Bitar, Dr. Fadel Jaber, Kamal Kaawach, Karim Baalbaki, Khaled Mneimneh, Hassan Mansour, Haytham El Zein, Loulou Hilal, Majd Eid, Ryan Sabbah, Roaida Hilal, Sarah Hilal, Samih El Khatib, and Zeina Youssef.

Abbass Sharif

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Actigraphy is an emerging technology for measuring human activity/rest levels over time. Actigraphy data are collected by an actigraph unit which is a non-invasive watch-like device that consists of an accelerometer. Actigraphy is useful to evaluate sleeping patterns, fatigue, circadian rhythms, and general activity over a period of several weeks.

Actigraphy observations are recorded almost continuously over time. Today's actigraphs can measure human activity at different accumulation rates ranging from low (one or more minute intervals) over high (15 seconds intervals) to very high (one second interval). These data could be treated as functional data. Ramsay and Silverman (2006, p. 38), characterize functional data as follows: "The basic philosophy of functional data analysis is to think of observed data functions as single entities, rather than merely as a sequence of individual observations." In order to explore and interact with functional actigraphy data, we need some new and easy-to-use visualization techniques and interfaces. So far, a few visualization techniques have been provided by the manufacturers of actigraphs. These techniques have very limited features that might not allow the users of the software to view activity levels of one or more subjects from different perspectives. In addition, a user/programmer is not allowed to customize the software provided by the manufacturer to meet his/her needs. In this dissertation, we developed new visualization methods that utilize the object-oriented (OO) programming approach (Lafore, 2002), and then bundled this functionality as

an open source software to be published as a part of *The Comprehensive R Archive Network-CRAN*, http://cran.r-project.org. The main purpose behind using the OO paradigm is to make it easier for other programmers to reuse our suggested visualization techniques. For example, this would allow other programmers to create different user interfaces and customize these interfaces for the needs of their users.

1.2 Current Visualizations of Actigraphy Data: Literature Review

Figure 2.1 shows an actigram (sometimes also called actogram), a visual display of the daily activity/rest patterns, of a certain subject. This is a commonly used visualization technique for actigraphy data and can be found in numerous actigraphy related publications, e.g., Figures 1 & 2 in Slaven et al. (2009), and Figure 2 in Labyak and Bourguignon (2002). Figure 2.1 was produced using the software developed by the manufacturer of the Actical actigraphy device, Mini Mitter Company Incorporated. (2005). The monitoring of this subject started on 4/26/2007 and ended on 5/7/2007. Each row of the actigram represents the flow of activity during one day. The black spikes represent the level of activity, the red dots/segments at the horizontal axis tell whether there is activity or not at a specific time of the day, and the blue upside-down triangles indicate time points marked by the subject when a major new activity has started or ended, such as waking up or going to bed.

From a display such as in Figure 2.1, a viewer should be able to detect if there is major minute-to-minute variability during a certain day or significant day-to-day variability among the days. Here, for example, on 5/4/2007 the subject's overall activity is very low compared to his/her overall activity on 5/6/2007. Although this graph allows us to explore the data visually, it only offers limited insight. It is not a very powerful tool for studying the variability within a subject and among multiple subjects. For example, if a medical doctor wants to check if a certain treatment

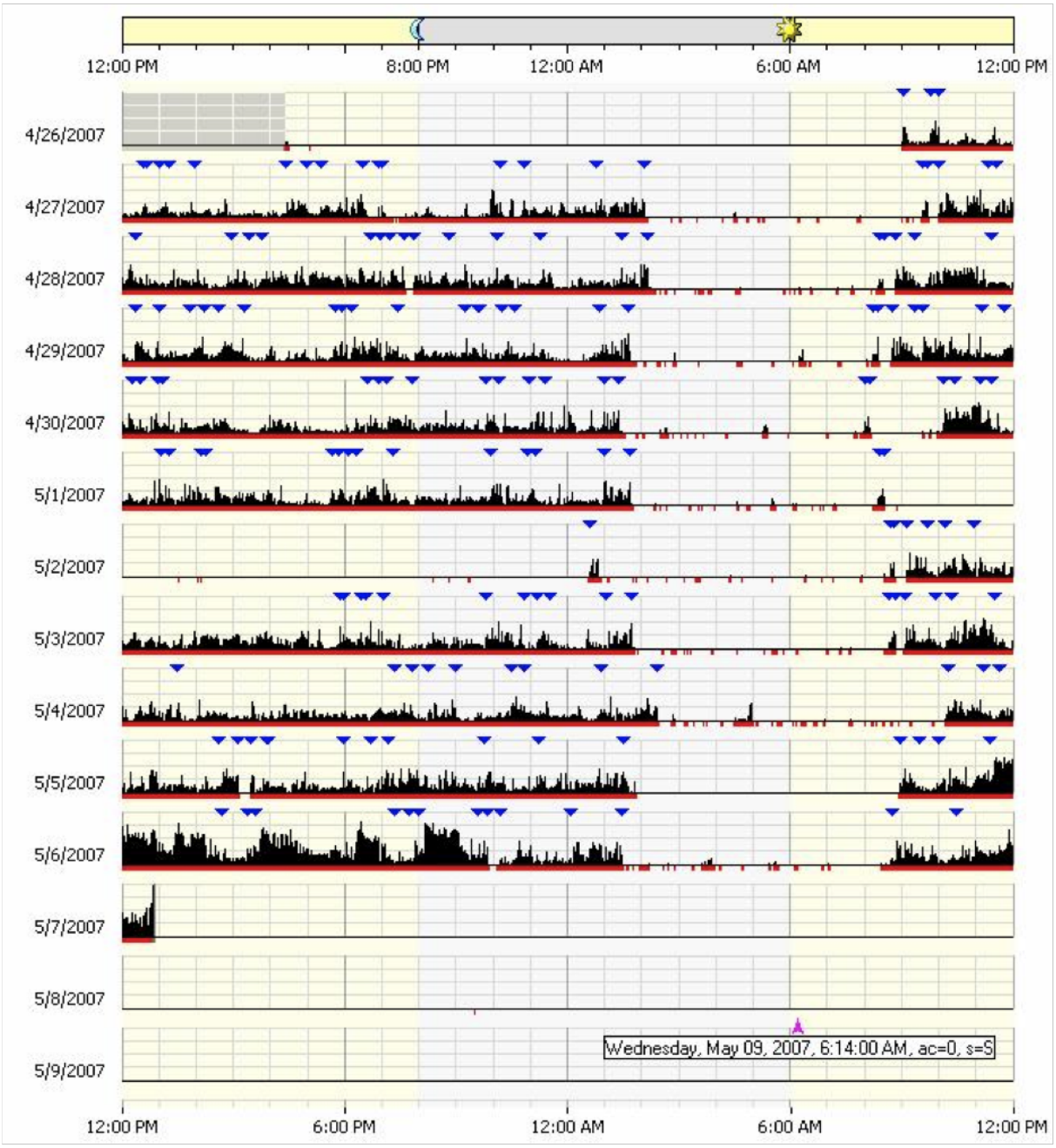


Fig. 1.1: An actigram as produced by the Actical software from Mini Mitter Company Incorporated. (2005). The horizontal axis represents the time from noon (far left) to noon 24 hours later (far right). Vertically, 14 days are shown. Data have been collected on twelve of those days. The solid black area represents the activity levels of the subject

or disease is affecting a patient's activity, then the doctor needs to compare data from before and after the treatment. The actigram won't allow us to do detailed comparisons. Furthermore, it does not allow us to do group comparisons such as control vs. experimental groups.

Histograms and numerical summary statistics have been also used to explore actigraphy data. These are not very useful and need to be extended. For instance, Figure 1.2, taken from Symanzik and Shannon (2008), shows the histogram display for four patients. The horizontal axis represent the activity levels for each patient (from low to high activity level), and the vertical axis represents the percentage of time a particular patient exhibits a certain level of activity. Patients A and B (the bottom two histograms) have similar levels of activity that range from 0 (probably during sleep) to 400, while patients C and D (the top two histograms) have very low levels of activity. This is shown by the high bars on the left side of the histogram and no bars on the right side. This interpretation is confirmed by summary statistics of mean activity levels of 5.4, 20.4, 206.5, and 209.7 for patients C, D, B, and A, respectively. These histograms and summary statistics describe how the patients differ in activity levels, but fail to capture when the patients exhibit different levels of activity levels and patterns.

A variety of new or improved visualization methods for actigraphy data have been suggested in Symanzik and Shannon (2008), such as the raw data plot, smoothed data plot, velocity plot, acceleration plot, cumulative sums plot, and sorted cumulative sums plot. Some of these plots have been previously introduced, such as the cumulative sums plot that resembles the cumulative actigram in Figures 2(B) & 3 in Labyak and Bourguignon (2002). These visualization techniques for actigraphy data are useful when doing comparisons for a single subject over time (e.g., baseline, during treatment, and after treatment).

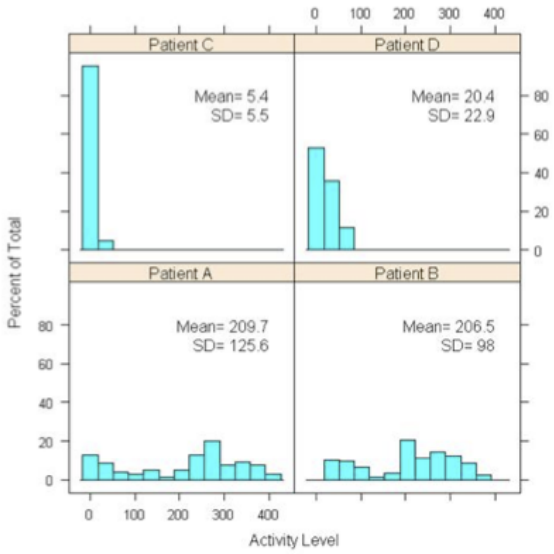


Fig. 1.2: Histograms and numerical summaries of actigraphy data of four different patients (Symanzik and Shannon, 2008). The x-axis indicates the level of activity (higher represents more active) and the y-axis shows the percentage of time the patient exhibits that level of activity.

Figure 1.3 shows six of these recently introduced visualization techniques (Sharif et al., 2010), produced in R (R Core Team., 2012). Each of these graphs provides a different insight for the actigraphy level of a certain subject, from Day 3 through Day 6 of monitoring. The x-axis shows the time of the day, except for the sorted cumulative sums plot in (f). The y-axis shows the activity level or a derived measure for the subject.

In the raw data plot (Figure 1.3(a)), a viewer may speculate that there is a pattern in the activity levels of this subject, but this pattern is not clearly visible because of the extensive overplotting of the points. The smoothed data plot (Figure 1.3(b)) would be a better approach, especially when taking into account that actigraphy data can be considered as functional data. In this plot, each day is represented as a function by showing (locally weighted scatterplot smoothing) lowess (Cleveland, 1979, 1981) smoothed curves (with parameter $f = 0.1$). Here, we can see a clear pattern of

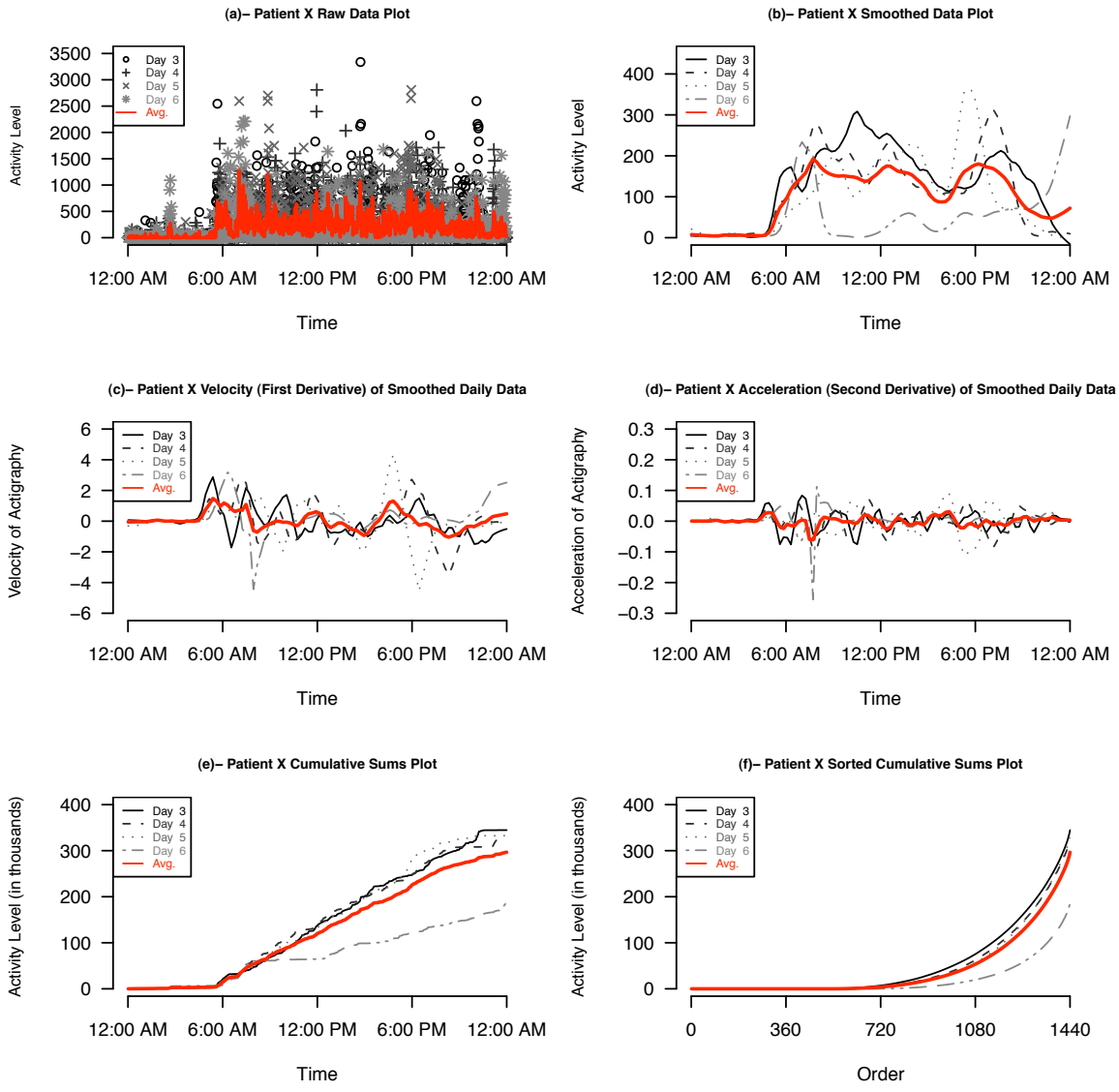


Fig. 1.3: Recent visualization techniques for actigraphy data (Sharif et al., 2010). (a) raw data plot, (b) smoothed data plot, (c) velocity plot, (d) acceleration plot, (e) cumulative sums plot, (f) sorted cumulative sums plot. Shown are data for a single subject, called Patient X, for four consecutive days. The horizontal axis represents a 24-hour period from midnight to midnight. The thick red curves represent the averages (Avg.) which are calculated from the raw data in plots (a), (e), and (f), and from the smoothed data in plots (b), (c), and (d). Some curves are partially hidden due to overplotting

typical activity levels for this subject; active during the day and resting during the night for all days except for Day 6 where the subject is inactive most of the day. The velocity plot (Figure 1.3(c)), i.e., the first derivative of the smoothed daily activity data, tells us how the activity level is changing over time. In other words, it tells us when the subject is becoming more active (positive), less active(negative), or staying at the same activity level (zero). If a viewer looks at the average velocity (thick red curve), he/she can see that the activity level starts increasing at 6 am, and then it stays almost at the same level between 9 am and 5 pm, and then decreases to become constant again when the subject goes to sleep. The acceleration plot (Figure 1.3(d)), i.e., the second derivative of the smoothed daily activity data, tells us how quickly the changes in velocity are occurring. If the acceleration is positive, then the rate of the change in activity levels over time is increasing; if the acceleration is negative, then the rate of change in activity levels over time is decreasing; and if the acceleration is zero, then the rate of change is constant.

If we focus on Day 6 in the smoothed data plot (Figure 1.3(b)), a viewer can see that this particular subject has low activity throughout this day compared to the three other days. To check whether this day is an outlier, we can use the cumulative sums plot (Figure 1.3(e)). This plot shows accumulated activity obtained by adding up the activity counts as one moves across the horizontal time axis from midnight (far left) to midnight 24 hours later (far right). This plot is useful to show total activity of a subject up to a particular time of the day. Indeed, the subject had accumulated very little activity this day compared to the other three plotted days. We can also use the sorted accumulated activity plot (Figure 1.3(f)) obtained by adding up the activity counts from smallest to largest. It should be noted that the horizontal axis no longer represents time, but order, i.e., minutes from 1 to 1440 (24 hours $\times$ 60 minutes = 1440 minutes). Statistically, we are summing up the observed order statistics $x_{(1)}$

to $x_{(1440)}$ to create this plot. This sorted cumulative sums plot might also be helpful to check if the overall activity of a particular subject for a certain day is similar to the overall activity for the remaining days. Perhaps, a subject might have shifted his/her main activity during a certain day from morning to afternoon.

A detailed description of these plots with variation assessment tools could be found in Ding et al. (2011).

1.3 Goals of this Dissertation

The work in this dissertation is developed in order to produce reusable statistical tools for visualizing actigraphy data with a user-friendly interface. The American Academy of Sleep Medicine recommends the use of actigraphy as a useful measure for detecting sleep in healthy individuals through assessing specific aspects in insomnia and restless leg syndrome (Ancoli-Israel et al., 2003; Morgenthaler et al., 2007). They also recommend actigraphy as a tool for objectively measuring fatigue. In order to increase actigraphy as a tool for objectively measuring fatigue, and overcome the limitations of current visualization tools and software, we propose the following three specific goals:

1.3.1 Goal 1: Development of New Multivariate Visualization Techniques for Actigraphy Data

We enhanced the visualization techniques that were suggested in Symanzik and Shannon (2008) and presented a first implementation in Sharif et al. (2010). Those enhanced visualization techniques allow the clinician to view a single patient's actigraphy data to identify aberrant patterns of activity. In other words, these techniques explore the variability within individual patients, but not between multiple patients. To overcome this limitation, we introduced four multivariate visualization techniques

(Chapter 2).

1.3.2 Goal 2: Integration of New and Current Visualization Tools into an R Package Using Object-oriented Model Design

We published our preliminary object-oriented model in Sharif et al. (2010). In this dissertation, we enhanced this model to fit all of our visualization techniques from Goal 1. All of the programming was done in an object-oriented approach using R (Chapter 3). R is a free software environment for statistical computing and graphics, http://www.r-project.org. We bundled all of the code into an R package, which is a set of utility methods for managing, storing, visualizing, and exporting data and results. This package will be submitted to *The Comprehensive R Archive Network-CRAN*, http://cran.r-project.org. In addition, a set of help and tutorial documents will be written and made available following the R program developer's guideline and specifications.

1.3.3 Goal 3: Development of a User-friendly Web Interface for Actigraphy Software

Many end users of the R package described in Goal 1 and Goal 2 can be expected to have a background in the medical field, sports, or they might be individuals interested in their daily activity levels.[1] Those users are unlikely to know how to deal with computer code written in R. Furthermore, R does not have a user-friendly Graphical User Interface (GUI) with menus and buttons. These users want to focus on the results and graphics, and not on running computer code. Thus, we developed an easy-to-use web GUI for the underlying R functionality (Chapter 4).

[1] There is website for people interested in self-tracking devices to gather information and share knowledge and experiences with others (Quantified Self Labs., 2012)

CHAPTER 2
MULIVARIATE VISUAL DATA MINING TECHNIQUES FOR ACTIGRAPHY DATA

2.1 Introduction

Actigraphy is an emerging clinical technology for measuring human sleep, daytime activity, and circadian activity rhythms over time via a device called an actigraph. An actigraph is a non-invasive watch-like device usually attached to the wrist or the leg to measure the movements, via a sensor, in the form of activity counts recorded almost continuously over time. Due to its continuous nature, this type of data could be treated as functional time series data (Ramsay and Silverman, 2006). So far, a few visualization techniques have been provided by the manufacturers of actigraphs. These techniques have very limited features that might not allow the users of the software to view activity levels of one or more subjects from different perspectives. For example, Figure 2.1 shows an actigram (sometimes also called actogram), a visual display of the daily activity/rest patterns, of a certain subject. This is a commonly used visualization technique for actigraphy data and can be found in numerous actigraphy related publications, e.g., Figure 1 and Figure 2 in Slaven et al. (2009), and Figure 2 in Labyak and Bourguignon (2002). This visualization allows to study the variability of a subject's activity during a certain day or many days, but it can't be used as a tool to compare many subjects or study the activity of different groups (e.g. males vs. females, young vs. old, etc.). In addition, the following three issues rise while visualizing such type of data, especially when we deal with a large number of subjects and/or many days of data per subject.

- *Information Loss*: researchers use data smoothing algorithms to fit curves on actigraphy data (Ogbagaber et al., 2012; Wang et al., 2011; Ding et al., 2011;

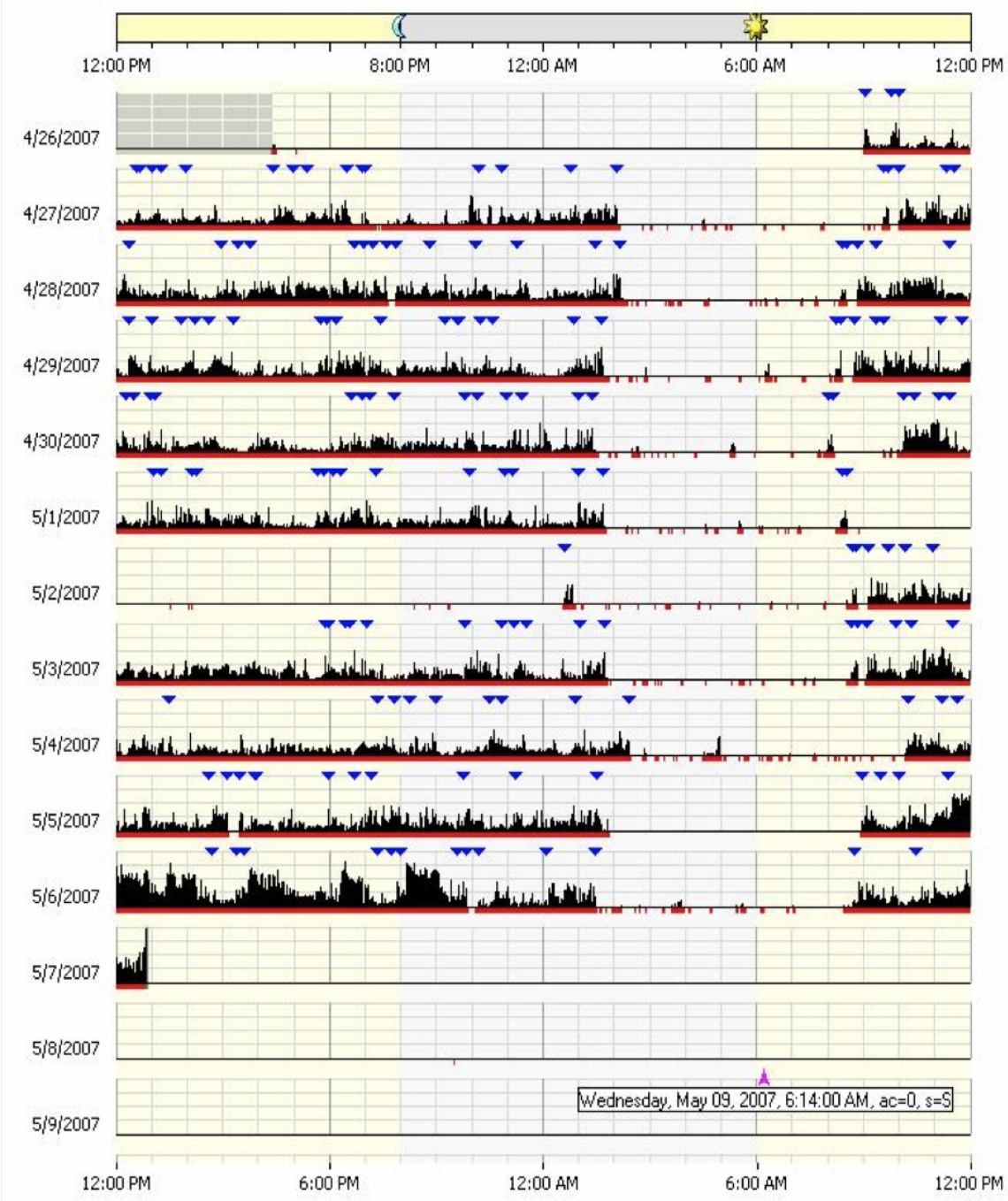


Fig. 2.1: An actigram as produced by the Actical software from Mini Mitter Company Incorporated. (2005). The horizontal axis represents the time from noon (far left) to noon 24 hours later (far right). Vertically, 14 days are shown. Data have been collected on twelve of those days. The solid black area represents the activity levels of the subject

Sharif et al., 2010; Symanzik and Shannon, 2008). Data smoothing is a tool used to reduce the noise and irregularities in order to capture and reveal interesting patterns present in large datasets. However; one of the main drawbacks for data smoothing is losing some insight for the variation in the data. Another critical issue in some of smoothing algorithms is that they are not robust against outliers, and therefore the smoothed curve is pulled towards those outliers (Rice, 2004). This problem is present in almost every smoothing technique except for the LOWESS (Locally Weighted Scatterplot Smoothing) algorithm (Cleveland, 1979, 1981).

- *Measurement Bias*: using actigraphs to measure activity levels of humans might produce substantial measurement error. The actigraphy devices might be biased for many different reasons. In Sherick's study (Sherick et al., 2010) on the accuracy of the Actical actigraphy devices that are manufactured by the Mini Mitter Company Incorporated (Mini Mitter Company Incorporated., 2005), it was suggested that one of the four sampled Actical devices which were used for measuring activity levels of patients in Sharif et al. (2010) were biased. Another study (Esliger and Tremblay, 2006) on Acticals suggests that even though those devices have small inter- and intra-instrument coefficient of variations, calibration and reliability of devices should not be assumed.

- *Curve Cluttering*: in exploratory data analysis, sometimes the researchers might want to compare different sets of groups (males vs. females, different age groups, different races, etc.). Laying the smoothed data curves of all groups on the same plot might not produce the desired "clear-cut" grouping of objects with close characteristics.

In order to overcome the above mentioned issues we have adopted and enhanced some techniques to help visualize functional datasets, and in particular actigraphy

datasets. Four main techniques are introduced: (i) density-based plots such as repeated density strips, (ii) data enveloping methods ($min-max$, $25^{th}-75^{th}$ percentile, and $40^{th}-60^{th}$ percentile) to summarize common features, (iii) data summing over time (10, 20, 30 and 60 minutes), and (iv) multivariate time series plots such as data images. These techniques are applied to raw data, i.e., unprocessed actigraphy data. No filtering, smoothing, or any other statistical techniques are required at this stage. Therefore, the visual data mining approach could be seen as an exploratory data analysis (EDA) phase for functional actigraphy data.

The EDA concept was first introduced by Tukey (1977) to encourage statisticians to visually explore their datasets in order to find structure, outliers, trends, patterns, and/or unexpected behavior, etc. in them. This chapter introduces EDA techniques for functional actigraphy data, and is inspired by Wegman's article on visual data mining (Wegman, 2003), where he presents three tools for visualizing large datasets: parallel coordinates, the d-dimensional grand tour, and saturation brushing. According to him, visual data mining is the process of discovering the unknown structure of the dataset through graphical methods and techniques that help in depicting statistical patterns, trends and information which is hidden in data.

In Section 2.2, we present four tools for the visual data mining of functional data. Section 2.3 describes simulated actigraphy-like data. In Section 2.4, we demonstrate how to apply the four techniques on the simulated data. Section 2.5 presents real actigraphy data, and Section 2.6 describes the use of the four suggested techniques on these data. Finally, Section 2.7 concludes the chapter with a discussion.

2.2 Techniques for Visualizing Functional Data

Consider the scenario in time series data where we have to graph observations recorded every minute over multiple days. This means that we end up having 1440 observations per day (24 hours $\times$ 60 minutes). This data usually have lots of spikes and variations depending on the application. In such scenarios, scientists usually

try to visualize the pattern that is present in the data by plotting all observations on one figure where the x-axis (horizontal axis) represents time (in minutes) and the y-axis (vertical axis) represents the measurement of a particular quantity under investigation. If the scientist wants to compare how those measurements vary among different days, then the traditional way would be to overlay all days on the same plot. This action might produce messy plots with lots of interweaving spikes and lots of data overplotting. Thus, there is no immediate way to distinguish between different days' patterns even if the scientist uses color to differentiate between days. A solution would be to smooth the curves, which has the disadvantage of losing information. In this section, we will present four visual data mining techniques to explore functional time series data that could be used either alone or combined with some of the other techniques depending on how messy or noisy the data are.

2.2.1 Density-based Plots

The idea of density-based plots is based on Jackson's density strip plots (Jackson, 2008) which display uncertainty with shading. A density strip plot is usually a horizontal rectangle with color shading that ranges from dark to white. It is shaded with darkness proportional to the probability density of the measured quantity at a point, darkest at points of highest probability density, and white at points of zero density. This kind of plots is usually very helpful when comparing distributions arising from parameter estimation.

Density strip plots could be utilized to visualize variability and trends for functional data over time. The main purpose is to make the trends look clearer on the plot without using smoother functions that might lead to loss of information. The proposed density-based plots are simply stacked vertical density strips with equal widths, where each strip shows data for a given period of time (e.g., 1 minute, 10 minutes, 1 hour, etc.). In order to have a proportional shading scheme for all of the strips, the shading level for for each strip is multiplied by its density divided by the

maximum density over all strips. This technique is similar to Miller and Wegman (1991), where the author proposed to have a "line density plot" instead of drawing individual lines using a binning technique where the plot is divided into zones, and the number of lines passing through each zone is counted. These zones are then color shaded based on the obtained counts.

These density-based plots could be used with either raw data or cumulative sums data. Cumulative sums plots show accumulated activity obtained by adding up activity counts as one moves across the horizontal time axis from midnight (far left) to midnight 24 hours later (far right) (Sharif et al., 2010). They are helpful to show the total activity of a group up to a particular time of the day.

2.2.2 Data Enveloping

Data enveloping is the process of subsetting the dataset into different classes, drawing bands around each class of data observations, and then filling each with a different color. The idea of enveloping data was first introduced by Inselberg et al. (1987) as a tool to reduce noise in Parallel Coordinate Plots (PCPs), and then it was enhanced by Moustafa et al. (2011).

This technique is very helpful to clearly see trends followed by each class or visually validate cluster analysis results. These bands often range from the minimum observation to the maximum observation for every minute ($min - max$ envelopes), but sometimes these extreme observations might be outliers, and thus cause heavy overlapping between the classes. In order to overcome this problem, two things could be done: (1) the bands could be drawn with a narrower range; for example, from the first quartile Q_1 to the third quartile Q_3 of a given class of data, or from the 40^{th} percentile to the 60^{th} percentile, etc., and (2) use alpha blending techniques (Porter and Duff, 1984) for the colors to create transparency effects in order to be able to see the hidden parts of class bands.

2.2.3 Data Summing

Data summing is the process of combining a collection of data observations into one single observation. In time series data analysis, we can sum up many observations into 10, 20, 30, or even 60 minute intervals instead of looking at minutely data. This process will help in data reduction, and automatically reduce the number of spikes and make the graph looks smoother . For example, if we decided to sum up each 10 observations into one observation, we will reduce the daily dataset by 90% (from 1440 to 144). In order to get a better view of data clusters, this technique could be used in combination with data enveloping.

2.2.4 Multivariate Time Series Plots

Multivariate time series plots become handy when we need to compare more than five or six time series data. In the traditional way, the comparison was done by stacking all time series plots in a fashion where it sometimes become difficult to fit all plots on one page or a computer screen. Peng (2008) visualized environmental data that are collected over time and multiple locations. Instead of producing the traditional time series stacked plots, he came up with a new visualization technique for plotting multivariate time series. This plot is based on the "data images" concept, which was first introduced by Wegman (1990) as colored histograms which are now called "data images" (Minnotte and West, 1998; Morphet and Symanzik, 2010). The idea is simple and similar to density based plots: each time series is split into different categories that are assigned to a color intensity range from low (few observations) to high (many observations). The number of categories vary depending on the nature of data. If the data are smooth, then we can have many categories, while if the data are noisy and spiky, then few categories are needed to give a smoother look for the plot. This kind of categorization or discretization allows the user to visualize variation in the data.

In addition to the basic data image, multivariate time series plots could be augmented with some summary information at the right and bottom sides of the plot. For example, a box plot could be drawn for each time series at the right hand side of the plot. Also, at the bottom of the plot, we can have some graphs using the previously mentioned techniques which would give summary information about values across all the time series for each specified time point.

2.3 Simulated Data

The techniques introduced in the previous section that will be applied to our real actigraphy data (Section 2.5) will first be demonstrated on simulated data set. One of the clinical questions that medical doctors investigate in actigraphy is whether activity patterns differ for people with different levels of depression? For that purpose, we simulated actigraphy-like data with five different classes representing five groups of people with different activity levels that are clearly separated from each other, based on the following Sine model:

$$Y_{ijkl} = \max(0, j \times \sin \tfrac{i\pi}{N} + k \times Z_{ijkl}),$$

where Y_{ijkl} represents an observation at time $i \in \{1, 2, 3, ..., N\}$, that belongs to class level $j \in \{200, 400, 600, 800, 1000\}$, with an induced random noise $k \in \{1, 50, 100, 200\}$ for $N = 1440$ (24 hours $\times$ 60 minutes) and with $l = 10$ functional data observations. Here, Z_{ijkl} is a standard normal random variable.

Thus, the simulated data represent some noisy actigraphy-like data with little to no actigraphy for small and large values of i (representing early mornings and late evenings), and peak actigraphy towards $i \approx \frac{N}{2}$ (representing times around noon). Different magnitudes (represented by j) and noise levels (represented by k) have been modeled. Simulated data smaller than zero were replaced with zero.

The motivation behind simulating actigraphy-like data is to see how the techniques that were introduced in Section 2.2 work with data classes that are clearly

separated from each other. In particular, these simulated datasets were created to answer the following questions:

- Does data enveloping help in detecting patterns in the data curves?
- Does summing of data over time help detecting patterns in the data curves?
- Does density-based plots and multivariate time series plots detect clusters and patterns in data?

2.4 Techniques Applied to Simulated Data

In this section, we demonstrate how the four techniques discussed in Section 2.2 can be applied to the simulated data that are described in Section 2.3. We use the same dataset to illustrate all of the techniques, with 10 functional data observations for each class level (j).

2.4.1 Density-based Plots

Figure 2.2 shows the density-based plots for the simulated data with four different noise levels ranging from zero noise ($k = 1$) to high noise ($k = 200$). Although it would be hard to detect a clear separation between the five classes of simulated levels, this kind of plot helps us to see the trend that the data are following. The activity levels are close to zero at midnight, and then they start growing up until they reach a peak at the middle of the day, and then they start falling down again. Density-based plots help in giving a smoother look at the data by blending the spikes together via its color shading technique. Notice that at midnight, the activity levels are concentrated near the zero level. Therefore, the color shading is the darkest in that region, and then it goes lighter towards grey at the middle of the day where the simulated activity levels vary most.

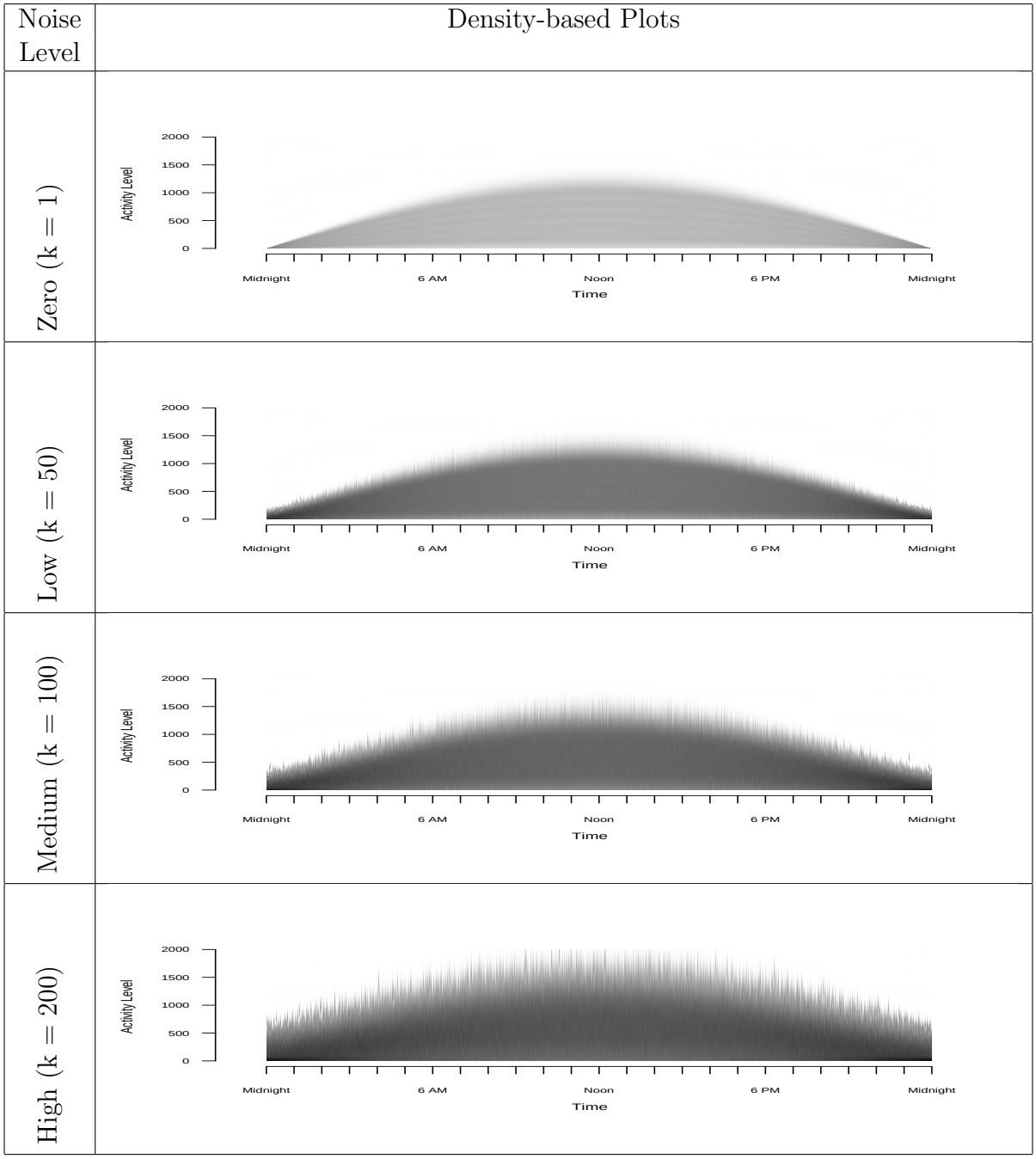


Fig. 2.2: Density-based Plots: Simulated data with different noise levels ($k = 1, 50, 100, 200$)

2.4.2 Data Enveloping

Figures 2.3 and 2.4 shows the $min-max$ and Q_1-Q_3 envelopes for the simulated

data. In each set of envelopes, there are four graphs (zero noise to high noise). Trying to get information from the $min - max$ envelopes does not help much, unless there is not much overlapping between the classes (zero and low noise). Narrowing the bands of the envelope to range from Q_1 to Q_3 in each class helps the viewer to better detect patterns at the medium noise level. The $Q_1 - Q_3$ envelope does not work well at the high noise level. Hence, we need a narrower envelope such as the $40^{th} - 60^{th}$ percentile envelope.

