*(this document is adopted from Supplementary Information Section S4 of the associated manuscript)*

**S4. Guidelines on implementing the R codes**

The R package "`CardiacDP`" includes two functions: the first function `collatedata` can be used to collate separate data files into a single data table for subsequent analyses; and the second function `computeHR` can be used to compute average heart rate from the cardiac data (which represents the algorithm presented in the main manuscript). Depending on the input file format (whether it is a .zip file that requires data collation, or a .csv file that is ready for heart rate computation), function `collatedata` will be automatically called within the function `computeHR`, and the overall workflow is summarized in **Figure S4.1**.

*S4.1 Function* `collatedata`*: data collation*

Some oscilloscope software (e.g. Picoscope) produces small and numerous data files as outputs instead of large, collated data files containing all measurements, presumably because large data files are difficult to operate. These numerous files, however, have to be collated into a single data table to facilitate subsequent data analysis procedures. The function `collatedata`, therefore, serves to collate separate data files to generate a single data table of cardiac data (each column represents measurements of one individual; referred to as "channels") measured across fixed time intervals.

This function automatically reads data files and collates them in chronological order as inferred by the file names. Taking the data files produced from Picoscope (v6, Pico Technology, UK) as an example (**Figure S4.2**), each 'page' as visualized on the software is saved as separate .csv files in one folder. After a defined number of pages, the software will reset, producing another folder with, again, separate .csv files representing different pages. All files are then nested inside one folder which can then be compressed (as a .zip file) for this function. As long as the folders and files are named in chronological order, this function is designed to automatically read them accordingly, and one by one in hierarchy (i.e. collate all files within one

folder before moving on to the next folder). After collating all files, time will be rewritten as a sequence increasing from zero at a fixed time interval. As such, only measurements taken across fixed time intervals should be collated as one data table for analyses, or else the data should be analyzed separately.

### *S4.2 Function* `computeHR`*: heart rate computation*

From the collated data table, the function `computeHR` analyzes channel (i.e. individual) by channel and computes heart rates of each sequence. Users can customize the parameters used in this function (**Table S4.1**; see Section S4.3 for recommendations on customization).

To minimize computation time, the cardiac data are first analyzed at a reduced time interval (default to be 0.01s; **Table S4.1**). As a result of inferring heart rates from time lags in ACF, the time interval used in the analysis determines the resolution of heart rates computed (which is implied by the difference in heart rate across a time lag difference of 1). Taking the time interval of 0.01s as an example (**Table S4.2** and **Figure S4.3**), the difference in heart rate decreases with time lag (i.e. the resolution increases with time lag) and such a relationship is not linear: for the same time lag difference of 1, the difference in heart rate computed is large at small time lags (coarse resolution), whereas the difference is small at large time lags (fine resolution). As such, the data are automatically re-analysed at the finest time interval when the corresponding percentage difference is > 2% (**Table S4.2** and **Figure S4.3**).

By employing ACF with a genetic algorithm framework, the positions and durations of the final periodic sub-sequences ("`finalsubseq`") and their candidate heart rates ("`candidateHR`") are determined from the cardiac data per sequence (**Table S4.3**; see also Section S4.4 for the examples). From these sub-sequences, the final heart rates are computed for each sequence, either with or without implementing the tracking index (i.e. "`results_ACF`" and "`results_TI`" refer to the "ACF + GA" and "ACF + GA + TI" approaches as described in the main manuscript respectively; **Table S4.3**; see also Section S4.4 for examples).

*S4.3 Parameter customization*

The default parameters have been tested on cardiac data from almost twenty invertebrate species and have shown to be widely applicable across diverse taxa. Consequently, users are recommended to start with the default parameters (as stated in **Table S4.1**). To determine whether further customization is necessary, users need to develop a good understanding of the cardiac data they are working with. As such, users are strongly recommended to conduct strategic spot-checks to compare manual counts to algorithm estimates. Across the same analysis interval as specified in the algorithm (one minute by default), users can count the number of full beats and divide it by the actual total duration across these beats (in seconds). Heart rate in bpm can then be computed as $\frac{number\ of\ beats}{total\ duration\ (s)/\ 60}$. To build an overall performance profile, users can analyze the data at a coarse time scale, say every 30 minutes (i.e. analyzing the 1st minute, the 31st minute, and so on), to verify whether the heart rate estimates derived using the current parameters are reliable. In addition, users can strategically select challenging or noisy sessions of data for comparison to identify potential issues and concerns. This approach allows users to efficiently validate algorithm performance and gain confidence in the results without conducting exhaustive manual analysis for the entire dataset.

