HRV analysis

Janice Tjeng

2018-01-03

Load packages

```
library(RMySQL)
library(tidyverse)
library(lubridate) # for manipulating date/time objects
library(robustbase) # for robust covariance estimate
library(stringr) # for manipulating regular expressions
library(Rssa) # for singular spectrum analysis
```

Read data from E4, firstbeat, & MSband

```
db <- dbConnect(MySQL(), user="", password="", dbname="", host="") # contact
for access to data
e4 <- dbSendQuery(db,"SELECT * FROM viewl_e4_rr")
e4dat <- fetch(e4,n=-1)
fb <- dbSendQuery(db, "SELECT * FROM viewl_firstbeat_rr")
fbdat <- fetch(fb, n=-1)
mb <- dbSendQuery(db, "SELECT * FROM viewl_msband_rr")
mbdat <- fetch(mb, n=-1)
```

Read timing data

```
timing <- read_csv("timing_data.csv")
df <- timing %>%
  select (Event, LWP2_0019, Activity) %>%
  rename("Time"="LWP2_0019")
n <- nrow(df)
dte <- mdy(as.character(df[1,"Time"]))
start <- as.character(df[2,"Time"])
end <- as.character(df[n,"Time"])
```

Select rows for user 19 from E4

```
user19_l <- e4dat %>%
  filter(user=="lwp2_0019", device_location=="Left") %>%
  mutate(readable_timestamp=as.POSIXct(readable_timestamp),
         e4_rr=e4_rr*1000)
filter_time_day1_e4 <- user19_l %>%
  filter(strftime(readable_timestamp, "%H:%M:%S") >= start & strftime(readabl
e_timestamp, "%H:%M:%S") <= end, date(readable_timestamp)==dte)
```

Select rows for user 19 from ECG

```
user19_fb <- fbdat %>%
  filter(user=="lwp2_0019") %>%
  mutate(readable_timestamp=as.POSIXct(readable_timestamp))
filter_time_day1_fb <- user19_fb %>%
  filter(strftime(readable_timestamp, "%H:%M:%S") >= start & strftime(readabl
e_timestamp, "%H:%M:%S") <= end, date(readable_timestamp)==dte)
```

Select rows for user 19 from MSband

```
user19_l_mb <- mbdat %>%
  filter(user=="lwp2_0019", device_location=="Left") %>%
  mutate(readable_timestamp=as.POSIXct(readable_timestamp),
         mb_rr=(mb_rr*1000))
filter_time_day1_mb <- user19_l_mb %>%
  filter(strftime(readable_timestamp, "%H:%M:%S") >= start & strftime(readabl
e_timestamp, "%H:%M:%S") <= end, date(readable_timestamp)==dte)
```

Clean data

Detect outlier, short RR-interval, based on z-score standardization.

```
# Standardize function. Standard deviation based on robust covariance estimat
e
standardize <- function(rr){
  return ((rr-mean(rr))/sqrt(covMcd(rr)$cov[1])) # robust covariance estimate
from covMcd function, robustbase package
}

# Function to identify outlier
outlier_short <- function(rr){
  z <- standardize(rr)
  ind <- which(z<(-3)) # -3: limit in sd for outlier detection of short rr in
terval
  return (ind)
}

# Function to remove outlier, then add to subsequent beat
rm_short <- function(x,y){ # x is the index of outliers, y is the column of r
r-interval
  for (i in 1:length(x)){
    if (x[i]<length(y)){
      y[x[i]+1] <- y[x[i]+1] + y[x[i]]
      y[x[i]] <- NA
    }
    else{  # if rr-interval is at the end of the array, just remove it withou
t adding it to the next beat
      y[x[i]] <- NA
```

```
    }
  }
  return (y)
}

# Combine all functions
rm_short_comp <- function (rr){
  indx <- outlier_short(rr)
  clean <- rm_short(indx,rr)
  return (clean)
}
```

Impute long rr-interval, applied after removing short

```
# Function to impute long rr-intervals
impute_long <- function(rr){
  dmax <- mean(rr) + (2*sqrt(covMcd(rr)$cov[1]))
  dp <- numeric()
  dp1 <- numeric()
  ind_long <- which(rr>dmax) # Index of long outliers
  for (i in 1:length(ind_long)){     # imputation if rr>dmax
    dp <- sum((rr[ind_long][i]-dmax), dp)
  }
  dp <- dp/length(ind_long)          # imputation if rr<=dmax
  rr[ind_long] <- rr[ind_long]-dp
  for (i in 1:length(rr[-ind_long])){
    dp1 <- sum((rr[-ind_long][i]-dmax), dp1)
  }
  dp1 <- dp1/length(rr[-ind_long])
  rr[-ind_long] <- rr[-ind_long]-dp1
  return (rr)
}

# Iterate each rr-interval and expand window. Apply impute_long function when
average of window is less than or equal to dmax
window_impute_long <- function(rr){
  k <- 1
  dmax <- mean(rr) + (2*sqrt(covMcd(rr)$cov[1]))
  for (i in 2:length(rr)){
    if ((mean(rr[i-k]:rr[i+k]))<=dmax){
      return (impute_long(rr))
    }
    else {
      k <- k+1
    }
  }
}
```