Even though data envelopes help in revealing data clusters, this technique does not help in reducing the spikes in the plots, in particular sharp spikes that are present at the medium to high noise levels.

This technique would be very useful if implemented in an interactive software environment where the envelope bands range as a parameter that could be changed by the user. In our simulated data, when the noise level was low, a $Q_1 - Q_3$ envelope was sufficient to depict a clear separation between the five classes. That is not the case when the noise level was medium or high. We need to have narrower envelopes such as the $30^{th} - 70^{th}$ percentile or maybe the $40^{th} - 60^{th}$ percentile to be able to distinguish the patterns of the five classes.

2.4.3 Data Summing

Data enveloping helps in showing the separation between different group clusters in the simulated data, but it fails to smooth the spikes. In order to reduce the spikes from the envelope plots in Figures 2.3 and 2.4, we aggregate the data and do four levels of summing over time (10, 20, 30, and 60 minutes summing). Figures 2.5, 2.6, 2.7, and 2.8 show the graphs of the raw simulated data with different noise levels before and after aggregation (10, 20, 30 and 60 minutes summing) with $min - max$ envelopes. For the zero noise level, it is obvious that there is no need for data summing (Figure 2.5). For the low level noise (Figure 2.6) and the medium level noise (Figure 2.7), we can see spikes being smoothed out with just 10 minutes of aggregation. This

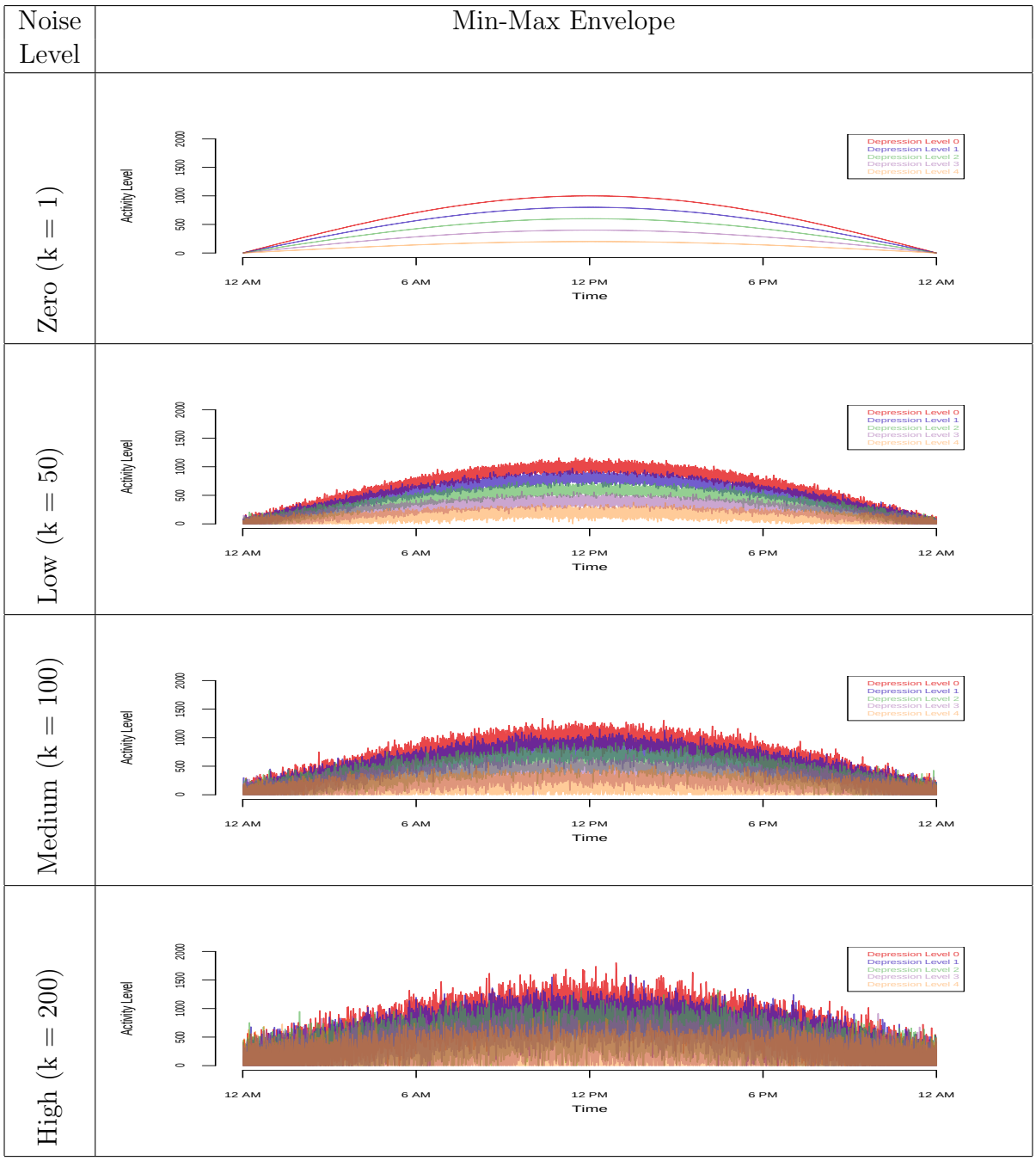


Fig. 2.3: Data Envelopes: $Min - max$ envelopes for simulated data with different noise levels

turned out to be enough for smoothing low and medium level noise data because there is no further improvement done with 20, 30, and 60 minutes of aggregation. Figure

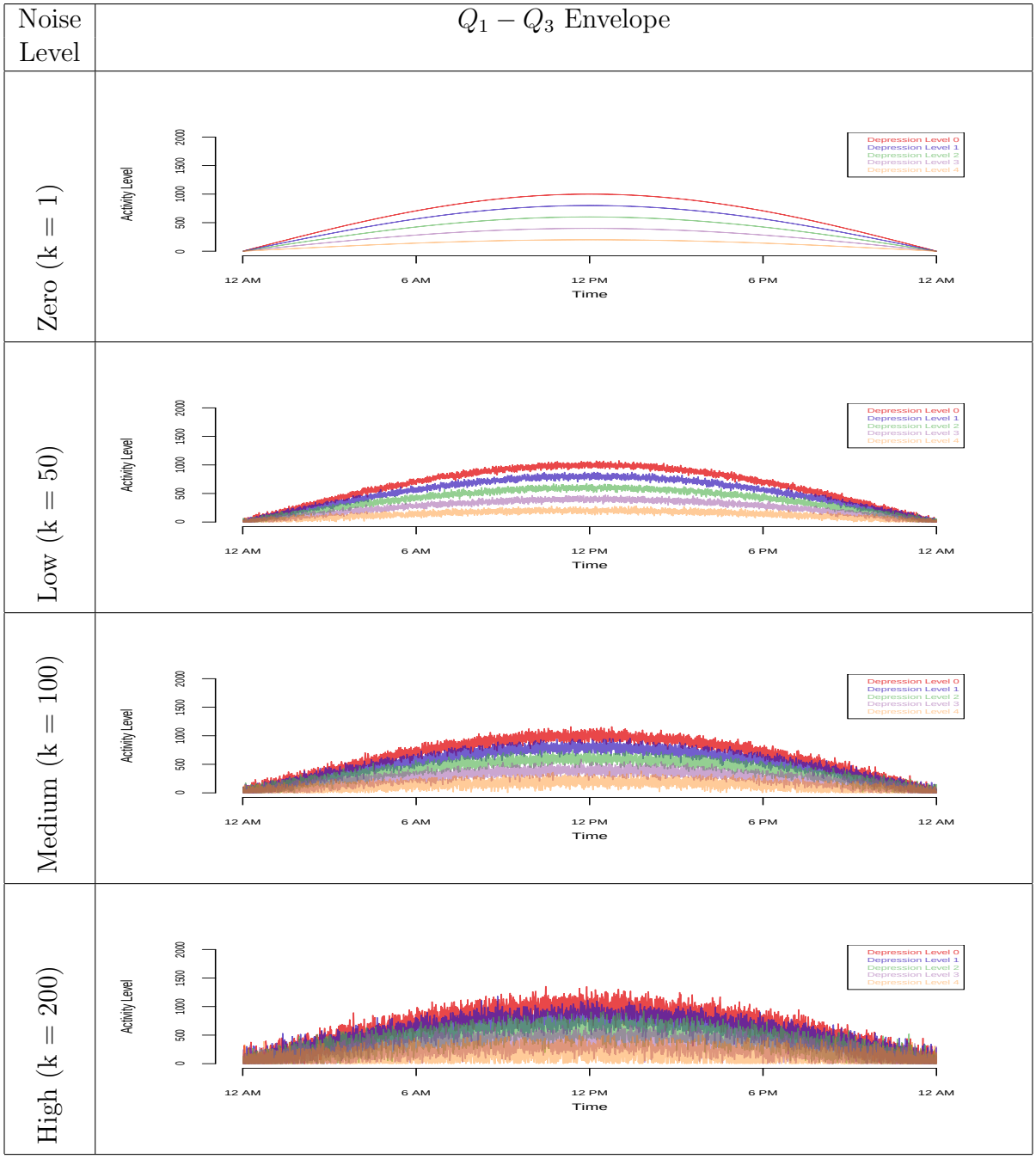


Fig. 2.4: Data Envelopes: $Q_1 - Q_3$ envelopes for simulated data with different noise levels

2.8 shows that for data with the high noise level 10 minutes of aggregation do not result in a clear separation of clusters and spikes are still present. Things get better

with 20 minutes of aggregation, but for this particular case, 30 minutes of aggregation is the best.

2.4.4 Multivariate Time Series Plots

In Figure 2.9, we plot the simulated actigraphy-like data with medium noise level (k=100). The top left part of the graph is an image plot with 50 horizontal strips, each represents a time series for one subject's average activity level over a ten-days period ($l = 10$). In this plot, each time series is discretized into three categories using a diverging palette of colors which assigns purple to low activity values, grey to medium activity, and green to high activity. The discretization of the data is done using quantiles of the time series values. Therefore, using three levels implies dividing the data into tertiles with roughly an equal number of points in each (Peng, 2008).

This kind of plot, clearly categorizes subjects into five classes separated by a horizontal black line. According to our simulated data, a subject with low class level (Level 0), has very low activity at mid night (purple), and then its activity grows gradually (grey) until it reaches its maximum in the middle of the day (green). Notice that the green areas shrink the higher the class level is. A subject with very high class level (Level 4) is barely active, and this is clear because most of the Level 5 block is purple.

The right hand side display of the graph is a set of box plots which shows the distribution of average activity levels for each subject. We can see that subjects with very high class level (Level 5) have very low variation in their activity, while subjects with very low class level (Level 0) have more activity variation. The plot at the bottom is a display for the median activity level across all the time series for each time point. This bottom area could be replaced with any of the previously discussed visualizations in this chapter such as in Figure 2.10.

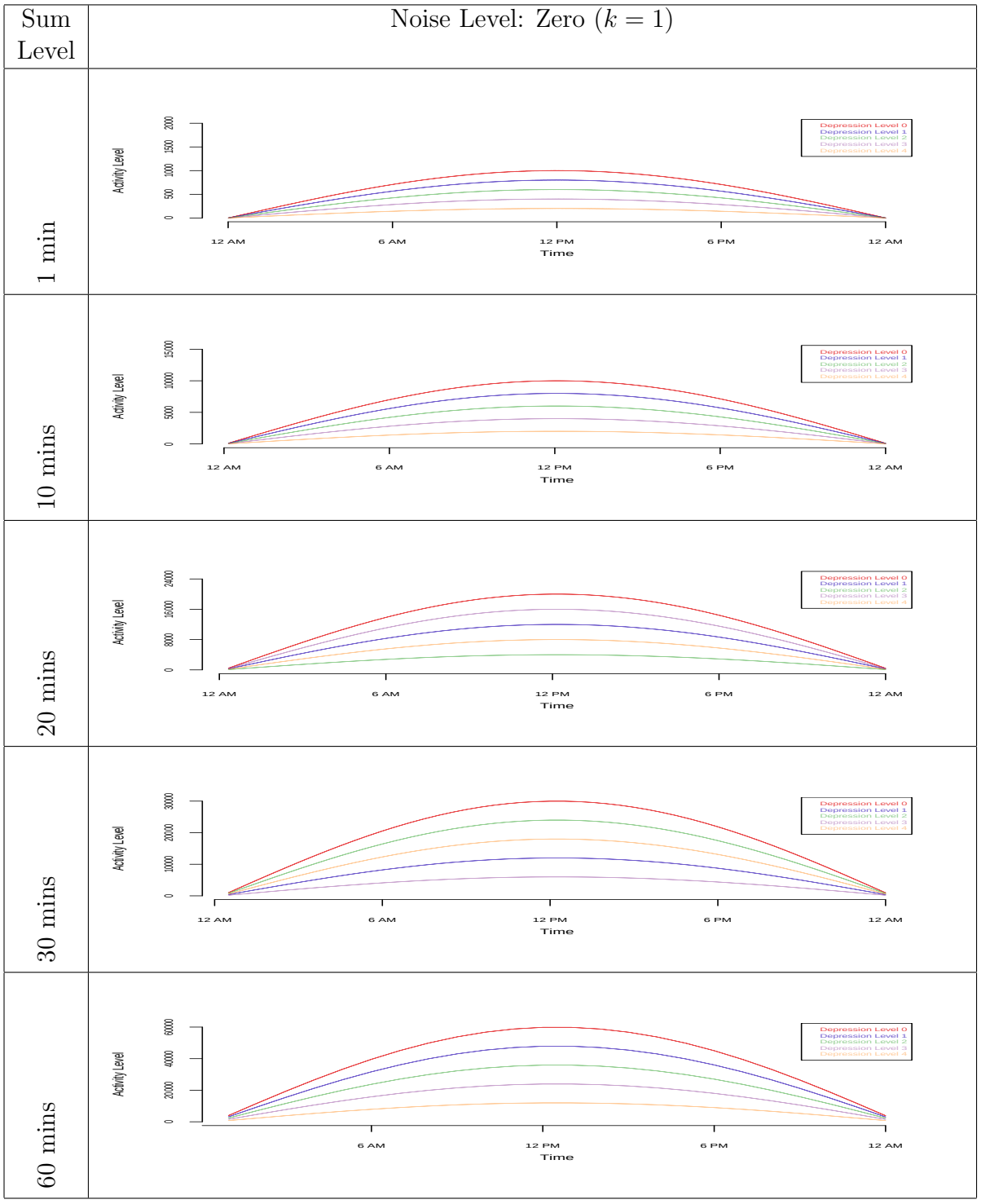


Fig. 2.5: Data Summing with Enveloping (Zero noise): raw data (1 minute) vs. data sums of 10, 20, 30, and 60 minutes

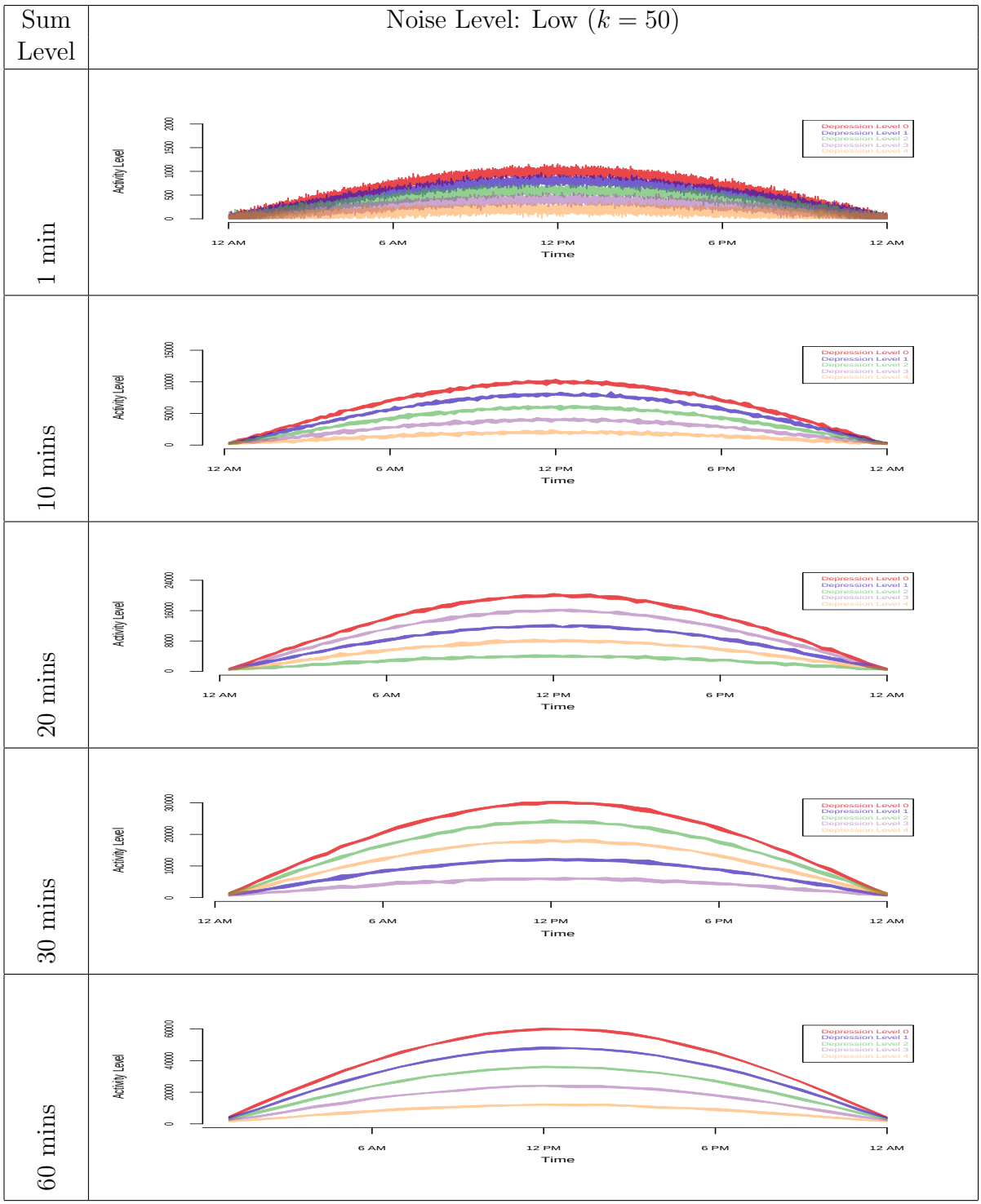


Fig. 2.6: Data Summing with Enveloping (Low noise): raw data (1 minute) vs. data sums of 10, 20, 30, and 60 minutes

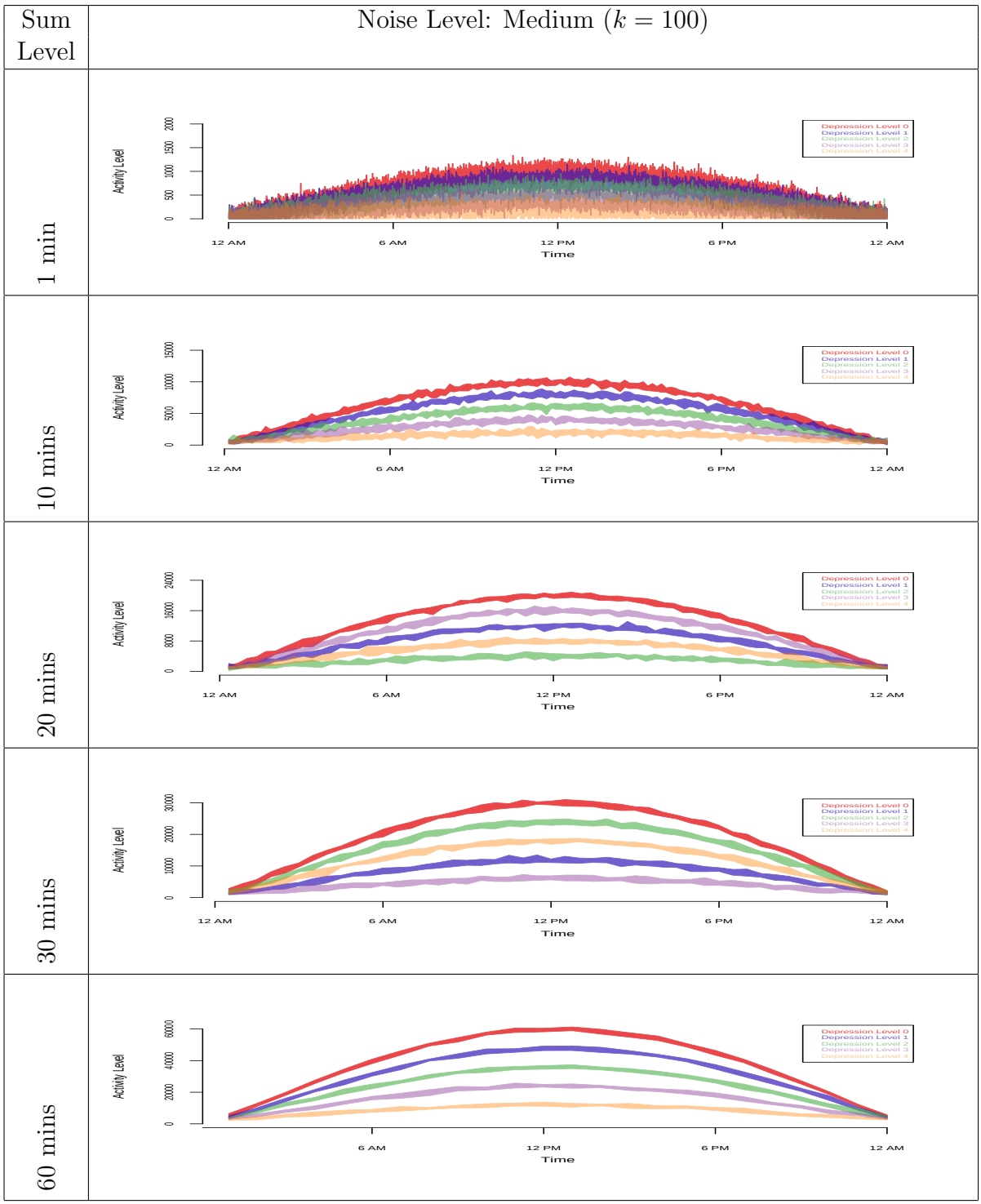


Fig. 2.7: Data Summing with Enveloping (Medium noise): raw data (1 minute) vs. data sums of 10, 20, 30, and 60 minutes

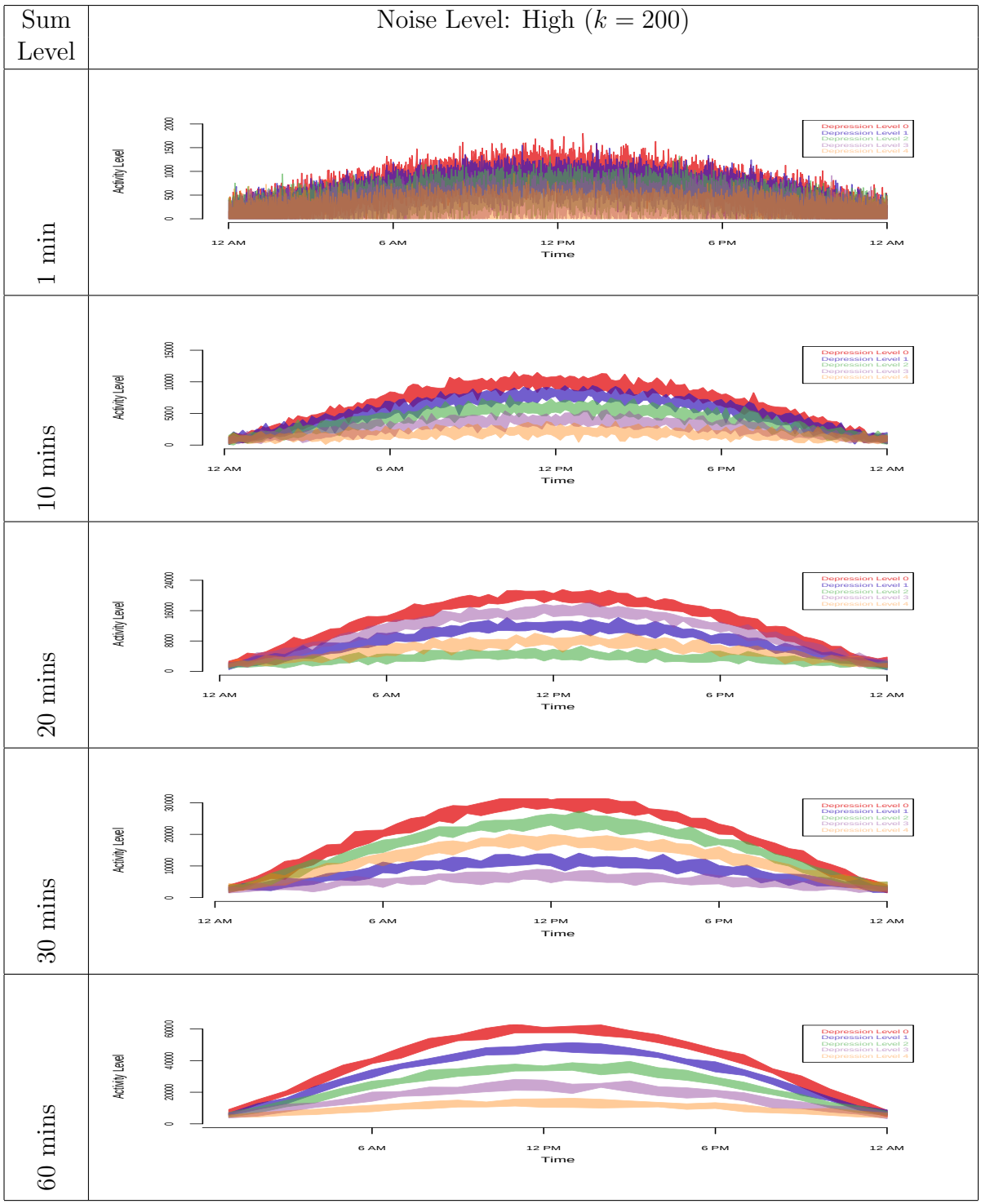


Fig. 2.8: Data Summing with Enveloping (High noise): raw data (1 minute) vs. data sums of 10, 20, 30, and 60 minutes

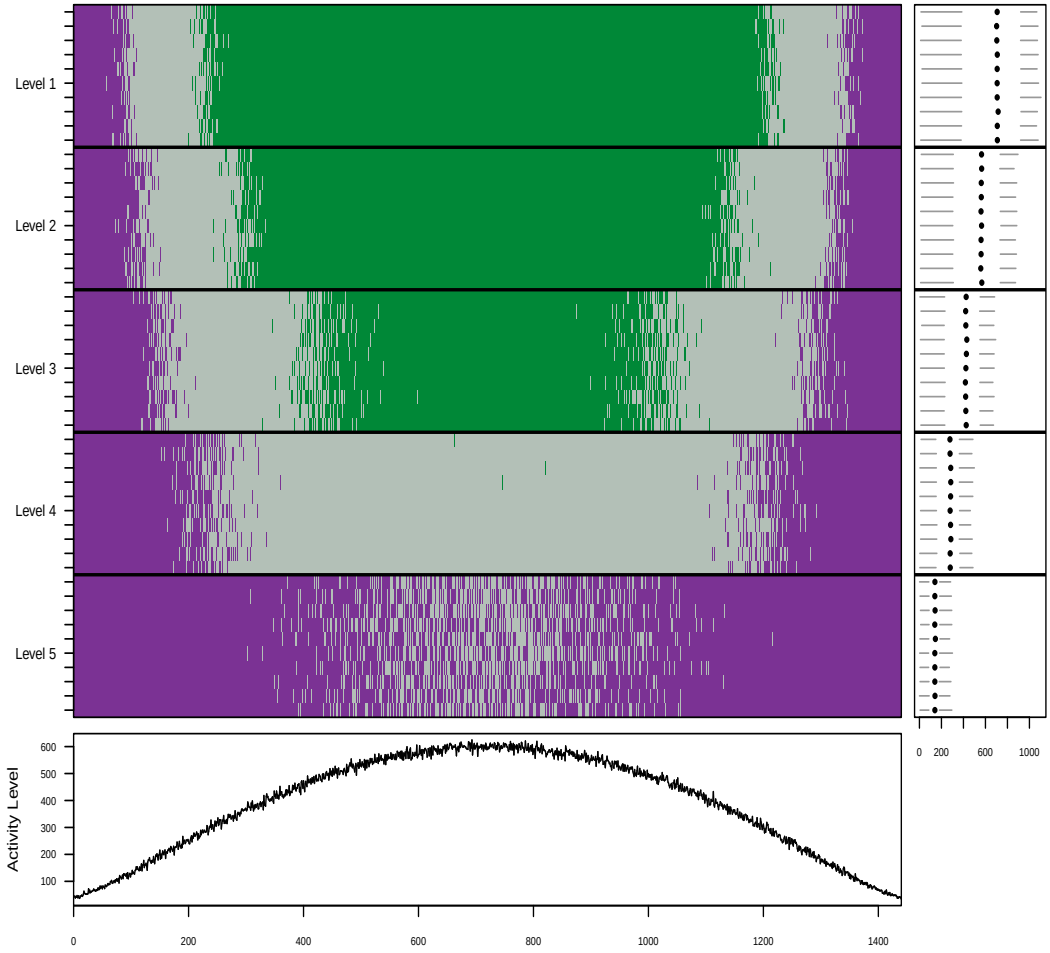


Fig. 2.9: Multivariate Time Series Plot: Simulated data with different levels. The right hand side shows a box plot, and the bottom side shows a plot for the median activity level across all the times series for each time point.

2.5 Actigraphy Data

The real data used in this chapter is based on a small sample of 55 patients with insomnia, sleep apnea, or restless leg syndrome, collected at the Washington University Sleep Medicine Center. Two types of data were collected for each patient: actigraphy level data and depression level data. (For more information about this data, please refer to Ding et al. (2011).) The actigraphy level data were collected via an actigraph device manufactured by the Mini Mitter Company Incorporated Mini

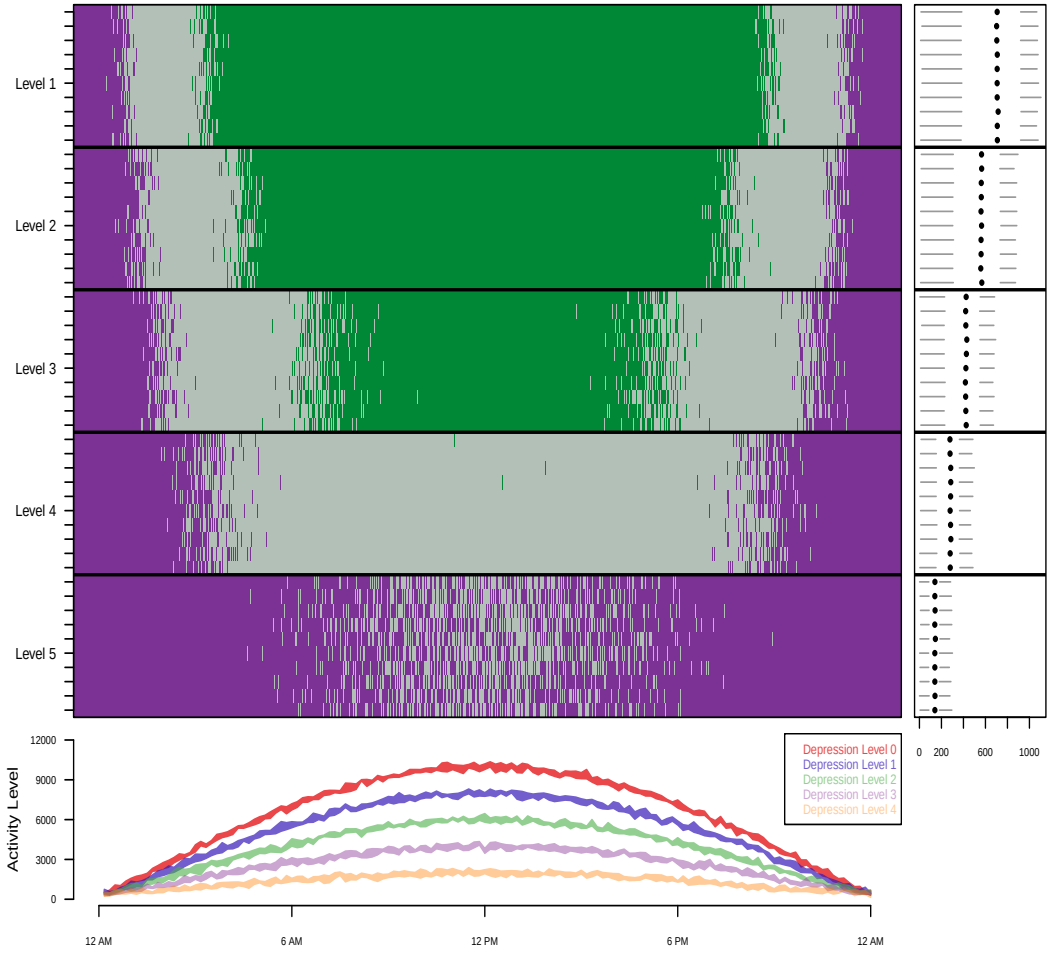


Fig. 2.10: Multivariate Time Series Plot: Simulated data with different levels. The right hand side plot shows a box plot, and the bottom side shows a plot for the $25^{th} - 75^{th}$ percentile envelopes.

Mitter Company Incorporated. (2005) that the patient wore on his/her wrist for a period of seven days. Some of the actigraphs collected data every 15 seconds, and others collected data every minute, but for the purpose of this study, we aggregated the 15 seconds level data into one minute level data. The depression level data were collected to investigate patterns in activity levels in different patient subgroups. Each patient filled out the Patient Health Questionnaire (PHQ-9) (Kroenke and Spitzer, 2002), one of several existing ways to evaluate the level of depression. On the PHQ-9

scale, the higher the depression score, the more depressed the patient is. The following are some descriptive statistics for the collected data in terms of demographics and depression levels:

- Gender: 17 males, 38 females
- Depression level: 15 patients with no depression (Level 0), 13 with mild depression levels (Level 1), 15 have moderate depression levels (Level 2), 8 have moderately severe depression levels (Level 3), and 4 are severely depressed (Level 4).

2.6 Techniques Applied to Actigraphy Data

2.6.1 Density-based Plots

Figures 2.11 and 2.12 show the density-based plots for the actigraphy data we described in Section 2.5 for groups of patients with different depression levels (Figure 2.11) and gender (Figure 2.12), respectively. These plots look very rugged. It is difficult to compare the activity patterns for these groups of patients. The "disadvantage" of such plots is that we have to separate the groups into different plots. We can see that patients with very high depression levels (Level 4) are active during the night and have low activity levels early during the morning while the other four groups (Levels 0, 1, 2, and 3) have normal activity pattern- active during the day and are passive during the night. This kind of plot does not show us clearly if there is a difference in activity levels of some groups.Thus, to obtain a clearer picture for all of the groups, we can look at the cumulative sums plots for the actigraphy data. These plots show accumulated activity obtained by adding up activity counts as one moves across the horizontal time axis from midnight (far left) to midnight 24 hours later (far right) (Sharif et al., 2010). They are helpful to show the total activity of a group up to a particular time of the day. Figures 2.13 and 2.14 show the density-based plots for the cumulative sums for actigraphy data based on depression levels

(Figure 2.13) and gender (Figure 2.14), respectively. People with higher depression levels accumulate higher activity counts during the night and early morning (Figure 2.13). Also, females accumulate higher activity counts than males (Figure 2.14).

