Published in final edited form as:
Behav Res Methods. 2013 March ; 45(1): 125–131. doi:10.3758/s13428-012-0233-x.

PhysioScripts: An Extensible, Open Source Platform for the Processing of Physiological Data

Israel C. Christie and **Peter J. Gianaros**
Israel Christie, Department of Psychiatry, University of Pittsburgh; Peter Gianaros, Department of Psychology, University of Pittsburgh.

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 paper describes a new tool to aide in the processing of physiological data: PhysioScripts. Key elements of PhysioScripts include a graphical interface to view and edit results of processing steps, as well as flexible framework to automate the creation of uniform or variable length epochs. The software is comprised of 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 (israelchristie.com/software).

Keywords

Software; R; Open Source; Physiological Data Processing

PhysioScripts: A Collection of Open Source R Scripts for the Processing of Physiological Data

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, in press), or the quantification of heart rate variability as an index of cardiac autonomic control during hotflashes in mid-life 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 (e.g., see 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

Correspondence concerning this article should be addressed to Israel Christie, Department of Psychiatry, University of Pittsburgh, 201 North Craig Street. Rm. 201, Pittsburgh, PA 15213. ichristie@gmail.com.

(e.g., extended ambulatory recordings in human or non-human 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 paper is to describe a collection of freely available scripts[1] for processing and editing physiological data, called PhysioScripts, which consists 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 upon 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 preprocessing steps [e.g., the conversion of electrocardiogram to interbeat interval (IBI) series] or 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 analysis 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 there are a number of similar software packages and routines 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, Massachusetts) or Labview (National Instruments; Austin, Texas). 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 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 can be augmented by the addition of contributed packages, which currently number nearly 3000. 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 place user numbers in the range of 1 to

[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.

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 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 doubled clicking the RData file, which initiates a new R session and loads the functions into memory. Most functions contain imbedded 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 graphical interface or 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, with variable names (e.g., "time", "ecg") in the first row. By default, files are compressed using the gzip format, though 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, though perhaps inefficient, task using a text editor[2]. An example function[3] is included which 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 timepoint '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 hours are being processed.

[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.

[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.

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 where 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 4 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 minute baseline beginning at 10:00 into the recording, a 5 minute stressor task beginning at 20:30, and a 10 minute recovery beginning at 25:30. If the experimenter desired 1 minute epochs for both the baseline and stressor conditions, and 5 minute 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
```

[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.

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
```

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, “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 the on the type of signal being viewed, either the maximum, 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 phenomenon ranging from attentional capacity (Porges, 1992) and emotion regulation (Calkins and 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 raw electrocardiogram (ECG) data file to 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 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 basis for many studies involving HR 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 & Artifacting

Interbeat intervals (IBI), 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: percent or 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 msec); and a range of values outside of which will be considered artifact (the limits argument, in this case 300 and 1500). IBI data are then reviewed using the third function call, plotting the IBI data (see Fig. 2) with highlighted data points here indicating those IBI flagged as artifact 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, 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 artifact (nartifact and tartifact). This information allows for the simple calculation of the percentage of artifact 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 msec), and the band limited variance estimates called for in the f.bands argument above (in this case, hf and lf). Detailed information as to 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 subject differences in respiratory parameters may confound the interpretation of RSA. Opinions vary considerably as to the degree and nature of experimental and/or statistical control of respiration necessary to validly interpret RSA, and the subject remains a point of contention among many methodologists (see Allen et al., 2007 for review). Respiration data is routinely collected alongside the ECG so that respiratory parameters can be used to confirm 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 Hz, 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 paper 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, though it should be noted that nearly all require Matlab and many do not provide cross-platform support (e.g., run only in Windows). Lacking suitable existing software options, the user 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, file import/export, etc.), 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 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, future directions include both developing additional modules for other physiological data types and measures (e.g., tonic and phasic electrodermal activity, 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.

Acknowledgments

This research was supported by a grant from the National Institutes of Health (R01-HL089850). The authors would like to thank Elizabeth Mezick and Kristen Stedenfeld for testing early versions of the software.

References

Allen JJ, Chambers AS, Towers DN. The many metrics of cardiac chronotropy: a pragmatic primer and a brief comparison of metrics. Biological Psychology. 2007; 74:243–262. [PubMed: 17070982]

Berntson GG, Stowell JR. ECG artifacts and heart period variability: don't miss a beat. Psychophysiology. 1998; 35:127–132. [PubMed: 9499713]

Bradley MM, Codispoti M, Cuthbert BN, Lang PJ. Emotion and motivation I: defensive and appetitive reactions in picture processing. Emotion. 2001; 1:276–298. [PubMed: 12934687]

Chambers, JM. Programming with data: A guide to the S language. Springer; New York: 1998.