SSA cleaning based on the following links:

https://www.r-bloggers.com/wheres-the-magic-emd-and-ssa-in-r/

https://www.researchgate.net/publication/228092069_Basic_Singular_Spectrum_Analysis_and_Forecasting_with_R

Function to use reconstructed value is residuals > 0.5

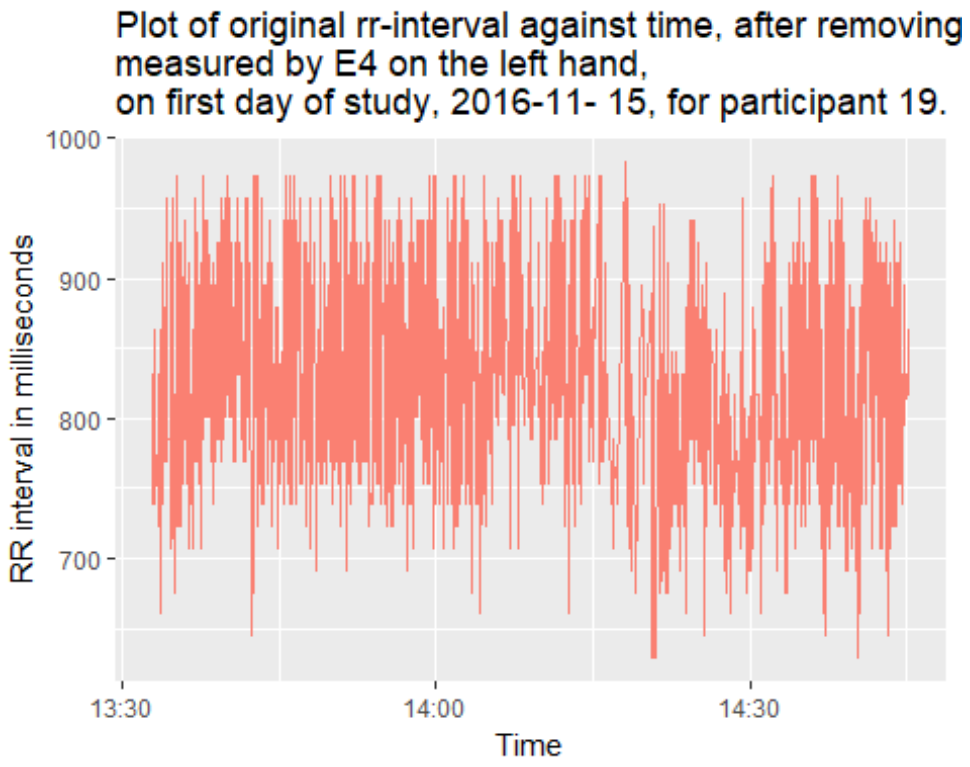
```
reconstructed_rr <- function(residuals, reconstruct, original){
  int <- seq(1, length(residuals), 1)
  indx.res <- which(residuals>0.5)
  indx.or <- int[!int %in% indx.res]
  res <- data.frame(Index=indx.res, Label="Reconstruct")
  or <- data.frame(Index=indx.or, Label="Original")
  for (i in 1:length(indx.res)){
    res$RR[i] <- reconstruct[indx.res[i]]
  }
  for (i in 1:length(indx.or)){
    or$RR[i] <- original[indx.or[i]]
  }
  df <- rbind(res,or) %>%
    arrange(Index) %>%
    select(-Index)
  return (df)
}
```