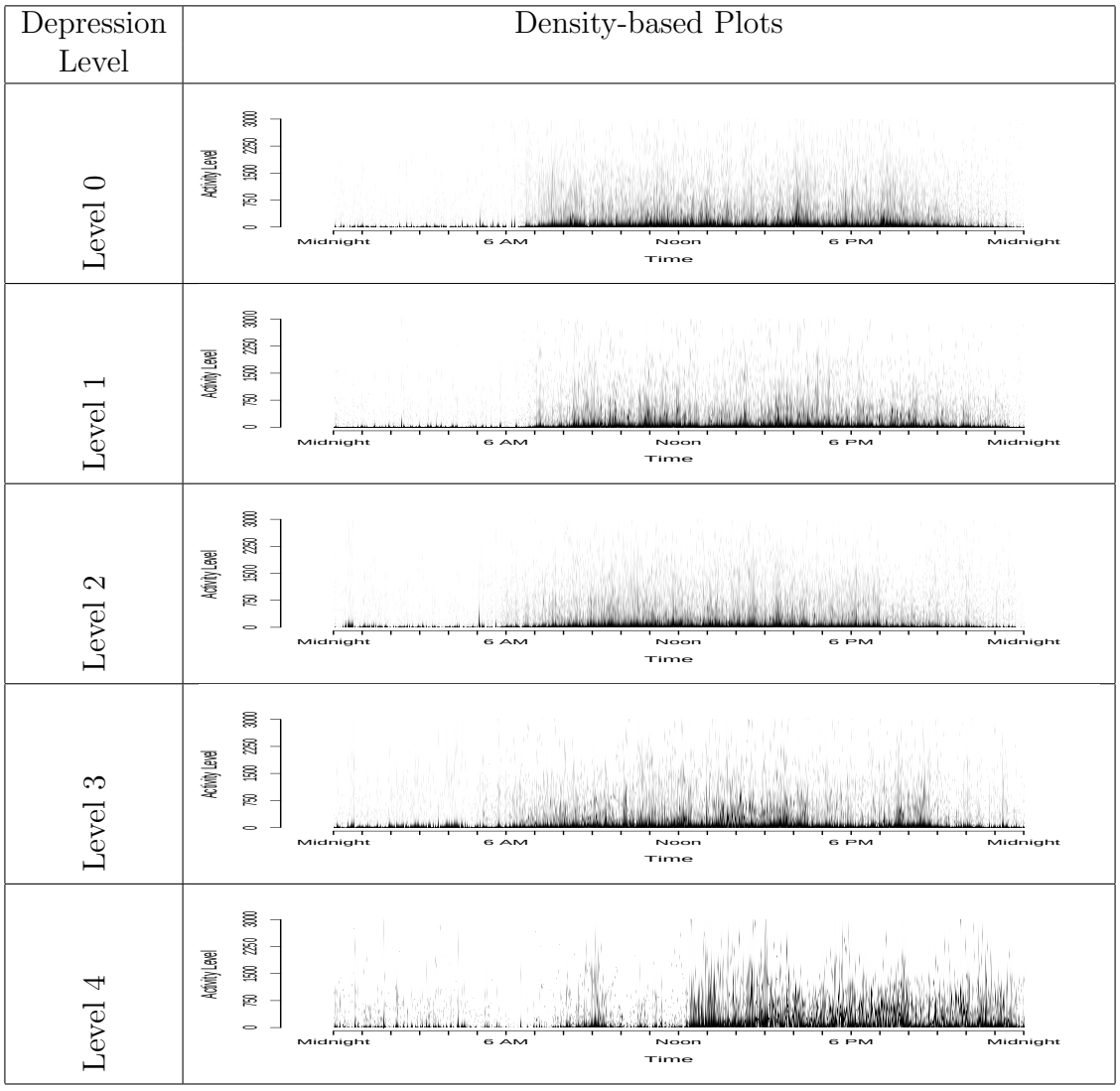


Fig. 2.11: Density-based Plots: Actigraphy data grouped by patients' depression levels

2.6.2 Data Enveloping

Figures 2.15 and 2.16 show the actigraphy data of the 55 patients clustered according to the patients' depression level (Figure 2.15) and gender (2.16), respectively.

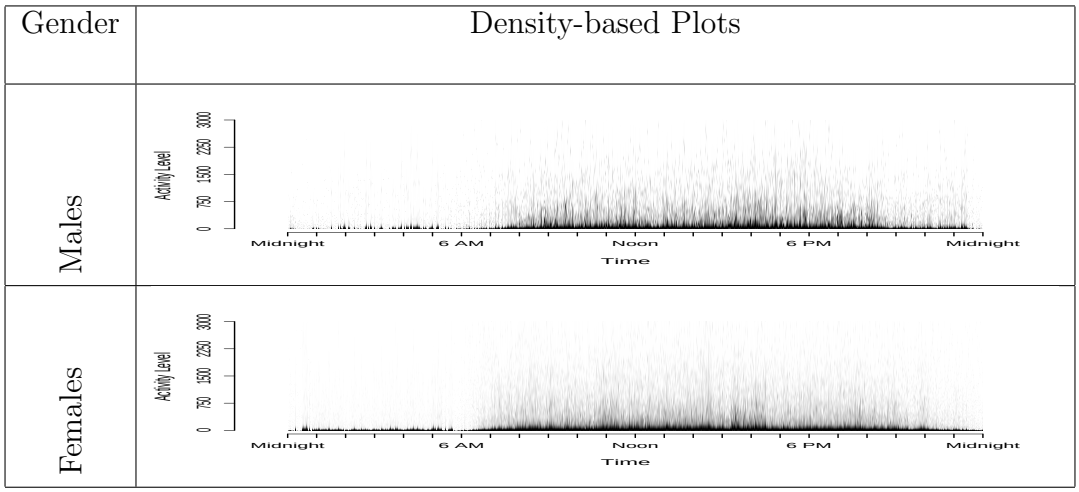


Fig. 2.12: Density-based Plots: Actigraphy data grouped by patients' gender

Figures 2.15 (top) and 2.16 (top) show a $25^{th} - 75^{th}$ percentile envelope for the raw actigraphy data, while Figures 2.15 (bottom) and 2.16 (bottom) show a narrower $40^{th} - 60^{th}$ percentile envelope. Figure 2.15 shows that patients with high depression levels are most active during the night compared to the other groups of patients, and this group is least active during the day. It is not easy to compare the other groups of patients even when we at look a narrower envelope (see Figure 2.15 (bottom)). As for gender, (Figure 2.16), men and women in general have the same activity pattern - active during the day, and rest during the night (from 12 am until 6 am).

Figures 2.17 and 2.18 show the cumulative sums plots for the accumulated sums of actigraphy clustered according to depression levels (Figure 2.17) and gender (Figure 2.18), respectively. Figure 2.17 (top) uses a $25^{th} - 75^{th}$ percentile envelope which shows some distinction between the five groups of patients with different depression levels. As anticipated, patients at Level 0 depression have the highest accumulated activity during the whole day, followed by patients at Level 1, then Level 2, then Level 4. It seems that there is a high variability for the Level 3 depression patients. To see a clearer picture, we plotted the $40^{th} - 60^{th}$ percentile envelope (Figure 2.17 (bottom)). This plot shows that there might be an outlier in the group with depression Level 3 that was affecting the variability of this group. Opposite to what we anticipated,

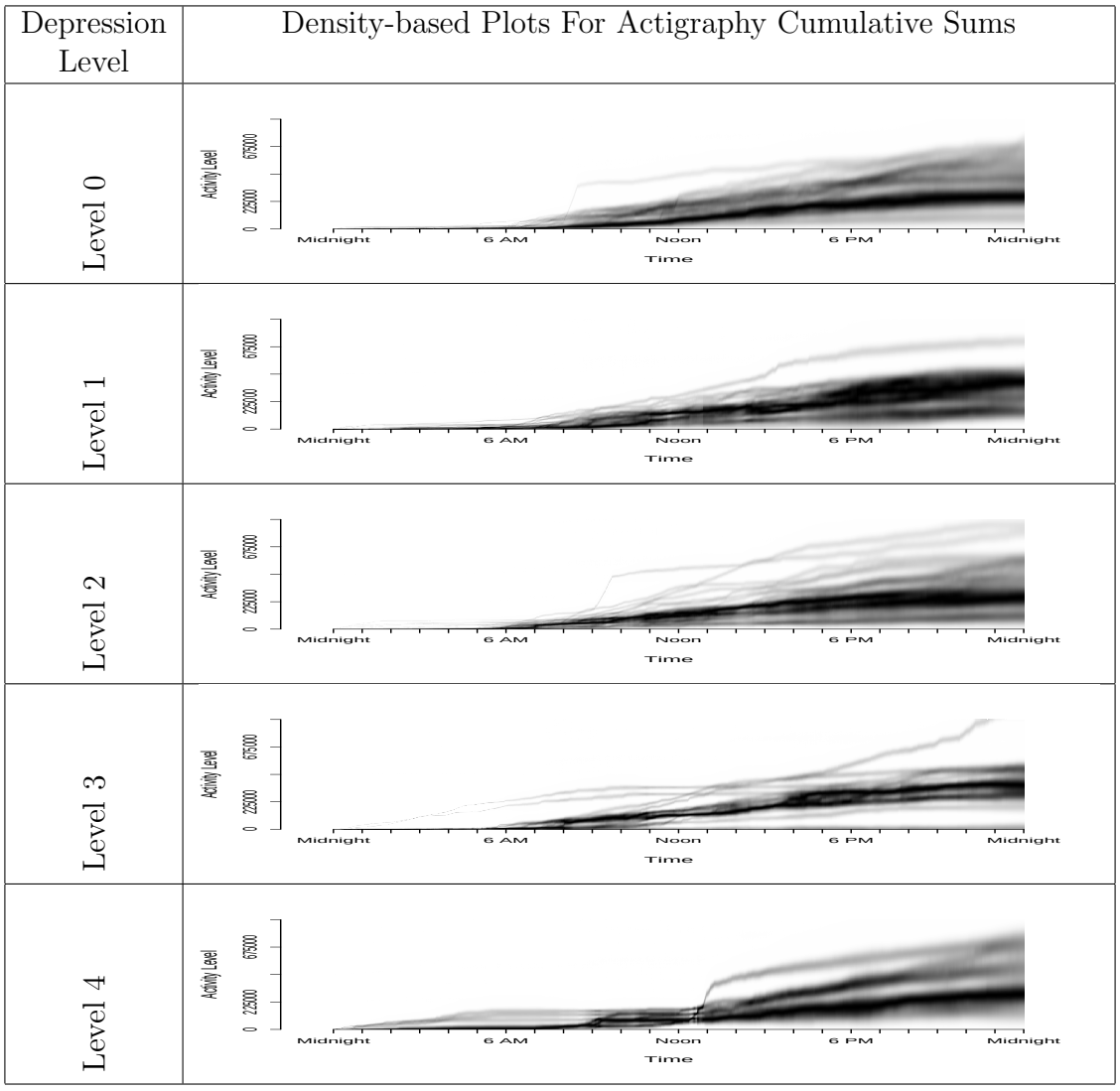


Fig. 2.13: Density-based Plots: Cumulative sum plots for actigraphy data grouped by patients' depression levels

patients that were diagnosed with depression Level 3 have the highest activity. This result might be due to the small sample of just 8 patients with depression Level 3. Looking at the gender grouping for the cumulative sums plots (Figure 2.18), we can see that women have higher accumulated activity counts than men.

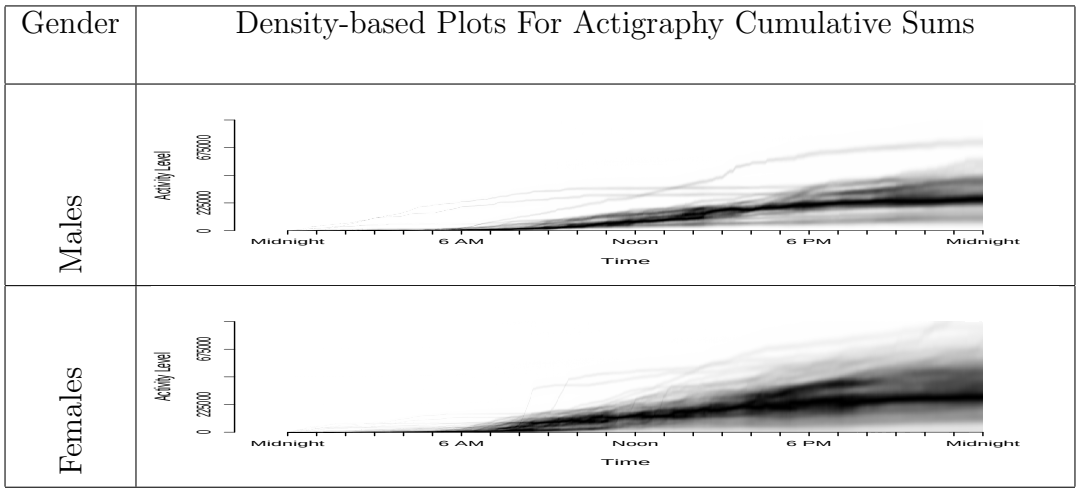


Fig. 2.14: Density-based Plots: Cumulative sum plots for actigraphy data grouped by patients' gender

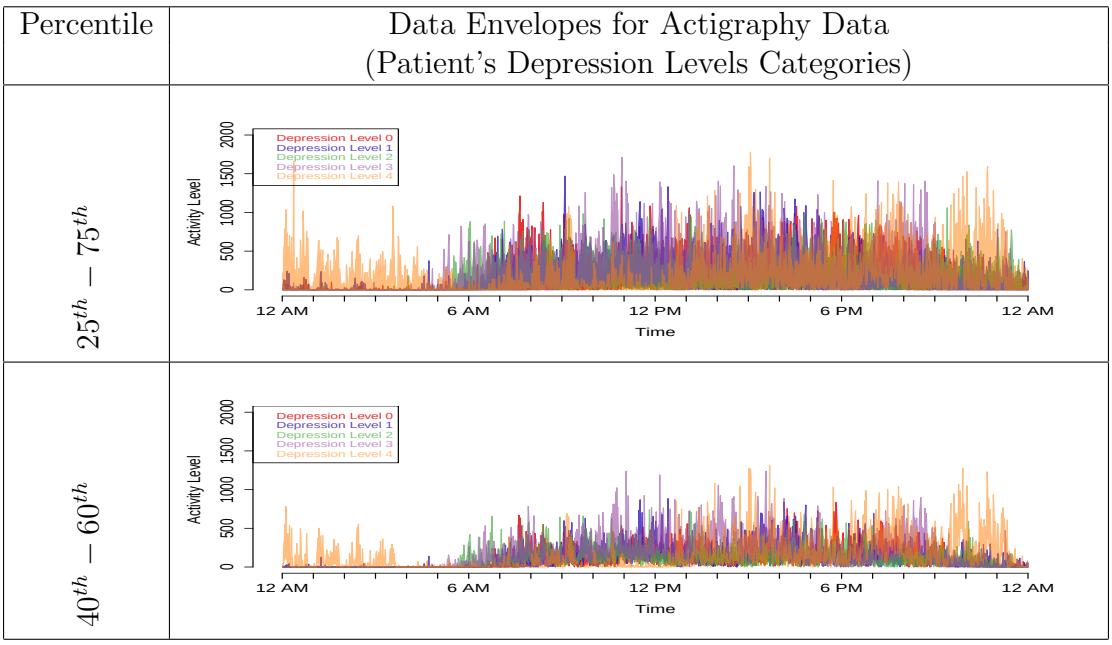


Fig. 2.15: Data Envelopes: $25^{th} - 75^{th}$ percentile vs. $40^{th} - 60^{th}$ percentile envelopes for actigraphy data grouped by patients' depression levels

2.6.3 Data Summing

In Figures 2.15 and 2.16, the actigraphy plots had lots of spikes and it was hard to see the patterns for each class of patients even after we applied enveloping

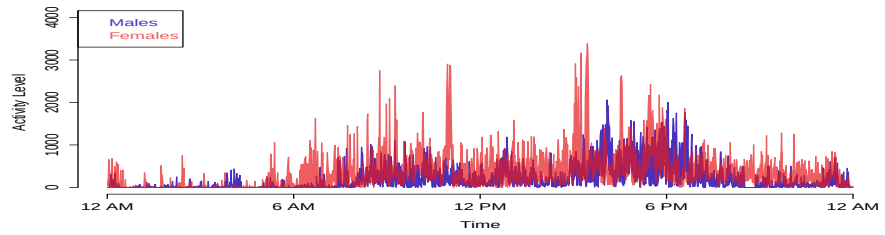
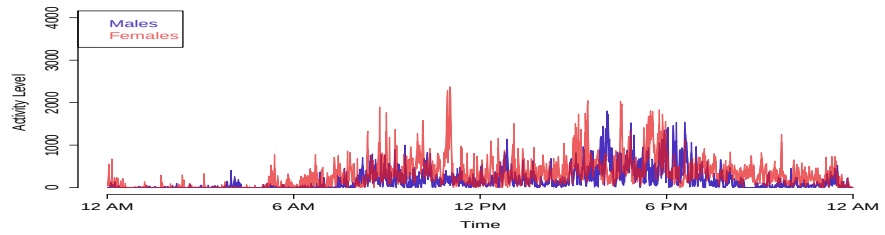
Percentile	Data Envelopes for Actigraphy Data (Patient's Gender Categories)
$25^{th} - 75^{th}$	
$40^{th} - 60^{th}$	

Fig. 2.16: Data Envelopes: $25^{th} - 75^{th}$ percentile vs. $40^{th} - 60^{th}$ percentile envelopes for actigraphy data grouped by patients' gender

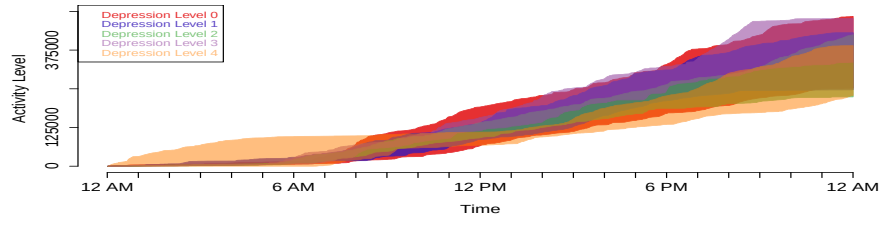
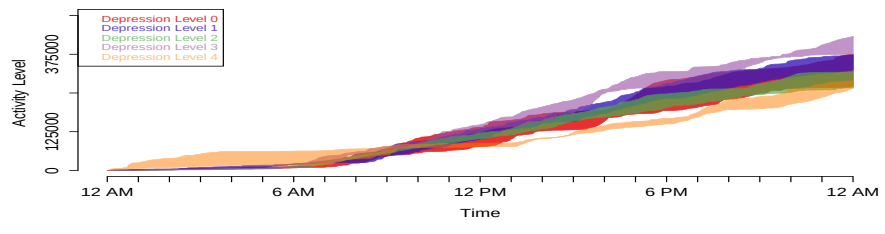
Percentile	Data Envelopes for Cumulative Sums of Actigraphy Data (Patient's Depression Levels Categories)
$25^{th} - 75^{th}$	
$40^{th} - 60^{th}$	

Fig. 2.17: Data Envelopes: $25^{th} - 75^{th}$ percentile vs. $40^{th} - 60^{th}$ percentile envelopes for cumulative sums of actigraphy data grouped by patients' depression levels

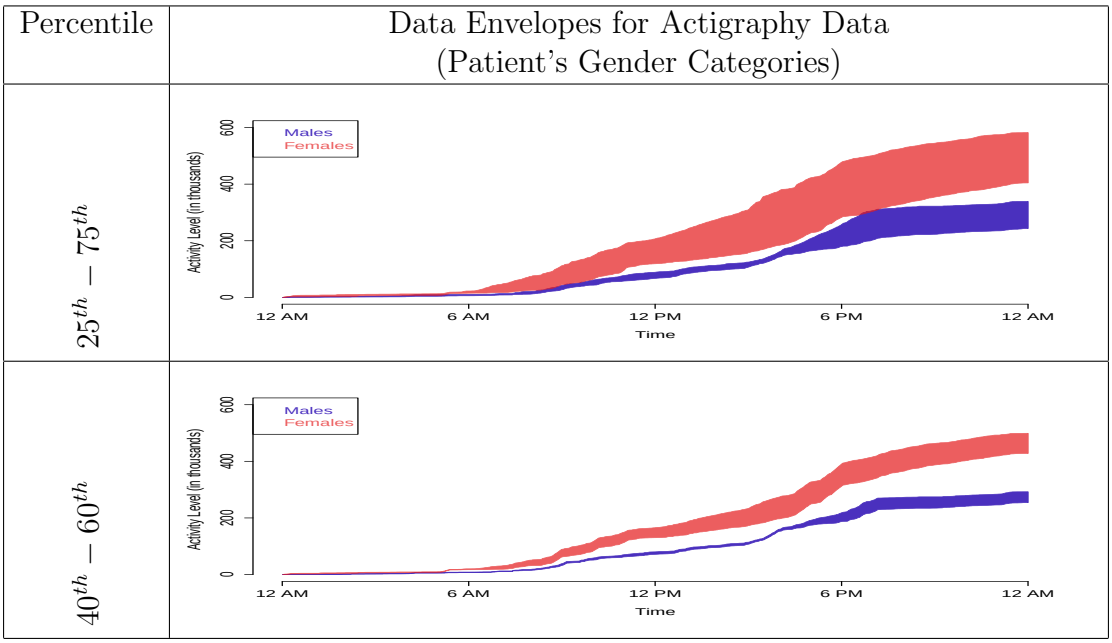


Fig. 2.18: Data Envelopes: $25^{th} - 75^{th}$ percentile vs. $40^{th} - 60^{th}$ percentile envelopes for cumulative sums of actigraphy data grouped by patients' gender

techniques. To make the plots look smoother and obtain a better separation among different patients' groups, we aggregated the data and summed it over time in addition to using enveloping techniques. Figures 2.19 (top) and 2.20 (top) show $25^{th} - 75^{th}$ percentile envelopes but with data summing every 20 mins. In other words, the activity levels for each patient are accumulated into one observation every 20 mins. This technique, combined with enveloping, helps to smooth the plot. Figures 2.19 (bottom) and 2.20 (bottom) show the actigraphy data in a $40^{th} - 60^{th}$ percentile envelope for all patients' groups based on depression levels and gender, respectively.

2.6.4 Multivariate Time Series Plots

Figure 2.21 shows the multivariate time series plot for all patients. There are 55 horizontal time series strips in the image plot (top left) divided into five depression categories that are separated with bold horizontal black lines. For this dataset, it is not clear whether patients with different depression levels have different activity

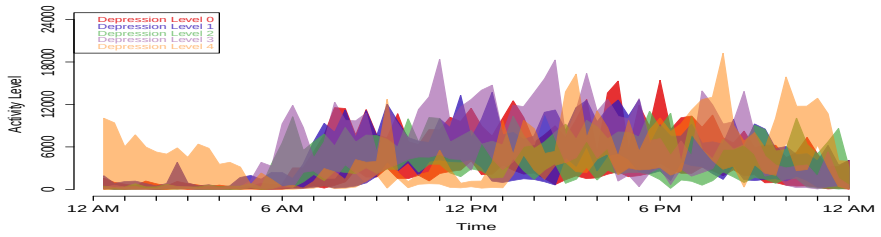
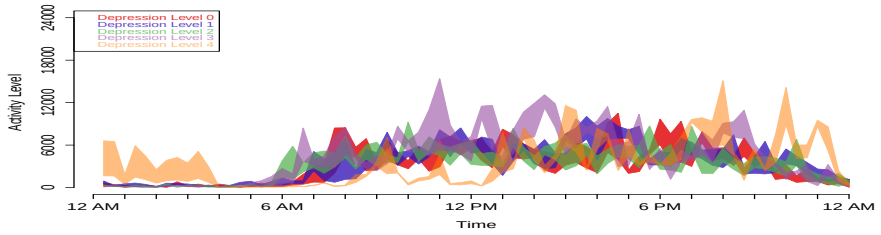
Percentile	Data Sums with Envelopes for Cumulative Sums of Actigraphy Data (Patient's Depression Levels Categories)
$25^{th} - 75^{th}$	
$40^{th} - 60^{th}$	

Fig. 2.19: Data Sums with Envelopes: $25^{th} - 75^{th}$ percentile vs. $40^{th} - 60^{th}$ percentile envelopes for actigraphy data grouped by patients' depression levels

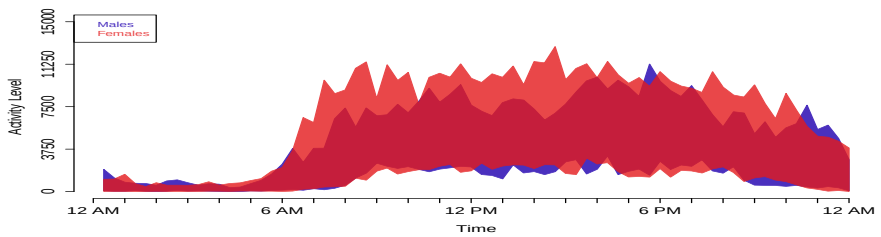
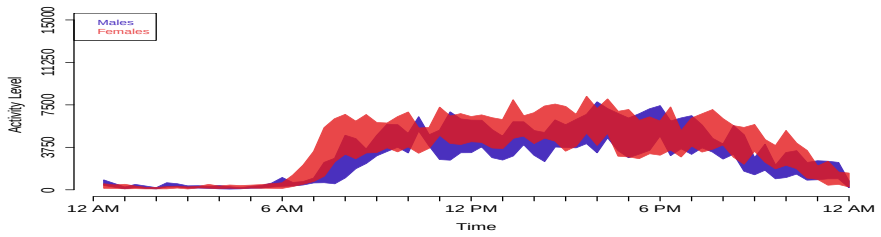
Percentile	Data Sums with Envelopes for Actigraphy Data (Patient's Gender Categories)
$25^{th} - 75^{th}$	
$40^{th} - 60^{th}$	

Fig. 2.20: Data Sums with Envelopes: $25^{th} - 75^{th}$ percentile vs. $40^{th} - 60^{th}$ percentile envelopes for actigraphy data grouped by patients' gender

levels due to the high noise. From the box plots on the right hand side of the image plot, we can see that the variation in activity levels for patients with depression levels 1 and 3 is high when compared to that of patients with levels 2 and 4. This might be due to the presence of outlier data.

Figures 2.21 and 2.22 show the same image plot and the same box plots, but the bottom parts are different. Multivariate time series plots can be augmented with any of the plots discussed earlier. Figure 2.21 shows the median activity level across all the time series for each time point, while Figure 2.22 shows five regular time series plots where each represents the activity level for one of the five depression levels.

2.7 Discussion

In this chapter, we presented exploratory data analysis (EDA) techniques to reduce noise and irregularities in visualizing large functional datasets, without using any smoothing method, in order to reveal interesting patterns and trends in these data. Four main techniques were introduced: (i) density-based plots, (ii) data enveloping methods, (iii) data summing over time, and (iv) multivariate time series plots.

These techniques were applied to raw actigraphy data for a small sample of 55 patients with insomnia, sleep apnea, or restless leg syndrome. We wanted to study how the activity of patients with different depression levels and genders vary. We also looked at simulated actigraphy-like data to see how these techniques work with data classes that are clearly separated from each other.

With density-based plots (Figures 2.11, 2.12, 2.13, and 2.14), we were able to see that patients with very high depression levels (Level 4) are active during the night and have low activity levels early during the morning while the four groups (Level 0, 1, 2, and 3) have normal activity pattern- active during the day and are passive during the night. This kind of plot does not show clearly if there is a difference in activity levels of some groups. Looking at the the density plots for cumulative sums data, we can see that people with higher depression levels accumulate higher activity

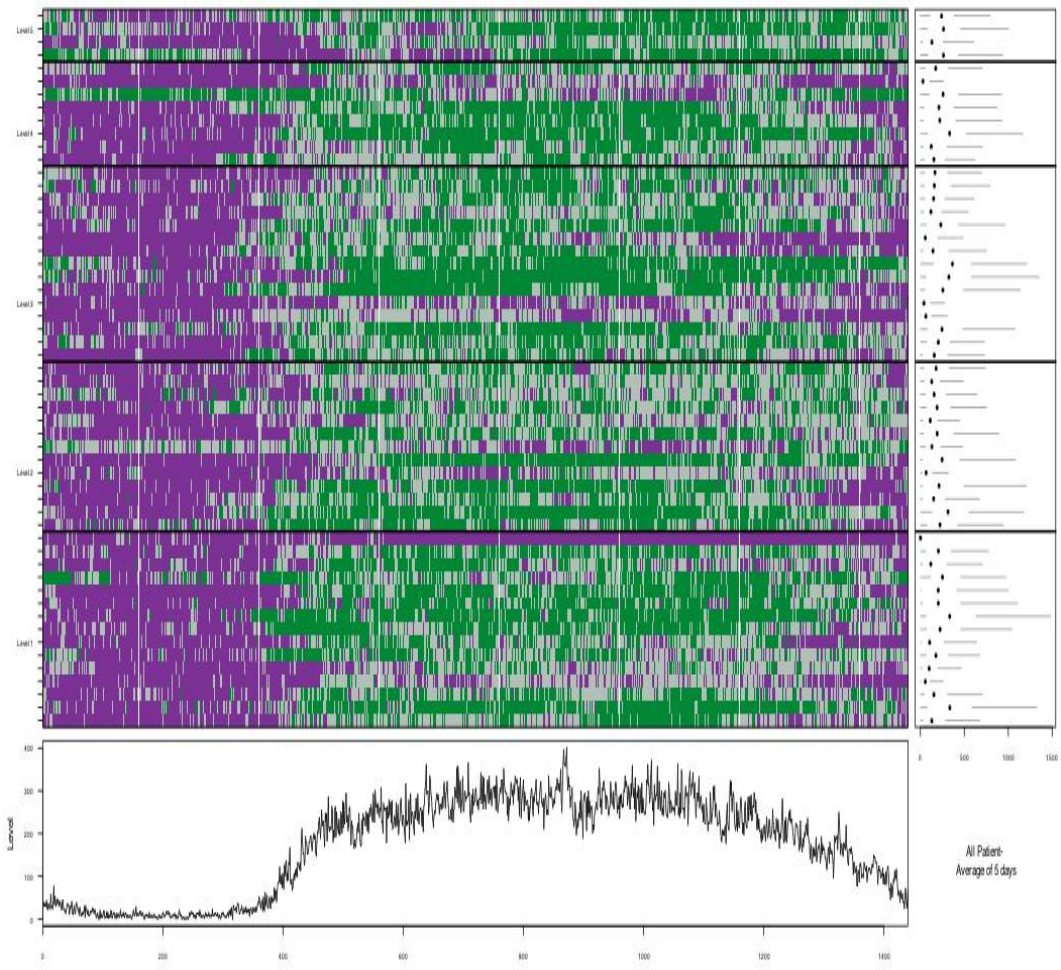


Fig. 2.21: Multivariate Time Series Plot: Actigraphy data with different levels of depression -low(level 1) up to high (level 5). The right hand side plot shows a box plot, and the bottom side shows a plot for the median activity level across all the times series for each time point.

counts during the night and early morning. Also, females accumulate higher activity counts than males. Density-based plots could be helpful to smooth the look of such data and thus make it easy to detect trends over time. This tool has a disadvantage that it's hard to detect clusters for different groups in the data on one plot, and thus we need to graph each group separately.

The data enveloping technique (Figures 2.15, 2.16, 2.17, and 2.18) proved to be very useful in comparing different groups in the data especially when we look at

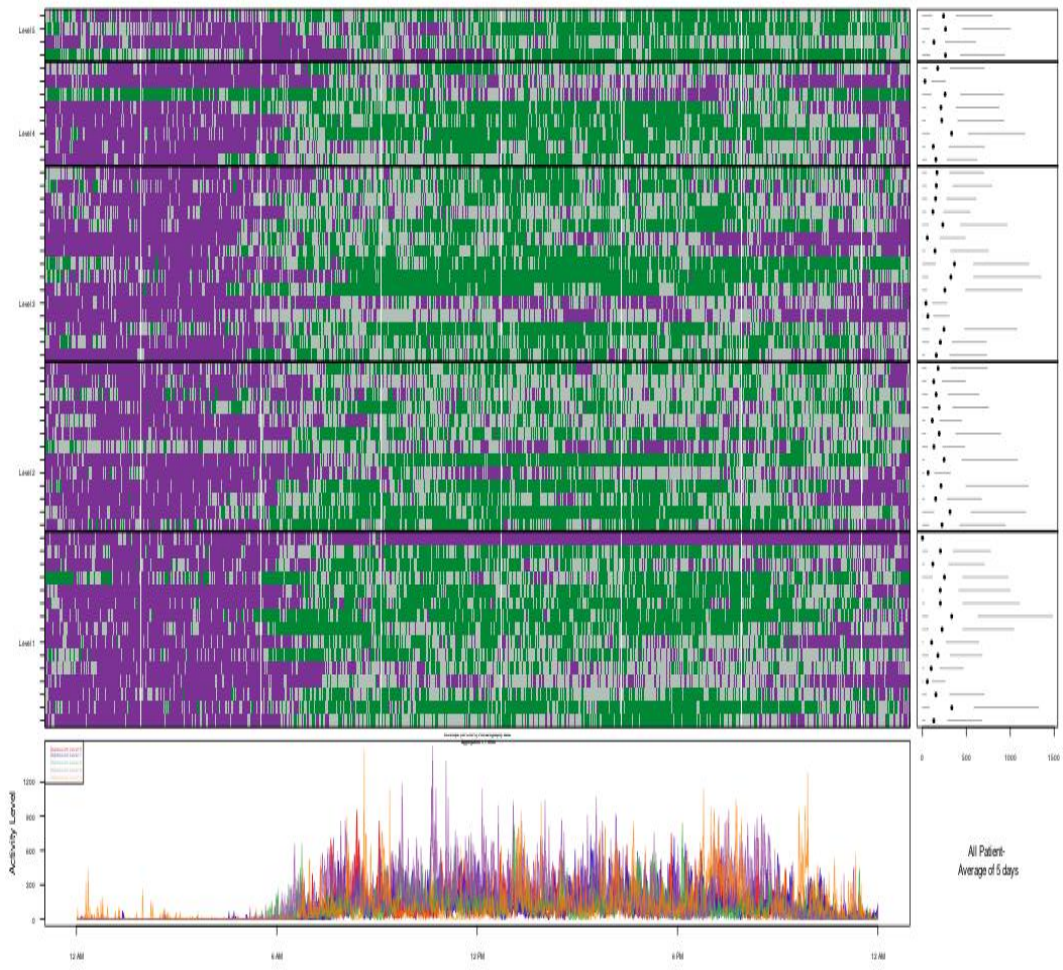


Fig. 2.22: Multivariate Time Series Plot: Actigraphy data with different levels of depression -low(level 1) up to high (level 5). The right hand side plot shows a box plot, and the bottom side shows a plot for the $25^{th} - 75^{th}$ percentile envelopes.

the cumulative sums plots. As anticipated, patients at Level 0 depression have the highest accumulated activity during the whole day, followed by patients at Level 1, then Level 2, then Level 4. It seems that there is a high variability for the Level 3 depression patients. With the data enveloping technique, we were able to detect in the group with depression Level 3 a possible outlier that was affecting the variability of the whole group, by making it the most active. As for gender, the plots showed that women have higher accumulated activity counts than men.

This data enveloping technique (Figures 2.19, and 2.20) could be combined with the data summing technique to give the data a smoother look by getting rid of all of the spikes. It helps in data reduction, and automatically reduce the number of spikes and make the graph look smoother. If the noise level is very low, then there is no need for summing. On the other hand, 20-minutes sums are enough for medium to high noise actigraphy data.

The last technique introduced was the multivariate time series plot (Figures 2.21, and 2.22) which proved to be a useful tool when comparing more than five or six time series plots, i.e. up to 40 or 50 time series. It also gives a smoother look of different classes.

In Chapter 3, we discuss an R package called ActiVis which bundles all of these techniques. To makes it easy for medical doctors who do not write or run computer code, we created an interactive web application (Chapter 4) that interfaces the ActiVis package.

CHAPTER 3
ACTIVIS: AN R PACKAGE FOR VISUALIZING ACTIGRAPHY DATA

3.1 Introduction

This chapter consists of two main parts: the first part discusses the development process and software design for the visualization tools of actigraphy data that we presented in Chapters 1 and 2, and the second part presents a manual for the actigraphy package that bundles all of these tools together. ActiVis R package consists of a set of utility methods for managing, storing, importing and exporting actigraphy data and results. It also includes the plots discussed in Section 1.2 such as raw data plot, smoothed data plot, velocity plot, acceleration plot, cumulative sums plot, and sorted cumulative sums plot. In addition, it also implements multivariate exploratory data analysis (EDA) techniques, which were discussed in Section 2.2, to reduce noise and irregularities in visualizing large functional actigraphy datasets, without using any smoothing method, in order to reveal interesting patterns and trends in these data. Four main techniques were introduced: (i) density-based plots, (ii) data enveloping methods, (iii) data summing over time, and (iv) multivariate time series plots.

Our preliminary object-oriented (OO) model has been described in Sharif et al. (2010). In this chapter, we present a full model that also utilizes the OO programming approach (Lafore, 2002), and extend the description of the low level interaction among the different software entities. The main purpose behind using the OO paradigm is to make it easier for other programmers to reuse our suggested visualization techniques. For example, this would allow other programmers to create different user interfaces and customize these according to the needs of different users. For those who are not familiar with object-oriented design, the ActiVis R package also includes all functionality written in the usual procedural paradigm.

The implementation of the visualization functions for actigraphy data is done using R (R Core Team., 2012), which is a free software environment for statistical computing and graphics, available at http://www.r-project.org. R compiles and runs on a wide variety of UNIX platforms, Windows, and MacOS. Over the last ten years, R has become one of the most widely used statistical software packages among statisticians and researchers. It currently provides more than 4000 specialized packages.

There exists another R package called Actigraphy (Shannon et al., 2012). While that package deals with functional linear modeling and analysis for actigraphy data, our ActiVis R package provides techniques for visualizing functional actigraphy data. Thus, they have different purposes, but they complement each other.

In Section 3.2, we discuss why we chose R to implement the visualization techniques. In Section 3.3, we introduce the object oriented programming approach, and how we used it in our R implementation. Then, in Section 3.4, we talk about the format of actigraphy files and clinical data files. In Section 3.5, we look at a case study to show how our ActiVis package work, and we finish with a discussion in Section 3.6. Appendix A contains the OO functionality of the ActiVis R package. Appendix B contains the same functionality in the usual procedural paradigm.

3.2 Why to use R?

R is unique because it gives the developer the power to do three things in one single tool:

- Data Manipulation: R allows the data scientist to shape the dataset into a format that could be easily accessed and analyzed by slicing large multivariate datasets. It is also one of the very few tools that has great indexing techniques.

- Data Analysis: Any kind of statistical data analysis could be found in R. R is an open source development tool that is supported by a large community of statisticians, computer scientists, and applied scientists from all disciplines. It has over 4000 packages that implement various statistical analysis tools related

to hypothesis testing, model fitting, clustering techniques, machine learning, and so on.

- Data Visualization: R is "the software" for visualization. There are many on-the-shelf graphic functions in R that are ready to be used.The best part of R is that it gives the developer capabilities to implement any visualization idea for any dataset. In addition, animated and interactive graphs can be implemented easily in R.

In addition to the above mentioned advantages, R is also free, runs on any operating system, and can read data in any format (Excel, CSV, text, xml, json, etc.) Unfortunately, R's main disadvantage is that its graphical user interface (GUI) is limited to command line interactions. The user has to write down the commands to load data, perform statistical analyses and create plots. This might be a huge turn off for some users, but in my opinion, it is the biggest advantage of using R. It gives the developers complete control over the system, and thus, they can build statistical systems that fit their users' needs, and they can even make R interact with other programming languages such as C, C++, java, HTML, PHP, Ruby, etc.

3.3 Object-Oriented Programming

Object-oriented programming is a way of thinking differently from the widely used procedural programming approach. Thus, instead of implementing data structures and procedures, the programmer now implements *objects* that operate through a set of *methods* and are described by a set of *attributes*. In the OO approach, data are and combined with *methods* (which are supposed to have access to the data) into one single component called an *object*. This makes any unauthorized access to the data by a different component of the software almost impossible. The OO approach binds data closely to the methods that operate on them and protects the data from any accidental modifications from outside methods. The problem with procedural

programming lies in the separation between the data and procedures, because this provides a poor model for real world applications (Lafore, 2002).