S4.3.1 Customization on the optimization procedures

In order to initiate the optimization procedures in the genetic algorithm framework, recurrent heartbeats must first be determined in each sub-sequence using autocorrelation. This initiation may fail when the period of heartbeats is comparable to the initial duration of the sub-sequences, resulting in no usable data. Consequently, the ratio of heartbeat periods to the initial duration of sub-sequences is critical to ensure successful initiation A rule of thumb is that the initial sub-sequence duration should be at least double, or more conservatively triple, of the expected heartbeat period (so that at least 2 – 3 full heartbeats occur within each sub-sequence).

Two parameters, "`an_in`" (as in 'analysis interval', default to be a minute) and "`pop_size`" (as in 'population size', default to be ten), are primarily related to this initiation, as they collectively determine the initial duration of sub-sequences (which equals to one-tenth of a minute, i.e. 6s by default). When working with animals with slow heart rates (< 30 bpm), users may consider extending the analysis interval to two or five minutes to ensure adequate heartbeat coverage within sub-sequences. An alternative approach is to reduce the number of sub-sequences (population size), but this is not recommended as it may overcome the initiation failure but weaken the overall optimization process (i.e. the optimization will terminate prematurely with a low population size, potentially compromising the quality of results).

S4.3.2 Customization on data utilization

The parameter "`acf_thres`" (default to be 0.5) is a threshold used in ACF to classify periodic data from aperiodic noise. Essentially, data with periods exhibiting autocorrelation values >= 0.5 are considered informative and usable for heart rate estimation, and are thus retained in the optimization procedures. Conversely, data with periods exhibiting autocorrelation values < 0.5 are considered irregular oscillations and are removed during the optimization.

The autocorrelation value reflects the degree of similarity between the cardiac data and its time-delayed counterpart at a certain time lag, and can be reduced in cases of variations across beats (e.g. vertical shift in amplitude) and noise interference. As such, increasing this threshold tightens the screening criteria for recurrent beats, permitting less variation in waveforms across successive heartbeats, which typically results in greater data loss. In contrast, decreasing this threshold accepts more variable waveforms as recurrent beats, which leads to higher data utilization but potentially includes more erroneous data in computation.

After repeated testing across multiple species and individuals, the default ACF threshold has been shown to provide a good balance between accuracy and data utilization. Users are generally not recommended to decrease this threshold as doing so may compromise the accuracy of heart rate estimates. In cases of strong and neat

cardiac data with minimal noise interference, there is likewise no need to increase the threshold, as the default value is already sufficient to determine the heart rate accurately.

S4.3.3 Customization on computation time

In cases of limited computational power or when seeking to reduce computation time, several parameters can be adjusted. Firstly, users can reduce the total number of sequences analysed by extending the analysis interval (`an_in`). However, it should be noted that this will produce outputs at coarser time intervals representing estimates averaged over longer durations, which may obscure rapid variations in heart rates. This option is most suitable when working with long cardiac datasets where heart rate changes are not rapid or when high temporal resolution is not required.

Secondly, the parameter "`max_gen`" (as in maximum generations, default to be 20) specifies the maximum number of rounds in the optimization procedures. While this parameter can be reduced to minimize computation time, it is often not the limiting factor in practice. In most cases with clean cardiac data, the optimization process terminates before reaching the maximum number of rounds, either when the full sequence is classified as periodic or when there is no further increase in the duration of any periodic sub-sequences for two consecutive rounds (see **Table S2.1**). Consequently, reducing this parameter may not yield observable improvements in computation time for typical datasets, and users should be cautious to avoid prematurely terminating potentially beneficial optimization iterations.

*S4.4 A step-by-step example*

The functions of this package are illustrated with the examples below.

When a .zip file (with the corresponding file path) comprising folders and files named in chorological order is provided, the function `collatedata` will automatically run to read and collate the separate files, before proceeding with the computation analysis function `computeHR`. By default, neither the collated data table nor the analysis output is saved.

```
output <- computeHR (
  file_path = "~/20210518A.zip",
  save_outputs = FALSE,
  output_dir = NULL,
  verbose = TRUE)
```

Alternatively, the user can choose to manually run the function `collatedata` (where a copy of the collated data table can be saved) and then provide the saved .csv for subsequent analysis.

```
collatedata(
  file_path = "~/20210518A.zip",
  output_file = "~/20210518A.csv", # PATH_TO_SAVE_COLLATED_CSV
  verbose = TRUE)


output <- computeHR(
  file_path = "~/20210518A.csv", # csv_path
  save_outputs = TRUE,
  output_dir = "~/", # PATH_TO_STORE_OUTPUTS
  verbose = TRUE)
```

