Behav Res (2013) 45:125–131
DOI 10.3758/s13428-012-0233-x

PhysioScripts: An extensible, open source platform for the processing of physiological data

Israel C. Christie · Peter J. Gianaros

Published online: 6 September 2012

Abstract A commonality across research involving physiological measures is the need to process large amounts of data. Such data processing typically involves the use of software tools to achieve several methodological steps, including identifying and correcting artifacts and defining epochs of time for the reduction and analysis of one or more physiological measures. This article describes a new tool to aid in the processing of physiological data: PhysioScripts. Key elements of PhysioScripts include a graphical interface to view and edit the results of processing steps, as well as a flexible framework to automate the creation of uniform or variable length epochs. The software comprises freely available scripts implemented in the R computing environment. Consequently, PhysioScripts can be readily modified to process other data types through the addition of new subroutines that can be plugged into the existing data processing framework. For illustrative purposes, we describe the steps involved in two data processing examples: (1) heart rate variability from the electrocardiogram and (2) respiratory rate derived from a chest strain gauge. The software, accompanying documentation, and an example data set are available online at israelchristie.com/software.

Keywords Software · R · Open source · Physiological data processing

Numerous disciplines within the psychological sciences and allied fields rely, in large part, upon the processing and analysis of continuous physiological time series as a major source of data. Whether the goal is the description of heart rate reactivity during affective picture processing (Bradley, Codispoti, Cuthbert, & Lang, 2001), the derivation of baroreflex function during stress and the underlying brain systems involved (Gianaros, Onyewuenyi, Sheu, Christie, & Critchley, 2012), or the quantification of heart rate variability as an index of cardiac autonomic control during hot flashes in midlife women (Thurston, Christie, & Matthews, 2012), a commonality across such research goals is the need to ensure that physiological time series are adequately cleaned (i.e., artifacted) if valid inferences are to be made (see, e.g., Berntson & Stowell, 1998). In addition, a typical data processing step is to break the continuous time series into time periods or epochs based on either fixed time points or events of interest. Both of these tasks, artifacting and epoching, can be laborious, particularly when recordings are of longer duration and/or the probability of artifacts is high (e.g., extended ambulatory recordings in human or nonhuman animal studies). Moreover, the analysis of physiological time series is nearly invariably performed with the assistance of software tools, either proprietary packages bundled with the data recording equipment or third-party applications.

The goal of this article is to describe a collection of freely available scripts[1] for the processing and editing physiological data, called PhysioScripts, which consist primarily of (1) a graphical interface to facilitate the viewing and artifacting of data; (2) a highly flexible framework to automate the creation of uniform or variable-length epochs based on information provided by the user in simple text files called *epoch lists*; and (3) modules that handle specific types of physiological signals that perform either preliminary

I. C. Christie (✉)
Department of Psychiatry, University of Pittsburgh,
201 North Craig Street. Rm. 201,
Pittsburgh, PA 15213, USA
e-mail: ichristie@gmail.com

P. J. Gianaros
Department of Psychology, University of Pittsburgh,
201 North Craig Street. Rm. 201,
Pittsburgh, PA 15213, USA

[1] Text-based files containing R code are typically referred to as scripts. In this case, the scripts define the functions that perform data processing or editing tasks. Thus, the term "scripts" and "functions" are used interchangeably.

preprocessing steps [e.g., the conversion of electrocardiogram to interbeat interval (IBI) series] or the derivation of summary measures within epochs [e.g., the estimation of heart rate variability (HRV)]. The software is designed to be easily extended to handle other data types through the addition of new processing subroutines that can be plugged into the existing framework. Two such sets of subroutines, one for the analysis of heart rate and HRV and the other for the analysis of respiration, will be described here for illustrative purposes. The software, accompanying documentation, and an example data set are available online (israelchristie.com/software).

