plethy vignette

Daniel Bottomly*

May 3, 2016

Contents

1 The plethy package

The goal of the *plethy* package is to enable efficient import, storage and retrieval of plethysmography and other related data for statistical analysis in R. Currently, the *plethy* package contains functionality to parse .csv files derived from Buxco whole body plethysmography instruments and import the resulting data into a SQLite database for convenient access. Once created, the database contains an organized representation of the Buxco file that can be easily queried through pre-defined R code or through lower-level SQL access using the RSQLite package. One of the features is the ability to add new data and/or annotations based on existing data within the database without modifying the old data. An example of this is the computation of the number of days past the first measurement as well as the assignment of experimental labels such as whether the measurement is part of acclimation or experimental data as is demonstrated below. Subsets of the data are retrieved in *data.frame* form for use with statistical analysis software packages in R such as *nlme*.

2 Exporting the Buxco CSV file

The Buxco Whole Body Plethysmographer is run through the Finepointe software suite (Buxco Research Systems, Wilmington, NC). Finepointe has explicit experimental set-up tools that should be used to set up the structure of the Buxco experiment. We suggest setting the number of measurments output to be every 2 seconds as opposed to every 150 events during this experimental set-up. Experimental treatments (e.g. viral infection) can alter respiratory function and animal activity, which impacts the number of extracted datapoints if events are used, as opposed to time. Following completion of the experiment, data can be extracted as a .csv or .txt file from the tool menu in Finepointe. Depending on whether the user included gas analysis or other measures during their study, the user might need to trim some header lines from the extracted file before being useable in *plethy*.

*bottomly@ohsu.edu

3 Description of the Buxco file format

The parsing code contained within this package is designed to handle Buxco files which contain data of the form:

`Time,Subject,Phase,Recording,f,TVb,MVb,Penh,PAU,Rpef,Comp,PIFb,PEFb,Ti,Te,EF50,Tr,Tbody,Tc,RH,Rinx`

Which is further organized into subsections containing either experimental or acclimation data based on the following pattern:

`table.delim` table name
header line
acclimation (ACC) for animal 1
`burn.in.lines`
acclimation (ACC) for animal 2
experimental readings (EXP) for animal 1
`burn.in.lines`
acclimation (ACC) for animal 3
experimental readings (EXP for animal 2

Where the `indicated` lines correspond to default parameters in the current implementation of the `parse.buxco` function. Unless the file format changes, these should not have to be modified.

4 An initial example

To begin we will parse the example file 'BuxcoR_sample.csv' into an SQLite database. Note that the `chunk.size` parameter controls the number of lines read in at a given time. Although this can be useful for limiting the memory consumption of the program at the expense of runtime, there is a limit to its benefit as the data for several entire file sections are currently held in memory at a given time.

```
> file.name <- buxco.sample.data.path()
> chunk.size <- 500
> db.name <-  file.path(tempdir(), "bux_test.db")
> parse.buxco(file.name=file.name, chunk.size=chunk.size, db.name=db.name, verbose=FALSE)
```

```
BuxcoDB object:
Database: /tmp/RtmpInllX0/bux_test.db
Annotation Table: Additional_labels
| PARSE_DATE: 2016-05-03 21:57:47
| DBSCHEMA: Buxco
| package: plethy
| Db type: BuxcoDB
| DBSCHEMAVERION: 1.0
```

```
>
```

The resulting database (bux_test.db) can be accessed conveniently through an S4 object oriented interface where a BuxcoDb object can be create and accessed through several defined methods. For instance the defined samples, variables and tables can be retrieved from the database as follows:

```
> bux.db <- makeBuxcoDB(db.name=db.name)
> samples(bux.db)
```

```
[1] "8034x13140_5"  "8034x13140_11" "8034x13140_2"  "8034x13140_3"  "8034x13140_1"
[6] "8034x13140_9"  "8034x13140_4"  "8034x13140_10"
```

```
> variables(bux.db)
```

```
 [1] "f"     "TVb"   "MVb"   "Penh"  "PAU"   "Rpef"  "Comp"  "PIFb"  "PEFb"  "Ti"
[11] "Te"    "EF50"  "Tr"    "Tbody" "Tc"    "RH"    "Rinx"
```

```
> tables(bux.db)
```

```
[1] "WBPth"
```

```
>
```