The data structure will be analyzed and displayed for the user's reference, and the progress of collation will be indicated.

```
Zip file name: 20210518A
Using extracted directory: …/20210518A
Subdirectories found: 20210518A-0001, 20210518A-0001 (2), 20210518A-0001
(3), 20210518A-0001 (4), 20210518A-0001 (5), 20210518A-0001 (6)
…
Number of files found: 90
Trying to read file: …/20210518A/20210518A-0001/20210518A-0001_01.csv
…
Number of columns in data: 9
Successfully read file with format: semicolon
Number of folders: 6
Number of channels: 8
```

```
Names of channels: Channel A, Channel B, Channel C, Channel D, Channel E,
 Channel F, Channel G, Channel H
```

```
Reading data: 17%
Reading data: 33%
Reading data: 50%
Reading data: 67%
Reading data: 83%
Reading data: 100%
Finalizing...
Total duration: 170.4 mins
```

After collation, the structure of the collated data table is displayed and the collated data table is saved to the designated file path.

```
Classes ‘data.table’ and 'data.frame':    7667044 obs. of  9 variables:
 $ Time     : num  0 0.00133 0.00267 0.004 0.00533 ...
 $ Channel A: num  -0.002198 -0.00058 -0.001129 -0.000061 -0.001648 ...
 $ Channel B: num  0.00369 0.00314 0.00424 0.00314 0.00314 ...
 $ Channel C: num  0.00424 0.00479 0.00424 0.00372 0.00479 ...
 $ Channel D: num  -0.000488 -0.000488 -0.001007 -0.001007 -0.001007 ...
 $ Channel E: num  -0.0266 -0.0277 -0.0282 -0.0277 -0.0304 ...
 $ Channel F: num  -0.023 -0.0246 -0.0246 -0.023 -0.0241 ...
 $ Channel G: num  0.00906 0.00906 0.01337 0.00748 0.008 ...
 $ Channel H: num  -0.00534 -0.00375 -0.00641 -0.00693 -0.00693 ...
```

```
Collated data table saved to: ~/20210518A.csv
```

With the collated data table, the heart rate computation analyses are conducted channel by channel and the progress will be indicated.

```
Calculating heart rate: Channel A (Duration: 170.4 mins, Sequences: 17
0)...
Generating output...
Saved CSV and PNG files for Channel A to ~/
Calculating heart rate: Channel B (Duration: 170.4 mins, Sequences: 17
0)...
Generating output...
…
```

The user can then access the output by the channel name. `finalsubseq` is a list showing the positions and durations of the final sub-sequences determined for each sequence. They are presented as data tables (s = start index of the sub-sequence; e = end index of the sub-sequence; p = which of the initial population the sub-sequence

is derived from; and f = duration of the sub-sequence), the rows of which represent separate final sub-sequences. The example below shows the first few sequences (e.g. `[[4]]` indicates at the 4th minute there are three final sub-sequences).

```
# positions (in indices) and durations of the final sub-sequences
output [["finalsubseq"]] [["Channel A"]]
```

```
[[1]]
       s     e     p     f
   <num> <num> <num> <num>
1:     1  6430     1  6429

[[2]]
       s     e     p     f
   <num> <num> <num> <num>
1:     1  4387     1  4386

[[3]]
       s     e     p     f
   <num> <num> <num> <num>
1:     1  5775     1  5774

[[4]]
       s     e     p     f
   <num> <num> <num> <num>
1:  2197  3856     6  1659
2:  3669  6430     8  2761
3:  3570  6363     8  2793
…
```

The corresponding candidate heart rates per sub-sequence can be found in `candidateHR` (ACF = autocorrelation value; lag = time lag; and hr = heart rate). Taking the 4th minute again as an example, there are two candidate heart rates for the first sub-sequence (as shown in `[[4]][[1]]`), and only one candidate heart rate for the other two sub-sequences (as shown in `[[4]][[2]]`and `[[4]][[3]]`).

```
# candidate heart rates of the final sub-sequences
output [["candidateHR"]] [["Channel A"]]
```

```
[[1]]
[[1]][[1]]
         ACF   lag       hr
       <num> <num>    <num>
1: 0.1438675   88 73.05174
2: 0.5748986  180 35.71418
```

```
[[2]]
[[2]][[1]]
         ACF   lag       hr
       <num> <num>    <num>
1: 0.5141086  170 37.81502


[[3]]
[[3]][[1]]
         ACF   lag       hr
       <num> <num>    <num>
1: 0.5002503  164 39.19849


[[4]]
[[4]][[1]]
         ACF   lag       hr
       <num> <num>    <num>
1: 0.4296444  156 41.20867
2: 0.5028381  307 20.93991

[[4]][[2]]
         ACF   lag       hr
       <num> <num>    <num>
1: 0.5149000  188 34.19443

[[4]][[3]]
         ACF   lag       hr
       <num> <num>    <num>
1: 0.5145251  186 34.56211

…
```