A brief note regarding what distinguishes this software from other options is warranted. While a number of similar software packages and routines are available, encompassing both single-purpose applications and more extensive software suites, the PhysioScripts package is arguably unique in that it is written using, and runs within, the open source and freely available R environment, as opposed to a commercial environments like MATLAB (The MathWorks Inc., Natick, MA) or Labview (National Instruments, Austin, TX). This feature alone represents a saving in costs stemming from licensing fees. The readily accessible R syntax, as well as its open source nature, should lend itself to the rapid development of additional modules with which to expand the PhysioScripts collection. Finally, for those who choose to also use R for their data analysis needs, it is difficult to overstate the convenience that comes with using the same statistical package for both data processing and analysis.

About R

The R computing environment (Ihaka & Gentleman, 1996; R Development Core Team, 2010) is a free, open source, cooperatively developed implementation of the S statistical programming language developed at Bell Laboratories (formerly AT&T, now Lucent Technologies; Chambers, 1998). The term "environment" is intentionally used when discussing R because it connotes a fully planned and coherent system, as opposed to a collection of specific and inflexible tools characteristic of other data analysis software. Hence, R is designed around a true computer language and affords users a high degree of extensibility through the creation of user-defined functions and the installation of contributed packages. The base R installation comes equipped with capabilities roughly comparable to, for example, a basic installation of SPSS or SAS, but it can be augmented by the addition of contributed packages, which currently number nearly 3,000. Since its introduction in the mid-1990s, R has rapidly become one of the most widely used platforms for statistical computing and is considered to have broader coverage of statistical methods than any other statistical software (Fox, 2006). Because of its open source nature, usage statistics are difficult to obtain. But, recent estimates have placed user numbers in the range of 1–2 million (Vance, 2009), across both business and academic domains. Another benefit is the fact that R is available for a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), as well as for Windows and MacOS. At present, we are unaware of any other integrative set of scripts for R that have been developed for the purposes of physiological data reduction and analysis, as described in this report.

About the PhysioScripts functions

Installation Because PhysioScripts is not presently distributed as a formal package, there is no need to "install it," per se. Rather, the PhysioScripts functions are distributed within a single binary RData file (e.g., "PhysioScripts.####.RData," where #### is a unique version identifier). PhysioScripts is started by simply double-clicking the RData file, which initiates a new R session and loads the functions into memory. Most functions contain embedded help and usage instructions and can be viewed by simply typing the name of the function into the R console, without parentheses, and pressing enter. Also, as a general rule, data paths to data files should not contain spaces.

Interactive versus batch processing modes PhysioScripts functions are designed to operate in one of two processing modes: interactive or batch. Interactive mode, the default, involves manual selection of data files via a graphical interface and should be immediately familiar to any user. Batch mode provides a more efficient means of handling a large number of files by performing the processing task on all files contained in a given directory, which is identified either through the graphical interface or by specification of a path during the function call. In addition, the default behavior for PhysioScripts functions (which can be overridden in the function call) is not to overwrite existing files or processing steps. This not only protects prior work, but also makes the batch processing mode more useful, in that a single function call can perform a given processing step on all unprocessed files within a directory.

PhysioScripts data and resource files The default data format used by PhysioScripts is comma-delimited text file, with variable names (e.g., "time," "ecg") in the first row. By default, files are compressed using the gzip format, although uncompressed files can also be employed, and may be more useful if input data files are created by hand. Most modern physiological recording software will allow users to export raw data as ASCII text. The preparation of data files to use with the PhysioScripts functions should be a trivial, although perhaps inefficient, task using a text editor.[2] An example

[2] Microsoft Word is not a text editor. Notepad on a Windows machine or TextEdit on a Mac will serve this purpose. A Web search for "text editor" will reveal numerous free options for editing text files on any computing platform.

function[3] is included that converts exported ASCII text to the PhysioScripts file format and can be modified to meet the data formatting characteristics of specific recording equipment. Presently, all columns in the input data files are presumed to represent independent channels or physiological signals, each sampled at identical (uniform) sampling rates. A time column should be included in all data files and should be expressed in seconds, with the time point "zero" referenced to either the beginning of the file or midnight on the day of recording onset, the latter being used when recordings of more than 12 h are being processed.