Fox J. Structural Equation Modeling With the sem Package in R. Structural Equation Modeling. 2006; 13:465–486.

Friedman BH. An autonomic flexibility-neurovisceral integration model of anxiety and cardiac vagal tone. Biological Psychology. 2007; 74:185–199. [PubMed: 17069959]

Friesen GM, Jannett TC, Jadallah MA, Yates SL, Quint SR, Nagle HT. A comparison of the noise sensitivity of nine QRS detection algorithms. IEEE Transactions in Biomedical Engineering. 1990; 37:85–98.

Gianaros PJ, Onyewuenyi IC, Sheu LK, Christie IC, Critchley HD. Brain systems for baroreflex suppression during stress in humans. Human Brain Mapping. (in press).

Ihaka R, Gentleman R. R: A Language for Data Analysis and Graphics. Journal of Computational and Graphical Statistics. 1996; 5:299–314.

R Development Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing; Vienna, Austria: 2010. ISBN 3-900051-07-0, URL http://www.R-project.org/

Rottenberg J. Cardiac vagal control in depression: a critical analysis. Biological Psychology. 2007; 74:200–211. [PubMed: 17045728]

Thayer JF, Lane RD. The role of vagal function in the risk for cardiovascular disease and mortality. Biological Psychology. 2007; 74:224–242. [PubMed: 17182165]

Thurston RC, Christie IC, Matthews KA. Hot flashes and cardiac vagal control during women's daily lives. Menopause. 2012; 19:406–412. [PubMed: 22095062]

Vance, A. [Retrieved May 31, 2011] R You Ready for R?. Jan 8. 2009 from http://bits.blogs.nytimes.com/2009/01/08/r-you-ready-for-r/

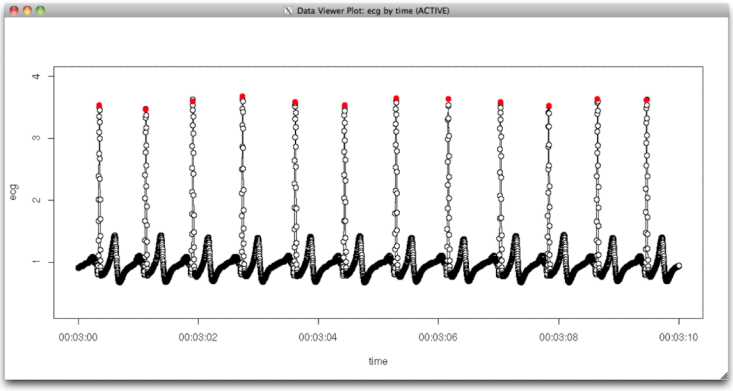

Figure 1.
The data viewer plot window displaying a period of ECG data. The highlighted data points indicate R-spikes detected by QRS detection during the ECG processing step.

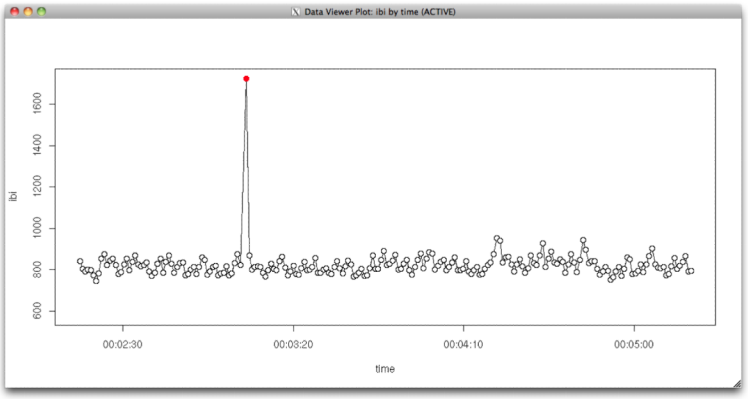

Figure 2.
The data viewer plot window displaying a period of IBI data. The highlighted data point indicates artifact detected during the IBI artifacting step.

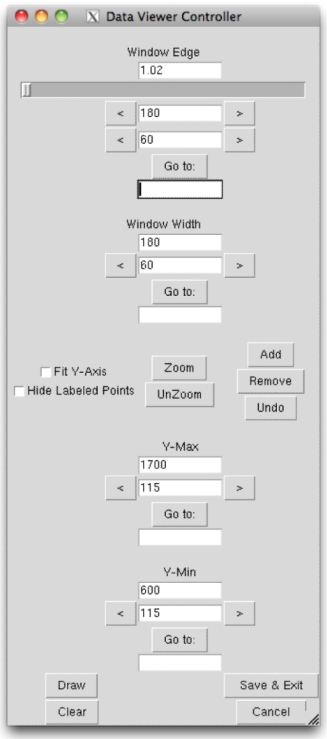

Figure 3.
The data viewer control window.