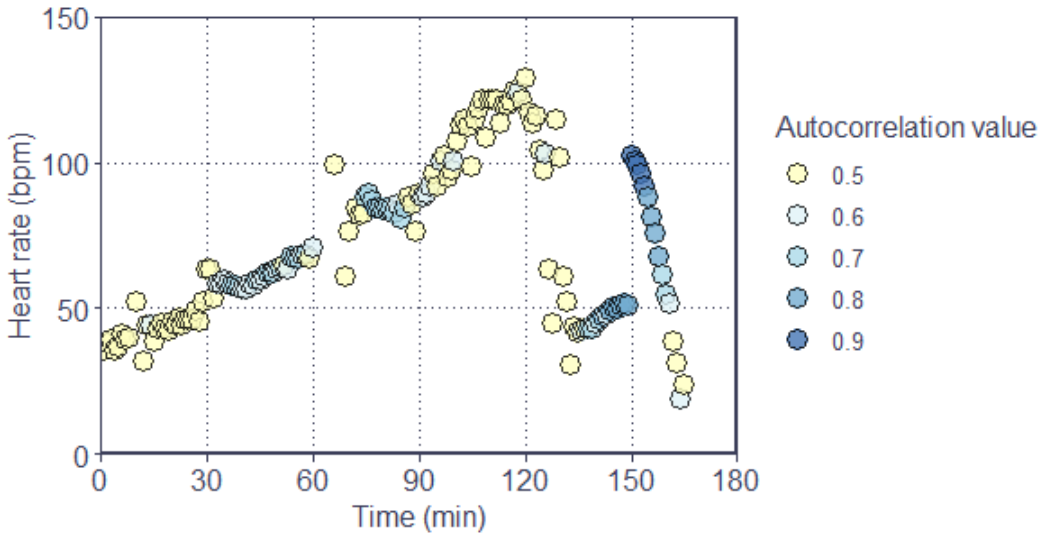
The final results after evaluating the candidate heart rates, checking for resolution and weighting for durations can eventually be obtained from `results_ACF` and `results_TI`. These results refer to evaluating the candidate heart rates by merely autocorrelation values (i.e. the “ACF + GA” approach in the publication) or the tracking index (i.e. the “ACF + GA +TI” approach in the publication) respectively. Each of them consists of 1) the details of the sub-sequences (ix = sequence (in minute), res = time interval (resolution) used in the final analysis, and the remaining variables are defined as above; `subseqHR`); 2) the weighted heart rate per sequence (`weightedHR`); and 3) a plot of weighted heart rate against time (`plot`).

```
# results obtained from evaluating the candidate heart rates by autocorrelati
on values
output [["results_ACF"]] [["Channel A"]]
```

```
$subseqHR
       ix   win     s     e     p     f       ACF   lag       hr         res
    <int> <num> <num> <num> <num> <num>     <num> <num>    <num>       <num>
  1:    1     1     1  6430    10  6429 0.5748986  180  35.71418 0.00933336
  2:    2     1     1  4201     1  4200 0.5240022  170  37.81502 0.00933336
  3:    3     1     1  5725     1  5724 0.5000011  164  39.19849 0.00933336
  4:    4     1  3215  3856     6   641 0.5241728  156  41.20867 0.00933336
  5:    4     2  3548  6430     8  2882 0.5040846  188  34.19443 0.00933336
 ---
227:  166     1    NA    NA    NA    NA        NA   NA        NA 0.00933336
228:  167     1    NA    NA    NA    NA        NA   NA        NA 0.00933336
229:  168     1    NA    NA    NA    NA        NA   NA        NA 0.00933336
230:  169     1    NA    NA    NA    NA        NA   NA        NA 0.00933336
231:  170     1    NA    NA    NA    NA        NA   NA        NA 0.00933336

$weightedHR
       ix      wACF      whr
    <int>     <num>    <num>
  1:    1 0.5748986 35.71418
  2:    2 0.5240022 37.81502
  3:    3 0.5000011 39.19849
  4:    4 0.5071297 35.25769
  5:    5 0.5134750 36.53428
 ---
166:  166        NA       NA
167:  167        NA       NA
168:  168        NA       NA
169:  169        NA       NA
170:  170        NA       NA

$plot
```