Similar objects are described via a template called a *class*. *Encapsulation* hides the data from unauthorized operations. In addition, a class can be reused by other programmers working on the same project or in a different application. Another important concept in object-oriented programming is *inheritance*. A new class can inherit the capabilities of any existing class and add more features to it. Gentleman (2009, Chapter 3), following Freidman et al. (2001), states four elements that make up an object-oriented programming language:

- Objects: encapsulate state information and control behavior.
- Classes: describe general properties for groups of objects.
- Inheritance: new classes can be defined in terms of existing classes.
- Polymorphism: a function/method has different behaviors depending on the class of one or more of its arguments.

3.3.1 Object-Oriented Programming in R

The R programming language is widely used among statisticians as a procedural programming language, but it can also be tailored to support object-oriented programming. R has three internal object-oriented systems: S3, S4 (Gentleman, 2009, Chapter 3) and R5 (Wickham, 2012). In addition, there exists an add-on package called R.oo that supports OO development in R (Bengtsson, 2003).

The first OO system, S3, is the easiest to use, but it is not fully object-oriented. There is no formal specification for classes, and hence, there is little control of objects and inheritance (Gentleman, 2009, Chapter 3). Classes are defined as lists. Thus the programmer has to do a lot of checking of arguments to ensure that all instances of a certain class have the correct slots, the correct types of values in those slots, and the

correct class attributes. More details about S3 can be found in Chambers and Hastie (1992, Appendix A).

The second OO system, S4, is more object-oriented than S3. S4, is designed in a way to overcome the weaknesses of S3. There is an explicit definition of classes. In addition, tools have been developed to automatically do the checking of class definitions and properties. Thus, S4 is more stable than S3. More details about S4 can be found in Chambers (1998, Chapters 7 and 8). A comparison between S3 and S4 classes can be found in Chambers (2008, Chapter 6).

The third OO approach is via an R package called R.oo (Bengtsson, 2003). The purpose of this package is to have an object-oriented design and implementation that makes use of references. In other words, this package should help in developing scalable and maintainable code that takes into consideration efficiency in memory usage by allowing objects to be *passed by reference* to other objects. Thus, any change or update to a certain object would be directly reflected on the object itself, instead of creating a new copy of that object and passing it to the other calling object. Another advantage of *passing by reference* results in programming-friendly methods with fewer arguments that have to be specified. The R.oo package extends the S3 system with an extra layer to provide reference variables and a more formal way of defining classes, similar to the one in S4. It also makes object-oriented design and programming in R easier and more robust.

The fourth and most recent OO system is R5 or Reference Classes. Its purpose is very similar to the R.oo package, but it is a built-in part of R (since R 2.12), i.e., there is no need to install any additional package. Objects have attributes that could be accessed and modified by reference without the need to have a new updated copy as in S3 and S4. We decided to use the R5 approach because: (i) it implements a full object-oriented design when compared to the S3 approach; (ii) the classes could be accessed and modified by reference when compared to the S3 and S4 approaches; (iii) there is no need to install any additional package when compared to the R.oo

approach.

There are few R packages on CRAN that use the object-oriented approach in their design. The latest are (i) distrMod (Kohl and Ruckdeschel, 2010), which provides an S4-style implementation of probability models; (ii) simecol package (Petzoldt and Rinke, 2007), which provides an open structure to implement, simulate and share ecological models using S4 class system. Todorov and Filzmoser (2009) developed an object-oriented framework for robust multivariate analysis using the S4 class system. This framework resides in some existing packages and includes a set of algorithms for computing robust multivariate location and scatter, robust methods for principal component analysis, and robust linear and quadratic discriminant analysis.

3.3.2 Object-Oriented Design for ActiVis R Package

The ActiVis R package consists of eleven classes. These are outlined in the class diagram in Figure 3.1. A class diagram displays the structure of the system by showing its classes, their attributes and methods, and the relationships between the classes.

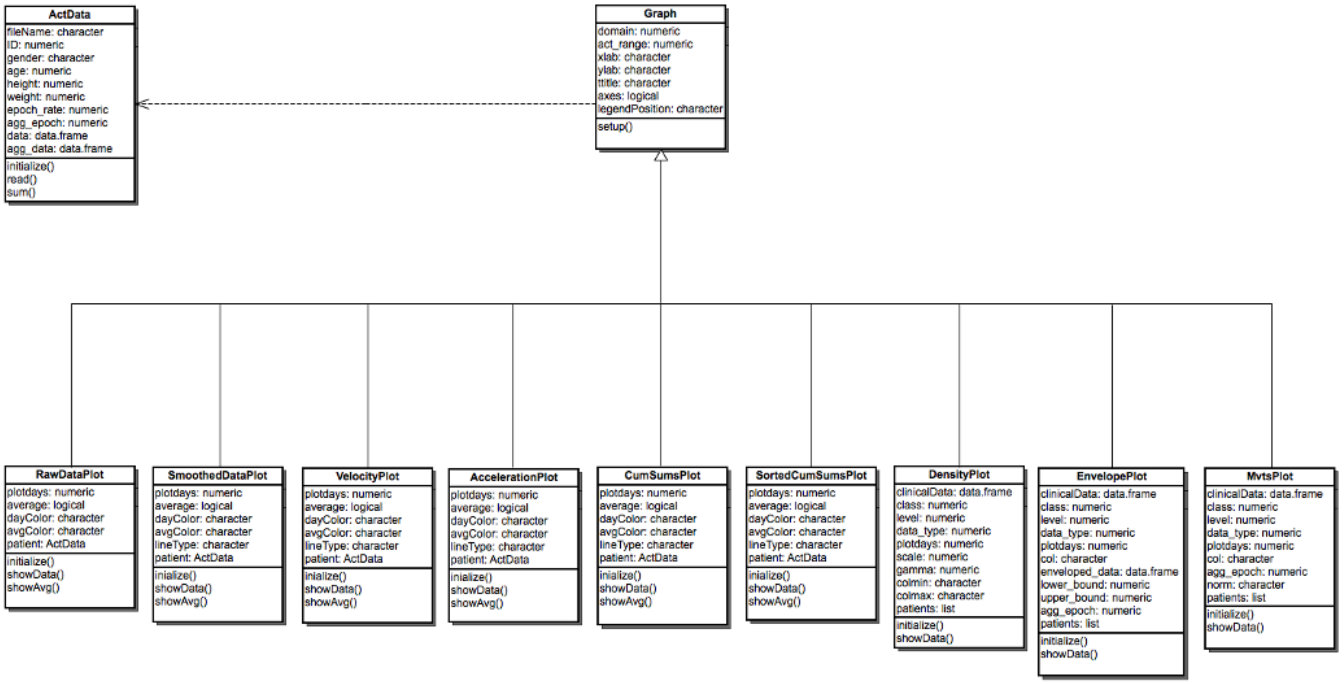


Fig. 3.1: Class diagram for the ActVis R package. Shown are eleven different classes and the interactions among them. The dashed arrow represents a dependency relation, while the non-dashed arrows represent inheritance relations. Inheritance indicates that one of the two related classes is considered to be a specialized form of the other, while a dependency relationship indicates that one class depends on another because it uses it at some point of time

- *ActData*: This class is responsible for reading, storing, and manipulating actigraphy data as well as patient-level data. Table 3.1 describes its 10 attributes and 3 methods.

- *Graph*: This class uses *ActData* and it is responsible for setting up the plotting area for visualizing the actigraphy data. Table 3.2 describes its 7 attributes and 1 method. All of the following classes inherit from *Graph*.

- *RawDataPlot, SmoothedDataPlot, VelocityPlot, AccelerationPlot, CumSumsPlot, SortedCumSumsPlot*: These six classes are for visualizing single-patient data. *RawDataPlot* produces a graph that shows the raw actigraphy data over time (Figure 1.3(a)); *SmoothedDataPlot* produces a graph that shows smoothed curves for the actigraphy data (Figure 1.3(b)); *VelocityPlot* produces a velocity graph, i.e., the first derivative of the smoothed daily activity data, and tells us how the activity level is changing over time (Figure 1.3(c)); *AccelerationPlot* produces an acceleration graph, i.e., the second derivative of the smoothed daily activity data, and it tells us how quickly the changes in velocity are occurring (Figure 1.3(d)); *CumSumsPlot* produces cumulative sums graph, which shows accumulated activity obtained by adding up the activity counts as one moves across the horizontal time axis from midnight (far left) to midnight 24 hours later (far right) (Figure 1.3(e)); and finally, *SortedCumSums* produces a sorted cumulative sums graph, which is obtained by adding up the activity counts from smallest to largest (Figure 1.3(f)).

  All of these classes have the following attributes and methods (Table 3.3). Even though the methods have the same names among the different classes, they do different tasks because of polymorphism in OO programming.

- *DensityPlot*: This class is responsible for plotting density graphs, which help visualize variability and trends for functional data over time. *DensityPlot* has 10 attributes and 2 methods (Table 3.4).

Table 3.1: Description of the attributes and methods for the *ActData* class. The AWC data format is described in Section 3.4

Attributes	**Description**
fileName	This is the name of the file that contains the actigraphy data (AWC format). It is of type "character".
ID	This is the identification number for the patient. It is of type "numeric".
gender	This is the gender of the patient. It is of type "character".
age	This is the age of the patient. It is of type "numeric".
height	This is the height of the patient. It is of type "numeric".
weight	This is the weight of the patient. It is of type "numeric".
epoch_rate	This is the rate for which the actigraph collected the patient's data. It is of type "numeric".
agg_epoch	This field of type "numeric". It represents the aggregation level for the data. It takes values of 1, 10, 20, 30 or 60.
data	This is where the actigraphy data of the patient are stored after being read. It is of type "data.frame".
agg_data	This is where the summed aggregated actigraphy data are stored. It is of type "data.frame".
Methods	
initialize()	This method populates the attributes of the class with default values when instantiated. It is a private method, and it is called by the software when an object of this class is instantiated.
read()	This method reads data from data files in AWC format, and stores them in *ActData* attributes.
sum()	This method is used to aggregate (sum up) raw actigraphy data.

- *EnvelopePlot*: This class is responsible for computing, storing, and graphing data envelopes. It visually subsets the dataset into different classes by drawing bands around each class of data observations, and then fill each with a different color. *EnvelopePlot* has 11 attributes and 2 methods (Table 3.5).

- *MvtsPlot*: This class is responsible for generating multivariate time series plots. They have the same concept of image plots, and they become handy when

Table 3.2: Description of the attributes and methods for the *Graph* class.

Attributes	**Description**
domain	This field stores the time range of the data during the day. The default is the whole day, and it is of type "numeric".
act_range	This field stores the range of the actigraphy data to be shown on the graph. It is of type "numeric".
xlab	This field is for the horizontal axis label of the graph. It is of type "character".
ylab	This field is for the vertical axis label of the graph. It is of type "character".
title	This field is for the title of the graph. It is of type "character".
axes	This field is used to indicate whether to draw the default axes or not. It is of type "logical".
legendPosition	This field is to tell where to put the legend on the graph. It is of type "character".
Methods	
setup()	This method sets up a basic graph with no data.

we need to compare more than five or six time series data. *MvtsPlot* has 9 attributes and 2 methods (Table 3.6).

Table 3.3: Description of the attributes and methods for the *RawDataPlot, SmoothedDataPlot, VelocityPlot, AccelerationPlot, CumSumsPlot*, and *SortedCumSumsPlot* classes.

Attributes	**Description**
plotdays	This is a vector of the days to plot for each patient. It is of type "numeric".
average	This is a "logical" type field. If it is 'T', then the average activity levels curve will be overlaid on top of all plotted day curves.
dayColor	This is a vector of colors for the day curves. It is of type "character". This is either a built-in R color name (one of colors()) or an RGB hex value.
avgColor	This is the color for the average curve. It is of type "character". This is either a built-in R color name (one of colors()) or an RGB hex value.
lineType	This is a vector of line types for the day curves. It is of type "character". This is used for *RawDataPlot* only.
patient	This is of type "ActData". It stores a reference to the *ActData* object of a patient.
Methods	
initialize()	This method populates the attributes of the class with default values when instantiated. It is a private method, and it is called by the software when an object of this class is instantiated.
showData()	This method shows the curves for the days selected. The graph has to be "setup()" first.
showAvg()	This method shows only the curve for the average of selected days. The graph has to be "setup()" first.

3.4 Data

3.4.1 Actigraphy Data

The actigraphy data files (AWC files) that are generated by the actigraph devices are in ASCII text format. They contain three kinds of data: (1) the subject's (patient) information, (2) the device (actigraph) information, and (3) the raw actigraphy data. All of this information is stored in one column. Table 3.7 shows the format of AWC files: the first 11 lines are patient and device information, and the rest of the file

contains the raw actigraphy data.

3.4.2 Clinical Data

In order to look at the activity levels in different patient subgroups with different depression levels, our sample of 55 patients were asked to fill out the Patient Health Questionnaire (PHQ-9) (Kroenke and Spitzer, 2002), one of several existing ways to evaluate the level of depression. In addition to these depression scores, demographic patient data were also collected (e.g., gender and age). All of these data were compiled into a CSV format (Comma Separated Values) file (Table 3.8).

3.5 Case Study: Using the ActiVis R Package

This section serves as a short tutorial for using the ActiVis R package directly from R. It is divided into two parts: (1) single patient actigraphy visualizations, and (2) multiple patients actigraphy visualizations. A full user manual can be found in Appendix D. In Chapter 4, we will describe how the ActiVis functionality can be invoked via a Graphical User Interface (GUI).

3.5.1 Single Patient Actigraphy Visualization

Instantiating an ActData Object

In order to read and store actigraphy data for patients, we first need to instantiate an object of class *ActData*. This is done using the following line of code in R:

```
#inialize karli's actigraphy object
karli <- ActData$new()
```

In object-oriented programming, an object can access a class method using the '$' sign. In this example, we defined a new patient object "karli" of class "*ActData*", and we initialized it by calling the "new()" method that is defined internally in R5 for

the purpose of creating objects. This method will automatically call the "initialize()" method to set attributes to given defaults. Figure 3.2 shows the first 48 lines of the actual content of Karli's AWC file (Karli.AWC).

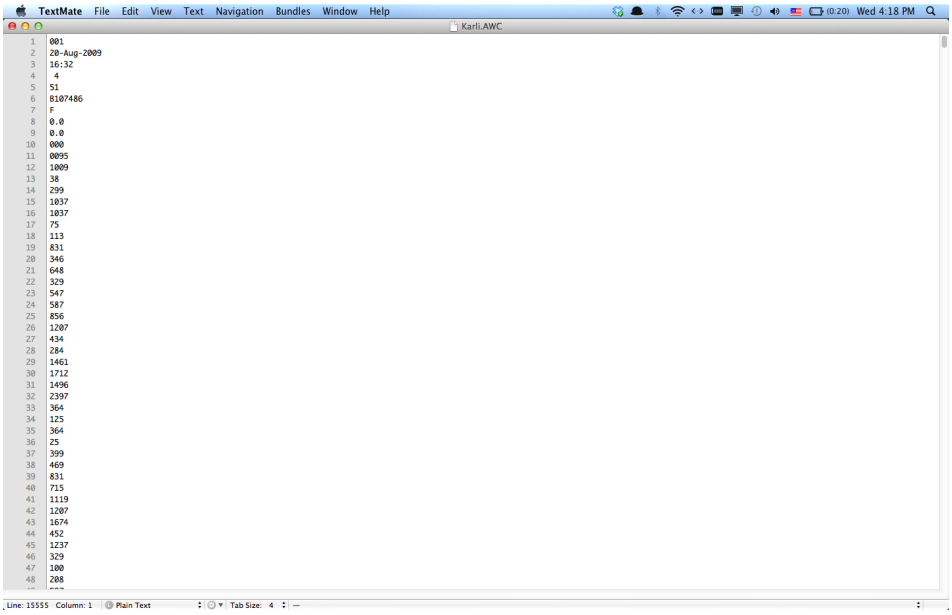


Fig. 3.2: Actual content of Karli's AWC file (Karli.AWC). Shown are the first 48 lines.

Reading and Aggregating Actigraphy Data

In order to populate the fields of Karli's *ActData* object with Karli's data, the user needs to specify the path to Karli's AWC file, and then use the *read()* method using the following code:

```
#specify path of the awc file for Karli
setwd("path/goes/here")
karli$fileName <- "Karli.AWC"
#read the patient data
karli$read()
```

After issuing the command for the read() method, the other fields in Karli's *ActData* object are populated. To read the value of the fields in Karli's object, a user

can use the '$' notation. The following code will print Karli's height, weight, and the first few values of the actigraphy data.

```
karli$height
karli$weight
head(karli$data)
```

The user has also an option to aggregate the raw actigraphy data. For example, the following code sums up every 10 activity levels into one:

```
karli$agg_epoch <- 10
karli$sum()
```

Instantiating a RawDataPlot, SmoothedDataPlot, Velocity, Acceleration, Cumsums, and Sorted Cumsums Objects

In order to use the visualizations from the ActiVis package, the user needs to set up a *Graph* object. The following chunk of code initializes a *Graph* object of class *RawDataPlot* for Karli's object of class *ActData*, and then plots it.

```
#create a new raw data plot object
karli_raw <- RawDataPlot$new()
karli_raw$legendPosition <- "topright"
karli_raw$act_range <- c(0, 4000)
#setup the graph by calling the graphics device
karli_raw$setup()
#show the average of data for the days
karli_raw$average <- T
#what days to plot?
karli_raw$plotdays <- 2:3
#store Karli's object in 'patient' field
karli_raw$patient <- karli
```

```
#show the data on the raw data plot
karli_raw$showData()
```

The resulting plot is shown in Figure 3.3.

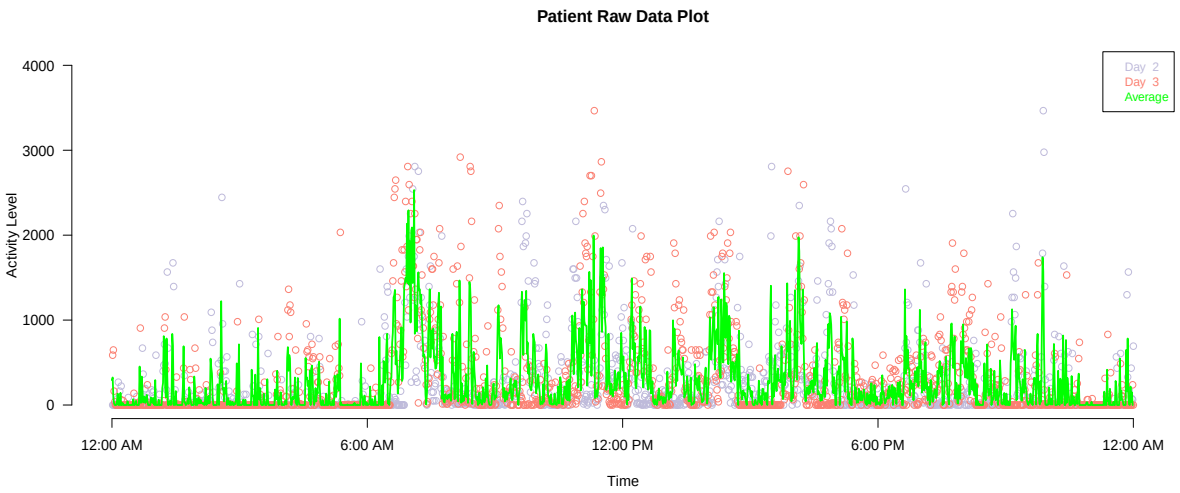


Fig. 3.3: Raw data plot for of Karli's actigraphy data (days 2 and 3) , with the average of these two days (solid green line).

Instantiating and plotting *SmoothedDataPlot, VelocityPlot, AccelerationPlot, CumSumsPlot,* and *SortedCumsumsPlot* objects could be done similar to the *RawDataPlot* in this example.

3.5.2 Multiple Patient Actigraphy Visualization

Reading Actigraphy and PHQ_9 Scores Data

To visually explore actigraphy data for multiple patients, we need to create an *ActData* object for each of the patients to store their data. The following chunk of code reads actigraphy data files for six patients into six different *ActData* objects and stores all of them into a vector data structure. In addition, PHQ_9 scores are read and stored into a data-frame structure in R.

```
files <- c("001 31Aug09.AWC", "002_01SEP09.AWC", "003_01Sep09.AWC",
```

```
   "004 09Sep09.AWC","005_17Sep09.AWC", "006_10SEP09.AWC")

 allPatients <- NULL
 for (file in files){
   print(file) #to show the progress
   Patient <- ActData$new()
   Patient$fileName <- file
   Patient$read() #read the patient data
   Patient$agg_epoch <- 10
   Patient$sum() #aggregating actigraphy data using a 10 mins epoch
   allPatients <- c(allPatients, Patient)
  }

 clinicalData <- read.csv("PHQ9_scores_new.csv")
 clinicalData <- as.data.frame(
   cbind(ID   = clinicalData$ID,
         Level  = clinicalData$level.new,
         gender = clinicalData$Gender
   )
)
```

Instantiating a DensityPlot, EnvelopPlot, and MvtsPlot Objects

The code for calling density, envelope, and Mvts plots is similar, once the data are stored into a vector of *ActData* objects. The following chunk of code instantiates an *MvtsPlot* object for all patients whose data was read in the previous chunk of code. The classification of patients is based on gender, and the actigraphy data used are the "sums" and not the "raw" data.

```
allPatients_mvts_plot <- MvtsPlot$new()
allPatients_mvts_plot$plotdays <- 2:6
#classification level (0: gender, 1: depression)
allPatients_mvts_plot$class <- 0
# ActSums and not raw data
allPatients_mvts_plot$data_type <- 0
allPatients_mvts_plot$title <- "MVTS Plot"
allPatients_mvts_plot$setup()
#show your data on the density plot
allPatients_mvts_plot$patients <- allPatients
allPatients_mvts_plot$clinicalData <- clinicalData
allPatients_mvts_plot$showData()
```

The resulting plot is shown in Figure 3.4.

3.6 Discussion

In this chapter, we presented the ActiVis R package for the visualization of actigraphy data that we discussed in Chapter 2. To implement this package, we chose the R development environment because it is very flexible, and it also gives the developer capabilities to implement any visualization idea for any dataset. We have almost 6000 lines of R code. To have a maintainable, scalable, and reusable code, we followed the object-oriented approach in our design and implementation, by encapsulating both data and methods that act on the data in one object. This feature makes it easy to maintain, update, and/or reuse the code in new systems.

The ActiVis R package needs to be tested using software engineering techniques to validate and verify that it meets the requirements of the CRAN repository. In addition, the techniques proposed in this work could be adapted to visualize any kind

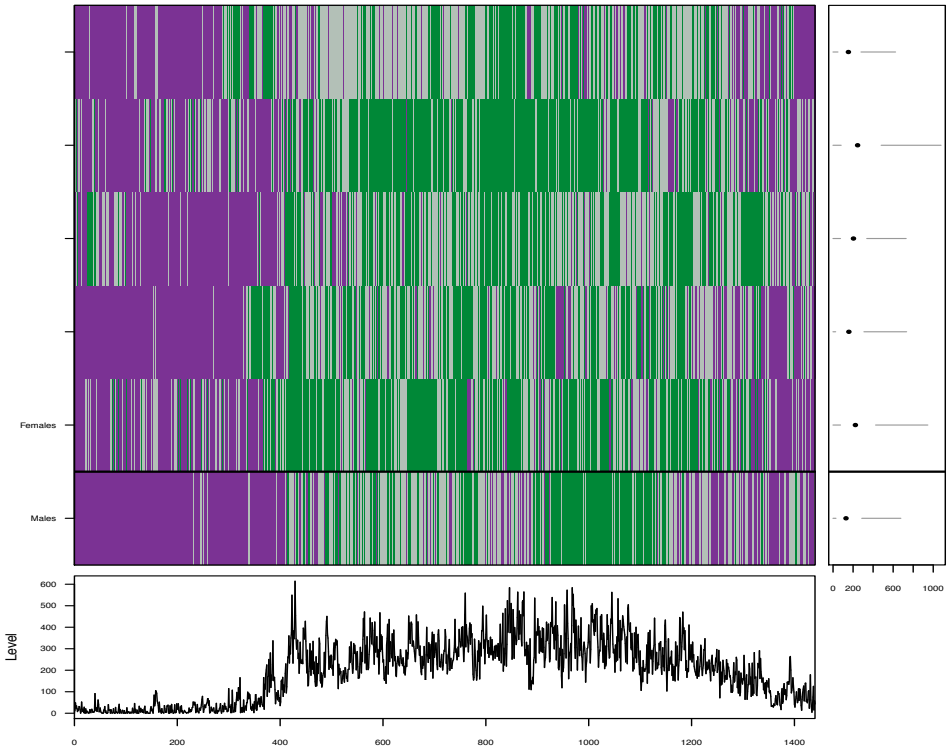


Fig. 3.4: MVTS plot for 6 patients (5 females, and 1 male).

of functional data, and not just actigraphy data. In the future, we will test ActiVis techniques on different functional datasets.

Many end users of the ActiVis R package can be expected to have a background in medical field, sports, or they might be individuals interested in observing their daily activity levels. Most of them won't have enough background in dealing with R code, so we created an adaptive web application that interfaces the ActiVis package (see Chapter 4).

Table 3.4: Description of the attributes and methods for the *DensityPlot* class.

Attributes	**Description**
clinicalData	This field stores a reference to the clinical data used to classify different patients (e.g. PHQ-9 scores). It is of type "data.frame".
class	It is a "numeric" field used to indicate how to classify data (0 for gender classification, and 1 for depression classification).
level	It is a "numeric" field to indicate which class level to plot. Classification could be done based on either depression (levels go from 0 to 4) or gender classification (0: male, 1: female).
data_type	It is a "numeric" field to indicate what type of data to use: 0: raw, 1: cumsums, 2: sorted raw, 3: sorted cumsums.
plotdays	This is a vector of the days to plot for each patient. It is of type "numeric".
scale	Proportion of colmax to shade the maximum density, for example scale=0.5 with colmax="black" for a mid-grey color.
gamma	Gamma correction to apply to the color palette. The default of 1 should give an approximate perception of darkness proportional to density, but this may need to be adjusted for different displays. Values of gamma greater than 1 produce colors weighted towards the lighter end, and values between 0 and 1 produce darker colors.
colmin	This is either a built-in R color name (one of colors()) or an RGB hex value. It represents the lower shade color for the density plot.
colmax	This is either a built-in R color name (one of colors()) or an RGB hex value. It represents the higher shade color for the density plot.
patients	This is a list of objects to plot. These objects are of type *ActData*.
Methods	
initialize()	This method populates the attributes of the class with default values when instantiated. It is a private method, and it is called by the software when an object of this class is instantiated.
showData()	This method shows the density plot for the days selected. The graph has to be "setup()" first.

Table 3.5: Description of the attributes and methods for the *EnvelopePlot* class.

Attributes	**Description**
clinicalData	This field stores a reference to the clinical data used to classify different patients (e.g. PHQ-9 scores). It is of type "data.frame".
class	It is a "numeric" field to indicate how to classify data (0 for gender classification, and 1 for depression classification).
level	It is a "numeric" field to indicate which class level to plot. Classification could be done based on either depression (levels go from 0 to 4), or gender classification (0: male, 1: female).
data_type	It is a "numeric" field to tell what type of data to use: 0: raw, 1: cumsums, 2: sorted raw, 3: sorted cumsums.
plotdays	This is a vector of the days to plot for each patient. It is of type "numeric".
col	This is either a built-in R color name (one of colors()) or an RGB hex value.
enveloped_data	This field is of type "data.frame". It stores the enveloped data for plotting.
lower_bound	This field is of type "numeric" and takes values between 0 and 1. It represents the lower bound of the envelope.
upper_bound	This field is of type "numeric" and takes values between 0 and 1. It represents the upper bound of the envelope.
agg_epoch	This field of type "numeric". It represents the aggregation level for the data. It takes values of 1, 10, 20, 30 or 60.
patients	This is a list of objects to plot. These objects are of type *ActData*.
Methods	
initialize()	This method populates the attributes of the class with default values when instantiated. It is a private method, and it is called by the software when an object of this class is instantiated.
showData()	This method shows the enveloped data plot for the days selected. The graph has to be "setup()" first.

Table 3.6: Description of the attributes and methods for the *MvtsPlot* class.

Attributes	**Description**
clinicalData	This field stores a reference to the clinical data used to classify different patients (e.g. PHQ-9 scores). It is of type "data.frame".
class	It is a "numeric" field to indicate how to classify data (0 for gender classification, and 1 for depression classification).
level	It is a "numeric" field to indicate which class level to plot. Classification could be done based on either depression (levels go from 0 to 4), or gender classification (0: male, 1: female).
data_type	It is a "numeric" field to tell what type of data to use: 0: raw, 1: cumsums, 2: sorted raw, 3: sorted cumsums.
plotdays	This is a vector of the days to plot for each patient. It is of type "numeric".
col	This is either a built-in R color name (one of colors()) or an RGB hex value.
agg_epoch	This field of type "numeric". It represents the aggregation level for the data. It takes values of 1, 10, 20, 30 or 60.
norm	For the normalization, specifying "internal" means that each time series is categorized into colors based on the range of values in each time series individually. Therefore, under this scenario, the same color in two different time series will have two different meanings. If "global" is specified, then each time series will be categorized based on the range of values for the entire collection of time series. In this case, the colors are comparable across series.
patients	This is a list of objects to plot. These objects are of type *ActData*.
Methods	
initialize()	This method populates the attributes of the class with default values when instantiated. It is a private method, and it is called by the software when an object of this class is instantiated.
showData()	This method shows the Mvts plot for the days selected. The graph has to be "setup()" first.

Table 3.7: The format of the AWC data files that are produced by the actigraphy devices (Mini Mitter Company Incorporated., 2005)

Line	**Name**	**Format**	**Description**
1	Subject's Identity	String, e.g. "001"	This is a unique value that identifies each patient.
2	Start Date	dd-mth-yyyy, e.g. 24-Jul-2009	This is the date when the patient started wearing the actigraphy device.
3	Start Time	hh:mm, e.g. 14:25	This is the time of the day when the patient started wearing the actigraphy device.
4	Epoch Rate	Integer, e.g. 1	This represents the interval at which the actigraph is recording data. The rates are: 1 = 15 seconds, 2 = 30 seconds, 4 = 60 seconds.
5	Age	Integer, e.g. 27	This is the age of the patient.
6	Serial Number	String, e.g. B107496	This is the actigraphy device serial number.
7	Gender	Char, e.g. 'M'	This is the gender of the patient.
8	Height	Float, e.g. 179.5	This is the height of the patient in cm.
9	Weight	Float, e.g. 72	This is the weight of the patient in kg.
10	-	-	For technical support use
11	Battery life	Integer e.g. 0012	This shows the actigraphy device battery life remaining in days.

Table 3.8: The format of the CSV clinical data files that are collected by the PHQ-9 questionnaire.

Header	**Format**	**Description**
ID	Numeric, e.g. 1	This is a unique value that identifies each patient.
Gender	Numeric, e.g. 1	This is the gender of the patient (1: Males, 2: Females)
Age	Numeric, e.g. 56	This is the age of the patient.
PHQ9_score	Numeric, e.g. 15	This score tells about the level of depression of a particular patient.

CHAPTER 4
A WEB-BASED STATISTICAL FRAMEWORK FOR THE VISUALIZATION OF ACTIGRAPHY DATA USING R

4.1 Introduction

Many end users of the ActiVis R package described in Chapter 3 can be expected to have a background in medical field, sports, or they might be individuals interested in observing their daily activity levels. Those users are unlikely to know how to deal with computer code written in R (R Core Team., 2012). Furthermore, R does not have a user friendly Graphical User Interface (GUI) with menus and buttons. These users most likely want to focus on the results and graphics, and not on running computer code. This chapter discusses how we developed an adaptive web interface that facilitates the use of our ActiVis R package without the need to write or run any R code.

In Section 4.2, we present about different approaches for building an interface to the R engine. In Section 4.3, we review existing statistical web applications that inspired our work. In Section4.4, we summarize about the different technologies and programming languages we utilized to develop our web interface, and in Section 4.5 we discusses the architecture of the ActiVis system. In Section 4.6, we demonstrate how the web interface works through screen snapshots, and finally, in Section 4.7, we conclude the chapter with some remarks and lessons learned from our experience with building the ActiVis system. Appendix C contains the source code for the GUI for the ActiVis package.

4.2 Graphical Interfaces to R

Initially, we developed two easy-to-use GUI prototypes for the underlying actigraphy functionality in R. The first is via a windows interface, and the second is via a web interface.

4.2.1 Windows Interface to R

Knowing that many people nowadays use Microsoft Office, we bridged R to Microsoft Excel using RExcel and Rcmdr. RExcel (Baier and Neuwirth, 2007; Heiberger and Neuwirth, 2009) is an add-in for Microsoft Excel. The main intention behind developing RExcel is to provide a general user-friendly interface to R, thus making it easy to non-programmers to use R. RExcel allows data transfer between R and Excel in both directions. It also allows to run R code directly from Excel cells and to write Excel macros to automatically run R commands. Most importantly, it makes R accessible through a menu-driven environment via the Rcmdr R package (Fox, 2005). Figure 4.1 shows a snapshot of Microsoft Excel with an "Actigraphy" drop-down menu added to the RExcel tool bar. The "control panel" window (Figures 4.2 and 4.3) is a prototype of what pops up when a user selects an option from the "Actigraphy" menu in Microsoft Excel. In Figure 4.2, the user produced the raw data plot (see Section 1.2) of the activity levels for a subject with id number 003 for days 2, 3, and 4. In Figure 4.3, the user produced a smoothed data plot (see Section 1.2) for the same subject in order to better visualize this subject's different activity patterns for these days.

4.2.2 Web Interface To R

The main idea behind developing a web interface to the underlying actigraphy functionality in R is to make our new tools available to wide audience. Thus, the decision was to focus on developing a web interface that would be accessible by any one at anytime and from any place, without the hassle of installing any software (a.k.a

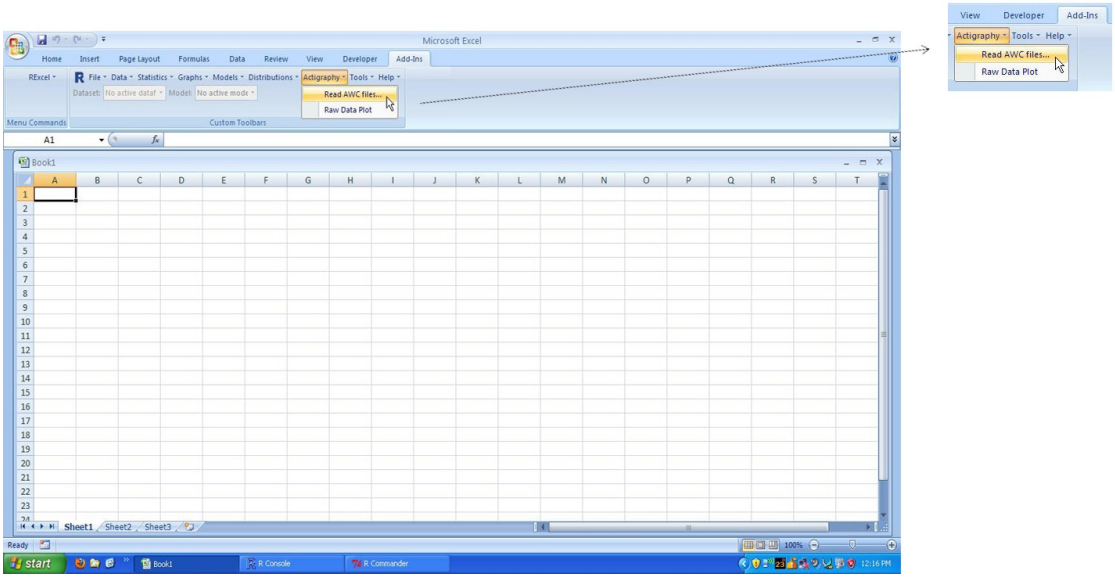


Fig. 4.1: Actigraphy plugin in Microsoft Excel

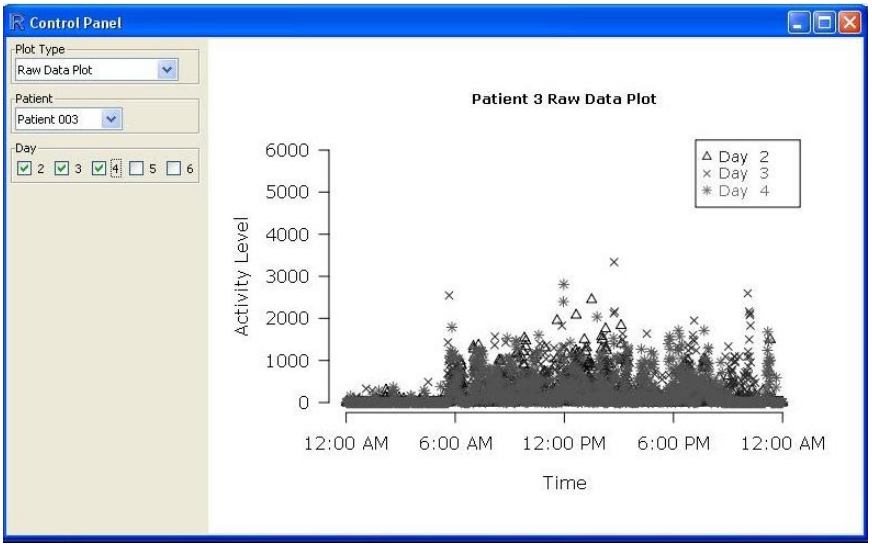


Fig. 4.2: A control panel window based on the Rcmdr package showing the raw data plot for a certain subject for three different days

Excel and R). The users only need a standard web browser to access our software, and they could do it from any machine running Windows, Mac, or Linux. This kind of design makes it easier for both users and developers at the same time. The user has not to worry about configuring the software, and the developers can more easily and

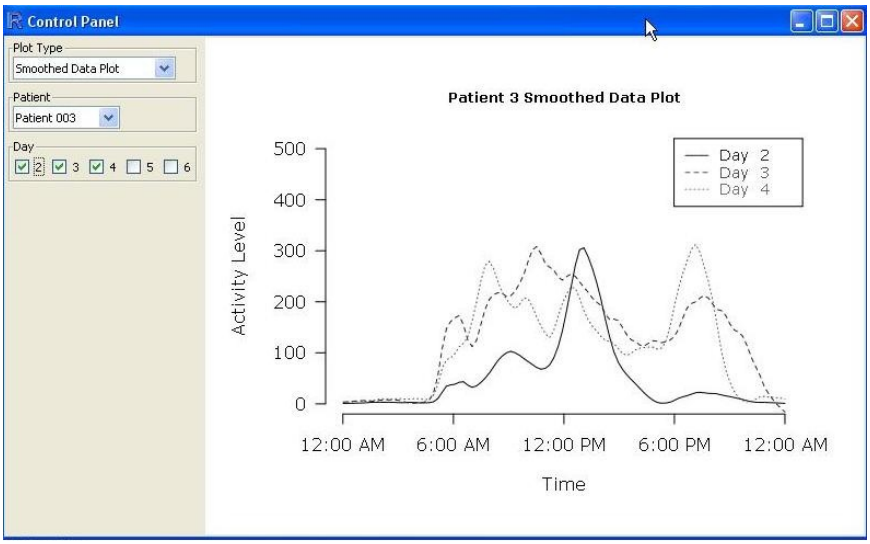


Fig. 4.3: A control panel window based on the Rcmdr package showing the smoothed data plot for a certain subject for three different days

efficiently maintain of the software because of its centralized design. In the process of developing the web interface, we tried three routes: Rpad, Rook, and RApache.