Aside from the initial input data file, two other files should accompany each data file: an info file and an epoch list. Both files are similarly formatted as comma-delimited text with column names in the first row, although both of these files should be saved as simple text files[4] (i.e., with the ".txt" file extension). The info files contain file-specific information and should minimally include "origin," the date on which the recording was started for a given data file, in the "YYYY-MM-DD" format (for studies in which the day of recording is unimportant, origin can simply be entered as "NA"), and "fs," the sampling rate, in hertz, at which the data file was recorded. Info files can be created by hand using a text editor. The contents of an example info file are printed below:

```
origin,fs
NA,500
```

The epoch list contains information used by some PhysioScripts functions, specifically those performing summary processing steps on discrete epochs derived from an existing data file. Epochs are specified in blocks of related, uniform-length segments of data, with each line corresponding to a user-specified number of epochs. All epoch lists consist of five columns:

(1) *name*, the label used to identify epochs of a given block in the resulting output file;
(2) *time*, the time of onset for the first epoch in a given block, which can be specified as seconds, clock time using the format "HH:MM:SS," or date and time using the format "YYYY-MM-DD HH:MM:SS";
(3) *length*, the duration, specified in either seconds or the "HH:MM:SS" format, of all epochs in a given block; and
(4) *before* and *after*, which specify the number of uniform-sized epochs to be created prior to and after the initial epoch created by the time and length variables, respectively.

Hence, the time and length variables specify the onset and duration of the initial epoch (identified in the output data as epoch zero), and the before and after variables specify the number of epochs preceding and following the initial epoch. If only one epoch is desired, then both before and after should be set to zero. Each line in an epoch list specifies a block of non-overlapping epochs, though blocks of epochs are handled independently and are allowed to overlap (i.e., a given period of data may be included in more than one line of the epoch list). For example, an experimenter could be interested in constructing epochs from data recorded during a laboratory protocol consisting of a 10-min baseline beginning at 10:00 into the recording, a 5-min stressor task beginning at 20:30, and a 10-min recovery beginning at 25:30. If the experimenter desired 1-min epochs for both the baseline and stressor conditions, and 5-min epochs for the recovery period, the following epoch list would specify this epoching:

```
name,     time,  length,   before, after
base,     10:00, 00:01:00, 0,      9
stress,   20:30, 00:01:00, 0,      4
recovery, 25:30, 00:05:00, 0,      1
```

The manner in which PhysioScripts will interpret a given epoch list can be double-checked using the function call

```
import.epoch.list(print.epochs = TRUE)
```

which prompts the user to select an epoch list and, for the example epoch list above, prints out the following in the console:

```
          name    start     stop duration
1       base.0 00:10:00 00:11:00       60
2       base.1 00:11:00 00:12:00       60
3       base.2 00:12:00 00:13:00       60
4       base.3 00:13:00 00:14:00       60
5       base.4 00:14:00 00:15:00       60
6       base.5 00:15:00 00:16:00       60
7       base.6 00:16:00 00:17:00       60
8       base.7 00:17:00 00:18:00       60
9       base.8 00:18:00 00:19:00       60
10      base.9 00:19:00 00:20:00       60
11    stress.0 00:20:30 00:21:30       60
12    stress.1 00:21:30 00:22:30       60
13    stress.2 00:22:30 00:23:30       60
14    stress.3 00:23:30 00:24:30       60
15    stress.4 00:24:30 00:25:30       60
16 recovery.0 00:25:30 00:30:30      300
17 recovery.1 00:30:30 00:35:30      300
```

[3] The function, vernier.to.gz(), converts data files recorded using ASCII data files exported from Vernier physiological recording software (Vernier Software & Technology, Beaverton, OR) to the default PhysioScripts file format.