Also, we can retrieve data for analysis. This is done most conveniently through the `retrieveData` method. This method without any arguments will retrieve all the data and return it in a *data.frame*. This can take a lot of memory and will be a little slower than specifying the data subsets the user is interested in. By passing in character or numeric variables as arguments to the method as shown below then smaller subsets of the data can be retrieved quickly and then be used for downstream statistical or graphical analysis.

```
> data.1 <- retrieveData(bux.db, samples="8034x13140_1", variables="Penh")
> head(data.1)
```

```
   Sample_Name              P_Time Break_sec_start Variable_Name Bux_table_Name
1 8034x13140_1 2012-09-28 10:07:17               0          Penh          WBPth
2 8034x13140_1 2012-09-28 10:07:19               0          Penh          WBPth
3 8034x13140_1 2012-09-28 10:07:21               2          Penh          WBPth
4 8034x13140_1 2012-09-28 10:07:23               4          Penh          WBPth
5 8034x13140_1 2012-09-28 10:07:25               6          Penh          WBPth
6 8034x13140_1 2012-09-28 10:07:27               8          Penh          WBPth
  Rec_Exp_date Break_number     Value
1          D-3            5 0.7851210
2          D-3            6 0.7239651
3          D-3            6 0.8044858
4          D-3            6 0.5304968
5          D-3            6 0.4664128
6          D-3            6 0.7347313
```

```
>
```

values can also be specified as vectors

```
> data.2 <- retrieveData(bux.db, samples=c("8034x13140_1", "8034x13140_10"), variables=c("Penh", "f"))
> head(data.2)
```

```
   Sample_Name              P_Time Break_sec_start Variable_Name Bux_table_Name
1 8034x13140_1 2012-09-28 10:07:17               0             f          WBPth
2 8034x13140_1 2012-09-28 10:07:17               0          Penh          WBPth
3 8034x13140_1 2012-09-28 10:07:19               0             f          WBPth
4 8034x13140_1 2012-09-28 10:07:19               0          Penh          WBPth
5 8034x13140_1 2012-09-28 10:07:21               2             f          WBPth
6 8034x13140_1 2012-09-28 10:07:21               2          Penh          WBPth
  Rec_Exp_date Break_number       Value
1          D-3            5 598.2654419
2          D-3            5   0.7851210
3          D-3            6 582.1941528
4          D-3            6   0.7239651
5          D-3            6 606.6564941
6          D-3            6   0.8044858
```

```
> table(data.1$Sample_Name, data.1$Variable_Name)
```

```
               Penh
  8034x13140_1  151
```

```
>
```

Note that at this point the data in the database consists of values directly parsed from the raw file. Adding additional labels or computations can be done through the `addAnnotation` method by specifying the *BuxcoDB* object and a function returning the SQL query. There are currently two functions that are defined, though users can define their own as well with some familiarity of the database structure. The first, `day.infer.query` computes the number of days past the first measurement for a given animal. The second `break.type.query` labels the measurements as 'ACC' for acclimation, 'EXP' for experimental, 'UNK' for unknown (where there is only one set of measurements for a given animal and given time point) or 'ERR' which likely represents an error in the parsing or query. Examples are shown below.

```
> addAnnotation(bux.db, query=day.infer.query, index=FALSE)

[1] TRUE

> addAnnotation(bux.db, query=break.type.query, index=TRUE)

[1] TRUE

>
```

The index argument specifies whether an index should be added to the columns. There should not really be any harm in adding them at each stage though as the method tries to create a set of covering indices to maximize retrieval speed, I would recommend waiting until the last call to `addAnnotation`. Again we can extract the data as before using `retrieveData` though this time in order to specify constraints on any column defined through `addAnnotation` we need to type in the column name and the constraint vector at the end.