Rpad

We started using an R package called Rpad (Short and Grosjean, 2007). Rpad is a combination of an analysis package, a web-page designer, and a GUI designer. Rpad makes it easy to develop powerful data analysis applications that can be shared with others over the internet (through a web server like Apache) or locally over the intranet (using a built-in web server). Our web interface prototype was tested over the local host. This could be done by simply issuing a single line command *Rpad()* from within R. This runs a mini web server in the background and launches the default web browser to an Rpad startup page.

Figure 4.4 shows a snapshot of our prototype web interface using Rpad. We followed one single subject and compared his/her activity levels for five days at the baseline with his/her activity levels for five days after a certain treatment after six months. Based on the lowess fit (Section 1.2), it appears that the average daily

activity levels after treatment are somewhat higher than the levels at baseline.

Unfortunately, the authors of the Rpad package stopped updating it while we were working on our web interface. The Rpad web interface is functional up to R 2.9.2, but not beyond this R version.

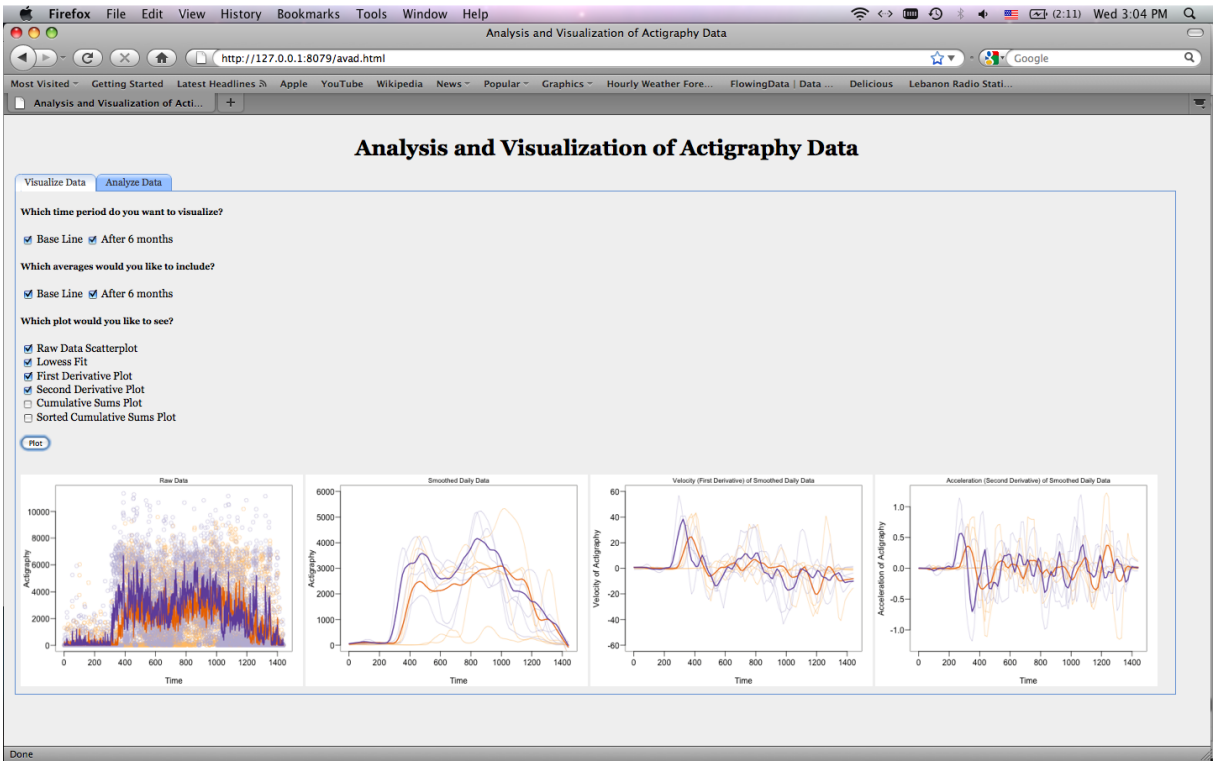


Fig. 4.4: A web interface showing four different types of plots (raw data, lowess fit, first derivative, and second derivative plot, introduced in Section 1.2), based on user selections via "check boxes" menu items. Faint orange symbols and lines represent five days at the baseline; faint purple symbols and lines represent five days after treatment. Solid orange/purple lines represent averages at the baseline/after treatment for this subject

Rook

Fortunately, Rook (Horner, 2011), a new web server interface for R was published at the end of 2011. It can be used to run web applications which can run in R 2.13's new built-in web server named Rhttpd. Rook allows embedding HTML and javascript code into R scripts. We created some functionality to upload actigraphy data files to

a web-server, send it to R's computing environment, produce some plots, and then post them on the web interface. As Rpad, Rook is an R package that needs to be installed into R, and thus, R needs to be running on the server side in order to make the ActiVis web application functional. One of the problems that we faced during our development process while using Rook was the ability to upload more than one data file. Also, in order to open the interface, we had to issue commands through R just the same way we had to do with Rpad. This considerably slowed down the process of interaction between the user and the system. During our development process, Rook's software distribution only allowed website access from the server's local host, which limits our ability to share our web application to the public.

rApache

In order to overcome the issues we faced while using Rpad and Rook, we utilized rApache (Horner, 2012). rApache is a project supporting web application development by realizing the full potentials of the R statistical environment and the Apache HTTP web server. rAppache consists of two modules: (1) mod_R which is responsible for loading the R interpreter and (2) libapreq which is responsible for manipulating client request data. The current version of rApache runs only on UNIX/ Linux and Mac OS X operating systems. Notice that this does not mean that the end user of the web site has to use a UNIX/Linux or Mac OS X machine. On the contrary, the client (end user) can be using any operating system, but the server machine has to running on UNIX/Linux or Mac OS X. We followed the installation procedure for rApache in Horner (2012) step by step, and the process is a bit tedious for non-programmers. In order to run the web application on the server, the server does not need to have R open as it was the case for Rpad and Rook. On the contrary, rApache accesses R's interpreter silently through the mod_R module.

Table 4.1: List of other approaches for bridging R to web interfaces

Software	**Description**
RinRuby	RinRuby is a library that embeds the R interpreter in Ruby (Dahl and Crawford, 2009). Ruby is a scripting language, and thus, it needs another library to integrate it in HTML to have a web interface. Erubis (Berube, 2007) is a Ruby library which allows the developer to weave Ruby code into HTML pages.
PypeR	PypeR (Xia et al., 2010) is a Python package for using R in Python. It has the same functionality as RinRuby, but again a web interface is needed.
JRI	JRI is a Java/R Interface, which loads R's dynamic library into Java applications. Java applications could be loaded on the web, but the client machine should run a Java Virtual Machine.
CGIwithR	CGIwithR (Firth, 2003) is an R package that facilitates processing web-based forms and reporting of results in HTML for display on the client browser. CGIwithR is not available on CRAN anymore, and it has not been updated since 2003.
Rserve	Rserve (Urbanek, 2003) is a TCP/IP server which allows other programs to use R facilities without the need to initialize R or link to the R library. It has not been updated since 2006.

Other Approaches

In addition to Rpad, Rook, and rApache, there exist several other approaches that we did not pursue because they either were very old and had not been updated since 2006, or because they were hard to configure with R and required additional software. Table 4.1 shows a list of these approaches, based on Saunders (2009) and our own research.

4.3 Existing Statistical Web Applications Using R

Most of the existing statistical web applications are simple interfaces to the R environment. Many of them (e.g., Rweb and webbioc) allow users to write R

scripts and submit them through their interfaces. Few others (Table 4.2) have more specific purposes with a user friendly graphical interface, such as the web interface by Jeroen Ooms (Ooms, 2010*b*) to the ggplot2 R package (Wickham, 2009). The ActiVis web interface presented in this chapter was inspired by Oom's web applications and interfaces to R (Ooms, 2009, 2010*b*).

4.4 Technologies Used

In this section, we describe the tools and technologies we used in the process of our web development. The readers might find them handy while designing and implementing an interactive statistical web interface.

4.4.1 Brew

Brew is an R package used as a templating framework for report generation (**?**). It gives the developer the ability to mix R code and text, and thus, producing Hyper Text Markup Language (HTML) syntax for displaying web pages in a web browser. The main intention behind the creation of Brew is to facilitate reproducible research (Kuhn, 2012).

4.4.2 Javascript

Javascript is a client-side scripting language that the server[1] could use in order to add functionality to web pages, validate forms, and communicate with the server (Negrino and Smith, 2011). Javascript makes web pages more dynamic by customizing the output for the user, adding interactivity to a web page (buttons, animations, form validation, etc.), and giving feedback to users.

[1]Client-side script is a program that is sent by the server to run on the user's machine, while a server-side script is a program that runs on the server.

4.4.3 CSS

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colors, spacing) to web documents (Boss, 2012). The purpose behind these style sheets is to separate the content of a web page written in HTML from its presentation, and thus allowing easier website development and maintenance because it reduces repetition in the structural content.

4.4.4 AJAX

In order to improve the user's experience interacting with the web site, and make it respond in a faster way, we adopted AJAX (Garret, 2012). With AJAX (Asynchronous JavaScript and XML), communication between the client and the server is done asynchronously. This means that data is send to and retrieved from the server without any interference with the display and behavior of the existing page. Thus, the waiting time is reduced, and the user no longer has to wait for the entire page to get rebuild and transmitted back by the server. Instead, only relevant content gets changed on the web interface. Google's search "autocomplete" feature is an example of an AJAX application. This application changes the content of the interface asynchronously as users type their queries by suggesting suitable search terms.

4.4.5 rjson

rjson is an R package that converts R objects to JavaScript Object Notation (JSON) and vice-versa (Couture-Beil, 2012). For advanced web applications, with a lot of data communication between the server and other applications such as R, we need to use a standard format for data interchange, simply because the Javascript language won't understand R object notations and vice-versa. Extensible Markup Language (XML) is another candidate for such a standardization, but JSON is more preferred for large datasets, and it is much simpler than XML (Ooms, 2010*a*).

4.4.6 jQuery

jQuery is free, open source javascript library that makes it easier to traverse HTML documents, handle website events, create animations, and AJAX interactions for rapid web development (Resig, 2012). jQuery has also a user interface library (jQuery-ui) that facilitates the creations of customizable widgets (accordion, tabs, progress bar, etc.), interactions (drag and drop, resizing, sorting, etc.) and effects (animated transitions).

Table 4.2: List of some statistical R web applications

Name	Description
R-Node	R-Node (Galili, 2012) is an open source web front-end to R. It supports most of R's standard commands, including the ones which provide textual as well as graphical feedback. It is limited to the existing commands, and thus, it could not be extended through installing new R packages. R-Node can be accessed through this link http://69.164.204.238:2904/.
Rweb	Rweb is one of the oldest attempts to bridge R to the web. The main purpose behind developing Rweb is educational. It could be used as a "complete computing environment for advanced courses, a simple point and click interface for introductory courses, or it can be incorporated as an interactive component in online assignments for any course" (Banfield, 1999).
webbioc	webbioc (Smith, 2009) is a "web interface for some of the Bioconductor microarray analysis packages. It is designed to be installed at local sites as a shared bioinformatics resource."
Rwui	Rwui (Wernisch, 2007) is a web application that allows R programmers to create web interfaces to run R scripts. It is mainly aimed at bioinformaticians who want to automate their analysis in a user friendly way. Statistics educators might find Rwui very helpful for creating teaching applications.
Concerto	Concerto (The Psychometrics Centre., 2008) is an adaptive testing platform developed at the Psychometrics Centre of Cambridge University. It allows R users with no web application experience to develop psychometric tests. Concerto can be accessed through the link http://code.google.com/p/concerto-platform/

4.5 Web Application Setup

The ActiVis R package that we introduced in Chapter 3 consists of almost 6000 lines of code. In order to reduce the hassle of running computer code by users and also

reach a wide audience, we developed a web interface for this package. Through this interface, the users are free from downloading or installing any kind of extra software on their computers. Not even the R environment has to be installed on their side. All they need is a standard web browser such as Firefox, Google Chrome, Safari, or Internet Explorer.

Before discussing the design of our web interface and how the message exchange takes place between the user and the server, we will first explain how the client-server model works in general.

4.5.1 Client-Server Architecture

Client-Server computing is a distributed application model, where the server acts as the "brain" which does heavy computations, and offer its services and resources to clients. According to Yadav and Singh (2009), this model is based on "distribution of functions between two types of independent processes: Client and Server. A Client is any process that requests specific service from the server process. A server is a process that provides requested services for the Client. Client and server processes can reside in the same computer or in different computers linked by a network." Figure 4.5 shows the basic Client-Server Model. The client computer requests a service from the server such as printing, faxing, or computing, and then the server responds back.

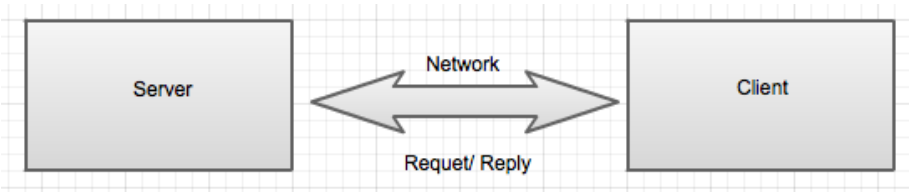


Fig. 4.5: Basic Client-Server Model.

4.5.2 High Level Actigraphy Web Application Design Model

The ActiVis web application presented in this chapter is based on a client-server architecture as presented in Section 4.5.1. Figure 4.6 shows a scenario of interaction

between the medical doctor and the web interface on the client side. The client machine sends the medical doctor's request (e.g., type and parameters of a plot) to the server that runs R, which in turn decodes the request and plots the graph. Then, the server responds back to the client with the plot file, which will be posted on the client's web interface.

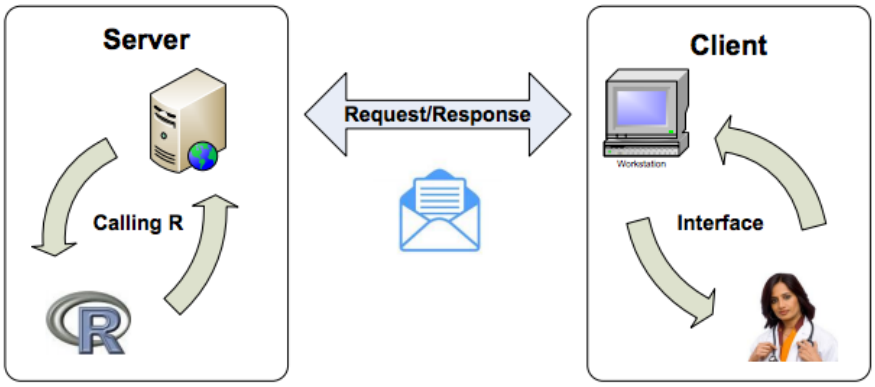


Fig. 4.6: ActiVis Client-Server Model (Ooms, 2010*a*).

4.5.3 Low Level Actigraphy Web Application Design Model

Building advanced web applications with easy-to-use interfaces is a complex task. The architecture for the ActiVis web application is shown in Figure 4.7 as a flow diagram for data between the client and the server: Once the application is loaded, the user is asked to upload data files, i.e., AWC and PHQ-9 scores files as described in Section 3.4, and select the desired graph type and parameters. Once these data are submitted, an HTTP request object is created by the AJAX engine in response to a javascript call (triggered by the user's submission) to encapsulate and send them to the server.

To communicate with the server, The AJAX engine sends the request to the server and waits for its response. The web server (Apache) receives the HTTP request in JSON format and sends it to the R engine via the rApache module after

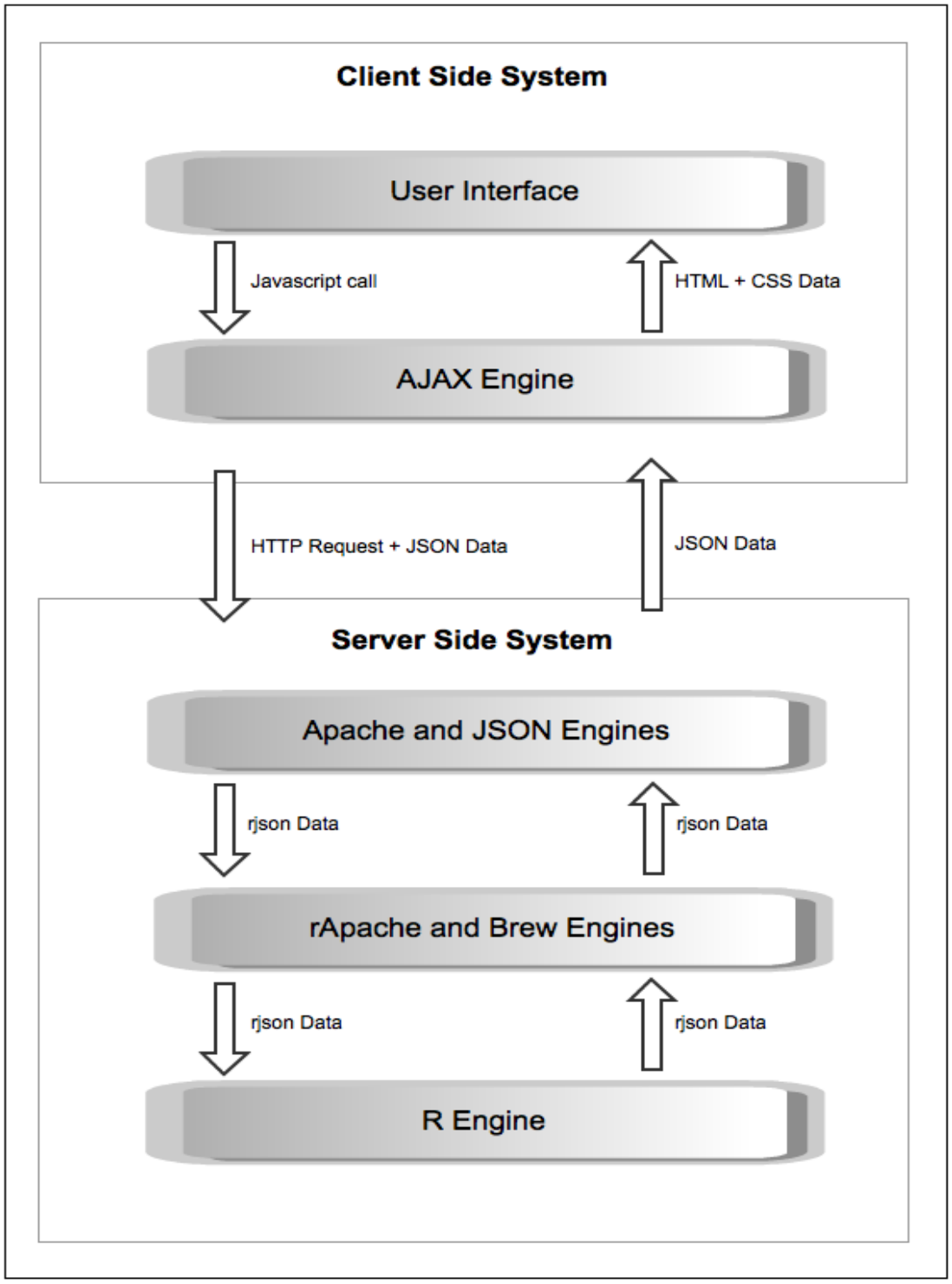


Fig. 4.7: Low Level Architecture for the ActiVis Client-Server Model

transforming it into an rjson object that could be read in R. The Brew engine is responsible for this transformation at this level.

R decodes the rjson data HTTP request, does the necessary data manipulation, and generates the requested plot which is saved temporarily on the server side. After the request has been processed, a JSON formatted data encapsulating the plot's address path on the server is sent to the HTTP request object on the client side.

When the HTTP request object receives the address of the plot from the server, the AJAX engine downloads the plot file (PNG format) to the client side, and presents it on the web interface through HTML and CSS formatting techniques.

Notice that this is not just a simple HTML document presented to the user, but it is an application with lots of computations and message passing happening in the background. With AJAX, the user can even do minor changes on the page efficiently, and without having to reload the whole web page. For example, if the user decides to change the title of a generated plot, the system can do this without making the user upload the data files or input other parameters again.

4.6 The ActiVis Graphical User Interface

The main idea of having a web application is to make it easy for the users to access and use the ActiVis package discussed in Chapter 3, without the hassle of installing any kind of software on the user's computer. Figure 4.8 shows a snapshot how our graphical user interface looks before any kind of interaction happens. It has two main parts: the left-hand side is the user interaction area, and the right-hand side is the plot display area.

The user interaction area is designed to make it easy for the user to follow the steps needed for producing actigraphy graphics, described in Chapter 2. For this purpose, we used an "accordion" GUI control, which is a stacked list of items that could be expanded either automatically or manually to reveal the content associated with each option. The user interaction area for the ActiVis web interface has three

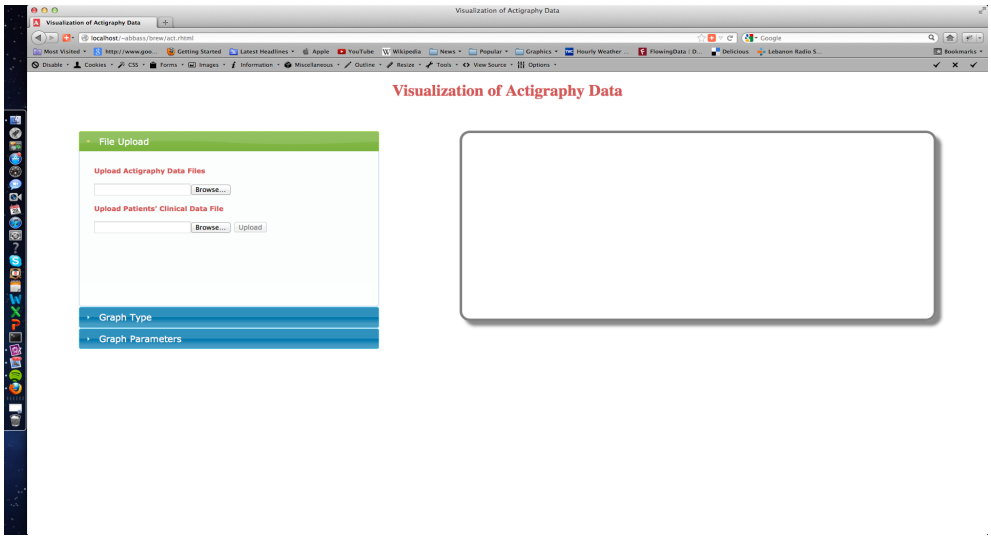


Fig. 4.8: ActiVis Web Application

items: File Upload (shown), Graph Type (hidden), and Graph Parameters (hidden). Figure 4.9 shows a snapshot of the application when the user clicks on the "Browse" button. Notice that the "Upload" button (Figure 4.8) is inactive until the user browses for files. When the "Browse" button is clicked, a file browser window pops up and prompts the user to select one or more actigraphy data files for uploading. The second file upload field is optional, and it is for uploading the PHQ9 scores in case the user wants to do cluster visualization. After the selection of files is done, the "Upload" button becomes active (Figure 4.10), and the user can click on it to submit his/her request.

In order to keep the interface clean and clear, we decided to make it adaptive. In other words, the interface presents the "next" steps based on the user's input. For example, if the user uploads only one data file, this means that he/she is planning to do visualizations for only one patient. Thus, the plots for the visualization of one patient (see Section 1.2) will show up as options. Otherwise, if more than one file is uploaded, the system will automatically detect that the user is planning to do cluster analysis, and thus multivariate visualization options (from Section 2.2) will show up.

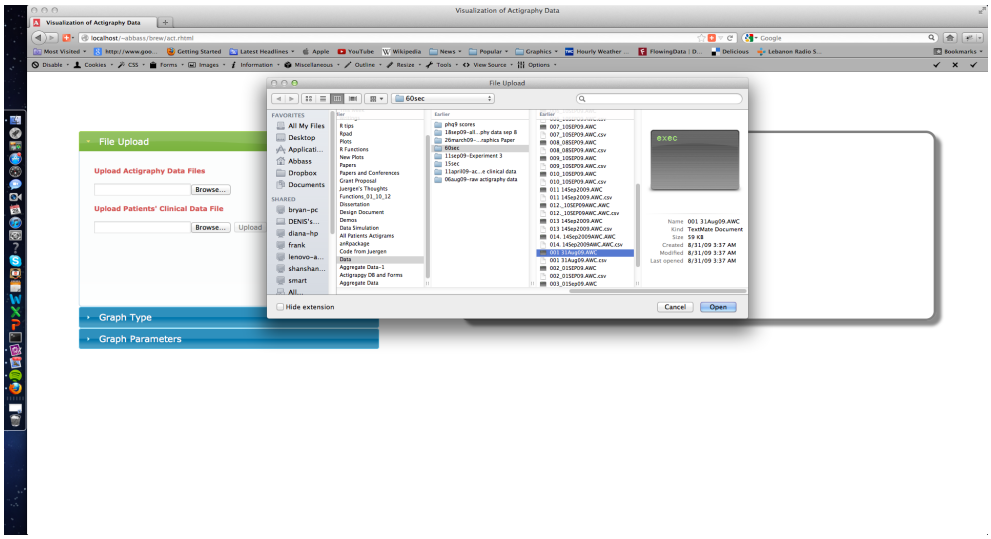


Fig. 4.9: ActiVis Web Application: Browsing for Data Files

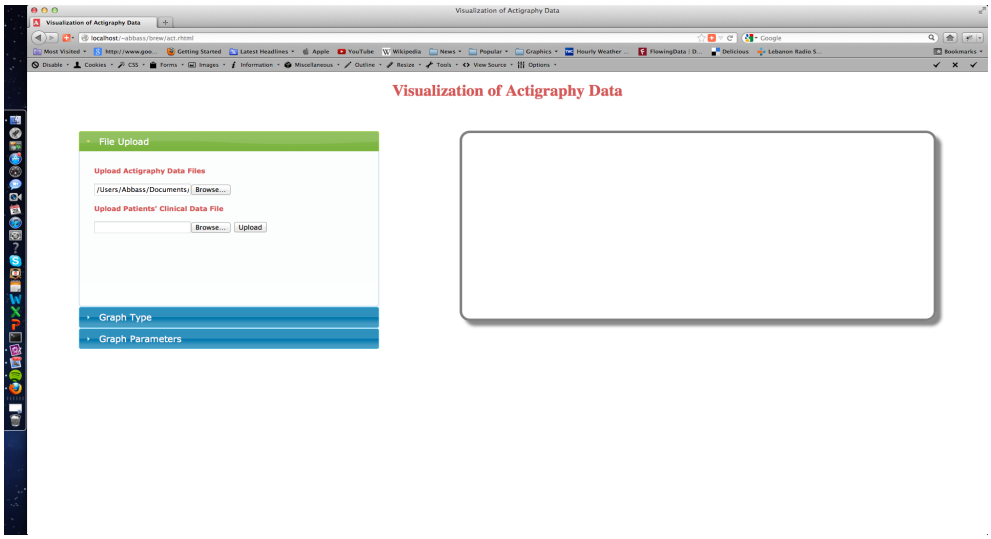


Fig. 4.10: ActiVis Web Application: Uploading Data Files

The transmission between the three steps of user interaction is smooth and automatic. When the system finishes uploading the files to the server, the "Graph Type" area of the accordion GUI control slides up in an animated way (Figure 4.11), and hides the "File Upload" area. This ensures that the user knows that the uploading is done, and in addition he/she will be notified how many files have been uploaded in the title of the "File Upload" section. This kind of feedback is important to keep the

user updated and active.

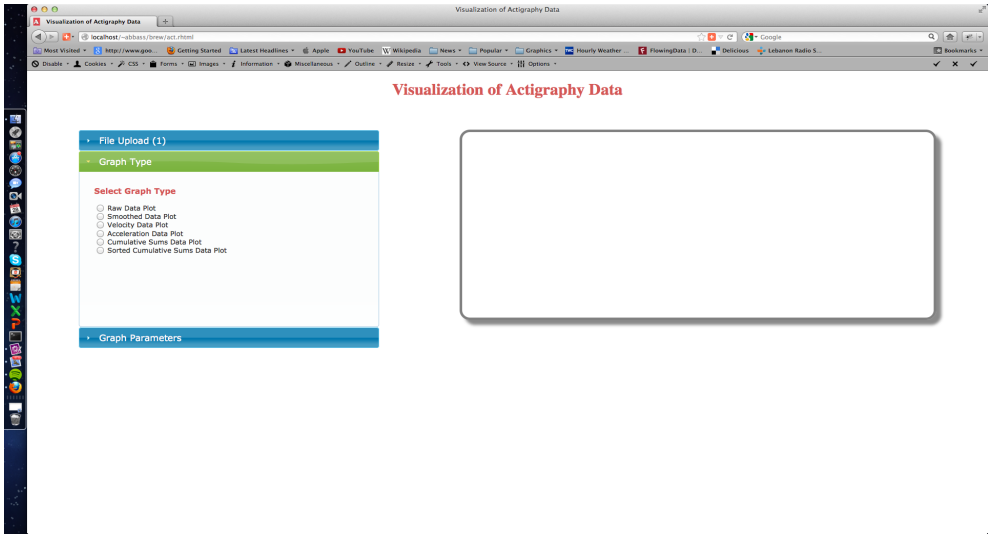


Fig. 4.11: ActiVis Web Application: Selecting a Graph Type

In this example, the user has uploaded one file. The visualization options given to the user are for examining a single patient's behavior and not for cluster analysis. When the user selects a graph type, the "Graph Parameter" area of the accordion GUI control slides up and hides the "Graph Type" area, and the system shows a feedback message to the user about the type of visualization selected (Figure 4.12).

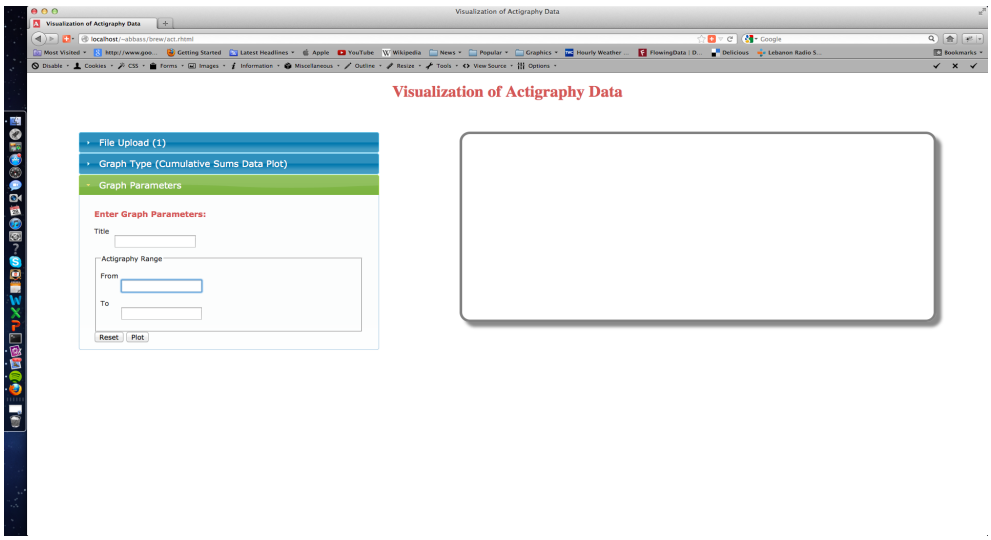


Fig. 4.12: ActiVis Web Application: Inputting Graph Paramters

The third and last step before rendering and showing the graph to the user is to get some graph parameters. We included this step to let the user manipulate the graph according to his/her preferences. For this version of the web interface, the parameters include the title of the plot and the range for the data. Figure 4.13 shows that the user has submitted all parameters and waits for the graph to be shown. In the mean time, and to give feedback to the user while the server is doing the computation, a progress bar is shown beside the "Plot" button.

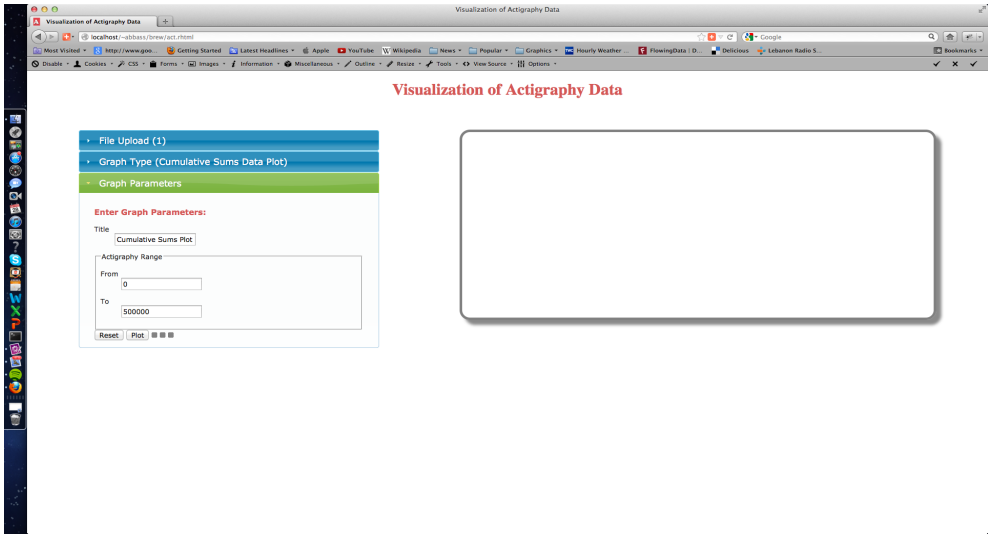


Fig. 4.13: ActiVis Web Application: Rendering the Visualization

When the server is done with the computation, the progress bar disappears, and the selected graph for the uploaded data is shown on the graph display area of the web interface (Figure 4.14). If the user double clicks on the graph, a new window will pop up showing only that graph. Users also have the option of downloading graphs to their local hard drive.

Because of the technologies used for this web application, users do not need to start from the beginning if they decide to have a different visualization for the uploaded data. A user would click on the "Graph Type" accordion GUI control area and change the choice, and then the system will automatically take him/her through the necessary steps to produce the selected graph. This kind of interaction makes the

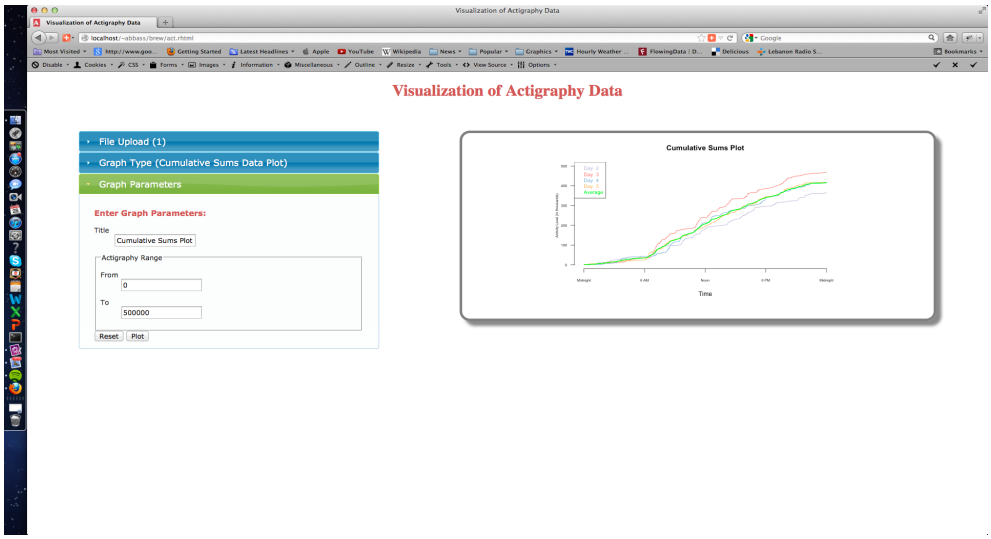


Fig. 4.14: ActiVis Web Application: Showing the Visualization

web application dynamic and adaptive, and at the same time efficient and easy to use because users do not need to reload the whole application to change the graph type or even one parameter in the chosen graph.

4.7 Discussion

Users are the most important entity in any software system, and thus, they are a priority in software engineering. Of course, the system itself should function efficiently and correctly, but if users are not satisfied with what they see, then the whole system fails. A graphical user interface for any software should be user friendly and easy to use. Keeping that in mind, we designed a simple interactive interface for the ActiVis R package that adapts to the user's input. We ensured to have a clean and clear design by showing only necessary information to the users based on their previous selections.

At a lower level, we did an extensive search to decide what approach to take for building the AcitiVis web application. At the beginning, we started using R packages for this purpose (Rpad and Rook), but then we decided that it would be better to separate the web interface (HTML code) from its functionality (R code), and thus,

we used the rApache approach for this bridging. In addition to code separation, the latter approach allows us to upload many data files to the server unlike Rpad and Rook (at least at the time of development).

One of the limitations that we faced in producing our multivariate plots (density, envelope, and mvtsplot) on the web is the speed in processing the necessary computations, because it involves transferring the data over the network. To eliminate this problem, we started looking at parallel computing and "Big Data". Hellerstein (2008) was the first to define this problem. He stated that "machines are the main generators of data. Billions of data is being generated every minute, and this Big Data is too huge or heavy to be stored on a desktop machine and then analyzed."

