Examples With Peak Finder Script

Michael Olvera

September 8, 2016

Using Peak Finder

This is a quick guide on how to use the Peak Finder script. Feel free to follow along with the code in this file to make sure everything is working well. For this file, R code is in boxes, and outputs are boxes with ## before the code.

Before starting, include the following line of code at the top of your script:

```
source("R/peakFinder.R")
```

Loading data straight from the microscope.

Data from a Ziess experiment incudes several metrics we are not interested in, as well as artifacts introduced by loading the data into R. Here is an example of raw data from the Ziess:

```
raw_data <- read.csv(file = "./data/CMEC_75.25-1_5.16.17_1.csv")
colnames(raw_data) <- NULL
head(raw_data)
```

```
##
## 1          S NA       NA       NA     NA  NA       NA       NA     NA  NA
## 2  20.205059 NA 3475.907 110.5332 909357 142 3478.865 110.8166 912464 144
## 3 20.2150665 NA 3475.907 110.5836 909771 139 3478.865 110.8251 912534 144
## 4 20.2250739 NA 3475.907 110.5225 909269 138 3478.865 110.7590 911990 141
## 5 20.2350814 NA 3475.907 110.4265 908479 142 3478.865 110.7340 911784 143
## 6 20.2450889 NA 3475.907 110.4226 908447 143 3478.865 110.6909 911429 142
##
## 1       NA       NA     NA  NA       NA       NA     NA  NA       NA
## 2 3482.245 110.7084 912459 141 3474.64 113.2736 931562 154 3476.753
## 3 3482.245 110.6740 912175 141 3474.64 113.3372 932085 144 3476.753
## 4 3482.245 110.6879 912290 134 3474.64 113.3134 931889 141 3476.753
## 5 3482.245 110.4661 910462 135 3474.64 113.2645 931487 142 3476.753
## 6 3482.245 110.5500 911153 138 3474.64 113.2422 931304 142 3476.753
##
## 1       NA     NA  NA       NA       NA     NA  NA      NA       NA     NA
## 2 110.8344 912056 135 3483.935 110.6613 912513 144 3476.33 109.5576 901440
## 3 110.9593 913084 138 3483.935 110.7195 912993 141 3476.33 109.5965 901760
## 4 110.8493 912179 139 3483.935 110.6398 912336 144 3476.33 109.4147 900264
## 5 110.7295 911193 134 3483.935 110.6179 912155 142 3476.33 109.4533 900582
## 6 110.7790 911600 152 3483.935 110.6139 912122 160 3476.33 109.3927 900083
##
## 1  NA       NA       NA     NA  NA       NA       NA     NA  NA
## 2 137 3472.105 111.6027 917151 144 3476.753 109.7079 902786 145
## 3 144 3472.105 111.5426 916657 135 3476.753 109.5897 901814 141
## 4 138 3472.105 111.4501 915897 139 3476.753 109.6593 902386 137
## 5 142 3472.105 111.6452 917500 137 3476.753 109.6029 901922 147
## 6 137 3472.105 111.4326 915753 139 3476.753 109.5609 901577 136
```

PeakFinder has a build in function to automatically clean the data, leaving only the mean intensity values.

```
raw_data <- readZiessData(path_to_csv = "./data/CMEC_75.25-1_5.16.17_1.csv", time_index = 1, grep_keywo
colnames(raw_data) <- NULL
head(raw_data)
```

```
##
## 2 20.20506 110.5332 110.8166 110.7084 113.2736 110.8344 110.6613 109.5576
## 3 20.21507 110.5836 110.8251 110.6740 113.3372 110.9593 110.7195 109.5965
## 4 20.22507 110.5225 110.7590 110.6879 113.3134 110.8493 110.6398 109.4147
## 5 20.23508 110.4265 110.7340 110.4661 113.2645 110.7295 110.6179 109.4533
## 6 20.24509 110.4226 110.6909 110.5500 113.2422 110.7790 110.6139 109.3927
## 7 20.25510 110.4937 110.6689 110.5588 113.0637 110.9266 110.6927 109.4358
##
## 2 111.6027 109.7079
## 3 111.5426 109.5897
## 4 111.4501 109.6593
## 5 111.6452 109.6029
## 6 111.4326 109.5609
## 7 111.5001 109.5156
```

Note that I am nullifying the column names for printing purposes only. You do not nulify column names normally, and you are recommended to change column names after running *readZiessData()*.