For this it is useful to figure out the names and different values of the annotation columns.

```
> annoCols(bux.db)

[1] "Days"             "Break_type_label"

> annoLevels(bux.db)

$Days
[1] 0

$Break_type_label
[1] "ACC" "EXP"

>
```

First we can get data as before and examine its structure.

```
> data.3 <- retrieveData(bux.db, samples="8034x13140_2", variables="Penh")
> with(data.3, table(Days, Break_type_label))

    Break_type_label
Days ACC EXP
   0   1 150

>
```

Similar to above but placing constraints on the annotation columns.

```
> data.4 <- retrieveData(bux.db, samples="8034x13140_2", variables="Penh", Days = 0)
> with(data.4, table(Days, Break_type_label))

    Break_type_label
Days ACC EXP
   0   1 150

> data.5 <- retrieveData(bux.db, samples="8034x13140_2", variables="Penh", Days = 0,
+         Break_type_label = 'EXP')
> with(data.5, table(Days, Break_type_label))
```

```
    Break_type_label
Days EXP
   0 150
```

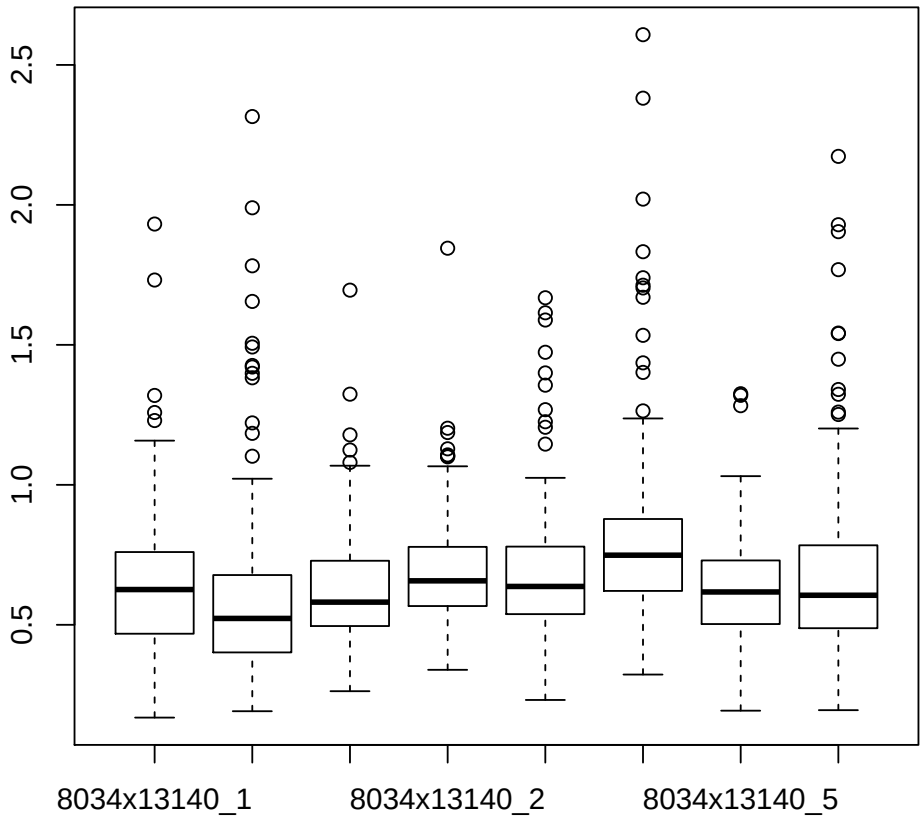
```
>
```

Putting it all together we can easily perform descriptive analyses comparing the different samples using the enhanced pause (Penh) phenotype.