Finally, the ActiVis web interface could be enhanced to have more interactive capabilities such as zooming into the graphs and also selecting points or lines. In addition, we can add more graph parameters (line types, colors, and text fonts) for the user to manipulate, and have more control over the produced visualization.

CHAPTER 5
SUMMARY AND CONCLUSIONS

In this dissertation, we presented visual techniques for exploring functional actigraphy data. In particular, we looked into ways to visualize the activity of patients who might be suffering from depression, and noticed how their activity levels varied from one day to another within one patient as well as among different patients during different days. These techniques were implemented in R following the object-oriented paradigm in order to facilitate the process of updating and reusing the developed computer code in the future. We also bundled our techniques into an "open source" R package which will be available on the CRAN website sometime soon. Finally, and to make our system accessible by a wide variety of audiences, we developed a web application with a graphical user interface that deploys all of our techniques.

One of the limitations that we faced in producing our multivariate plots (density, envelope, and mvtsplot) is the speed in processing the necessary computations. This speed would be even slower when the computations were done via the web application because it involves an extra step of transferring the data over the network. We have data for 55 patients, and on average each patient has 12 days of data. The data collection was done every minute, so in total our dataset has almost 1 million observations (55 patients $\times$ 12 days $\times$ 1440 minutes). The process of data clustering takes even more time when we look at a representative sample of 750 patients as planned for the full study in Ding et al. (2011). To eliminate this problem, we started looking at parallel computing and "Big Data". Hellerstein (2008) was the first to define this problem. He stated that "machines are the main generators of data. Billions of data is being generated every minute, and this Big Data is too huge or heavy to be stored on a desktop machine and then analyzed."

The ActiVis web interface could be enhanced to have more interactive capabilities

such as zooming into the graphs and also selecting points or lines. In addition, we can add more graph parameters (line types, colors, and text fonts) for the user to manipulate, and have more control over the produced visualization.

Visualizing functional actigraphy data gives us an insight of how activity levels of patients vary. The natural step that comes after visualization is analyzing these data, and one of our future goals is to quantitatively investigate the source of variation. For this purpose, we already started looking at Functional Data Analysis (FDA) techniques, and in particular Functional Principle Component Analysis (FPCA) of actigraphy data (Ding et al., 2011). This work was conducted in collaboration with researchers at the Washington University in St. Louis Sleep Center, who also created an R package called "Actigraphy" (Shannon et al., 2012) for this purpose.

The techniques proposed in this work could be adapted to visualize any kind of functional data, and not just actigraphy data. They could be very helpful for space agencies like NASA who would be interested to see how the activity patterns and trends for their astronauts are in space. Also, the military could use these techniques to check how soldiers are doing during war time, and send active one to the fight.

Finally, there has been an increasing interest in "quantified self" (Quantified Self Labs., 2012). There is a community of people around the world, who are interested in self-tracking devices to regularly gather information about themselves (e.g., sleeping patterns, blood pressure, and eating habits during the day, etc.) and share knowledge and experiences with others. We could adapt our web application for these individuals, and give them the capability to upload their data and visualize them.

REFERENCES

Ancoli-Israel, S., Cole, R., Alessi, C., Chambers, M., Moorcroft, W. and Pollak, C. (2003), "The Role of Actigraphy in the Study of Sleep and Circadian Rhythms," *Sleep* **26**(3), 342–392.

Baier, T. and Neuwirth, E. (2007), 'Excel::COM::R', *Computational Statistics* **22**(1), 91–108.

Banfield, J. (1999), 'Rweb:Web-based Statistical Analysis', *Journal of Statistical Software* **4**(1), 1–15.

Bengtsson, H. (2003), "The R.oo Package — Object-Oriented Programming with References Using Standard R Code," *in Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, Vienna, Austria.

Berube, D. (2007), *Practical Ruby Gems*, Apress, New York, NY.

Boss, B. (2012), 'Cascading Style Sheets'. http://www.w3.org/Style/CSS/.

Chambers, J. M. (1998), *Programming with Data: A Guide to the S Language*, Springer, New York, NY.

Chambers, J. M. (2008), *Software for Data Analysis: Programming with R*, Springer, New York, NY.

Chambers, J. M. and Hastie, T. J., eds. (1992), *Statistical Models in S*, Wadsworth & Brooks/Cole, Pacific Grove, CA.

Cleveland, W. S. (1979), 'Robust Locally Weighted Regression and Smoothing Scatterplots', *Journal of the American Statistical Association* **74**(368), 829–836.

Cleveland, W. S. (1981), 'LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression', *The American Statistician* **35**(1), 54.

Couture-Beil, A. (2012), *rjson: JSON for R*. R package version 0.2.8. http://CRAN.R-project.org.

Dahl, D. B. and Crawford, S. (2009), 'RinRuby: Accessing the R Interpreter from Pure Ruby', *Journal of Statistical Software* **29**(4), 1–18.

Ding, J., Symanzik, J., Sharif, A., Wang, J., Duntley, S. and Shannon, W. D. (2011), 'Powerful Actigraphy Data Through Functional Representation', *Chance* **24**(1), 30–36.

Esliger, D. W. and Tremblay, M. (2006), 'Technical Reliability Assessment of Three Accelerometer Models in a Mechanical Setup', *Medicine & Science in Sports & Exercise* **38**(12), 2173–2181.

Firth, D. (2003), 'CGIwithR: Facilities for processing web forms using R', *Journal of Statistical Software* **8**(10), 1–8.

Fox, J. (2005), 'The R Commander: A Basic-Statistics Graphical User Interface to R', *Journal of Statistical Software* **14**(9), 1–42.

Freidman, D. P., Wand, M. and Haynes, C. T. (2001), *Essentials of Programming Languages (2nd Edition)*, MIT Press, Cambridge, MA.

Galili, T. (2012), 'R-Node: a Web Front-end to R with Protovis'. http://www.r-statistics.com/2010/04/r-node-a-web-front-end-to-r-with-protovis/.

Garret, J. (2012), 'Ajax: A New Approach to Web Applications'. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.

Gentleman, R. (2009), *R Programming for Bioinformatics*, Chapman & Hall/CRC, London, UK.

Heiberger, R. M. and Neuwirth, E. (2009), *R Through Excel: A Spreadsheet Interface for Statistics, Data Analysis, and Graphics*, Springer, New York, NY.

Hellerstein, J. (2008), 'The Commoditization of Massive Data Analysis'. http://radar.oreilly.com/2008/11/the-commoditization-of-massive.html.

Horner, J. (2011), *Rook: Rook - A Web Server Interface for R.* R package version 1.0-2. http://CRAN.R-project.org.

Horner, J. (2012), *rApache: Web Application Development with R and Apache.*

Inselberg, A., Chomut, T. and Reif, M. (1987), 'Convexity Algorithms in Parallel Coordinate', *Journal of the ACM* **34**(4), 765–801.

Jackson, C. (2008), 'Displaying Uncertainty with Shading', *The American Statistician* **62**(4), 340–347.

Kohl, M. and Ruckdeschel, P. (2010), 'R Package distrMod: S4 Classes and Methods for Probability Models', *Journal of Statistical Software* **35**(10), 1–27.

Kroenke, K. and Spitzer, R. (2002), 'The PHQ–9: A New Depression Diagnostic and Severity Measure', *Psychiatric Annals* **32**(9), 509–515.

Kuhn, M. (2012), 'Reproducible Research'. http://cran.r-project.org/web/views/ReproducibleResearch.html.

Labyak, S. E. and Bourguignon, C. (2002), 'Measurement Issues Related to Actigraphy Use in Older Women', *Topics in Geriatric Rehabilitation* **18**(1), 68–79.

Lafore, R. (2002), *Object-Oriented Programming in C++*, Sams Pub., Indianapolis, IN.

Miller, J. J. and Wegman, E. (1991), Computing and graphics in statistics, Springer-Verlag New York, Inc., New York, NY, USA, chapter Construction of Line Densities for Parallel Coordinate Plots, pp. 107–123.

Mini Mitter Company Incorporated (2005), *Actical Physical Activity Monitoring System Instruction Manual*, Bend, OR.

Minnotte, M. C. and West, R. W. (1998), "The Data Image: A Tool for Exploring High Dimensional Data Sets," *in 1998 Proceedings of the Section on Statistical Graphics', American Statistical Association*, Alexandria, VA, pp. 25–33.

Morgenthaler, T., Alessi, C., Friedman, L., Owens, J., Kapur, V., Boehlecke, B., Brown, T., Chesson, A., Coleman, J., Lee-Chiong, T., Pancer, J. and Swick, T. (2007), 'Practice Parameters for the Use of Actigraphy in the Assessment of Sleep and Sleep Disorders: An Update for 2007', *Sleep* **30**(4), 519–529.

Morphet, W. and Symanzik, J. (2010), 'The Circular Dataimage, a Graph for High-resolution Circular-spatial Data', *International Journal of Digital Earth* **3**(1), 47–71.

Moustafa, R. I., Hadi, A. S. and Symanzik, J. (2011), 'Multi-Class Data Exploration Using Space Transformed Visualization Plots', *Journal of Computational and Graphical Statistics* **20**(2), 298–315.

Negrino, T. and Smith, D. (2011), *JavaScript: Visual QuickStart Guide*, Peachpit Press, Berkeley, CA.

Ogbagaber, S., Albert, P., Lewin, D. and Iannotti, R. J. (2012), 'Summer Activity Patterns Among Teenage Girls: Harmonic Shape Invariant Modeling to Estimate Circadian Cycles', *Journal of Circadian Rhythms*.

Ooms, J. (2009), *rweb.stat.ucla.edu/irttool/: A Web Interface for the R Package IRT*. rweb.stat.ucla.edu/irttool.

Ooms, J. (2010*a*), Web Development with R, Presented at Bay Area useR Group, San Francisco, CA.

Ooms, J. (2010*b*), *yeroon.net/ggplot2: A Web Interface for the R Package ggplot2*. Version 0.2. yeroon.net/ggplot2.

Peng, R. (2008), 'A Method for Visualizing Multivariate Time Series Data', *Journal of Statistical Software* **25**(1), 1–17.

Petzoldt, T. and Rinke, K. (2007), 'simecol: An Object-Oriented Framework for Ecological Modeling in R', *Journal of Statistical Software* **22**(9), 1–31.

Porter, T. and Duff, T. (1984), 'Compositing Digital Images', *SIGGRAPH Comput. Graph.* **18**(3), 253–259.

Quantified Self Labs. (2012), 'Quantified Self: Self Knowledge Through Numbers'.

R Core Team. (2012), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Ramsay, J. O. and Silverman, B. W. (2006), *Functional Data Analysis (2nd Edition)*, Springer, New York, NY.

Resig, J. (2012), 'jQuery'. http://www.jquery.com.

Rice, J. (2004), 'Functional and Longitudinal Data Analysis: Perspectives on Smoothing', *Statistica Sinica* **14**(3), 631.

Saunders, N. (2009), 'A Brief Survey of R Web Interfaces'. http://nsaunders.wordpress.com/2009/11/30/a-brief-survey-of-r-web-interfaces/.

Shannon, W., Li, T., Xian, H., Wang, J., Deych, E. and Gonzalez, C. (2012), *Actigraphy: Actigraphy Data Analysis*. R package version 1.0.

Sharif, A., Symanzik, J. and Shannon, W. D. (2010), 'An Object-Oriented Approach in R for the Visualization of Functional Actigraphy Data', *Computing Science and Statistics*. Forthcoming.

Sherick, P., Symanzik, J. and Shannon, W. (2010), Accuracy, Bias, and Notable Observation for Experimental Actigraphy Data, *in* '2011 JSM Proceedings', American Statistical Association, Alexandria, VA.

Short, T. and Grosjean, P. (2007), *Rpad: Workbook-style, Web-based Interface to R*. R package version 1.3.0. http://CRAN.R-project.org.

Slaven, J. E., Andrew, M. E., Mnatsakanova, A., Violanti, J. M., Burchfiel, C. M. and Vila, B. J. (2009), 'Statistical Modeling of Sleep', *Chance* **22**(1), 16–21.

Smith, C. (2009), *webbioc: Bioconductor Web Interface*. R package version 1.28.0. http://CRAN.R-project.org.

Symanzik, J. and Shannon, W. (2008), Exploratory Graphics for Functional Actigraphy Data, *in* '2008 JSM Proceedings', American Statistical Association, Alexandria, VA. (CD).

The Psychometrics Centre. (2008), *Concerto: An Open-source Online R-based Adaptive Testing Platform*, University of Cambridge, Cambridge, UK.

Todorov, V. and Filzmoser, P. (2009), 'An Object-Oriented Framework for Robust Multivariate Analysis', *Journal of Statistical Software* **32**(3), 1–47.

Tukey, J. W. (1977), *Exploratory Data Analysis*, Addison Wesley, Reading, MA.

Urbanek, S. (2003), Rserve: A Fast Way to Provide R Functionality to Applications, Proceedings of the third International Workshop on Distributed Statistical Computing, Vienna, Austria.

Wang, J., Xian, H., Licis, A., Deych, E., Ding, J., McLeland, C., Li, T., Duntley, S. and Shannon, W. (2011), 'Measuring the Impact of Apnea and Obesity on Circadian Activity Patterns Using Functional Linear Modeling of Actigraphy Data', *Journal of Circadian Rhythms* **9**(11).

Wegman, E. J. (1990), 'Hyperdimensional Data Analysis Using Parallel Coordinates', *Journal of the American Statistical Association* **85**, 664–675.

Wegman, E. J. (2003), 'Visual Data Mining', *Statistics in Medicine* **22**, 1383–1397.

Wernisch, N. (2007), 'Rwui: A Web Application to Create User Friendly Web Interfaces for R Scripts', *R News* **7**(2), 32–35.

Wickham, H. (2009), *ggplot2: Elegant Graphics for Data Analysis*, Springer New York, NY.

Wickham, H. (2012), 'R5'. https://github.com/hadley/devtools/wiki/R5.

Xia, M., McClelland, M. and Wang, Y. (2010), 'PypeR, A Python Package for Using R in Python', *Journal of Statistical Software, Code Snippets* **35**(2), 1–8.

Yadav, S. and Singh, S. (2009), *An Introduction To Client/Server Computing*, New Age Books, Daryaganj, Delhi, IND.

APPENDICES

APPENDIX A

PROCEDURAL PARADIGM (R CODE)

A.1 Read AWC Data Format File

```
read.awc <- function(files){

pats.data = NULL
for(file in files) {

    id <-  read.fwf(file, width=12, skip=0, n=1)
    date <- read.fwf(file, width=12, skip=1, n=1)
    time <- read.fwf(file, width=12, skip=2, n=1)
    epoch <- read.fwf(file, width=12, skip=3, n=1)
    # Epoch records the frequency of the data recording
    activity <- read.fwf(file, width=12, skip=11,na.string="0 M")[,1]

    x <- paste(date$V1, time$V1)
    x <- strptime(x, "%d-%b-%Y %H:%M")
    times  <- rep(x, length(activity))+ seq(0, length(activity)-1)*(15*epoch$V1)

    id  <- as.numeric(id$V1)
epoch_rate <-as.numeric(epoch$V1)
# To check whether the data are recorded every 15 secs, 30 secs, or 1 min
start_time_hour <- as.numeric(substring(time$V1, 1, 2))
# The starting hour of the experiment (hh)
start_time_min <- as.numeric(substring(time$V1, 4, 5))
# The starting minute of the experiment (mm)
data_size <- length(activity)
# The size of the dataset

if(epoch_rate == 1) {j <- 4}
if(epoch_rate == 2) {j <- 3}
if(epoch_rate == 3) {j <- 2}
if(epoch_rate == 4) {j <- 1}

# Calculate when the experiment started on day 1

epoch_start  <-  ((start_time_hour *60) + start_time_min)*j

if(data_size <1440*j){
# If the experiment was run for one incomplete day
length_first_day  <- data_size
# Length of day 1 (which is the only day in the experiment)
```

```
length_last_day  <- length_first_day
# The last day of the experiment is the first day...
remaining_epoch <- 1440*j - epoch_start - length_last_day
# This is how many NA's to append to the last day if it is not complete
} else {
# If the experiment is run for more than one day
length_first_day  <- 1440*j - epoch_start
# Length of day 1
length_last_day <- (data_size - length_first_day) %% (1440*j)
# Length of the last day in the experiment
remaining_epoch <- 1440*j - length_last_day
# This is how many NA's to append to the last day if it is not complete
}

# Calculate the overall number of days of the experiment
num_days <- 2
# Initialize number of days to 2...
#Taking into account day 1 and the last day

length_without_day1_and_lastday <- data_size -length_first_day -length_last_day

if( length_without_day1_and_lastday > 1440*j){
num_days  <- num_days + ceiling(length_without_day1_and_lastday/(1440*j))
}else if (length_without_day1 >0){
num_days  <- num_days + 1
}

day <- NULL
# Initialize day array

day <- matrix(rep(1, length_first_day), ncol = 1)
# Fill in the days array

for(i in 2:(num_days-1)){
day <-  rbind(day, matrix(rep(i,1440*j), ncol = 1))
}

day <- rbind(day, matrix(rep(num_days,length_last_day), ncol = 1))

epoch_array <- NULL
### initialize epoch array

epoch_array = matrix(c(1:length_first_day), ncol = 1)
# Fill in the Epoch array

for(i in 2:(num_days-1)){
epoch_array <- rbind(epoch_array, matrix(c(1:(1440*j)), ncol = 1))
}
epoch_array <- rbind(epoch_array,
     matrix(c(1:length_last_day), ncol = 1))
```

```
  activity[is.na(activity)] <- 0
# Get rid of NA's

      data <- data.frame(
ID=rep(as.numeric(substr(file,start=1,stop=3)),
   length(activity)),
    Date=times,
DayNumber = day,
     Time = epoch_array
     )

       # Add weekdays names to the data using "weekdays" function

    data <- cbind (data, Act = activity)
   ActSum <- NULL

for (i in 1:num_days){
ActSum <- rbind(
ActSum,
matrix(cumsum(data$Act[(data$DayNumber == i)]), ncol = 1))
}

   ActSort <- NULL

for (i in 1:num_days){
ActSort <- rbind(
ActSort,
matrix(sort(data$Act[(data$DayNumber == i)]), ncol = 1))
}

data <- cbind(data, ActSum, ActSort)

   ActSortSum <- NULL

for (i in 1:num_days){
ActSortSum <- rbind(
ActSortSum,
matrix(cumsum(data$ActSort[(data$DayNumber == i)]), ncol = 1))
}

data <- cbind(data, ActSortSum)

pats.data <- rbind(pats.data, data)

   patients_data <- data.frame( ID = pats.data[,1],
                                Date = pats.data[,2],
                                DayNumber = pats.data[,3],
                                Time = pats.data[,4],
                                Act = pats.data[,5],
                                 ActSum = pats.data[,6],
                                ActSort = pats.data[,7],
```

```
                                ActSortSum = pats.data[,8])

} #end of first for loop

return(patients_data)

} #End function
```

A.2 Aggregate 15-seconds Data to 1-minute Data

```
aggregate_sum_to_1min <- function(data){

 act = data$Act
 act_size  = length(act)
 new_act  = data$Act
 start_time <- data$Date[1]

 remainder  = act_size %% 4

 if (remainder != 0){
 new_act = head(act, -remainder)
 # deletes the last extra elements
 }

 # Put data in matrix form, and then aggregate
 new_act = matrix(new_act, nrow = 4, byrow = FALSE)
 aggregated_act = apply(new_act, 2, sum)

 # Add "Day" column to the aggregated data

 num_of_days_all  = data$DayNumber[length(data$DayNumber-remainder)]
 # just look at the last day.
 num_of_obs_first_day  = length(data$DayNumber[data$DayNumber == 1])
 num_of_obs_last_day = length(data$Day[data$DayNumber == num_of_days_all])

 new_num_of_obs_first_day = floor(num_of_obs_first_day / 4)
 new_num_of_obs_last_day = floor(num_of_obs_last_day / 4)

 Day = rep(1,new_num_of_obs_first_day)

 for(i in 2:(num_of_days_all - 1)){
Day = c(Day, rep(i, (1440)))
}

 Day = c(Day, rep(i+1,new_num_of_obs_last_day) )
 # Time and Date

 x = strptime(start_time, "%Y-%m-%d %H:%M")
 times = NULL
```

```
 times = rep(x, length(Day))
 times = times + (seq(0, length(Day)- 1) * 60)

 epoch_array = NULL
 #initialize epoch array

 epoch_array = matrix(c(1:new_num_of_obs_first_day), ncol = 1)
 # Fill in the Epoch array

 for(i in 2:(num_of_days_all-1)){
   epoch_array = rbind(epoch_array, matrix(c(1:(1440)), ncol = 1))
 }
 epoch_array = rbind(epoch_array, matrix(c(1:new_num_of_obs_last_day), ncol = 1))

 new_data = data.frame (
ID = rep(data$ID[1], length(aggregated_act)),
    Date = times,
    DayNumber = Day,
    Time = epoch_array,
    Act = aggregated_act
  )

 num_days <- num_of_days_all

 ActSum = NULL
 for (i in 1:num_days){
   ActSum = rbind(
ActSum,
matrix(cumsum(new_data$Act[(new_data$DayNumber == i)]), ncol = 1))
 }
 ActSort = NULL

 for (i in 1:num_days){
   ActSort = rbind(
ActSort,
matrix(sort(new_data$Act[(new_data$DayNumber == i)]), ncol = 1))
 }

 new_data = cbind(new_data, ActSum, ActSort)

 ActSortSum = NULL

 for (i in 1:num_days){
   ActSortSum = rbind(
ActSortSum,
matrix(cumsum(new_data$ActSort[(new_data$DayNumber == i)]), ncol = 1))
 }
 new_data = cbind(new_data, ActSortSum)

 return(new_data)
}
```

A.3 Aggregate Sum of Minutely Data

```
aggregate_sum <- function(data, agg_epoch = 10){

 date =   as.POSIXlt(strptime(data$Date, '%Y-%m-%d %H:%M'))

 #date = as.POSIXlt(as.Date(data$Date))
 time = as.integer(date$min)
 act = data$Act

###Initialize some variables to be used later in the program
 start_NA_count = 0
 end_NA_count = 0
 new_act  = act ### set it to the whole data set

### Determine the start
 if(time[1] %% agg_epoch != 0){
start_NA_count = agg_epoch - (time[1]%%agg_epoch)
}

### Determine the end
 if(time[length(time)] %% agg_epoch != agg_epoch - 1){
      end_NA_count = (time[length(time)]%%agg_epoch) + 1
}

### Define a new array without the fractions of times
 if(start_NA_count != 0)
new_act = act[-(1:start_NA_count)]
 if(end_NA_count !=0)
new_act = new_act[-((length(new_act) - end_NA_count + 1):length(new_act))]

### Put data in matrix form, and then aggregate
 new_act = matrix(new_act, nrow = agg_epoch, byrow = FALSE)
 aggregated_act = apply(new_act, 2, sum)

### Append NA's at the beginning and end of the aggregated data
 if(start_NA_count != 0)
aggregated_act = c(0, aggregated_act)
 if(end_NA_count !=0)
aggregated_act = c(aggregated_act, 0)

### Add "Day" column to the aggregated data

 num_of_days_all  = data$DayNumber[length(data$DayNumber)]
 ### just look at the last day.
 num_of_obs_first_day  = length(data$DayNumber[data$DayNumber == 1])
 num_of_obs_last_day = length(data$DayNumber[data$DayNumber == num_of_days_all])

 new_num_of_obs_first_day = ceiling(num_of_obs_first_day / agg_epoch)
 new_num_of_obs_last_day = ceiling(num_of_obs_last_day / agg_epoch)

 Day = rep(1,new_num_of_obs_first_day)
```

```
 for(i in 2:(num_of_days_all - 1)){
Day = c(Day, rep(i, (1440/agg_epoch)))
}

 Day = c(Day, rep(i+1,new_num_of_obs_last_day) )

 new_data = data.frame (ID = rep(data$ID[1], length(aggregated_act)),
DayNumber = Day,
Act = aggregated_act)

 num_days <- num_of_days_all

 ActSum = NULL
 for (i in 1:num_days){
   ActSum = rbind(
ActSum,
matrix(cumsum(new_data$Act[(new_data$DayNumber == i)]), ncol = 1))
 }
 ActSort = NULL

 for (i in 1:num_days){
   ActSort = rbind(
ActSort,
matrix(sort(new_data$Act[(new_data$DayNumber == i)]), ncol = 1))
 }
 new_data = cbind(new_data, ActSum, ActSort)
 ActSortSum = NULL

 for (i in 1:num_days){
   ActSortSum = rbind(
ActSortSum,
matrix(cumsum(new_data$ActSort[(new_data$DayNumber == i)]), ncol = 1))
 }

 new_data = cbind(new_data, ActSortSum)

 return(new_data)
}
```

A.4 Raw Data Plot

```
rawdata.plot <- function(data, title= "Patient Raw Data Plot",
                         plotdays = -1, average = T, act_range = c(0,6000),
                         dayColor = c(rgb(141, 211, 199, max = 255),
                                      rgb(190, 186, 218, max = 255),
                                      rgb(251, 128, 114, max = 255),
                                      rgb(128, 177, 211, max = 255),
                                      rgb(253, 180, 98, max = 255),
                                      rgb(179, 222, 105, max = 255),
                                      rgb(252, 205, 229, max = 255),
                                      rgb(217, 217, 217, max = 255),
                                      rgb(188, 128, 189, max = 255),
                                      rgb(204, 235, 197, max = 255),
                                      rgb(255, 255, 179, max = 255),
                                      rgb(255, 237, 111, max = 255)
                                    ),
                         avgColor = c(rgb(0, 1, 0)),
                         legendPosition = "topright")

{

#### Read number of days for each patient
act = NULL
epoch = NULL
day = NULL
activity = NULL


if(plotdays[1] != -1) {
    for (i in 1:length(plotdays)){
epoch = c(data$Time[data$DayNumber == plotdays[length(plotdays)-i + 1]],
epoch)
    day = c(data$DayNumber[data$DayNumber == plotdays[length(plotdays)-i + 1]],
day)
    activity = c(data$Act[data$DayNumber == plotdays[length(plotdays)-i + 1]],
  activity)
}
act = data.frame(epoch, day, activity)
} else {
act = data
plotdays = 1:range(data$DayNumber)[2]
}


act$activity[act$activity <act_range[1]] = NA
act$activity[act$activity >act_range[2]] = NA

epoch = c(1:1440)

for(i in plotdays)
{
```

```
activity = as.numeric(act$activity[act$day==i])
points(epoch, activity, col = dayColor[i])
}

# create avgerages

if (average){
avgbase = rep(0, 1440)

epoch = act$epoch[act$day ==i] - (i-1)*1440

for (i in 0:1439)
{
avgbase[i] = mean(act$activity[act$epoch == i], na.rm=T)
}

lines(0:1439, avgbase, col = avgColor, lwd = 2)
}

axis(1, , at = seq(0,1440, 360),
labels = c("12:00 AM","6:00 AM","12:00 PM","6:00 PM","12:00 AM"))
axis(2, las=2)
mtext("Activity Level",
at = ((act_range[2] + act_range[1])/2),line = 3, side = 2, cex = 1)

#define the legend
legendColor = NULL
legendText = NULL
for (i in plotdays){
  legendText = c(legendText, paste("Day ", i))
  legendColor = c(legendColor,dayColor[i])
}

if(average){
  legendText = c(legendText, "Average")
  legendColor = c(legendColor,avgColor)
}
legend(legendPosition,legendText, text.col = legendColor,
   cex =0.85, col = legendColor)
}
```

A.5 Smoothed Data Plot

```
smoothdata.plot <- function(data, title= "Patient Smoothed Data Plot",
                            plotdays = -1, average = T, act_range = c(0,4000),
                            dayColor = c(rgb(141, 211, 199, max = 255),
                                         rgb(190, 186, 218, max = 255),
                                         rgb(251, 128, 114, max = 255),
                                         rgb(128, 177, 211, max = 255),
                                         rgb(253, 180, 98, max = 255),
                                         rgb(179, 222, 105, max = 255),
                                         rgb(252, 205, 229, max = 255),
                                         rgb(217, 217, 217, max = 255),
                                         rgb(188, 128, 189, max = 255),
                                         rgb(204, 235, 197, max = 255),
                                         rgb(255, 255, 179, max = 255),
                                         rgb(255, 237, 111, max = 255)
                            ),
                            avgColor = c(rgb(0, 1, 0)),
  lineType, legendPosition = "topright")
  {

      lowess_act = NULL
      lowess_data = NULL # to include time and lowess_act

      #epoch = c(1:1440)
      for (i in plotdays)  {
        lines(
          data$Time[(data$DayNumber == i)],
          lowess(data$Act[(data$DayNumber == i)], f = .1)$y,
          lty=lineType,
          lwd = 1.2,
          col=dayColor[i]
        )
        lowess_act = c(lowess_act,
lowess(data$Act[(data$DayNumber == i)], f = .1)$y)
      } # end for

      time = rep(1:1440, length(plotdays))
      lowess_data = data.frame(time, lowess_act)



      ### overplot the average
      if(average){
        avg = rep(1, 1440)

        for (i in 1:1440)
        {
          avg[i] = mean(lowess_data$lowess_act[lowess_data$time == i],
na.rm=T)
        }
```

```
        lines(1:1440, avg, col = avgColor, lwd = 2)
      }


      axis(1, , at = seq(0,1440, 360),
          labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
  cex.axis = 0.6)


      axis(2, las=2, cex.axis = 0.5)
      mtext("Activity Level",
at = ((act_range[2] + act_range[1])/2),line = 2,
side = 2, cex = 1)

      #define the legend
      legendColor = NULL
      legendText = NULL
      for (i in plotdays){

        legendText = c(legendText, paste("Day ", i))
        legendColor = c(legendColor,dayColor[i])
      }



      if(average){
        legendText = c(legendText, "Average")
        legendColor = c(legendColor,avgColor)
      }

      legend(legendPosition,legendText, text.col = legendColor,
cex =0.85, col = legendColor)
}
```

A.6 Velocity Plot

```
velocity.plot <- function(data, title= "Patient Velociyu Data Plot",
                          plotdays = -1, average = T, act_range = c(0,4000),
                          dayColor = c(rgb(141, 211, 199, max = 255),
                                       rgb(190, 186, 218, max = 255),
                                       rgb(251, 128, 114, max = 255),
                                       rgb(128, 177, 211, max = 255),
                                       rgb(253, 180, 98, max = 255),
                                       rgb(179, 222, 105, max = 255),
                                       rgb(252, 205, 229, max = 255),
                                       rgb(217, 217, 217, max = 255),
                                       rgb(188, 128, 189, max = 255),
                                       rgb(204, 235, 197, max = 255),
                                       rgb(255, 255, 179, max = 255),
                                       rgb(255, 237, 111, max = 255)
                          ),
                          avgColor = c(rgb(0, 1, 0)),
  lineType, legendPosition = "topright")
  {


      lowessbase = NULL
      diffbase = NULL
      diff_data = NULL
      velocity_data = NULL

      epoch = c(1:1440)
      for (i in plotdays)  {
        #Compute lowess data
        lowessbase = lowess(data$Act[ (data$DayNumber == i)], f = .1)$y
        #Compute first derivative of lowess data
        diffbase = c(diff(lowessbase), lowessbase[1439])

        lines(
          epoch,
          lowess(diffbase, f = .02)$y,
          lty=1,
          lwd = 1.2,
          col=dayColor[i]
        )
        diff_data = c(diff_data,
lowess(diffbase, f = .02)$y)
#Get all derivative data into one col.

      } # end for

      ### overplot the average
      if(average){
        time = rep(1:1440, length(plotdays))
        velocity_data = data.frame(time, diff_data)
```

```
# dataframe derivative data and time

        avg = 1:1440

        for (j in 1:1440)
        {
          avg[j] = mean(velocity_data$diff_data[velocity_data$time == j],
na.rm=T)
        }
        lines(1:1440, avg, col = avgColor, lwd = 2)

        ### done plotting the average

      }


      axis(1, , at = seq(0,1440, 360),
           labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
   cex.axis = 0.6)


      axis(2, las=2, cex.axis = 0.5)
      mtext("Activity Level", at = ((act_range[2] + act_range[1])/2),
line = 2, side = 2, cex = 1)

      #define the legend
      legendColor = NULL
      legendText = NULL
      for (i in plotdays){

        legendText = c(legendText, paste("Day ", i))
        legendColor = c(legendColor,dayColor[i])
      }

      if(average){
        legendText = c(legendText, "Average")
        legendColor = c(legendColor,avgColor)
      }

      legend(legendPosition,legendText, text.col = legendColor,
cex =0.85, col = legendColor)
}
```

A.7 Acceleration Plot

```
acceleration.plot <- function(data, title= "Patient Acceleration Data Plot",
                           plotdays = -1, average = T, act_range = c(0,4000),
                           dayColor = c(rgb(141, 211, 199, max = 255),
                                        rgb(190, 186, 218, max = 255),
                                        rgb(251, 128, 114, max = 255),
                                        rgb(128, 177, 211, max = 255),
                                        rgb(253, 180, 98, max = 255),
                                        rgb(179, 222, 105, max = 255),
                                        rgb(252, 205, 229, max = 255),
                                        rgb(217, 217, 217, max = 255),
                                        rgb(188, 128, 189, max = 255),
                                        rgb(204, 235, 197, max = 255),
                                        rgb(255, 255, 179, max = 255),
                                        rgb(255, 237, 111, max = 255)
                           ),
                           avgColor = c(rgb(0, 1, 0)),
  lineType, legendPosition = "topright")
  {


        lowessbase = NULL
        diffbase = NULL
        diffbase2 = NULL
        diff_data = NULL
        velocity_data = NULL
        epoch = c(1:1440)

        for (i in plotdays)  {
          #Compute lowess data
          lowessbase = lowess(data$Act[ (data$DayNumber == i)], f = .1)$y
          #Compute first derivative of lowess data
          diffbase = c(diff(lowessbase), lowessbase[1439])
          #Compute Second derivative of lowess data
          diffbase2  = c(diff(lowess(diffbase, f = .02)$y),
diffbase[1440] - diffbase[1439])

          lines(
            epoch,
            lowess(diffbase2, f = .02)$y,
            lty=1,
            lwd = 1.2,
            col=dayColor[i]
          )
          diff_data = c(diff_data, lowess(diffbase2, f = .02)$y)
 #Get all derivative data into one col.

        } # end for

        ### overplot the average
```

```
        if(average){
          time = rep(1:1440, length(plotdays))
          acceleration_data = data.frame(time, diff_data)
# dataframe derivative data and time

          avg = 1:1440

          for (j in 1:1440)
          {
            avg[j] = mean(
  acceleration_data$diff_data[acceleration_data$time == j],
    na.rm=T)
          }
          lines(1:1440, avg, col = avgColor, lwd = 2)

          ### done plotting the average

        }

        axis(1, , at = seq(0,1440, 360),
             labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
             cex.axis = 0.6)

        axis(2, las=2, cex.axis = 0.5)
        mtext("Acceleration of Activity Level",
  at = ((act_range[2] + act_range[1])/2),
  line = 2, side = 2, cex = 1)

        #define the legend
        legendColor = NULL
        legendText = NULL
        for (i in plotdays){

          legendText = c(legendText, paste("Day ", i))
          legendColor = c(legendColor,dayColor[i])
        }


        if(average){
          legendText = c(legendText, "Average")
          legendColor = c(legendColor,avgColor)
        }

        legend(legendPosition,legendText, text.col = legendColor,
 cex =0.85, col = legendColor)
}
```

A.8 Cumulative Sums Plot

```
cumsums.plot <- function(data, title= "Patient CumSums Data Plot",
                         plotdays = -1, average = T, act_range = c(0,4000),
                         dayColor = c(rgb(141, 211, 199, max = 255),
                                      rgb(190, 186, 218, max = 255),
                                      rgb(251, 128, 114, max = 255),
                                      rgb(128, 177, 211, max = 255),
                                      rgb(253, 180, 98, max = 255),
                                      rgb(179, 222, 105, max = 255),
                                      rgb(252, 205, 229, max = 255),
                                      rgb(217, 217, 217, max = 255),
                                      rgb(188, 128, 189, max = 255),
                                      rgb(204, 235, 197, max = 255),
                                      rgb(255, 255, 179, max = 255),
                                      rgb(255, 237, 111, max = 255)
                         ),
                         avgColor = c(rgb(0, 1, 0)),lineType,
 legendPosition = "topright")
  {

            epoch = c(1:1440)

            for (i in plotdays)  {
              lines( data$Time[(data$DayNumber == i)],
                     data$ActSum[(data$DayNumber == i)],
                     lty=1,
                     lwd = 1.2,
                     col=dayColor[i]
              )


            } # end for

            plotted_data <- data[data$DayNumber %in% plotdays,]
#only plotted data
            ### overplot the average

            if(average){
              cumavgbase = rep(0, 1440)

              for (i in 1:1440)
              {
                cumavgbase[i] = mean(
plotted_data$ActSum[(plotted_data$Time == i)])
              }

              lines(1:1440, cumavgbase, col = avgColor, lwd = 2)

              ### done plotting the average
            }
```

```
            axis(1, , at = seq(0,1440, 360),
                 labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
 cex.axis = 0.6)

            axis(2, at = seq(0,500000, 100000),
labels = c("0","100","200","300", "400", "500"),
las=2, cex.axis = 0.6)
            mtext("Activity Level (in thousands)",
   line = 2, side = 2, cex = 0.6)


      #define the legend
      legendColor = NULL
      legendText = NULL
      for (i in plotdays){

        legendText = c(legendText, paste("Day ", i))
        legendColor = c(legendColor,dayColor[i])
      }



      if(average){
        legendText = c(legendText, "Average")
        legendColor = c(legendColor,avgColor)
      }

      legend(legendPosition,legendText, text.col = legendColor,
cex =0.85, col = legendColor)
}
```