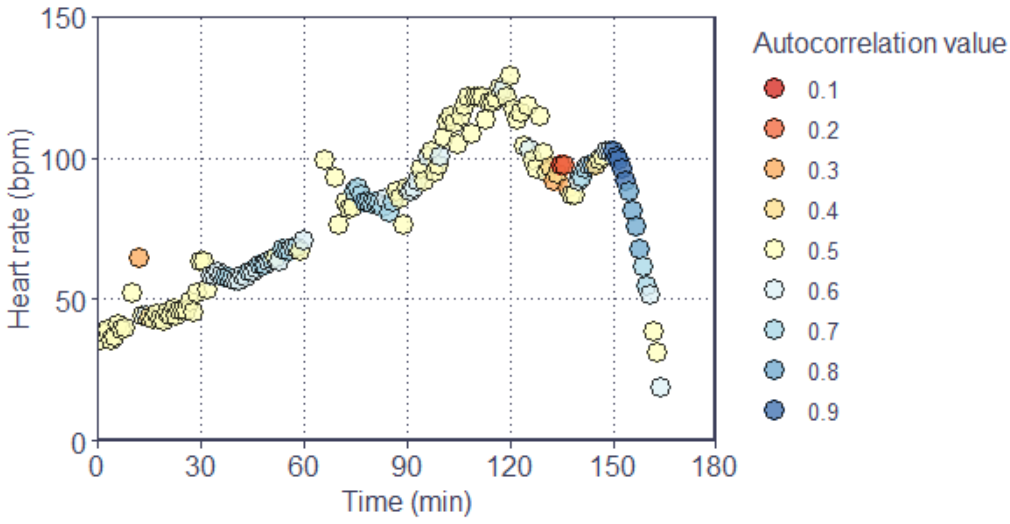


```
# results obtained from evaluating the candidate heart rates by the tracking
index
output [["results_TI"]] [["Channel A"]]
```

```
$subseqHR
       ix   win     s     e     p     f       ACF   lag       hr         res
    <int> <num> <num> <num> <num> <num>     <num> <num>    <num>       <num>
  1:     1     1     1  6430    10  6429 0.5748986   180 35.71418 0.00933336
  2:     2     1     1  4201     1  4200 0.5240022   170 37.81502 0.00933336
  3:     3     1     1  5725     1  5724 0.5000011   164 39.19849 0.00933336
  4:     4     1  3215  3856     6   641 0.5241728   156 41.20867 0.00933336
  5:     4     2  3548  6430     8  2882 0.5040846   188 34.19443 0.00933336
 ---
227:   166     1    NA    NA    NA    NA        NA    NA       NA 0.00933336
228:   167     1    NA    NA    NA    NA        NA    NA       NA 0.00933336
229:   168     1    NA    NA    NA    NA        NA    NA       NA 0.00933336
230:   169     1    NA    NA    NA    NA        NA    NA       NA 0.00933336
231:   170     1    NA    NA    NA    NA        NA    NA       NA 0.00933336

$weightedHR
       ix      wACF      whr
    <int>     <num>    <num>
  1:     1 0.5748986 35.71418
  2:     2 0.5240022 37.81502
  3:     3 0.5000011 39.19849
  4:     4 0.5071297 35.25769
  5:     5 0.5134750 36.53428
 ---
166:   166        NA       NA
167:   167        NA       NA
168:   168        NA       NA
169:   169        NA       NA
170:   170        NA       NA

$plot
```