[4] Although comma-delimited text can be read into nearly any spreadsheet software, the automatic formatting performed by most spreadsheet software (e.g., Microsoft Excel) can be problematic for even advanced users. Creating or editing such text files is most easily done in a text editor, hence the use of the ".txt" file extension.

A final point that should be addressed in reference to data and resource files is the adherence to a uniform naming convention. All files read into or output by the PhysioScripts functions should conform to the three-field format "###.type.ext," where "###" is a unique file identifier. This identifier is typically a subject number and can include alphanumeric characters, as well as underscores and dashes, but should not include periods or commas. The second field, "type", identifies the contents of the file with "phys" used for (possibly multichannel) raw input data files. The output data files created by some of the processing steps will create file names with the appropriate type identifiers (e.g., "ibi" for IBI data files, "hrv" for HRV results files, or "resp" for respiration results files). The third field, "ext", identifies the format of the file and should be "gz" for compressed data files, "csv" for uncompressed data files, and "txt" for info files and epoch lists.

PhysioScripts data viewer The data viewer provides an efficient means by which to view the raw physiological waveforms and review and edit the results of otherwise automated data processing functions. At present, the data viewer plots single waveforms as a function of time and displays annotated data points reflecting events or characteristics of the data at hand. As a general rule, the raw data are never edited within the PhysioScripts system; rather, data files are annotated, and the file annotations are edited using the data viewer. The data viewer consists of a plot window (e.g., see Figs. 1 and 2) and a controller window (see Fig. 3). The slice of data displayed in the plot window is determined by two values, both of which can be changed in the control window. First, the window edge corresponds to the time, in seconds, of the leftmost edge of the data slice displayed in the plot window. Second, the window width specifies the length, also in seconds, of the data slice displayed in the plot window. Highlighted data points, representing annotations from prior processing steps, are displayed in a different color and can be manually modified using the Add and Remove buttons from the control window (a specific example is presented in the following section). Identification of data point(s) involves two clicks describing the opposing corners of a square selection area. Depending on the type of signal being viewed, the maximum, the minimum, or all points within the selection area are selected.

Example data processing: Heart rate variability

Heart rate variability, particularly the portion of variability linked to respiration (respiratory sinus arrhythmia; RSA), has become a widely used index of autonomic control of the heart and has been related to both physical health (Thayer & Lane, 2007) and a range of psychological phenomena, ranging from attentional capacity (Porges, 1992) and emotion regulation (Calkins & Johnson, 1998) to depression (Rottenberg, 2007) and anxiety (Friedman, 2007). The highly readable review by Allen and colleagues (Allen, Chambers, & Towers, 2007) serves as an excellent introduction to the theory- and measurement-related issues surrounding HRV and also describes, in greater detail than is suitable for presentation here, the band-limited variance method employed by PhysioScripts to obtain HRV estimates.

The following example describes the processing steps in a typical study involving heart rate variability, from the raw electrocardiogram (ECG) data file to the finished HRV data file. All function calls are initiated using the interactive mode (i.e., input data files are selected using a graphical interface), and default arguments for functions are not printed, though they are readily available in both accompanying documents and source code.

QRS detection The ECG is one of the most ubiquitous biomedical signals in psychophysiological research and serves as the most accurate measure of chronotropic cardiac function. As such, it serves as a basis for many studies involving heart rate and nearly all investigations of HRV. As a first processing step, the ECG is passed through a detection algorithm identifying QRS waves, the electrical signature of ventricular depolarization and the fiducial point for beat detection. The QRS detection algorithm employed by PhysioScripts uses both the amplitude of the digitally filtered ECG waveform and its first derivative, and is based on filtering and detection methods shown to be resistant to sources of noise typically encountered in ECG recordings (Friesen et al., 1990).