A.9 Sorted Cumulative Sums Plot

```
sortedcumsums.plot <- function(data, title= "PatientSorted CumSums Data Plot",
                           plotdays = -1, average = T, act_range = c(0,4000),
                           dayColor = c(rgb(141, 211, 199, max = 255),
                                        rgb(190, 186, 218, max = 255),
                                        rgb(251, 128, 114, max = 255),
                                        rgb(128, 177, 211, max = 255),
                                        rgb(253, 180, 98, max = 255),
                                        rgb(179, 222, 105, max = 255),
                                        rgb(252, 205, 229, max = 255),
                                        rgb(217, 217, 217, max = 255),
                                        rgb(188, 128, 189, max = 255),
                                        rgb(204, 235, 197, max = 255),
                                        rgb(255, 255, 179, max = 255),
                                        rgb(255, 237, 111, max = 255)
                           ),
                           avgColor = c(rgb(0, 1, 0)),lineType,
  legendPosition = "topright")
  {

            epoch = c(1:1440)

            for (i in plotdays)  {
              lines( data$Time[(data$DayNumber == i)],
                     data$ActSortSum[(data$DayNumber == i)],
                     lty=1,
                     lwd = 1.2,
                     col=dayColor[i]
              )


            } # end for

            plotted_data <- data[data$DayNumber %in% plotdays,]
#only plotted data
            ### overplot the average

            if(average){
              cumavgbase = rep(0, 1440)

              for (i in 1:1440)
              {
                cumavgbase[i] = mean(
plotted_data$ActSortSum[(plotted_data$Time == i)])
              }

              lines(1:1440, cumavgbase, col = avgColor, lwd = 2)

              ### done plotting the average
```

```
            }


            axis(1, , at = seq(0,1440, 360),
                 labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
 cex.axis = 0.6)

            axis(2, at = seq(0,500000, 100000),
 labels = c("0","100","200","300", "400", "500"),
 las=2, cex.axis = 0.6)
            mtext("Activity Level (in thousands)",
  line = 2, side = 2, cex = 0.6)


      #define the legend
      legendColor = NULL
      legendText = NULL
      for (i in plotdays){

        legendText = c(legendText, paste("Day ", i))
        legendColor = c(legendColor,dayColor[i])
      }



      if(average){
        legendText = c(legendText, "Average")
        legendColor = c(legendColor,avgColor)
      }

      legend(legendPosition,legendText, text.col = legendColor,
cex =0.85, col = legendColor)
}
```

A.10 Density Plot

```
plot.density <- function(actData, clinicalData, class = 1,
 level = 0, data_type= 0, plotdays =2:4,
 act_range, title, xlab, ylab,  adjust = 0.5,
 scale = 1, gamma = 1, colmin="#FFFFFF",
                         colmax="#000000"){

  #check if denstrip package is installed.
  if("denstrip" %in% rownames(installed.packages()) == FALSE)
{install.packages("denstrip")}
  library("denstrip") #load the package

  data <- NULL
  for(i in 1:length(actData)){
    data <- rbind(data, actData[[i]]$data)
  }
  actData <- data #now actData is combined

  if (class == 0){  #for gender levels

    #check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3
    if(data_type == 0){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$Act)
    }else if(data_type == 1){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSum)
    }else if(data_type == 2){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSort)
    }else if(data_type == 3){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSortSum)
    }


    files_id  <- clinicalData$ID
    num_of_patients <- length(unique(actData$ID))
#find out how many patients

    actData <- actData [actData$DayNumber %in% plotdays,]
# Read only data for day in "days"
    actData <- data.frame(time =rep(1:1440,
(num_of_patients*length(plotdays))) , actData)
# attach a time coloumn to data

    clinicalData_Level_male = clinicalData[which(clinicalData$gender ==1),]
    clinicalData_Level_female = clinicalData[which(clinicalData$gender ==2),]

    PatientsData_Level_male = NULL
```

```
    PatientsData_Level_female = NULL

    myAct <- actData

    for (i in files_id){

      mydata = myAct[myAct$ID == i,]

      if(i %in% clinicalData_Level_male$ID){
        PatientsData_Level_male = rbind(PatientsData_Level_male, mydata)
      }

      if(i %in% clinicalData_Level_female$ID){
        PatientsData_Level_female = rbind(PatientsData_Level_female, mydata)
      }

    } #end for loop

    if (level == 0){  #level males
      mydata <- PatientsData_Level_male
    }else if (level == 1 ){ #level females
      mydata <- PatientsData_Level_female
    }else{ #all levels of  depression
      mydata <- actData
      }

  }else if (class == 1){  #for depression levels

  #check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3
  if(data_type == 0){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$Act)
  }else if(data_type == 1){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSum)
  }else if(data_type == 2){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSort)
  }else if(data_type == 3){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSortSum)
  }

  files_id  <- clinicalData$ID
  num_of_patients <- length(unique(actData$ID))
  #find out how many patients

  actData <- actData [actData$DayNumber %in% plotdays,]
  # Read only data for day in "days"
  actData <- data.frame(time =rep(1:1440,
(num_of_patients*length(plotdays))) , actData)
```

```
  # attach a time coloumn to data

  #classify patients into 5 depression levels

  clinicalData_Level_0 = clinicalData[which(clinicalData$Level ==0 ),]
  clinicalData_Level_1 = clinicalData[which(clinicalData$Level ==1 ),]
  clinicalData_Level_2 = clinicalData[which(clinicalData$Level ==2 ),]
  clinicalData_Level_3 = clinicalData[which(clinicalData$Level ==3 ),]
  clinicalData_Level_4 = clinicalData[which(clinicalData$Level ==4 ),]


  #Initialize data frames actigraphy data for different depression levels
PatientsData_Level_0 = NULL
PatientsData_Level_1 = NULL
PatientsData_Level_2 = NULL
PatientsData_Level_3 = NULL
PatientsData_Level_4 = NULL

  myAct <- actData

  for (i in files_id){

   mydata = myAct[myAct$ID == i,]

   if(i %in% clinicalData_Level_0$ID){

   PatientsData_Level_0 = rbind(PatientsData_Level_0, mydata)
   }

    if(i %in% clinicalData_Level_1$ID){

   PatientsData_Level_1 = rbind(PatientsData_Level_1, mydata)
   }

   if(i %in% clinicalData_Level_2$ID){

   PatientsData_Level_2 = rbind(PatientsData_Level_2, mydata)
   }

   if(i %in% clinicalData_Level_3$ID){

   PatientsData_Level_3 = rbind(PatientsData_Level_3, mydata)
   }

   if(i %in% clinicalData_Level_4$ID){

   PatientsData_Level_4 = rbind(PatientsData_Level_4, mydata)
   }

  }# end of for loop
```

```
  if (level == 0){  #level 0 depression
      mydata <- PatientsData_Level_0
  }else if (level == 1 ){ #level 1 depression
      mydata <- PatientsData_Level_1
  }else if (level == 2){ #level 2 depression
      mydata <- PatientsData_Level_2
  }else if (level == 3){ #level 3 depression
      mydata <- PatientsData_Level_3
  }else if (level == 4){ #level 4 depression
      mydata <- PatientsData_Level_4
  }else{ #all levels of  depression
      mydata <- actData
  }

  } #end of depression levels


  plot(time, xlim=c(-50, 1450),
       ylim = c(act_range[1]-50, act_range[2]),
       main = title,
       xlab=xlab,
       ylab=ylab,
       axes = F,
       type="n")


  for (i in 1:1440)
  {
    act <- mydata$Act[which(mydata$time == i )]


    denstrip(density(act, from= act_range[1],
   to= act_range[2], adjust = adjust)$x,
              density(act,from=act_range[1], to=act_range[2],
 adjust = adjust)$y,
              horiz=FALSE,
              at = i,
              width = 1,
              colmin=colmin,
              colmax=colmax,
              scale = scale,
              gamma = gamma)

  }

  if(data_type == 0 | data_type == 1){
    axis(1, , at = seq(0,1440, 60),
labels = c("Midnight",rep(NA, 5),"6 AM",rep(NA, 5),"Noon",
   rep(NA, 5),"6 PM",rep(NA, 5),"Midnight"),
cex.axis = 0.6, pos = -100)
  }else if(data_type == 2 | data_type == 3){
```

```
     axis(1, , at = seq(0,1440, 60),
labels = c("0",rep(NA,5), "360", rep(NA,5), "720", rep(NA,5),
 "1080", rep(NA,5),"1440"), cex.axis = 0.6, pos = -100)
  }

  #For labeling
  min_range <- act_range[1]
  max_range <- act_range[2]
  first_quarter_range <- (max_range - min_range)/4
  half_range <- (max_range - min_range)/2
  second_quarter_range <- first_quarter_range + half_range


  axis(2, at = c(min_range, first_quarter_range,
half_range,second_quarter_range, max_range),
         labels = c(format(min_range, scientific = FALSE),
                  format(first_quarter_range, scientific = FALSE),
                  format(half_range, scientific = FALSE),
                  format(second_quarter_range, scientific = FALSE),
                  format(max_range, scientific = FALSE)))

}
```

A.11 Envelope Data

```
envelop.data <- function(actData, clinicalData, class = 1, data_type= 0,
agg_epoch =1, plotdays =2:4, lower_bound=0, upper_bound=1)
{

  data <- NULL
  for(i in 1:length(actData)){
    data <- rbind(data, actData[[i]]$data)
  }
  actData <- data #now actData is combined

  if(class == 0){ #for gender levels
  #check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3

  if(data_type == 0){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$Act)
  }else if(data_type == 1){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSum)
  }else if(data_type == 2){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSort)
  }else if(data_type == 3){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSortSum)
  }

  files_id = clinicalData$ID  #get files id for classification purposes

  #classify patients into 5 levels of depression
  clinicalData_Level_male = clinicalData[which(clinicalData$gender ==1
& clinicalData$ID %in% files_id),]
  clinicalData_Level_female = clinicalData[which(clinicalData$gender ==2
& clinicalData$ID %in% files_id),]


  ######## 3. Prepare data ########

  ### Initialize data frames actigraphy data for different gender levels
  PatientsData_Level_male = NULL
  PatientsData_Level_female = NULL

  for (i in files_id){

    patient_data = actData[actData$ID == i,]
# get data for patient of ID = i

    if(i %in% clinicalData_Level_male$ID){
```

```
      onePatientData = NULL
      ### Select the data corresponding to only days  defined above
      onePatientData = matrix(patient_data[patient_data$DayNumber
%in% plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

      ### Save data for all  male patients
      PatientsData_Level_male = rbind(PatientsData_Level_male, onePatientData)
    }

    if(i %in% clinicalData_Level_female$ID){

      onePatientData = NULL
      ### Select the data corresponding to only days
      onePatientData = matrix(patient_data[patient_data$DayNumber
%in% plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

      ### Save data for all female patients
      PatientsData_Level_female = rbind(PatientsData_Level_female, onePatientData)
    }


  }

  ### Produce 1440 * 10 matrices for PatientsData at all levels

  PatientsData_Level_male = as.vector(PatientsData_Level_male)
  PatientsData_Level_female = as.vector(PatientsData_Level_female)


  PatientsData_Level_male = t(matrix(PatientsData_Level_male,
1440/agg_epoch, length(PatientsData_Level_male)))
  PatientsData_Level_female = t(matrix(PatientsData_Level_female,
1440/agg_epoch, length(PatientsData_Level_female)))



  ### Filter Data

  #Males
  lower_bound_data <- apply(PatientsData_Level_male,
2, function(x) quantile(x, lower_bound, na.rm = TRUE))
  upper_bound_data <- apply(PatientsData_Level_male,
2, function(x) quantile(x, upper_bound, na.rm = TRUE))
  filtered_data_Level_male <- data.frame(lower_bound_data, upper_bound_data)

  #Females
  lower_bound_data <- apply(PatientsData_Level_female,
2, function(x) quantile(x, lower_bound, na.rm = TRUE))
  upper_bound_data <- apply(PatientsData_Level_female,
2, function(x) quantile(x, upper_bound, na.rm = TRUE))
  filtered_data_Level_female <- data.frame(lower_bound_data,
```

```
upper_bound_data)


  # combine all enveloped data into one data frame
  filtered_data_All <- rbind(filtered_data_Level_male, filtered_data_Level_female)

  return(filtered_data_All)

  }
  else if (class == 1){  #for depression levels


    #check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3

    if(data_type == 0){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$Act)
    }else if(data_type == 1){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSum)
    }else if(data_type == 2){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSort)
    }else if(data_type == 3){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSortSum)
    }

    files_id = clinicalData$ID  #get files id for classification purposes

    #classify patients into 5 levels of depression
    clinicalData_Level_0 = clinicalData[which(clinicalData$Level ==0
& clinicalData$ID %in% files_id),]
    clinicalData_Level_1 = clinicalData[which(clinicalData$Level ==1
& clinicalData$ID %in% files_id),]
    clinicalData_Level_2 = clinicalData[which(clinicalData$Level ==2
& clinicalData$ID %in% files_id),]
    clinicalData_Level_3 = clinicalData[which(clinicalData$Level ==3
& clinicalData$ID %in% files_id),]
    clinicalData_Level_4 = clinicalData[which(clinicalData$Level ==4
& clinicalData$ID %in% files_id),]


    ######## 3. Prepare data ########

    ### Initialize data frames actigraphy data for different depression levels
    PatientsData_Level_0 = NULL
    PatientsData_Level_1 = NULL
    PatientsData_Level_2 = NULL
    PatientsData_Level_3 = NULL
```

```
    PatientsData_Level_4 = NULL


    for (i in files_id){

      patient_data = actData[actData$ID == i,] # get data for patient of ID = i


      if(i %in% clinicalData_Level_0$ID){

        onePatientData  = NULL
        ### Select the data corresponding to only days  defined above
        onePatientData = matrix(patient_data[patient_data$DayNumber %in%
plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

        ### Save data for all patients
        PatientsData_Level_0 = rbind(PatientsData_Level_0, onePatientData)
      }

      if(i %in% clinicalData_Level_1$ID){

        onePatientData = NULL
        ### Select the data corresponding to only days
        onePatientData = matrix(patient_data[patient_data$DayNumber %in%
plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

        ### Save data for all patients
        PatientsData_Level_1 = rbind(PatientsData_Level_1, onePatientData)
      }

      if(i %in% clinicalData_Level_2$ID){

        onePatientData = NULL
        ### Select the data corresponding to only days
        onePatientData = matrix(patient_data[patient_data$DayNumber %in%
plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

        ### Save data for all patients
        PatientsData_Level_2 = rbind(PatientsData_Level_2, onePatientData)
      }

      if(i %in% clinicalData_Level_3$ID){

        onePatientData = NULL
        ### Select the data corresponding to only days
        onePatientData = matrix(patient_data[patient_data$DayNumber %in%
plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

        ### Save data for all patients
        PatientsData_Level_3 = rbind(PatientsData_Level_3, onePatientData)
      }
```

```
      if(i %in% clinicalData_Level_4$ID){

        onePatientData = NULL
        ### Select the data corresponding to only days
        onePatientData = matrix(patient_data[patient_data$DayNumber %in%
plotdays,]$Act, ncol = length(plotdays), byrow = FALSE )

        ### Save data for all patients
        PatientsData_Level_4 = rbind(PatientsData_Level_4, onePatientData)
      }


    }

    ### Produce 1440 * 10 matrices for PatientsData at all levels

    PatientsData_Level_0 = as.vector(PatientsData_Level_0)
    PatientsData_Level_1 = as.vector(PatientsData_Level_1)
    PatientsData_Level_2 = as.vector(PatientsData_Level_2)
    PatientsData_Level_3 = as.vector(PatientsData_Level_3)
    PatientsData_Level_4 = as.vector(PatientsData_Level_4)

    PatientsData_Level_0 = t(matrix(PatientsData_Level_0,
1440/agg_epoch, length(clinicalData_Level_0$Level)))
    PatientsData_Level_1 = t(matrix(PatientsData_Level_1,
1440/agg_epoch, length(clinicalData_Level_1$Level)))
    PatientsData_Level_2 = t(matrix(PatientsData_Level_2,
1440/agg_epoch, length(clinicalData_Level_2$Level)))
    PatientsData_Level_3 = t(matrix(PatientsData_Level_3,
1440/agg_epoch, length(clinicalData_Level_3$Level)))
    PatientsData_Level_4 = t(matrix(PatientsData_Level_4,
1440/agg_epoch, length(clinicalData_Level_4$Level)))
    ### Filter Data

    #Level 0
    lower_bound_data <- apply(PatientsData_Level_0, 2,
function(x) quantile(x, lower_bound, na.rm = TRUE))
    upper_bound_data <- apply(PatientsData_Level_0, 2,
function(x) quantile(x, upper_bound, na.rm = TRUE))
    filtered_data_Level_0 <- data.frame(lower_bound_data, upper_bound_data)

    #Level 1
    lower_bound_data <- apply(PatientsData_Level_1, 2,
function(x) quantile(x, lower_bound, na.rm = TRUE))
    upper_bound_data <- apply(PatientsData_Level_1, 2,
function(x) quantile(x, upper_bound, na.rm = TRUE))
    filtered_data_Level_1 <- data.frame(lower_bound_data, upper_bound_data)

    #Level 2
    lower_bound_data <- apply(PatientsData_Level_2, 2,
```

```
function(x) quantile(x, lower_bound, na.rm = TRUE))
    upper_bound_data <- apply(PatientsData_Level_2, 2,
function(x) quantile(x, upper_bound, na.rm = TRUE))
    filtered_data_Level_2 <- data.frame(lower_bound_data, upper_bound_data)

    #Level 3
    lower_bound_data <- apply(PatientsData_Level_3, 2,
function(x) quantile(x, lower_bound, na.rm = TRUE))
    upper_bound_data <- apply(PatientsData_Level_3, 2,
function(x) quantile(x, upper_bound, na.rm = TRUE))
    filtered_data_Level_3 <- data.frame(lower_bound_data, upper_bound_data)

    #Level 4
    lower_bound_data <- apply(PatientsData_Level_4, 2,
function(x) quantile(x, lower_bound, na.rm = TRUE))
    upper_bound_data <- apply(PatientsData_Level_4, 2,
function(x) quantile(x, upper_bound, na.rm = TRUE))
    filtered_data_Level_4 <- data.frame(lower_bound_data, upper_bound_data)

    # combine all enveloped data into one data frame
    filtered_data_All <- rbind(filtered_data_Level_0, filtered_data_Level_1,
filtered_data_Level_2, filtered_data_Level_3,
filtered_data_Level_4)

    return(filtered_data_All)
  }

}
```

A.12 Plot Envelope Data

```
plot.envelope <- function (actData, class = 1, data_type= 0, agg_epoch = 10,
title, act_range, xlab, ylab, col = -1){

  if (class == 0){ #for gender levels
  par(bg = "transparent")

  #check if color is specified, otherwise use default
  if(col[1] == -1){
    col = c(
      rgb((228/255),(26/255), (28/255), alpha =0.9),
      rgb((55/255),(26/255),(184/255), alpha =0.8),
      rgb((77/255),(175/255),(74/255), alpha =0.7),
      rgb((152/255),(78/255),(163/255),alpha =0.6),
      rgb((255/255),(127/255),(0/255), alpha =0.5)
    )
  }

  plot(1:(1440/agg_epoch),
       type = "n",
       xlab = xlab,
       ylab = ylab,
       axes = FALSE,
       main = title,
       ylim = act_range)


  xx = c(1:(1440/agg_epoch), (1440/agg_epoch):1)
  lower = 1
  upper = 1440/agg_epoch

  for (count in c(1:2)){

  yy = c(actData$lower_bound_data[lower:upper],
rev(actData$upper_bound_data[lower:upper]))
  polygon(xx,yy,col=col[count], border = col[count])

  lower = upper + 1
  upper = upper + (1440/agg_epoch)
  count = count + 1

  }

  # Check if the data is sorted or not...
  #if sorted, then use order, otherwise, it is time.

  if(data_type == 0 | data_type == 1){
       axis(1, at = c(0/agg_epoch, 60/agg_epoch, 120/agg_epoch, 180/agg_epoch,
240/agg_epoch, 300/agg_epoch, 360/agg_epoch,
                      420/agg_epoch, 480/agg_epoch, 540/agg_epoch,
600/agg_epoch, 660/agg_epoch, 720/agg_epoch,
```

```
                    780/agg_epoch, 840/agg_epoch, 900/agg_epoch,
960/agg_epoch, 1020/agg_epoch,1080/agg_epoch,
                    1140/agg_epoch, 1200/agg_epoch, 1260/agg_epoch,
1320/agg_epoch, 1380/agg_epoch,1440/agg_epoch),
               labels = c("12 AM",rep(NA,5), "6 AM", rep(NA,5), "12 PM",
rep(NA,5), "6 PM", rep(NA,5),"12 AM"))
  } else if (data_type == 2 | data_type == 3){
      axis(1, at = c(0/agg_epoch, 60/agg_epoch, 120/agg_epoch, 180/agg_epoch,
 240/agg_epoch, 300/agg_epoch, 360/agg_epoch,
                    420/agg_epoch, 480/agg_epoch, 540/agg_epoch,
 600/agg_epoch, 660/agg_epoch, 720/agg_epoch,
                    780/agg_epoch, 840/agg_epoch, 900/agg_epoch,
 960/agg_epoch, 1020/agg_epoch,1080/agg_epoch,
                    1140/agg_epoch, 1200/agg_epoch, 1260/agg_epoch,
 1320/agg_epoch, 1380/agg_epoch,1440/agg_epoch),
              labels = c("0",rep(NA,5), "360", rep(NA,5), "720", rep(NA,5),
  "1080", rep(NA,5),"1440"))

  }
  #For labeling
  min_range <- act_range[1]
  max_range <- act_range[2]
  first_quarter_range <- (max_range - min_range)/4
  half_range <- (max_range - min_range)/2
  second_quarter_range <- first_quarter_range + half_range


  axis(2, at = c(min_range, first_quarter_range,half_range,
second_quarter_range, max_range),
          labels = c(format(min_range, scientific = FALSE),
                     format(first_quarter_range, scientific = FALSE),
                     format(half_range, scientific = FALSE),
                     format(second_quarter_range, scientific = FALSE),
                     format(max_range, scientific = FALSE)))


  #define the legend

 legendText = c("Males", "Females")
 legendColor = col

 legend("topleft",legendText, text.col = legendColor,
cex = 0.7, col = legendColor)

  }else if (class == 1){ #for depression levels

    par(bg = "transparent")

    #check if color is specified, otherwise use default
    if(col[1] == -1){
      col = c(
```

```
        rgb((228/255),(26/255), (28/255), alpha =0.9),
        rgb((55/255),(26/255),(184/255), alpha =0.8),
        rgb((77/255),(175/255),(74/255), alpha =0.7),
        rgb((152/255),(78/255),(163/255),alpha =0.6),
        rgb((255/255),(127/255),(0/255), alpha =0.5)
      )
    }

    plot(1:(1440/agg_epoch),
         type = "n",
         xlab = xlab,
         ylab = ylab,
         axes = FALSE,
         main = title,
         ylim = act_range)


    xx = c(1:(1440/agg_epoch), (1440/agg_epoch):1)
    lower = 1
    upper = 1440/agg_epoch

    for (count in c(1:5)){

      yy = c(actData$lower_bound_data[lower:upper],
rev(actData$upper_bound_data[lower:upper]))
      polygon(xx,yy,col=col[count], border = col[count])

      lower = upper + 1
      upper = upper + (1440/agg_epoch)
      count = count + 1

    }


    # Check if the data is sorted or not...
    #if sorted, then use order, otherwise, it is time.

    if(data_type == 0 | data_type == 1){
      axis(1, at = c(0/agg_epoch, 60/agg_epoch, 120/agg_epoch, 180/agg_epoch,
 240/agg_epoch, 300/agg_epoch, 360/agg_epoch,
                     420/agg_epoch, 480/agg_epoch, 540/agg_epoch,
 600/agg_epoch, 660/agg_epoch, 720/agg_epoch,
                     780/agg_epoch, 840/agg_epoch, 900/agg_epoch,
 960/agg_epoch, 1020/agg_epoch,1080/agg_epoch,
                     1140/agg_epoch, 1200/agg_epoch, 1260/agg_epoch,
 1320/agg_epoch, 1380/agg_epoch,1440/agg_epoch),
           labels = c("12 AM",rep(NA,5), "6 AM", rep(NA,5), "12 PM",
rep(NA,5), "6 PM", rep(NA,5),"12 AM"))
    } else if (data_type == 2 | data_type == 3){
      axis(1, at = c(0/agg_epoch, 60/agg_epoch, 120/agg_epoch, 180/agg_epoch,
 240/agg_epoch, 300/agg_epoch, 360/agg_epoch,
```

```
                    420/agg_epoch, 480/agg_epoch, 540/agg_epoch,
 600/agg_epoch, 660/agg_epoch, 720/agg_epoch,
                    780/agg_epoch, 840/agg_epoch, 900/agg_epoch,
 960/agg_epoch, 1020/agg_epoch,1080/agg_epoch,
                    1140/agg_epoch, 1200/agg_epoch, 1260/agg_epoch,
 1320/agg_epoch, 1380/agg_epoch,1440/agg_epoch),
          labels = c("0",rep(NA,5), "360", rep(NA,5), "720", rep(NA,5),
  "1080", rep(NA,5),"1440"))

    }
    #For labeling
    min_range <- act_range[1]
    max_range <- act_range[2]
    first_quarter_range <- (max_range - min_range)/4
    half_range <- (max_range - min_range)/2
    second_quarter_range <- first_quarter_range + half_range


    axis(2, at = c(min_range, first_quarter_range,half_range,
   second_quarter_range, max_range),
         labels = c(format(min_range, scientific = FALSE),
                    format(first_quarter_range, scientific = FALSE),
                    format(half_range, scientific = FALSE),
                    format(second_quarter_range, scientific = FALSE),
                    format(max_range, scientific = FALSE)))


    #define the legend

    legendText = c("Depression Level 0", "Depression Level 1",
   "Depression Level 2", "Depression Level 3",
"Depression Level 4")
    legendColor = col

    legend("topleft",legendText, text.col = legendColor, cex = 0.7,
col = legendColor)
  }
}
```

A.13 Mvts Plot

```
plot.mvts <- function(actData, clinicalData, class = 1,
data_type= 0, plotdays =2:6,
                          title, norm = "global"){

  #check if denstrip package is installed.
  if("RColorBrewer" %in% rownames(installed.packages()) == FALSE)
{install.packages("RColorBrewer")}
  library("RColorBrewer") #load the package


  data <- NULL
  for(i in 1:length(actData)){
    data <- rbind(data, actData[[i]]$data)
  }
  actData <- data #now actData is combined

  if (class == 0){  #for gender levels

    #check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3
    if(data_type == 0){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$Act)
    }else if(data_type == 1){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSum)
    }else if(data_type == 2){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSort)
    }else if(data_type == 3){
      actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSortSum)
    }


    files_id  <- clinicalData$ID
    num_of_patients <- length(unique(actData$ID))
#find out how many patients

    actData <- actData [actData$DayNumber %in% plotdays,]
# Read only data for day in "days"
    actData <- data.frame(time =rep(1:1440,
(num_of_patients*length(plotdays))) , actData)
# attach a time coloumn to data

    clinicalData_Level_male = clinicalData[which(clinicalData$gender ==1),]
    clinicalData_Level_female = clinicalData[which(clinicalData$gender ==2),]

    PatientsData_Level_male = NULL
    PatientsData_Level_female = NULL
```

```
myAct <- actData

for (i in files_id){
  mydata = myAct[myAct$ID == i,]

  if(i %in% clinicalData_Level_male$ID){

    PatientsData_Level_male = rbind(PatientsData_Level_male, mydata)
  }

  if(i %in% clinicalData_Level_female$ID){

    PatientsData_Level_female = rbind(PatientsData_Level_female, mydata)
  }

} #end for loop

## Cluster the data based on the depression levels
sorted_data <- rbind( PatientsData_Level_male,  PatientsData_Level_female)

# This is used for grouping purposes in the mvtsplot
group0_length <- length(PatientsData_Level_male$ID)/(1440 * length(plotdays))
group1_length <- length( PatientsData_Level_female$ID)/(1440 * length(plotdays

act <- sorted_data$Act
act    = matrix(act, ncol = 1440, byrow = TRUE)

new_act <- NULL

start <- 1
for(i in 1:num_of_patients){
  end <- i*length(plotdays)
  aggregated_act  = apply(act[start:end,], 2, mean)
  new_act <- rbind(new_act, aggregated_act)
  start <- end + 1
}

names <- NULL
if(group0_length >0){
  males <- c("Males", rep(NA, group0_length - 1))
  names <- c(names, males)
  }
if(group1_length >0){
  females <- c("Females", rep(NA, group1_length - 1))
  names <- c(names, females)
  }

rownames(new_act) <- names

mvtsplot(t(new_act),
```

```
            group = c(rep(0,group0_length), rep(1,group1_length)),
            norm = norm ,main= title)
  }else if (class == 1){  #for depression levels

  #check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3
  if(data_type == 0){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$Act)
  }else if(data_type == 1){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSum)
  }else if(data_type == 2){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSort)
  }else if(data_type == 3){
    actData <- data.frame(ID=actData$ID,
DayNumber= actData$DayNumber, Act=actData$ActSortSum)
  }


  files_id  <- clinicalData$ID
  num_of_patients <- length(unique(actData$ID))
#find out how many patients

  actData <- actData [actData$DayNumber %in% plotdays,]
# Read only data for day in "days"
  actData <- data.frame(time =rep(1:1440,
(num_of_patients*length(plotdays))) , actData)
# attach a time coloumn to data

  #classify patients into 5 depression levels

  clinicalData_Level_0 = clinicalData[which(clinicalData$Level ==0 ),]
  clinicalData_Level_1 = clinicalData[which(clinicalData$Level ==1 ),]
  clinicalData_Level_2 = clinicalData[which(clinicalData$Level ==2 ),]
  clinicalData_Level_3 = clinicalData[which(clinicalData$Level ==3 ),]
  clinicalData_Level_4 = clinicalData[which(clinicalData$Level ==4 ),]


  ### Initialize data frames actigraphy data for different depression levels
PatientsData_Level_0 = NULL
PatientsData_Level_1 = NULL
PatientsData_Level_2 = NULL
PatientsData_Level_3 = NULL
PatientsData_Level_4 = NULL

  myAct <- actData

  for (i in files_id){

   mydata = myAct[myAct$ID == i,]
```

```
   if(i %in% clinicalData_Level_0$ID){

   PatientsData_Level_0 = rbind(PatientsData_Level_0, mydata)
   }

    if(i %in% clinicalData_Level_1$ID){

   PatientsData_Level_1 = rbind(PatientsData_Level_1, mydata)
   }

   if(i %in% clinicalData_Level_2$ID){

   PatientsData_Level_2 = rbind(PatientsData_Level_2, mydata)
   }

   if(i %in% clinicalData_Level_3$ID){

   PatientsData_Level_3 = rbind(PatientsData_Level_3, mydata)
   }

   if(i %in% clinicalData_Level_4$ID){

   PatientsData_Level_4 = rbind(PatientsData_Level_4, mydata)
   }

  }# end of for loop

  ## Cluster the data based on the depression levels

  sorted_data <- rbind(PatientsData_Level_4, PatientsData_Level_3,
                       PatientsData_Level_2, PatientsData_Level_2,
                       PatientsData_Level_0)

  # This is used for grouping purposes in the mvtsplot
  group0_length <- length(PatientsData_Level_0$ID)/
(1440 * length(plotdays))
  group1_length <- length(PatientsData_Level_1$ID)/
(1440 * length(plotdays))
  group2_length <- length(PatientsData_Level_2$ID)/
(1440 * length(plotdays))
  group3_length <- length(PatientsData_Level_3$ID)/
(1440 * length(plotdays))
  group4_length <- length(PatientsData_Level_4$ID)/
(1440 * length(plotdays))

  act <- sorted_data$Act
  act    = matrix(act, ncol = 1440, byrow = TRUE)

  new_act <- NULL
```

```
  start <- 1
  for(i in 1:num_of_patients){
    end <- i*length(plotdays)
    aggregated_act  = apply(act[start:end,], 2, mean)
    new_act <- rbind(new_act, aggregated_act)
    start <- end + 1
  }

  names <- NULL
  if(group4_length >0){
    level_5 <- c("Level 5", rep(NA, group4_length - 1))
    names <- c(names, level_5)
    }
  if(group3_length >0){
    level_4 <- c("Level 4", rep(NA, group3_length - 1))
    names <- c(names, level_4)}
  if(group2_length >0){
    level_3 <- c("Level 3", rep(NA, group2_length - 1))
    names <- c(names, level_3)
    }
  if(group1_length >0){
    level_2 <- c("Level 2", rep(NA, group1_length - 1))
    names <- c(names, level_2)}
  if(group0_length >0){
    level_1 <- c("Level 1", rep(NA, group0_length - 1))
    names <- c(names, level_1)
    }

  rownames(new_act) <- names


  mvtsplot(t(new_act),
           group = c(rep(0,group4_length), rep(1,group3_length),
rep(2, group2_length), rep(3, group1_length),
rep(4, group0_length)),
           norm = "global",main= title)
  } #end of depression levels

}
```

APPENDIX B

OBJECT ORIENTED PARADIGM (R CODE)

B.1 ActData

```
ActData <-setRefClass("ActData",

fields = list(
fileName <- "character",
#file name for the patient
ID <- "numeric",
#id of the patient
gender <-"character",
#gender of the patient
age <- "numeric",
#age of the patient
height <- "numeric",
#height of the patient
weight <- "numeric",
#weight of the patient
epoch_rate <- "numeric",
#how often is the data being recorded
data <- "data.frame",
#to hold raw actigraphy data
agg_epoch <- "numeric",
#how many minutes to sum? 10, 20, 30, or 60 mins
agg_data <- "data.frame"
#to hold aggregated actigraphy data ),

methods = list(

      initialize = function(...){
  agg_epoch <<- 1
  initFields(...)
         },  # end of initialize method

      read = function(){
       source("R_read_awc.R")
  ### change directory accordingly
       data <<- read.awc(fileName)
              ID <<- as.numeric(read.fwf(fileName,
width=12, skip=0, n=1))
       gender <<- ifelse(as.character(
read.fwf(fileName, width=12,
```

```
skip=6, n=1))=="1", 'M','F')
              age <<- as.numeric(read.fwf(fileName,
width=12, skip=4, n=1))
              height <<- as.numeric(read.fwf(fileName,
width=12, skip=7, n=1))
       weight <<- as.numeric(read.fwf(fileName,
width=12, skip=8, n=1))
              epoch_rate <<- as.numeric(read.fwf(fileName,
width=12, skip=3, n=1))

  if (epoch_rate == 1){
#if data is recorded every 15 secodns
           source("R Functions/R_aggregate_sum_to_1min.R")
  ### change directory accordingly
           data <<- aggregate_sum_to_1min(data)
  ### aggregate to 1 minute
         }
         }, # end of read method

      sum = function(){
              source("R_aggregate_sum.R")
  ### change directory accordingly
              agg_data <<- aggregate_sum(data, agg_epoch)

      }

      )
)
```

B.2 Graph

```
Graph <- setRefClass("Graph",

fields = list(
      domain ="numeric",
act_range  = "numeric",
xlab  = "character",
ylab  = "character",
title  = "character",
axes  = "logical",
legendPosition = "character"
# where to place the legend
),

methods = list(

 setup = function(){
plot(domain,
 act_range,
 xlab = xlab,
 ylab = ylab,
           xlim = domain,
           ylim = act_range,
 main = title,
           type = 'n',
 axes = axes
 )
} # end setup method

)
)
```

B.3 RawDataPlot

```
RawDataPlot <- setRefClass("rawDataPlot",

  contains = "Graph",  # inherits from class Graph

  fields = list(

    plotdays = "numeric",     # days to plot
    average  = "logical",     # should we plot the average
    dayColor = "character",   # list of colors for the day
    avgColor = "character",   # color for the average plot
    patient  = "ActData"      # object of class ActData to plot
              ),


methods = list(

  initialize = function(...){
    domain <<- c(1,1440)
    act_range <<- c(0,4000)
    xlab <<- "Time"
    ylab <<- ""
    title <<- "Patient Raw Data Plot"
    axes <<- F
    legendPosition <<- c("topright")
    dayColor <<- c(rgb(141, 211, 199, max = 255),
                   rgb(190, 186, 218, max = 255),
                   rgb(251, 128, 114, max = 255),
                   rgb(128, 177, 211, max = 255),
                   rgb(253, 180, 98, max = 255),
                   rgb(179, 222, 105, max = 255),
                   rgb(252, 205, 229, max = 255),
                   rgb(217, 217, 217, max = 255),
                   rgb(188, 128, 189, max = 255),
                   rgb(204, 235, 197, max = 255),
                   rgb(255, 255, 179, max = 255),
                   rgb(255, 237, 111, max = 255)
    )
    plotdays <<- 2:5   #all days by default
    average <<- T
    avgColor <<- c(rgb(0, 1, 0))

    initFields(...)
  },  #end of initialize() method

showData = function(){

        source("R Functions/R_rawdata.plot.R")
        rawdata.plot(patient$data, title= title,
                                  plotdays = plotdays, average = average,
```

```
                                    act_range = act_range,
                                    dayColor = dayColor,
                                    avgColor = avgColor,
                                    legendPosition = legendPosition
                                    )

           }, # end showData() method

showAvg = function(){

            act <- patient$data
avg = rep(1, 1440)

for (i in 1:1440)
  {
     avg[i] = mean(act$Act[act$Time == i], na.rm=T)
  }

  lines(1:1440, avg, col = avgColor, lwd = 2)


   axis(1, , at = seq(0,1440, 360),
labels = c("Midnight","6 AM","Noon","6 PM","Midnight"))
   axis(2, las=2)
               mtext("Activity Level",
at = ((act_range[2] + act_range[1])/2),
line = 3, side = 2, cex = 1)
     }# end showAvg() method

) # end list of methods

)
```