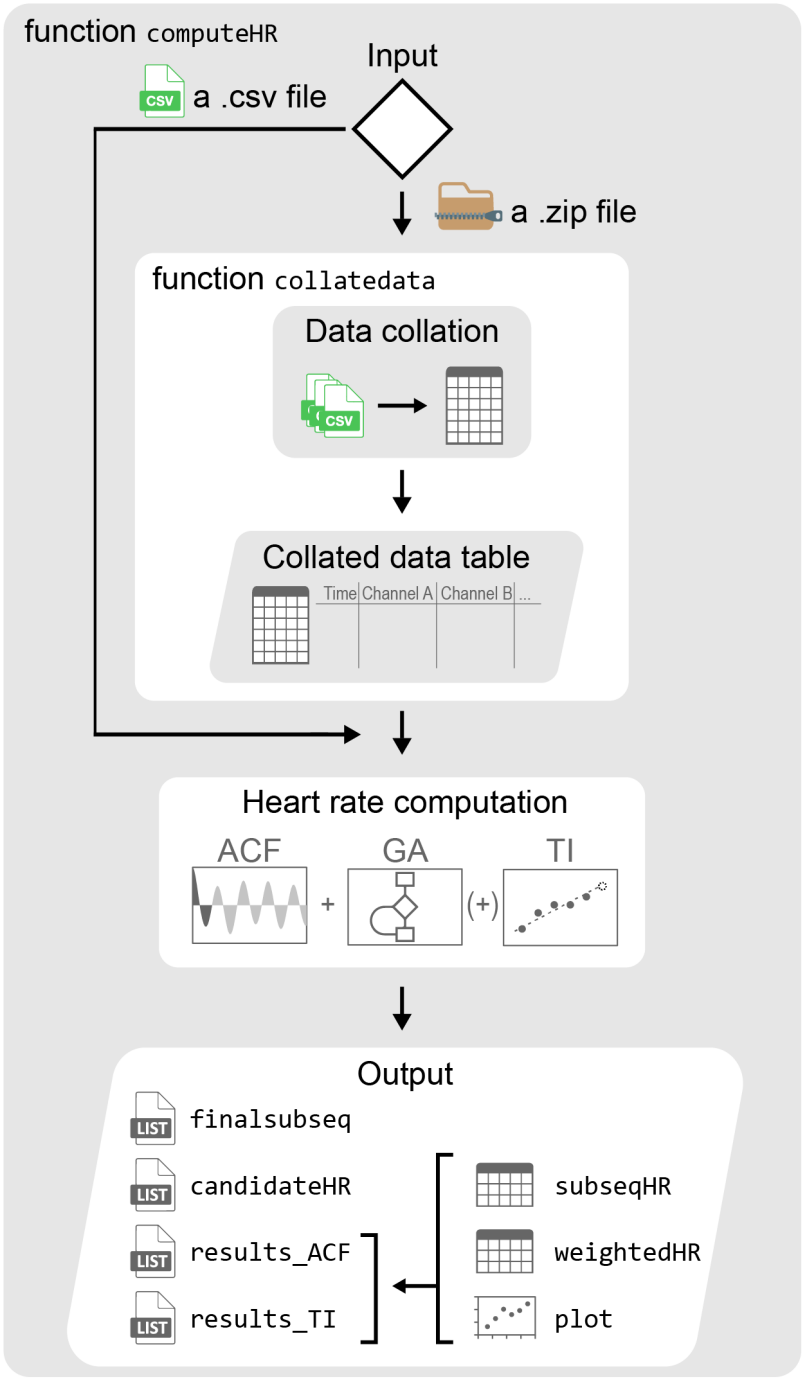



**Figure S4.1** Workflow of the two functions in `CardiacDP`. In function `computeHR`, depending on the input file format, either the function `collatedata` will be automatically called to read and collate separate files into a single data table in case of a .zip file, or the .csv file will proceed directly to the heart rate computation using autocorrelation function (ACF), genetic algorithm (GA) and tracking index (TI). Details of the output are summarized in **Table S4.3**.