To begin processing ECG data, the function call below initiates data file selection, import, and QRS detection and saves the results as a column in the phys data file indicating the points identified as R-spikes. (The default R prompt ">" is shown to indicate the beginning of a new line and should not be copied.)

```
> process.ecg()
```

To review the output of the QRS detection, the following function call initiates the data viewer displaying the ECG in the plot window (see Fig. 1). In this case, highlighted data points indicate detected R-spikes. Errors in QRS detection can be manually corrected and saved to the data file.

```
> review.ecg()
```

IBI extraction and artifacting Interbeat intervals (IBIs), or the time in milliseconds between successive R-spikes, are then derived from the annotated phys file by the first function call below and saved in an appropriately named ibi.gz data file in the same directory as the input file. The second function call below initiates the artifacting algorithm and specifies a number of criteria, including the percent or

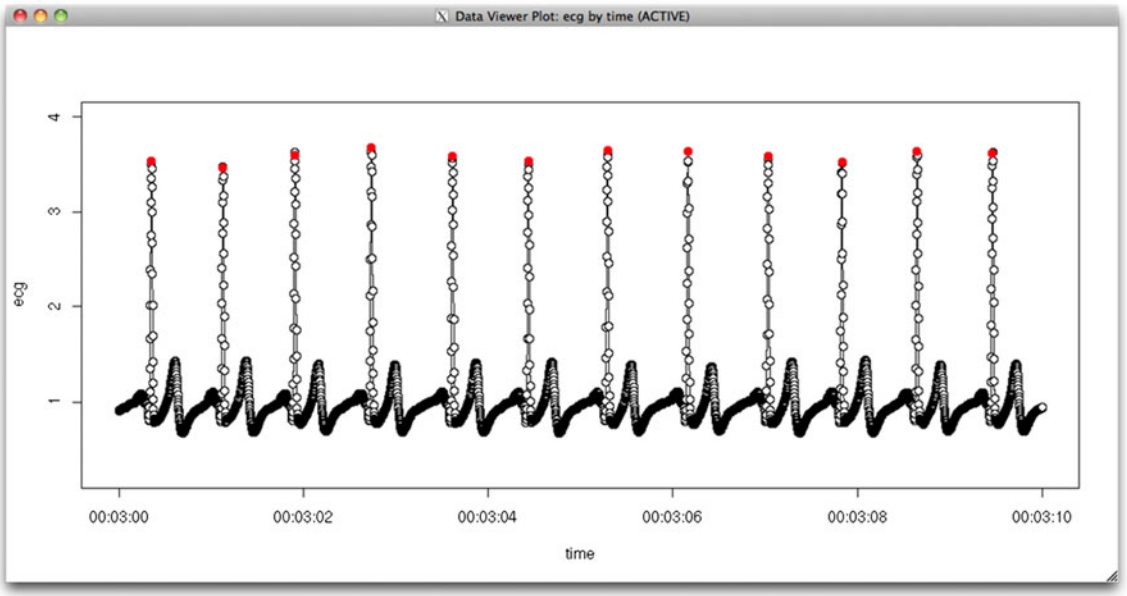

Fig. 1 Data viewer plot window displaying a period of electrocardiogram (ECG) data. The highlighted data points indicate R-spikes detected by QRS detection during the ECG processing step

absolute change from the previous beat (the threshold argument, in this case 25 %); the minimum percent or absolute deviation from a moving average, which assists greatly in limiting the number of false positives (the safe argument, in this case 100 ms); and a range of values outside of which data will be considered artifacts (the limits argument, in this case 300 and 1,500). IBI data are then reviewed using the third function call, plotting the IBI data (see Fig. 2), with highlighted data points here indicating those IBIs flagged as artifacts by the previous processing step. As before, highlighted data points can be added or removed manually and saved to the data file.

```
> extract.ibi()
> artifact.ibi(threshold = .25,
  safe = 100, limits = c(300,1500))
> review.ibi()
```

HRV estimation As a final step, the following function call uses the data file's accompanying epoch list to derive estimates of HRV for all specified epochs,