Apply algorithms to clean rr-interval, then apply SSA, for each device

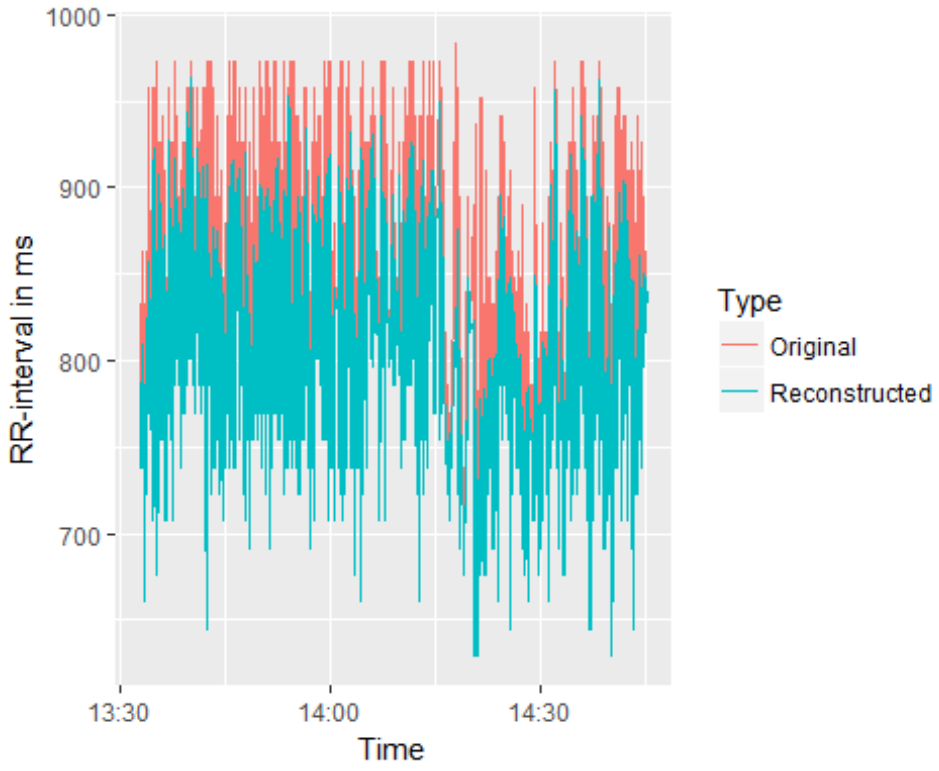
E4

```
filter_time_day1_e4.clean <- filter_time_day1_e4 %>%
  mutate(e4_rr=rm_short_comp(e4_rr)) %>%
  filter(!is.na(e4_rr)) %>%
  mutate(e4_rr=window_impute_long(e4_rr))

# SSA: 1st stage decomposition
e4.ssa <- ssa(L=15, x=filter_time_day1_e4.clean$e4_rr, kind="toeplitz-ssa", s
vd.method = "svd") # window, L=15
# summary(e4.ssa)

# 1st stage visual information for grouping
# plot(e4.ssa) # scree plot of singular values to identify trends and pairing
s

# 1st stage reconstruction
res1 <- reconstruct(e4.ssa, groups=list(Tr=1:4)) # choose first 4 trend/compo
nents to produce Poncaire's plot
res.trend <- residuals(res1) # Extract residuals, original values-predicted v
alues
```