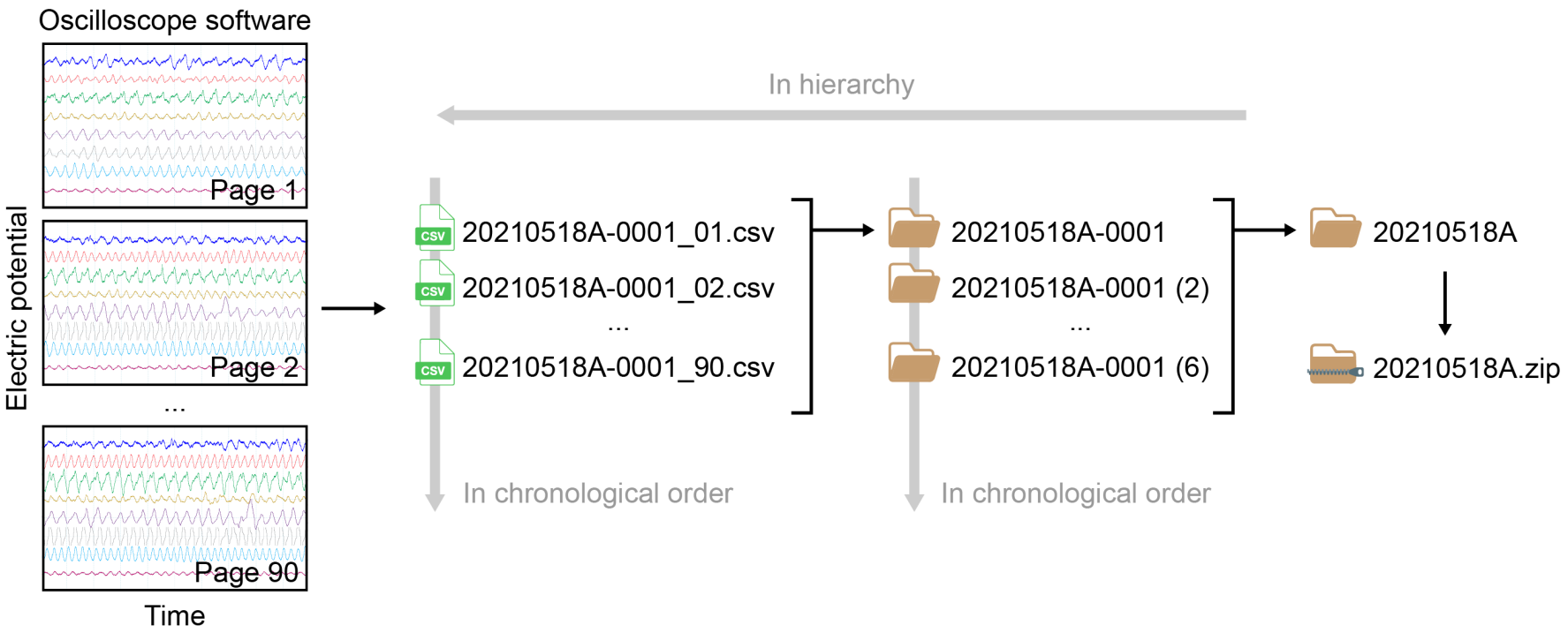



**Figure S4.2** An example data structure of the data files produced by the oscilloscope software Picoscope (v6, Pico Techology, UK). Each 'page' of the software is saved as separate .csv files nested within a folder. After a defined number of pages, the software resets and creates another folder in which 'pages' are again saved as separate .csv files. The files and folders are automatically named in chronological order and are finally nested within a single folder which is compressed to a .zip file for the function `collatedata`. This function reads the data in chronological order and in hierarchy (as indicated by the vertical and horizontal grey arrows respectively) and collates them to give a single data table for subsequent analyses.

**Table S4.1** Variables used in heart rate computation that allow user customization.

| Variable | Definition | Default value |
|---|---|---|
| `an_in` | As in 'analysis interval': the length of a sequence to be analyzed in parallel (minute). | 1 |
| `reduce_res` | The time interval of cardiac data to be analyzed at a reduced resolution (seconds). | 0.01 |
| `pop_size`[#] | As in 'population size*': the number of sub-sequences to analyze within each sequence in the genetic algorithm. | 10 |
| `max_gen`[#] | As in 'maximum generations*': the maximum number of rounds to repeat the selection and mutation procedures in the genetic algorithm. | 20 |
| `patience`[#] | The number of rounds to repeat when there are no further changes in the sub-sequence durations before terminating the genetic algorithm. | 2 |
| `acf_thres` | The threshold used in ACF to classify periodic and aperiodic data. Increasing this threshold may increase the power to screen off noises but also lead to greater data loss. | 0.5 |
| `lr_thres` | The threshold used in the linear extrapolation of when establishing the tracking index. | 0.7 |

*see **Table S2.1** for the equivalent terms used in the typical genetic algorithm and the present study

[#]variables that are in potential tradeoff with computation time (i.e. increasing these numbers may facilitate the optimization process of the genetic algorithm but increase the computation time)

**Table S4.2** The relationship between time lag, the corresponding heart rate and resolution (as implied by the difference in heart rate by a time lag difference of 1) when analyzing cardiac data with the time interval = 0.01s. The resolution increases with time lag and the relationship is not linear (see also **Figure S4.3**): for the same time lag difference of 1, the difference in heart rate computed is large at small time lags, whereas the difference is small at large time lags. As a result, all data are first analyzed at a reduced time interval to minimize computation time and are only re-analyzed at the finest time interval when the corresponding percentage difference is $\geq$ 2% (i.e. for results with time lag $\leq$ 49 in this example).

| Time lag | Period (s / beat) | Heart rate (bpm) | Difference in heart rate between time lag *t* and *t+1* (bpm) | % difference |
|---|---|---|---|---|
| 0 | NA | NA | NA | NA |
| 1 | 0.01 | 6000 | 3000 | 50.0 |
| 2 | 0.02 | 3000 | 1000 | 33.3 |
| 3 | 0.03 | 2000 | 500 | 25.0 |
| 4 | 0.04 | 1500 | 300 | 20.0 |
| 5 | 0.05 | 1200 | 200 | 16.7 |
| 6 | 0.06 | 1000 | 143 | 14.3 |
| 7 | 0.07 | 857 | 107 | 12.5 |
| 8 | 0.08 | 750 | 83 | 11.1 |
| 9 | 0.09 | 667 | 67 | 10.0 |
| 10 | 0.10 | 600 | 55 | 9.1 |
| … | | | | |
| 46 | 0.46 | 130.4 | 2.8 | 2.13 |
| 47 | 0.47 | 127.7 | 2.7 | 2.08 |
| 48 | 0.48 | 125.0 | 2.6 | 2.04 |
| **49** | **0.49** | **122.4** | **2.4** | **2.00** |
| 50 | 0.50 | 120.0 | 2.4 | 1.96 |
| 51 | 0.51 | 117.6 | 2.3 | 1.92 |
| 52 | 0.52 | 115.4 | 2.2 | 1.89 |