B.4 SmoothedDataPlot

```
SmoothedDataPlot <- setRefClass("smoothedDataPlot",

contains = "Graph",

   fields = list(
     plotdays = "numeric",     # days to plot
     average  = "logical",     # should we plot the average
     dayColor = "character",   # list of colors for the day
     avgColor = "character",   # color for the average plot
     lineType= "character",    # a list of different line types
     patient  = "ActData"      # object of class ActData to plot
     ),


 methods = list(

    initialize = function(...){
      domain <<- c(1,1440)
      act_range <<- c(0,4000)
      xlab <<- "Time"
      ylab <<- ""
      title <<- "Patient Smoothed Data Plot"
      axes <<- F
      legendPosition <<- c("topright")
      dayColor <<- c(rgb(141, 211, 199, max = 255),
                     rgb(190, 186, 218, max = 255),
                     rgb(251, 128, 114, max = 255),
                     rgb(128, 177, 211, max = 255),
                     rgb(253, 180, 98, max = 255),
                     rgb(179, 222, 105, max = 255),
                     rgb(252, 205, 229, max = 255),
                     rgb(217, 217, 217, max = 255),
                     rgb(188, 128, 189, max = 255),
                     rgb(204, 235, 197, max = 255),
                     rgb(255, 255, 179, max = 255),
                     rgb(255, 237, 111, max = 255)
      )
      plotdays <<- 2:5   #all days by default
      average <<- T
      avgColor <<- c(rgb(0, 1, 0))
      lineType <<- c("solid","dashed", "dashed", "solid", "solid", "solid")

      initFields(...)
        },  #end of initialize() method

    showData = function(){

      source("R Functions/R_smoothdata.plot.R")
      smoothdata.plot(patient$data, title= title,
```

```
                        plotdays = plotdays, average = average,
                        act_range = act_range,
                        dayColor = dayColor,
                        avgColor = avgColor,
                        lineType = lineType,
                        legendPosition = legendPosition
                        )

     }, # end showData() method

showAvg = function(){

          act <- patient$data
lowess_act = NULL
  lowess_data = NULL # to include time and lowess_act

 #epoch = c(1:1440)
   for (i in plotdays)  {
    lowess_act = c(lowess_act,
lowess(act$Act[(act$DayNumber == i)], f = .1)$y)
} # end for

     time = rep(1:1440, length(plotdays))
  lowess_data = data.frame(time, lowess_act)



  avg = rep(1, 1440)

for (i in 1:1440)
  {
   avg[i] = mean(lowess_data$lowess_act[lowess_data$time == i],
na.rm=T)
   }

  lines(1:1440, avg, col = avgColor, lwd = 2)

      axis(1, , at = seq(0,1440, 360),
          labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
cex.axis = 0.6)

      axis(2, las=2, cex.axis = 0.5)
      mtext("Activity Level", at = ((act_range[2] + act_range[1])/2),
line = 2, side = 2, cex = 1)

     }# end showAvg() method

) # end list of methods

)
```

B.5 VelocityPlot

```
VelocityPlot <- setRefClass("VelocityPlot",

  contains = "Graph",
  fields = list(

    plotdays = "numeric",     # days to plot
    average  = "logical",     # should we plot the average
    dayColor = "character",   # list of colors for the day
    avgColor = "character",   # color for the average plot
    lineType= "character",    # a list of different line types
    patient  = "ActData"      # object of class ActData to plot
    ),


  methods = list(

    initialize = function(...){
      domain <<- c(1,1440)
      act_range <<- c(0,4000)
      xlab <<- "Time"
      ylab <<- ""
      title <<- "Patient Velocity Data Plot"
      axes <<- F
      legendPosition <<- c("topright")
      dayColor <<- c(rgb(141, 211, 199, max = 255),
                     rgb(190, 186, 218, max = 255),
                     rgb(251, 128, 114, max = 255),
                     rgb(128, 177, 211, max = 255),
                     rgb(253, 180, 98, max = 255),
                     rgb(179, 222, 105, max = 255),
                     rgb(252, 205, 229, max = 255),
                     rgb(217, 217, 217, max = 255),
                     rgb(188, 128, 189, max = 255),
                     rgb(204, 235, 197, max = 255),
                     rgb(255, 255, 179, max = 255),
                     rgb(255, 237, 111, max = 255)
      )
      plotdays <<- 2:5   #all days by default
      average <<- T
      avgColor <<- c(rgb(0, 1, 0))
      lineType <<- c("solid","dashed", "dashed", "solid", "solid", "solid")

      initFields(...)
    },  #end of initialize() method

    showData = function(act){

      source("R Functions/R_velocity.plot.R")
      velocity.plot(patient$data, title= title,
```

```
                      plotdays = plotdays, average = average,
                      act_range = act_range,
                      dayColor = dayColor,
                      avgColor = avgColor,
                      lineType = lineType,
                      legendPosition = legendPosition
      )

     }, # end showData() method

showAvg = function(){

act <- patient$data

            lowessbase = NULL
diffbase = NULL
diff_data = NULL
velocity_data = NULL


avg = rep(1, 1440)

for (i in plotdays)  {
  #Compute lowess data
  lowessbase = lowess(act$Act[ (act$DayNumber == i)], f = .1)$y
  #Compute first derivative of lowess data
  diffbase = c(diff(lowessbase), lowessbase[1439])
  diff_data = c(diff_data,
lowess(diffbase, f = .02)$y)
#Get all derivative data into one col.

} # end for

time = rep(1:1440, length(plotdays))
velocity_data = data.frame(time, diff_data)
# dataframe derivative data and time

avg = 1:1440

for (j in 1:1440)
{
  avg[j] = mean(velocity_data$diff_data[velocity_data$time == j],
na.rm=T)
}
lines(1:1440, avg, col = avgColor, lwd = 2)

### done plotting the average

axis(1, , at = seq(0,1440, 360),
     labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
 cex.axis = 0.6)
```

```
axis(2, las=2, cex.axis = 0.5)
mtext("Activity Level", at = ((act_range[2] + act_range[1])/2),
line = 2, side = 2, cex = 1)

     }# end showAvg() method

) # end list of methods

)
```

B.6 AccelerationPlot

```
AccelerationPlot <- setRefClass("AccelerationPlot",

  contains = "Graph",
  fields = list(

    plotdays = "numeric",     # days to plot
    average  = "logical",     # should we plot the average
    dayColor = "character",   # list of colors for the day
    avgColor = "character",   # color for the average plot
    lineType= "character",    # a list of different line types
    patient  = "ActData"      # object of class ActData to plot
    ),


  methods = list(

    initialize = function(...){
      domain <<- c(1,1440)
      act_range <<- c(0,4000)
      xlab <<- "Time"
      ylab <<- ""
      title <<- "Patient Acceleration Data Plot"
      axes <<- F
      legendPosition <<- c("topright")
      dayColor <<- c(rgb(141, 211, 199, max = 255),
                     rgb(190, 186, 218, max = 255),
                     rgb(251, 128, 114, max = 255),
                     rgb(128, 177, 211, max = 255),
                     rgb(253, 180, 98, max = 255),
                     rgb(179, 222, 105, max = 255),
                     rgb(252, 205, 229, max = 255),
                     rgb(217, 217, 217, max = 255),
                     rgb(188, 128, 189, max = 255),
                     rgb(204, 235, 197, max = 255),
                     rgb(255, 255, 179, max = 255),
                     rgb(255, 237, 111, max = 255)
      )
      plotdays <<- 2:5   #all days by default
      average <<- T
      avgColor <<- c(rgb(0, 1, 0))
      lineType <<- c("solid","dashed", "dashed", "solid", "solid", "solid")

      initFields(...)
    },  #end of initialize() method

    showData = function(act){

      source("R Functions/R_acceleration.plot.R")
      acceleration.plot(patient$data, title= title,
```

```
                      plotdays = plotdays, average = average,
                      act_range = act_range,
                      dayColor = dayColor,
                      avgColor = avgColor,
                      lineType = lineType,
                      legendPosition = legendPosition
      )

     }, # end showData() method

showAvg = function(){

act <- patient$data

lowessbase = NULL
    diffbase = NULL
    diffbase2 = NULL
   diff_data = NULL
    velocity_data = NULL


avg = rep(1, 1440)

  for (i in plotdays)  {
#Compute lowess data
  lowessbase = lowess(act$Act[ (act$DayNumber == i)], f = .1)$y
#Compute first derivative of lowess data
 diffbase = c(diff(lowessbase), lowessbase[1439])
 #Compute Second derivative of lowess data
   diffbase2  = c(diff(lowess(diffbase, f = .02)$y),
diffbase[1440] - diffbase[1439])

        diff_data = c(diff_data, lowess(diffbase2, f = .02)$y)
#Get all derivative data into one col.

} # end for

time = rep(1:1440, length(plotdays))
  acceleration_data = data.frame(time, diff_data)
# dataframe derivative data and time

  avg = 1:1440

for (j in 1:1440)
  {
   avg[j] = mean(acceleration_data$diff_data[acceleration_data$time == j],
  na.rm=T)
  }
  lines(1:1440, avg, col = avgColor, lwd = 2)

### done plotting the average
```

```
      axis(1, , at = seq(0,1440, 360),
      labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
 cex.axis = 0.6)


       axis(2, las=2, cex.axis = 0.5)
       mtext("Acceleration of Activity", at = ((act_range[2] + act_range[1])/2),
line = 2, side = 2, cex = 1)
     }# end showAvg() method

) # end list of methods

)
```

B.7 CumSumsPlot

```
CumSumsPlot <- setRefClass("CumSumsPlot",

  contains = "Graph",
  fields = list(

    plotdays = "numeric",     # days to plot
    average  = "logical",     # should we plot the average
    dayColor = "character",   # list of colors for the day
    avgColor = "character",   # color for the average plot
    lineType= "character",    # a list of different line types
    patient  = "ActData"      # object of class ActData to plot
    ),


  methods = list(

    initialize = function(...){
      domain <<- c(1,1440)
      act_range <<- c(0,4000)
      xlab <<- "Time"
      ylab <<- ""
      title <<- "Patient Cumulative Sums Data Plot"
      axes <<- F
      legendPosition <<- c("topright")
      dayColor <<- c(rgb(141, 211, 199, max = 255),
                     rgb(190, 186, 218, max = 255),
                     rgb(251, 128, 114, max = 255),
                     rgb(128, 177, 211, max = 255),
                     rgb(253, 180, 98, max = 255),
                     rgb(179, 222, 105, max = 255),
                     rgb(252, 205, 229, max = 255),
                     rgb(217, 217, 217, max = 255),
                     rgb(188, 128, 189, max = 255),
                     rgb(204, 235, 197, max = 255),
                     rgb(255, 255, 179, max = 255),
                     rgb(255, 237, 111, max = 255)
      )
      plotdays <<- 2:5   #all days by default
      average <<- T
      avgColor <<- c(rgb(0, 1, 0))
      lineType <<- c("solid","dashed", "dashed", "solid", "solid", "solid")

      initFields(...)
    },  #end of initialize() method

    showData = function(act){

      source("R Functions/R_cumsums.plot.R")
      cumsums.plot(patient$data, title= title,
                     plotdays = plotdays, average = average,
```

```
                    act_range = act_range,
                    dayColor = dayColor,
                    avgColor = avgColor,
                    lineType = lineType,
                    legendPosition = legendPosition
      )

     }, # end showData() method

showAvg = function(){

data <- patient$data

act <- data[data$DayNumber %in% plotdays,]
#only plotted data
cumavgbase = rep(0, 1440)

for (i in 1:1440)
{
  cumavgbase[i] = mean(act$ActSum[(act$Time == i)])
}

lines(1:1440, cumavgbase, col = avgColor, lwd = 2)

### done plotting the average

axis(1, , at = seq(0,1440, 360),
     labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
cex.axis = 0.6)


axis(2, at = seq(0,500000, 100000),
labels = c("0","100","200","300", "400", "500"),
las=2, cex.axis = 0.6)
mtext("Activity Level (in thousands)", line = 2,
side = 2, cex = 0.6)

     }# end showAvg() method

) # end list of methods

)
```

B.8 SortedCumSumsPlot

```
SortedCumSumsPlot <- setRefClass("SortedCumSumsPlot",

  contains = "Graph",
  fields = list(

    plotdays = "numeric",     # days to plot
    average  = "logical",     # should we plot the average
    dayColor = "character",   # list of colors for the day
    avgColor = "character",   # color for the average plot
    lineType= "character",    # a list of different line types
    patient  = "ActData"      # object of class ActData to plot
    ),


  methods = list(

    initialize = function(...){
      domain <<- c(1,1440)
      act_range <<- c(0,4000)
      xlab <<- "Time"
      ylab <<- ""
      title <<- "Patient Sorted Cumulative Sums Data Plot"
      axes <<- F
      legendPosition <<- c("topright")
      dayColor <<- c(rgb(141, 211, 199, max = 255),
                     rgb(190, 186, 218, max = 255),
                     rgb(251, 128, 114, max = 255),
                     rgb(128, 177, 211, max = 255),
                     rgb(253, 180, 98, max = 255),
                     rgb(179, 222, 105, max = 255),
                     rgb(252, 205, 229, max = 255),
                     rgb(217, 217, 217, max = 255),
                     rgb(188, 128, 189, max = 255),
                     rgb(204, 235, 197, max = 255),
                     rgb(255, 255, 179, max = 255),
                     rgb(255, 237, 111, max = 255)
      )
      plotdays <<- 2:5   #all days by default
      average <<- T
      avgColor <<- c(rgb(0, 1, 0))
      lineType <<- c("solid","dashed", "dashed", "solid", "solid", "solid")

      initFields(...)
    },  #end of initialize() method

    showData = function(act){

      source("R Functions/R_sortedcumsums.plot.R")
      sortedcumsums.plot(patient$data, title= title,
```

```
                      plotdays = plotdays, average = average,
                      act_range = act_range,
                      dayColor = dayColor,
                      avgColor = avgColor,
                      lineType = lineType,
                      legendPosition = legendPosition
      )

     }, # end showData() method

showAvg = function(){

data <- patient$data

act <- data[data$DayNumber %in% plotdays,]
#only plotted data
cumavgbase = rep(0, 1440)

for (i in 1:1440)
{
  cumavgbase[i] = mean(act$ActSortSum[(act$Time == i)])
}

lines(1:1440, cumavgbase, col = avgColor, lwd = 2)

### done plotting the average

axis(1, , at = seq(0,1440, 360),
     labels = c("Midnight","6 AM","Noon","6 PM","Midnight"),
 cex.axis = 0.6)


axis(2, at = seq(0,500000, 100000),
labels = c("0","100","200","300", "400", "500"),
las=2, cex.axis = 0.6)
mtext("Activity Level (in thousands)",
line = 2, side = 2, cex = 0.6)

     }# end showAvg() method

) # end list of methods

)
```

B.9 DensityPlot

```
DensityPlot <- setRefClass("DensityPlot",

  contains = "Graph",
  fields = list(


    clinicalData = "data.frame",
#to hold clinical data for multivariate plots
    class = "numeric",
#classification level (0: gender, 1:depression)
    level = "numeric",
# 5 is for all depression levels, otherwise specify 0, 1, 2, 3, and 4
    # 3 is for all genders, 0:males, 1: females
    data_type = "numeric",
#check what type of data: raw:0, cumsum:1, sorted:2, sorted_cum_sum =3
    plotdays = "numeric",
# days to plot
    adjust   = "numeric",
    scale    = "numeric",
    gamma    = "numeric",
    colmin   = "character",
    colmax   = "character",
    patients  = "list"
    # list of objects to plot. These objects are of type ActData
    ),

  methods = list(

    initialize = function(...){
      domain <<- c(1,1440)
      act_range <<- c(0,4000)
      xlab <<- "Time"
      ylab <<- ""
      title <<- "Patients Data Density Plot"
      axes <<- F
      legendPosition <<- c("topright")
      class <<- 1 #depression level
      level <<- 5 #all levels by default
      data_type <<- 0 #raw data
      plotdays <<- 2:4   #by default
      adjust <<- 0.5
      scale <<- 1
      gamma <<- 1
      colmin <<- "#FFFFFF"
      colmax <<-"#000000"

      initFields(...)
    },  #end of initialize() method
```

```
    showData = function(){
      source("R Functions/R_density.plot.R")
      plot.density (patients, clinicalData, class ,
level , data_type, plotdays, act_range,
                title, xlab, ylab,  adjust , scale, gamma , colmin,
                colmax)

    } # end of showData method


) # end list of methods

)
```

APPENDIX C

WEB APPLICATION (HTML CODE)

C.1 Main Page

```
<% setContentType('text/html') %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Visualization of Actigraphy Data</title>
<link rel="shortcut icon" href="../A-icon.png" >

<script
src="https://ajax.googleapis.com/ajax/libs/prototype/1.7.0.0/prototype.js"
type="text/javascript">
</script>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.js"
type="text/javascript">
</script>
<script
src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.18/jquery-ui.js"
type="text/javascript">
</script>
<link
rel="stylesheet" type="text/css"
href="http://ajax.googleapis.com/ajax/libs/
jqueryui/1.8.18/themes/start/jquery-ui.css"
/>

<link
rel="stylesheet"
type="text/css"
href="../act.css"
/>

<script
src="../act.js" type="text/javascript">
</script>

<script type="text/javascript">
var $j = jQuery.noConflict();
```

```
//in  order not to have conflicts between jQuery and Prototype libraries
</script>
</head>

<body>

<h1>Visualization of Actigraphy Data</h1></br></br>

<div id="contentWrapper">
<div id="accordionContainer">

<h3>
<a href="#"><label id = "fileUploadHeader">File Upload</label></a>
</h3>
<div id="fileUpload">

<form enctype="multipart/form-data" method="POST" name = "dataForm">
<h4>Upload Actigraphy Data Files </h4>
<input type="file" name="datafile[]" multiple="multiple" id = "file"> </br>
<h4>Upload Patients' Clinical Data File</h4>
<input type="file" name="patientsData" id="patientsData"/>
<input type="submit" name="upload" value = "Upload"
id= "upload_button" disabled="disabled"
>

<input  type = "hidden" value="" id="inputField_dataFileName">
<input  type = "hidden" value="" id="inputField_tempDataFileName">
<input  type = "hidden" value="" id="inputField_phq9ScoresFileName">
<input  type = "hidden" value="" id="inputField_tempPhq9ScoresFileName">
<input  type = "hidden" value="" id="inputField_countFiles">
</form>
</div>

<h3><a href="#"><label id = "graphTypeHeader">Graph Type</label></a></h3>
<div id="graphType">
<h3>Select Graph Type</h3>
   <form method="POST" name="graphTypeForm">
<div id ="onePatientPlotTypeDiv">
<input type="radio" name="PlotType"
value="rawDataPlot"> Raw Data Plot<br>
<input type="radio" name="PlotType"
value="smoothDataPlot" > Smoothed Data Plot<br>
<input type="radio" name="PlotType"
value="velocityDataPlot" > Velocity Data Plot<br>
<input type="radio" name="PlotType"
value="accelerationDataPlot" > Acceleration Data Plot<br>
<input type="radio" name="PlotType"
value="cumsumsDataPlot" > Cumulative Sums Data Plot<br>
<input type="radio" name="PlotType"
value="sortedCumsumsDataPlot" >
Sorted Cumulative Sums Data Plot<br>
```

```
</div>

<div id="multiplePatientsPlotTypeDiv">
<input type="radio" name="PlotType"
value="densityPlot"> Density Data Plot<br>
<input type="radio" name="PlotType"
value="envelopPlot" > Envelop Data Plot<br>
<input type="radio" name="PlotType"
value="mvtsPlot" > MVTS Data Plot<br>
</div>
</form>
</div>
<h3><a href="#">Graph Parameters</a></h3>
<div id="paramInput">
<h3>Enter Graph Parameters:</h3>
<form name = "settingForm" method="POST">
<dl>
<dt>
<label for="title">Title</label>
</dt>
<dd><input type="text" name="title"
class="required" id="title" /></dd>
</dl>
<fieldset id="range">
<legend>Actigraphy Range</legend>
<dl>
<dt>
<label for="from">From</label>
</dt>
<dd><input type="text" name="from"
class="required" id="from" /></dd>
</dl>
<dl>
<dt>
<label for="to">To</label>
</dt>
<dd><input type="text" name="to"
 class="required" id="to" /></dd>
</dl>
</fieldset>
<div id="submit_buttons">
<button type="reset">Reset</button>
<input type="button"  onclick = "Plot()" value="Plot"></input>
<img style="display: none;"
 src="../progressBar.gif"
 alt="Plotting..."
id="plotProgressBar"/>
</div>
</form>
</div>
</div>
```

```
<div id="imgContainer">
<center>
<img style="display: none;"
 src="../progressBar.gif"
 alt="Plotting..." id="plotProgressBar"/>
</center>
</div>
</div>
</body>
</html>

<script type = "text/javascript">

//R code to upload the files to the server
<%
dataFileName <- NULL
tempDataFileName <- NULL

phq9ScoresFileName <- NULL
tempPhq9ScoresFileName <- NULL


#Upload AWC files
for(i in 1:(length(FILES)-1)){

dataFileName[i] = FILES[i]$datafile$name
#get all of the file names
tempDataFileName[i] = FILES[i]$datafile$tmp_name
#when data are uploaded, there is a temp file
#created for each. get the names of each.

destination <- file.path('/Users/Abbass/uploaded_files',dataFileName[i])
#where the data is uploaded
file.copy(tempDataFileName[i],destination,overwrite=TRUE)
#create a copy of the temp files, and place it in the
#destination defined above.

}


#Upload PHQ9 Scores file

phq9ScoresFileName = FILES$patientsData$name
#get all of the file names
tempPhq9ScoresFileName = FILES$patientsData$tmp_name
#when data are uploaded, there is a temp file created
#for each. get the names of each.

destination <- file.path('/Users/Abbass/uploaded_files',phq9ScoresFileName)
#where the data is uploaded
```

```
file.copy(tempPhq9ScoresFileName, destination,overwrite=TRUE)
#create a copy of the temp files, and place it in the destination defined above.

%>


//javascript code to save the name, location and number of files
//uploaded for use in actPlot-onePatient.rhtml and actPlot-multiplePatients.rhtml

var JSON_dataFileName = <%= cat(toJSON(dataFileName)) %>;
var JSON_tempDataFileName = <%= cat(toJSON(tempDataFileName)) %>;
var JSON_phq9ScoresFileName = <%= cat(toJSON(phq9ScoresFileName)) %>;
var JSON_tempPhq9ScoresFileName =  <%= cat(toJSON(tempPhq9ScoresFileName)) %>;
var JSON_countFiles = <%= length(FILES) %>;

var JSON_dataFileName_string = JSON.stringify(
JSON_dataFileName);
var JSON_tempDataFileName_string = JSON.stringify(
JSON_tempDataFileName);
var JSON_phq9ScoresFileName_string = JSON.stringify(
JSON_phq9ScoresFileName);
var JSON_tempPhq9ScoresFileName_string = JSON.stringify(
JSON_tempPhq9ScoresFileName);
var JSON_countFiles_string = JSON.stringify(
JSON_countFiles);

document.getElementById('inputField_dataFileName').setAttribute(
'value', JSON_dataFileName_string);
document.getElementById('inputField_tempDataFileName').setAttribute(
'value', JSON_tempDataFileName_string);
document.getElementById('inputField_phq9ScoresFileName').setAttribute(
'value', JSON_phq9ScoresFileName_string);
document.getElementById('inputField_tempPhq9ScoresFileName').setAttribute(
'value', JSON_tempPhq9ScoresFileName_string);
document.getElementById('inputField_countFiles').setAttribute(
'value', JSON_countFiles_string);

//check if the user has uploaded files.. if so,
//then open the second window in the accordion


$j('#file').change(function (){
   $j('#upload_button').removeAttr('disabled');
  });

if(JSON_countFiles > 2){

Element.hide('onePatientPlotTypeDiv');
$j( "#accordionContainer" ).accordion({ active: 1 });
var counter = JSON_countFiles -1;
    $j("#fileUploadHeader").html('File Upload' + ' (' + counter + ')');
```

```
} else if(JSON_countFiles == 2){

Element.hide('multiplePatientsPlotTypeDiv');
$j( "#accordionContainer" ).accordion({ active: 1 });
var counter = JSON_countFiles - 1;
$j("#fileUploadHeader").html('File Upload' + ' (' + counter + ')');

}

$j('input:radio[name=PlotType]').click(function(){
$j( "#accordionContainer" ).accordion({ active: 2 });

var graphSelected = $j('input[@name=PlotType]:checked').val();

if(graphSelected =="rawDataPlot") {
graphSelected = "Raw Data Plot";}
else if (graphSelected =="smoothDataPlot") {
graphSelected ="Smoothed Data Plot";}
else if (graphSelected =="velocityDataPlot") {
graphSelected ="Velocity Data Plot";}
else if (graphSelected =="accelerationDataPlot") {
graphSelected ="Acceleration Data Plot";}
else if (graphSelected =="cumsumsDataPlot") {
graphSelected ="Cumulative Sums Data Plot";}
else if (graphSelected =="sortedCumsumsDataPlot") {
graphSelected ="Sorted Cumulative Sums Data Plot";}
else if (graphSelected =="densityPlot") {
graphSelected ="Density Data Plot";}
else if (graphSelected =="envelopPlot") {
graphSelected ="Envelop Data Plot";}
else if (graphSelected =="mvtsPlot") {
graphSelected ="MVTS Data Plot";}


$j("#graphTypeHeader").html('Graph Type' + ' (' + graphSelected + ')');

});

</script>

<script type = "text/javascript">
      $j("#accordionContainer").accordion();  //define the accordion
</script>
```

C.2 Single Patient Page

```
<%

### fileUpload Selection
dataFileName = fromJSON(POST$dataFileName)
tempDataFileName = fromJSON(POST$tempDataFileName)
destination <- file.path('/Users/Abbass/uploaded_files',dataFileName[1])

### graphType Selection
graphType <- POST$graphType
### paramInput Selection
title <- ifelse(is.null(POST$title),"NoTitle" ,POST$title)
from <- as.numeric(POST$from)
to <- as.numeric(POST$to)


### load ActiVis package

library(ActiVis)

###create a new actigraphy data object for Karli
karli <- ActData$new() #inialize karli's actigraphy object
karli$fileName = destination #specify the awc file for Karli
karli$read() #read the patient data


if(graphType == "rawDataPlot"){
######################
## Raw data plot
######################

karli_raw <- RawDataPlot$new()

karli_raw$legendPosition = "topright"
karli_raw$act_range = c(from,to)
karli_raw$title = title
karli_raw$patient <- karli
karli_raw$avgPlot = FALSE
karli_raw$plotdays <- 2:4

PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
karli_raw$setup()
karli_raw$showData()
dev.off()

}else if (graphType == "smoothDataPlot"){
######################
```

```
## Smoothed Data plot
########################

karli_smooth <- SmoothedDataPlot$new()

karli_smooth$act_range = c(from,to) #0,600
karli_smooth$title = title
karli_smooth$plotdays <- 2:4
karli_smooth$patient <- karli


PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
karli_smooth$setup()
karli_smooth$showData()
dev.off()

} else if (graphType == "velocityDataPlot"){

########################
## Velocity plot
########################

karli_velocity <- VelocityPlot$new()

karli_velocity$act_range = c(from,to) #-10, 10
karli_velocity$title = title
karli_velocity$patient <- karli


PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
karli_velocity$setup()
karli_velocity$showData()
dev.off()

} else if (graphType == "accelerationDataPlot"){

########################
## Acceleration plot
########################

karli_accleration <- AccelerationPlot$new()

karli_accleration$act_range = c(from,to) #-0.5, 0.5
karli_accleration$title = title
karli_accleration$plotdays <- 2:3
```

```
karli_accleration$patient <- karli

PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
karli_accleration$setup()
karli_accleration$showData()
dev.off()

} else if (graphType == "cumsumsDataPlot"){
########################
## Cum Sums plot
########################

karli_cumsum <- CumSumsPlot$new()

karli_cumsum$act_range = c(from,to) #0, 5000000
karli_cumsum$title = title
karli_cumsum$legendPosition = "topleft"
karli_cumsum$patient <- karli


PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
karli_cumsum$setup()
karli_cumsum$showData()
dev.off()

} else if (graphType == "sortedCumsumsDataPlot"){
########################
## Sorted Cum Sums plot
########################

karli_sorted_cumsum<- SortedCumSumsPlot$new()

#setup your raw data plor
karli_sorted_cumsum$act_range = c(from,to) #0, 5000000
karli_sorted_cumsum$title = title
karli_sorted_cumsum$legendPosition = "topleft"
karli_sorted_cumsum$patient <- karli


PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
karli_sorted_cumsum$setup()
karli_sorted_cumsum$showData()
```

```
dev.off()

}

%>
```

C.3 Multiple Patient Page

```
<%

### fileUpload Selection
dataFileName = fromJSON(POST$dataFileName)
tempDataFileName = fromJSON(POST$tempDataFileName)
phq9ScoresFileName = fromJSON(POST$phq9ScoresFileName)
tempPhq9ScoresFileName = fromJSON (POST$tempPhq9ScoresFileName)



### graphType Selection
graphType <- POST$graphType
### paramInput Selection
title <- ifelse(is.null(POST$title),"NoTitle" ,POST$title)
#from <- as.numeric(POST$from)
#to <- as.numeric(POST$to)

# Load ActiVis package
library(AvtiVis)

### read all AWC files, and store it in allPatients
files <- file.path('/Users/Abbass/uploaded_files',dataFileName)
allPatients <- NULL
for (file in files){
  print(file) #to show the progress
  Patient<- ActData$new()
  Patient$fileName <- file
  Patient$read() #read the patient data
  Patient$agg_epoch <- 10
  Patient$sum()
  allPatients <- c(allPatients, Patient)
}



if(graphType == "densityPlot"){
#######################
## Density plot
#######################

allPatients_density_plot <- DensityPlot$new()

allPatients_density_plot$act_range <- c(0, 600000)
allPatients_density_plot$title = title
allPatients_density_plot$ylab <- "Activity Level"
allPatients_density_plot$data_type <- 1 # ActSums and not raw data
allPatients_density_plot$patients <- allPatients
```

```
PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
allPatients_density_plot$setup()
allPatients_density_plot$showData()
dev.off()

}else if(graphType == "envelopPlot"){
########################
## Envelope plot
########################

### read clinical data

clinicalFile <- file.path('/Users/Abbass/uploaded_files',phq9ScoresFileName)
clinicalData <- read.csv(clinicalFile)
clinicalData <- as.data.frame(
  cbind( ID   = clinicalData$ID,
         Level  = clinicalData$level.new,
         gender = clinicalData$Gender
   )
)

allPatients_envelope_plot <- EnvelopePlot$new()


allPatients_envelope_plot$lower_bound <- 0.4
allPatients_envelope_plot$upper_bound <- 0.6
allPatients_envelope_plot$data_type <- 1 # ActSums and not raw data
allPatients_envelope_plot$title <- title
allPatients_envelope_plot$patients <- allPatients
allPatients_envelope_plot$clinicalData <- clinicalData
allPatients_envelope_plot$envelopeData()

allPatients_envelope_plot$act_range <- c(0,500000)


PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
allPatients_envelope_plot$setup()
#show your data on the density plot
allPatients_envelope_plot$showData()
# plot(1:10, main = clinicalData$Level[3])
dev.off()

}else if(graphType == "mvtsPlot"){
########################
```

```
## MVTS plot
######################
### read clinical data
clinicalFile <- file.path('/Users/Abbass/uploaded_files',phq9ScoresFileName)
clinicalData <- read.csv(clinicalFile)
clinicalData <- as.data.frame(
cbind( ID   = clinicalData$ID,
     Level  = clinicalData$level.new,
     gender = clinicalData$Gender
)
)

allPatients_mvts_plot <- MvtsPlot$new()

allPatients_mvts_plot$norm <- "global"
allPatients_mvts_plot$plotdays <- 2:6
allPatients_mvts_plot$class <- 0 #classification level (0: gender, 1:depression)
allPatients_mvts_plot$data_type <- 0
allPatients_mvts_plot$title = "MVTS Plot"

allPatients_mvts_plot$patients <- allPatients
allPatients_mvts_plot$clinicalData <- clinicalData

PLOTDIR='/Users/Abbass/Sites/images'
plotname <- paste(title,'.png',sep='')
filename <- file.path(PLOTDIR,plotname)
png(filename, width = 600, height = 350)
allPatients_mvts_plot$setup()
#show your data on the density plot
allPatients_mvts_plot$showData()
dev.off()

}

%>
```

APPENDIX D

WEB APPLICATION (JAVASCRIPT CODE)

D.1 Javascript Controller

```
function Plot(){

//fileUpload Selection
var dataFileName =
document.getElementById('inputField_dataFileName').value;
var tempDataFileName =
document.getElementById('inputField_tempDataFileName').value;
var phq9ScoresFileName =
document.getElementById('inputField_phq9ScoresFileName').value;
var tempPhq9ScoresFileName =
document.getElementById('inputField_tempPhq9ScoresFileName').value;
var countFiles =
document.getElementById('inputField_countFiles').value;

//graphType Selection
var graphType = $j('input[@name=PlotType]:checked').val();

//paramInput Selection
var title = document.settingForm.title.value;
var from = document.settingForm.from.value;
var to = document.settingForm.to.value;

if (countFiles == 2) {    // The case of one patient

Element.show('plotProgressBar');
new  Ajax.Updater( 'plot', 'actPlot-onePatient.rhtml',
{
'method': 'post',
'parameters': {
   'dataFileName':dataFileName,
   'tempDataFileName':tempDataFileName,
   'graphType':graphType,
   'title': title,
   'from': from,
   'to': to},
'onSuccess': function(r){

  Element.hide('plotProgressBar')
          $j("#imgContainer").html(
```

```
'<center>
<a href="../images/'+title+'.png">
<img src="../images/'+title+'.png"/>
</a>
</center>');
      }
}
);

} else if(countFiles > 2 ){  // The case of multiple patients
Element.show('plotProgressBar');
new  Ajax.Updater( 'plot', 'actPlot-multiplePatients.rhtml',
{
'method': 'post',
'parameters': {
   'dataFileName':dataFileName,
   'tempDataFileName':tempDataFileName,
   'phq9ScoresFileName':phq9ScoresFileName,
   'tempPhq9ScoresFileName':tempPhq9ScoresFileName,
   'graphType': graphType,
   'title': title,
   //'from': from,
   //'to': to
},
'onSuccess': function(r){

 Element.hide('plotProgressBar')
          $j("#imgContainer").html(
'<center>
<img src="../images/'+title+'.png">
</center>');
      }
}
);
}
}
```

CURRICULUM VITAE

Abbass Sharif
October, 2012

Contact

7 Aggie Village, apt G
Logan, Utah, 84321
United States of America

Phone: (435) 757-3431
Email: abbass.sharif@gmail.com
Website: http://www.math.usu.edu/∼abbass.s

Education

Utah State University, Logan, UT, USA

Ph.D., Statistics — Dec. 2012

MS., Instructional Technology — Dec. 2012

Lebanese American University, Beirut, Lebanon

MS., Computer Science — May 2006

BS., Computer Science, Mathematics Minor — May 2004

Research Interests

Statistical Data Visualization
Statistical Computing
Web 2.0 Tools in Statistics Education
Animated Pedagogical Agents

Relevant Experience

Research

- Conducted research on statistical computing, data visualization, and technology in mathematics and statistics education
- Developed and implemented an R package for the visualization of functional data
- Used regression tools, experimental design, multivariate statistics tools, categorical data analysis tools in various projects
- (Co-) Authored two journal articles, two proceedings articles, and one magazine article

Teaching

- Taught, developed curriculum and instructional material, created and graded exams for the following classes: Introduction to Statistics (2 times, 150 students total), Business Statistics (1 time, 41 students total), Calculus I (3 times, 82 students total), College Algebra (3 times, 164 students total), Basic Mathematics (3 times 125 students total), and Computer Applications (2 times, 134 students total)

- Lead recitations and graded exams for the following classes: Statistics for Scientists (1 time, 115 students total), Experimental Design (1 time, 52 students)

- Graded assignments and lectured frequently in Advanced Mathematical Statistics class (1 time, 18 students)

Computer Programming

- Used the R programming language extensively throughout my graduate education and mainly for dissertation research
- Bridged the R programming language with C, Ruby, HTML, and Java script
- Used C, LaTeX, SAS, SPSS, Mathematica, Flash, Ruby on Rails, ArcGIS, OpenGl,
  VB .NET, and Assembly language in various projects throughout my undergraduate and graduate school years

Consulting

- Zeina Youssef, Department of Psychology, Iowa State University (2010). Assisted with quantitative analysis on *A Comparison of Forgiveness across College Students in Lebanon and the United States*
- Nabilah Haraty, Communications Arts Department, Lebanese American University (2008). Assisted with quantitative analysis on *Compliments: Receiving and Giving*

Work Experience

Utah State University, Logan, UT, USA

Department of Mathematics and Statistics

Lecturer	2011- Current
Research Assistant	2010
Teaching Assistant	2008-2010

Department of Instructional Technology and Learning Sciences

Software Engineer	2007-2008

Lebanese American University, Beirut, Lebanon

Department of Computer Science and Mathematics

Lecturer	2008-2010 (Su.)
Research Assistant	2004-2006

Publications

Journals

- Sharif, A. and Symanzik, J. (2012). Actigraphy: An Object Oriented R Package for the Visualization of Functional Actigraphy Data, *Journal of Statistical Software.* (In preparation).
- Sharif, A. and Symanzik, J. (2012). Visual Data Mining Techniques for Functional Actigraphy Data, *Statistics in Medicine.* (Submitted).
- Ding, J., Symanzik, J., Sharif, A., Wang, J., Duntley, S., and Shannon, W. D. (2011). Functional representation of actigraphy data, *CHANCE* 24(3):30–36.
- Sharif, A. and Symanzik, J. (2011). The Relationship between blogging and students' achievements in mathematics courses, *International Journal of Science and Mathematics Education* (Submitted).
- Kim, Y., Xu, B., and Sharif, A. (2008) Pedagogical agents as social models in an online learning environment MathGirls, *International Transactions on Systems Science and Applications*, 4(2), 99–106.
- Sharif, A. and Haraty, R. A. (2008) The Relationship between using of an intelligent tutoring system and class achievement in a basic mathematics course, *iJET*, 3, 20–23.

Conference Proceedings

- Sharif, A. and Symanzik, J. (2012). Graphical Representation of Clustered Functional Actigraphy Data, *Joint Statistical Meeting*, San Diego, CL. (To appear).
- Sharif, A., Symanzik, J., and Shannon, W. D. (2010). An object-oriented approach in R for the visualization of functional actigraphy data, *Computing Science and Statistics*, Seattle, WA. (To appear)
- Caswell, T. and Sharif, A. (2007). eduCommons in any language. *Open Education Conference 2007: Localizing and Learning*, Logan, UT.
- Sharif, A. and Habre, S. (2006). Students Dependence on Calculators in a Freshman Level Basic Mathematics Course. *International conference on the teaching of Mathematics at the undergraduate level*, Istanbul, Turkey.

Honors/Awards

Citation of Honor 2012

Utah State Legislature

Robins Awards Finalist - Man of the Year 2011

Utah State University

Graduate Teacher of the Year 2008

Department of Mathematics and Statistics, Utah State University

Dean's List Award for Outstanding Scholastic Achievements 2008-Present

Utah State University

President's Award Finalist 2004

Lebanese American University

Grants

College of Science Travel Award ($**400**) 2010

Utah State University

Department of Mathematics and Statistics Travel Award ($**300**) 2010

Utah State University

CREATE Travel Award ($**800**) 2008

Utah State University

Professional Memberships

American Statistical Association (ASA) 2008- Present

ASA Section on Statistical Graphics

ASA Section on Statistical Computing

Association of Computing Machinery (ACM) 2003- 2007

Media Interviews

CNN (Anderson Cooper 360° and Piers Morgan Tonight), ABC (The Ellen DeGeneres Show), People magazine, and the Associated Press	2011
Future TV	2007