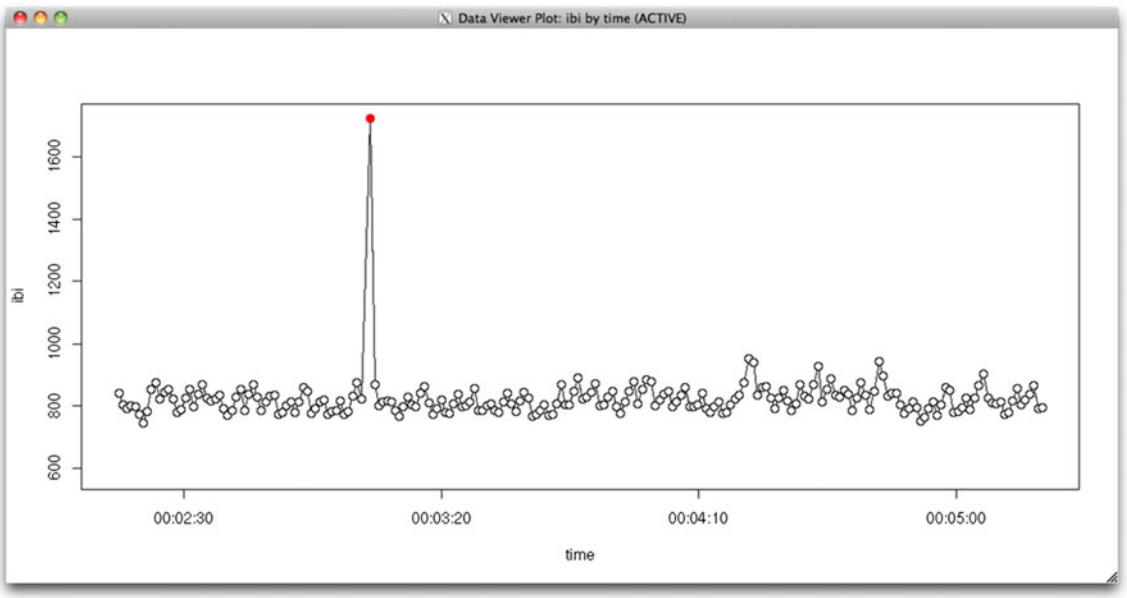

Fig. 2 Data viewer plot window displaying a period of interbeat-interval (IBI) data. The highlighted data point indicates an artifact detected during the IBI artifacting step

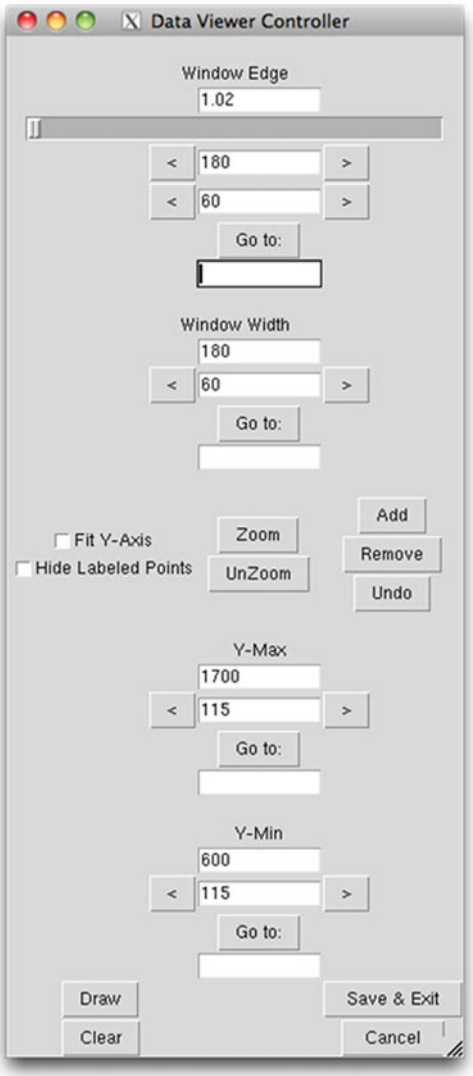

Fig. 3 Data viewer control window

with the f.bands argument specifying the frequency cutoffs employed in the band-limited variance routines.

```
> extract.hrv(f.bands = list(hf
  = c(.15,.4), lf = c(.04,.15)))
```