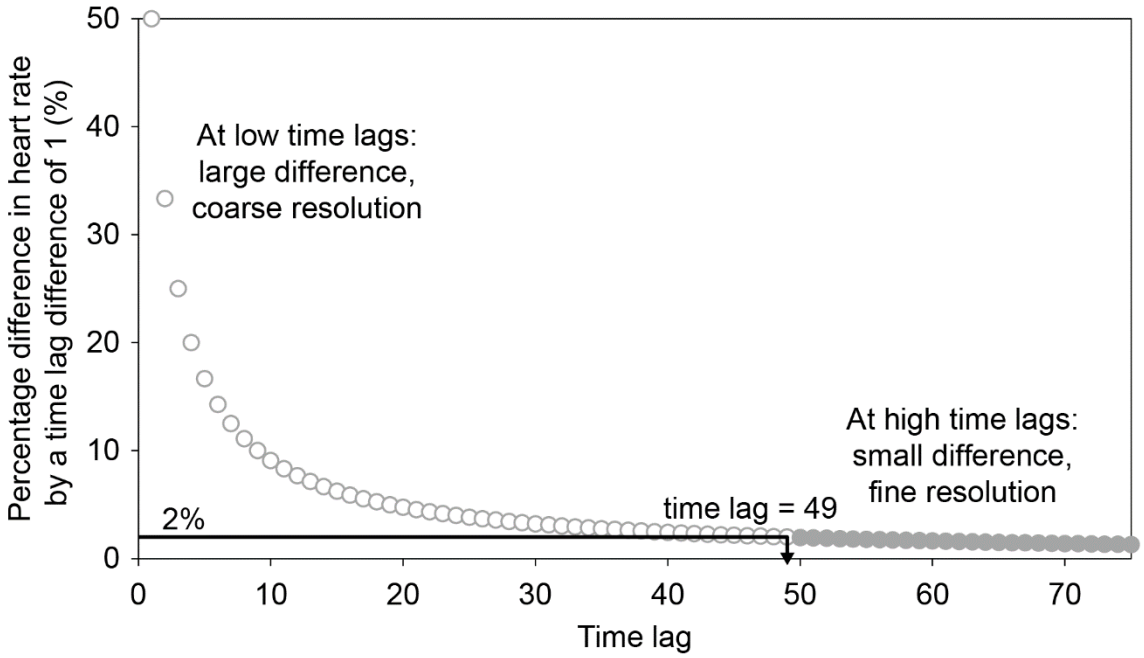



**Figure S4.3** The relationship between the percentage difference in heart rate computed across a time lag difference of 1 with time lag. All data are first analyzed at the reduced resolution to minimize computation time (as indicated by the closed dots) and are only re-analyzed at the finest time interval when the percentage difference is $\geq$ 2% (when time lag $\leq$ 49; as indicated by the open dots).

**Table S4.3** The output from function `computeHR` per channel consisting of four items, with `finalsubseq` and `candidateHR` indicating the results from the genetic algorithm, and `results_ACF` and `results_TI` indicating the final heart rates after evaluating the candidate heart rates, checking for resolution and weighting for durations.

| Variable | Content |
|---|---|
| `finalsubseq` | A list of positions and durations of the final periodic sub-sequences |
| `candidateHR` | A list of candidate heart rates extracted from ACF for each sub-sequence |
| `results_ACF` | Results obtained from evaluating the candidate heart rates of each sub-sequence based on autocorrelation values (i.e. following the "ACF + GA" approach as described in the main manuscript).<br>Consisted of three items: 1) `subseqHR`: a list of sub-sequences and the corresponding heart rates and durations; 2) `weightedHR`: a list of final heart rates per sequence after weighting; and 3) `plot`: a plot of final heart rates against time |
| `results_TI` | Results obtained from evaluating the candidate heart rates of each sub-sequence using a tracking index (i.e. following the "ACF + GA + TI" approach as described in the main manuscript).<br>Consisted of three items: 1) `subseqHR`: a list of sub-sequences and the corresponding heart rates and durations; 2) `weightedHR`: a list of final heart rates per sequence after weighting; and 3) `plot`: a plot of final heart rates against time |