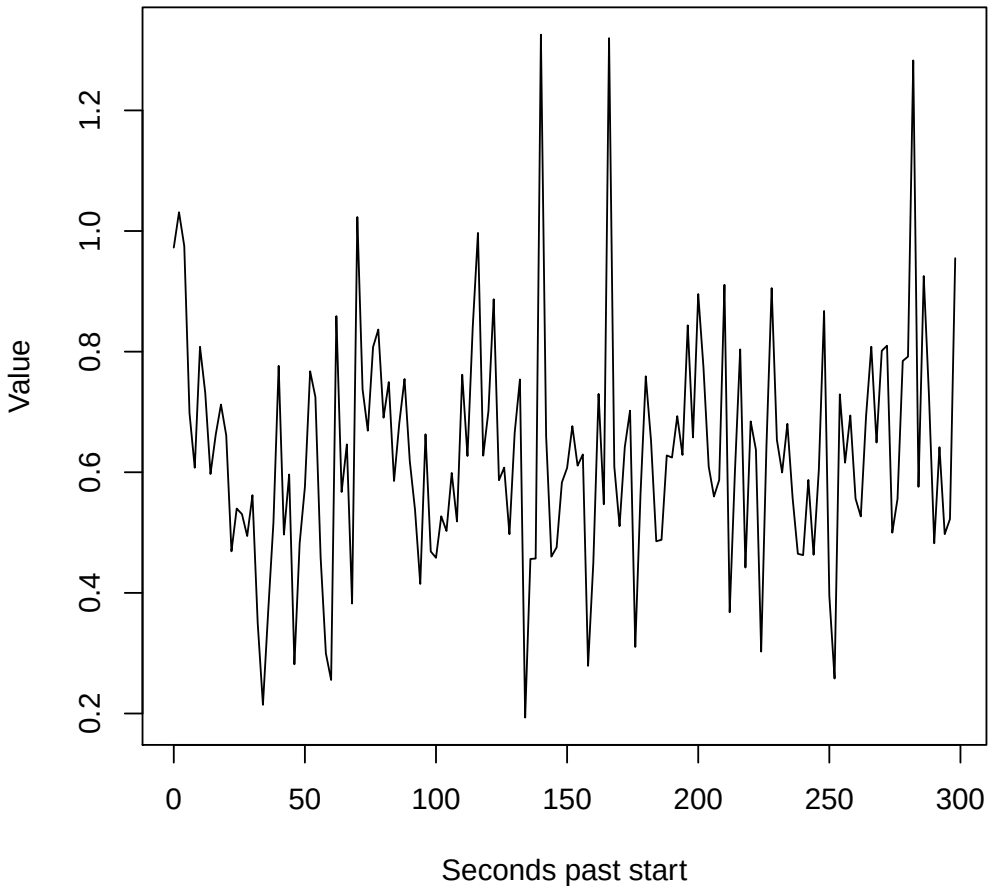
```
> exp.penh <- retrieveData(bux.db, variables="Penh", Break_type_label = 'EXP')
> head(exp.penh)
```

```
   Sample_Name              P_Time Break_sec_start Variable_Name Bux_table_Name
1 8034x13140_5 2012-09-28 10:07:19               0          Penh          WBPth
2 8034x13140_5 2012-09-28 10:07:21               2          Penh          WBPth
3 8034x13140_5 2012-09-28 10:07:23               4          Penh          WBPth
4 8034x13140_5 2012-09-28 10:07:25               6          Penh          WBPth
5 8034x13140_5 2012-09-28 10:07:27               8          Penh          WBPth
6 8034x13140_5 2012-09-28 10:07:29              10          Penh          WBPth
  Rec_Exp_date Break_number Days Break_type_label     Value
1          D-3            2    0              EXP 0.9725345
2          D-3            2    0              EXP 1.0310636
3          D-3            2    0              EXP 0.9750971
4          D-3            2    0              EXP 0.6979190
5          D-3            2    0              EXP 0.6075242
6          D-3            2    0              EXP 0.8080077
```

```
> boxplot(Value~Sample_Name, data=exp.penh)
>
```




```
> plot(Value~Break_sec_start, data=exp.penh, subset=Sample_Name=="8034x13140_5", type="l",
+         xlab="Seconds past start")
>
```


Value
0.2
0.4
0.6
0.8
1.0
1.2
0
50
100
150
200
250
300
Seconds past start