Quick Introduction

For simplicity's sake, I will be loading in another preprocessed dataframe. Samples must be preprocessed to be a dataframe with **Time** included, and all other columns the mean intensity values of samples. A preprocessed dataset is included in the zip folder:

```
example <- read.csv('data/CaHandUT1.csv')
head(example) # Prints first 6 rows
```

```
##      Time       R1       R2       R3       R4       R5       R6       R7
## 1  0.0000 187.6669 387.1738 176.6555 168.4455 167.4844 179.7831 184.2782
## 2 10.0075 187.4253 387.0821 176.1735 170.1468 169.4760 180.8299 184.5747
## 3 20.0149 186.4995 387.4237 176.3546 170.1221 169.2472 181.1079 184.0144
## 4 30.0224 186.3735 386.6219 175.9589 170.0625 169.4075 180.7528 184.4155
## 5 40.0298 185.2577 386.7345 176.1445 170.1175 169.1206 180.9563 183.9717
## 6 50.0373 185.3023 385.3170 175.7450 169.8538 169.0676 180.4453 184.2486
##         R8       R9      R10      R11      R12      R13      R14
## 1 351.8155 184.4244 340.8282 176.8191 213.6833 184.8174 182.6327
## 2 352.0044 184.4453 341.1342 178.9146 212.3781 185.0817 181.5208
## 3 353.9073 184.8587 340.2362 178.8213 211.2374 184.4887 181.5315
## 4 353.3250 184.4861 337.6030 178.8120 210.5008 184.8378 180.6417
## 5 355.3002 184.7740 336.4125 178.6491 209.4161 184.0433 180.7844
## 6 354.0300 184.2582 333.7079 178.8892 208.3467 184.4472 180.0346
```

As a side note, make sure all of your comlumes are numeric/integers. A common mistake with R is that .csv files are loaded with columns being factors.

```
str(example)
```

```
## 'data.frame':    1967 obs. of  15 variables:
##  $ Time: num  0 10 20 30 40 ...
##  $ R1  : num  188 187 186 186 185 ...
```

```
##  $ R2  : num  387 387 387 387 387 ...
##  $ R3  : num  177 176 176 176 176 ...
##  $ R4  : num  168 170 170 170 170 ...
##  $ R5  : num  167 169 169 169 169 ...
##  $ R6  : num  180 181 181 181 181 ...
##  $ R7  : num  184 185 184 184 184 ...
##  $ R8  : num  352 352 354 353 355 ...
##  $ R9  : num  184 184 185 184 185 ...
##  $ R10 : num  341 341 340 338 336 ...
##  $ R11 : num  177 179 179 179 179 ...
##  $ R12 : num  214 212 211 211 209 ...
##  $ R13 : num  185 185 184 185 184 ...
##  $ R14 : num  183 182 182 181 181 ...
```

As you can see above, all columns are of **type** numeric (**num**). Looking a the second row of the output, *$ Time* indicates the title of the column, then the datatype, followed by the first few data entries.

To analyze data, I have written a function to wrap the data in an object called *dataframeToExperiment()*. The 'experiment' object will hold all the intensity values, the time measurments, and all the computed readings.

```
example.object <- dataframeToExperiment(dataframe =  example, timeIndex = 1)
```

```
## No timing errors detected.
## Caution: Erronious intensity detected at index 1025.
## Attempting fix.
## Patching Sucessfull!
```

```
print(example.object)
```

```
## Data names : R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14
## Experimental object with 1967 measurments and 14 positions.
```

When the object is created, it will run a full test to try and detect errors in the data. Some errors will be patched automatically, while others might have to be excluded from further analysis.

Analysis

The main functions are *analyzeExperiemnt()* and *runTestGraph()*. You can call *analyzerExperiment()* on you experiment object to get physiologically relivant data, including: - Beats per minute - Amplitute - Upstroke and Downstroke T50 - Vmax (Up and Decay)