The output hrv file contains both an epoch label, used for identification in later analyses, and a number of variables useful for quality control. These include epoch length and the number and amount of time accounted for by both all beats (nibi and tibi), as well as those points identified as artifacts (nartifact and tartifact). This information allows for the simple calculation of the percentage of artifacts in relation to the total number of beats (nartifact/nibi) or time (tartifact/tibi). Importantly, such indices both allow for a simple means of quality control and enable greater transparency in the reporting of HRV findings. Other variables of interest include mean IBI, a number of time-domain measures of HRV (sdnn, the standard deviation of normal-to-normal beats; msd, the mean absolute difference between successive interbeat intervals; pnn50, the percentage of normal-to-normal beats greater than 50 ms); and the band-limited variance estimates called for in the f.bands argument above (in this case, hf and lf). Detailed information as to the interpretation of these variables can also be found in Allen et al. (2007). Once HRV processing is completed for all data files, the following function call will merge all hrv.gz data files in a specified directory, by default searching recursively through subdirectories.

```
> merge.data(pattern = ".hrv.gz",
  merged.prefix = "Merged.HRV")
```

The resulting merged data are written to disk in a comma-delimited ASCII text file with variable names in the first row. This output can be easily imported into any spreadsheet program or statistical analysis software.

Example data processing: Respiration

One point of consideration in the study of RSA, that component of HRV intrinsically linked to respiration, is the topic of respiration itself. Specifically at issue is whether within- and between-subjects differences in respiratory parameters may confound the interpretation of RSA. Opinions vary considerably as to the degree and nature of the experimental and/or statistical control of respiration necessary to validly interpret RSA, and the issue remains a point of contention among many methodologists (see Allen et al., 2007, for a review). Respiration data are routinely collected alongside the ECG so that respiratory parameters can be used to confirm that subjects are breathing within the expected frequency range and to possibly be used as covariates in subsequent analyses.

The PhysioScripts package includes basic functions to derive several indices of respiratory rate using a custom algorithm. Briefly, the respiratory waveform is bandpass filtered, and local maxima and minima, labeled as inspirations and expirations, respectively, are identified within a specified time window based on the shortest expected respiratory period (i.e., the fastest expected respiratory rate). Unbalanced inspirations and expirations—that is, two inspirations with no intervening expiration, or vice versa—are then corrected by removing the member of the paired values with the lesser absolute magnitude (i.e., the smaller inspiration or the larger expiration).

Respiration processing proceeds in a fashion similar to that used for HRV. The first function below detects inspirations and expirations, identifying their locations in the data file, and the second function allows for review and manual editing. The third function extracts several estimates of respiratory rate, in hertz, for each

epoch specified in the accompanying epoch list: mn.resp.rate and md.resp.rate, the mean and median of the inverse of all time periods between adjacent inspiratory peaks; and resp.freq, the peak frequency of the unsmoothed FFT-based periodogram. The final function merges the output data files just as with the HRV data files, this time targeting respiration output.

```
> process.resp()
> review.resp()
> extract.resp()
> merge.data(pattern = ".resp.gz",
  merged.prefix = "Merged.Resp")
```

Limitations and future directions

The functionality of PhysioScripts is presently limited to the cardiorespiratory variables discussed in this report, and when presented with a different data type (e.g., pupillometry), the end user's needs may be more readily met by other existing software. The software repository maintained by the Society for Psychophysiological Research (sprweb.org/repository) provides a list of available software for working with physiological data of varying types and may aid in the identification of suitable tools, although it should be noted that nearly all of these require MATLAB, and many do not provide cross-platform support (e.g., they run only in Windows). Lacking suitable existing software options, users may be required to develop their own applications, and can, of course, choose among many programming languages. It is our hope that the open source nature of PhysioScripts, particularly the core functionality (e.g., data visualization and file import/export), can facilitate the development of additional modules that will not only meet the user's needs, but also extend the functionality of the PhysioScripts package. It is in this regard that we view PhysioScripts as an extensible platform for physiological data processing. Furthermore, the expense of using PhysioScripts is effectively nil, so there is no cost impediment to testing the suitability of the software.

In view of the above facts, future directions will include both developing additional modules for other physiological data types and measures (e.g., tonic and phasic electrodermal activity and estimates of baroreflex function from continuous blood pressure data) and further automating the generation of epoch-based summary data by incorporating automated or manual event marks. Notwithstanding these future directions, we believe that this new collection of freely available and uniquely R-coded scripts for processing and editing physiological data provides an efficient and modifiable framework for executing critical data processing and analysis routines for a broad range of time series data.

Author note This research was supported by Grant No. R01-HL089850 from the National Institutes of Health. The authors thank Elizabeth Mezick and Kristen Stedenfeld for testing early versions of the software.

References

Allen, J. J., Chambers, A. S., & Towers, D. N. (2007). The many metrics of cardiac chronotropy: A pragmatic primer and a brief comparison of metrics. *Biological Psychology, 74,* 243–262.

Berntson, G. G., & Stowell, J. R. (1998). ECG artifacts and heart period variability: Don't miss a beat! *Psychophysiology, 35,* 127–132.

Bradley, M. M., Codispoti, M., Cuthbert, B. N., & Lang, P. J. (2001). Emotion and motivation I: Defensive and appetitive reactions in picture processing. *Emotion, 1,* 276–298. doi:10.1037/1528-3542.1.3.276

Calkins, S. D., Johnson, M. C. (1998). Toddler regulation of distress to frustrating events: temperamental and maternal correlates. *Infant Behavior and Development, 21*(3), 379–395.

Chambers, J. M. (1998). *Programming with data: A guide to the S language*. New York: Springer.

Fox, J. (2006). Structural equation modeling with the sem package in R. *Structural Equation Modeling, 13,* 465–486.

Friedman, B. H. (2007). An autonomic flexibility–neurovisceral integration model of anxiety and cardiac vagal tone. *Biological Psychology, 74,* 185–199.

Friesen, G. M., Jannett, T. C., Jadallah, M. A., Yates, S. L., Quint, S. R., & Nagle, H. T. (1990). A comparison of the noise sensitivity of nine QRS detection algorithms. *IEEE Transactions on Biomedical Engineering, 37,* 85–98.

Gianaros, P. J., Onyewuenyi, I. C., Sheu, L. K., Christie, I. C., & Critchley, H. D. (2012). Brain systems for baroreflex suppression during stress in humans. *Human Brain Mapping, 33,* 1700–1716. doi:10.1002/hbm.21315

Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics, 5,* 299–314.

Porges, S. W. (1992). Autonomic regulation and attention. In Campbell, B.A., Hayne, H. (Eds.), Attention and Information Processing in Infants and Adults: Perspectives from Human and Animal Research, (pp. 201–223). Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, USA.

R Development Core Team. (2010). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from www.R-project.org.

Rottenberg, J. (2007). Cardiac vagal control in depression: A critical analysis. *Biological Psychology, 74,* 200–211.

Thayer, J. F., & Lane, R. D. (2007). The role of vagal function in the risk for cardiovascular disease and mortality. *Biological Psychology, 74,* 224–242.

Thurston, R. C., Christie, I. C., & Matthews, K. A. (2012). Hot flashes and cardiac vagal control during women's daily lives. *Menopause, 19,* 406–412.

Vance, A. (2009, January 8). R you ready for R? Retrieved May 31, 2011, from http://bits.blogs.nytimes.com/2009/01/08/r-you-ready-for-r/