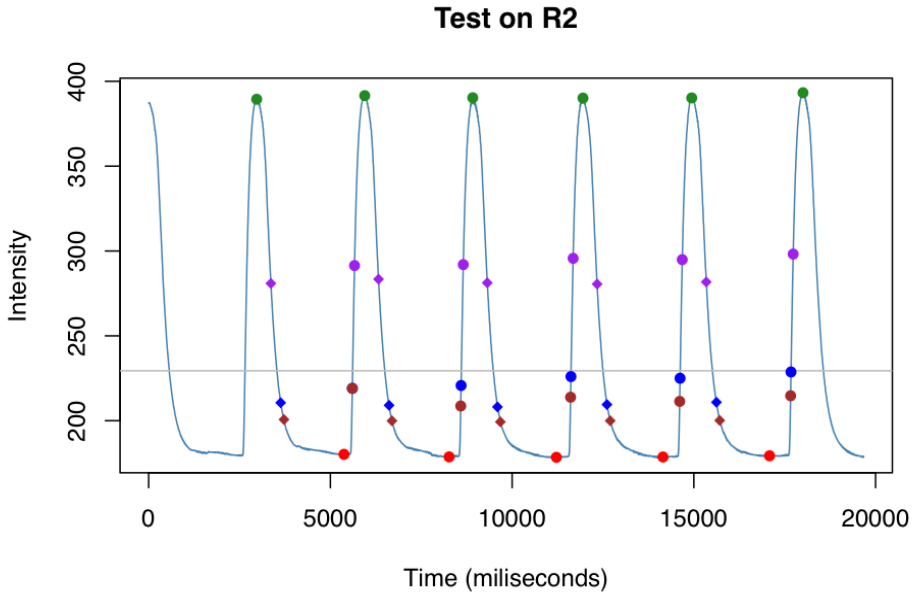
```
example.object <- analyzeExperiment(example.object)
head(example.object$results)
```

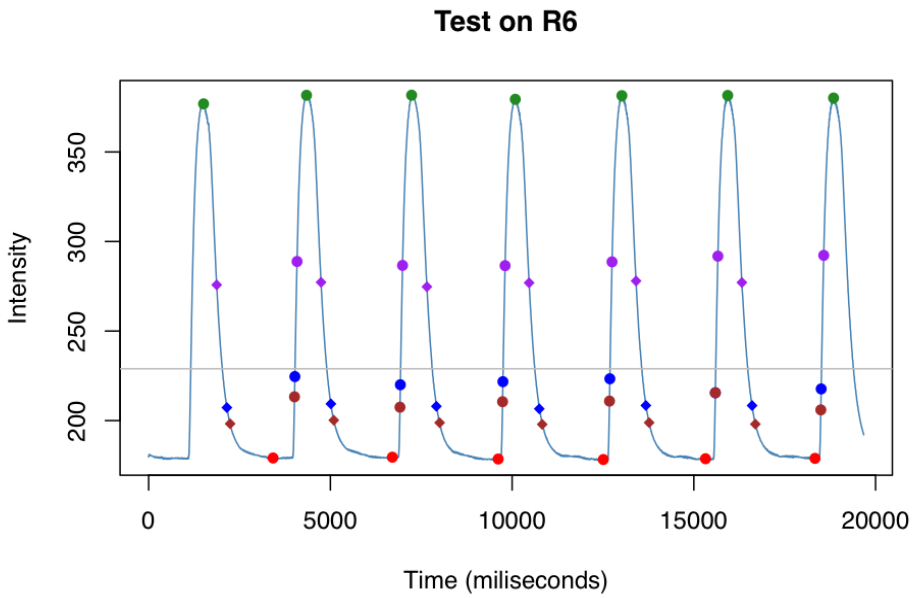
```
##   Peak (AU) Min (AU) F/Fn (Amplitude) Upstroke T50 (ms)
## 1   354.945  175.074            2.027           256.858
## 2   390.802  179.050            2.183           268.200
## 3   344.151  173.446            1.984           256.858
## 4   305.698  168.554            1.814           261.862
## 5   299.784  166.658            1.799           258.526
## 6   380.394  178.854            2.127           266.866
##   Downstroke T50 (ms) Vmax Up Vmax Decay    BPM
## 1             398.630   1.032     -0.274 25.116
## 2             392.292   1.173     -0.345 23.950
## 3             418.645   0.939     -0.286 24.372
## 4             405.302   0.797     -0.217 24.372
## 5             386.955   0.725     -0.176 24.372
```

```
## 6             390.291   1.138     -0.363 24.231
```

Individual readings can be called to a ploting funciton, for closer inspection of where the progrmaing is calling peaks, minimum values, etc. *runTestGraph()* can be called on the names of individual reading, or on the whole dataset (if the second argument is left blank).

```
runTestGraph(example.object, c("R2", "R6"))
```





The output can be saved and extracted to be used in Excel/Prism.

```
write.csv(example.object$results, "~/Desktop/results.csv")
```

If you have any questions or bugs, please let me know (michael.olvera@gladstone.ucsf.edu).

```
sessionInfo()
```

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.11.3 (El Capitan)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] backports_1.0.5 magrittr_1.5    rprojroot_1.2   tools_3.3.1
##  [5] htmltools_0.3.6 yaml_2.1.14     Rcpp_0.12.10    stringi_1.1.5
##  [9] rmarkdown_1.5   knitr_1.15.1    stringr_1.2.0   digest_0.6.12
## [13] evaluate_0.10
```