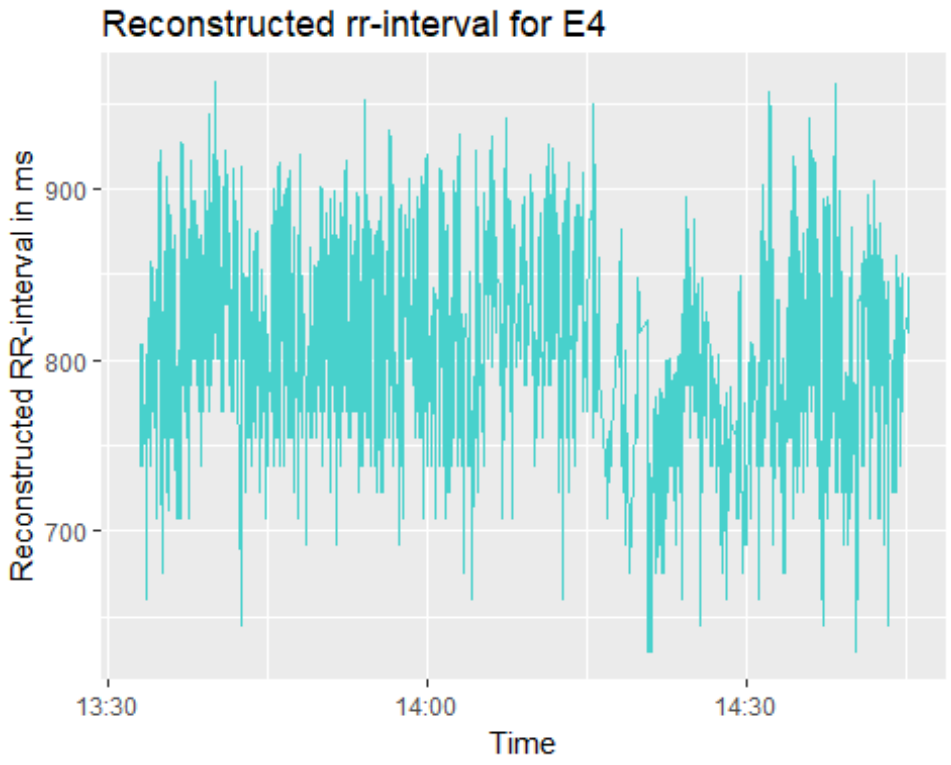
```
# spec.pgram(res.trend, detrend=F, log="no") # Periodogram, detect seasonalit
y. Unit of x-axis in Hz,y-axis represents power spectral density

# Apply formula that uses reconstructed value if residuals>0.5
e4_reconstructed <- reconstructed_rr(res.trend, res1$Tr, filter_time_day1_e4.
clean$e4_rr)

# Original plot
ggplot(filter_time_day1_e4.clean, aes(x=readable_timestamp, y=e4_rr)) + geom_
line(color="salmon") +
  ggtitle("Plot of original rr-interval against time, after removing outliers
, \nmeasured by E4 on the left hand, \non first day of study, 2016-11- 15, fo
r participant 19.") +
  ylab("RR interval in milliseconds") +
  xlab("Time")
```




```
df_e4 <- data.frame(Time=filter_time_day1_e4.clean$readable_timestamp, Origin
al=filter_time_day1_e4.clean$e4_rr, Reconstructed=e4_reconstructed$RR)

# Reconstructed and original plot combined
df_e4 %>%
  gather(Reconstructed, Original, key="Type", value="RR-interval") %>%
  ggplot(aes(x=Time, y=`RR-interval`)) +
  geom_line(aes(colour=Type)) +
  ylab("RR-interval in ms")
```



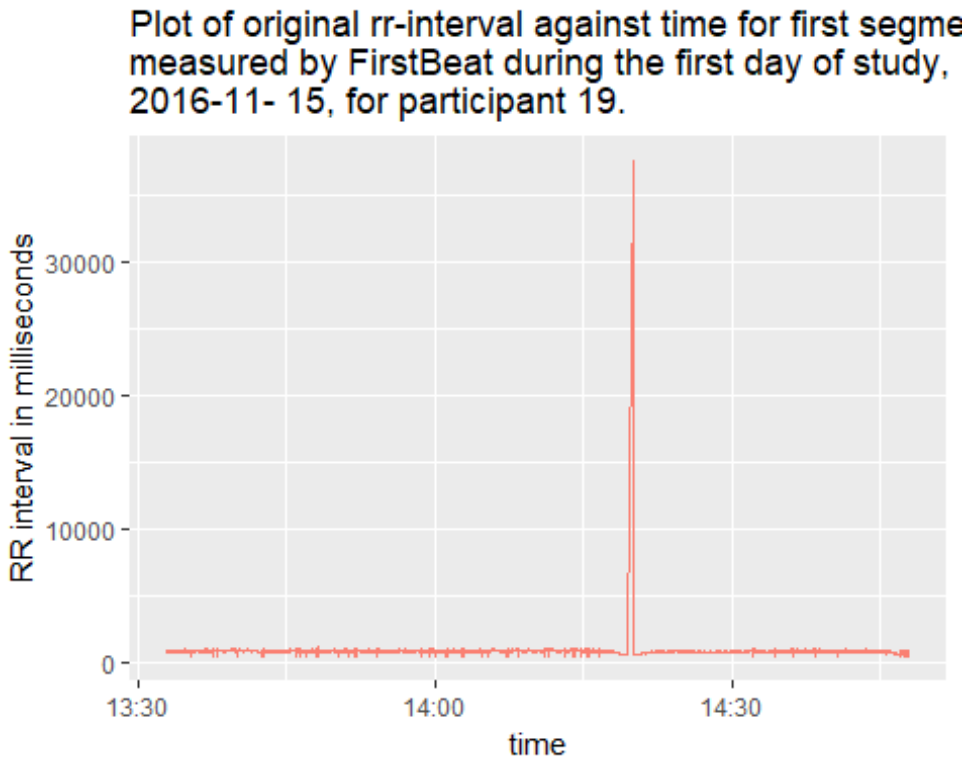

```
# Reconstructed plot
ggplot(df_e4, aes(x=Time, y=Reconstructed)) + geom_line(color="mediumturquois
e") + ylab("Reconstructed RR-interval in ms") + ggtitle("Reconstructed rr-int
erval for E4")
```



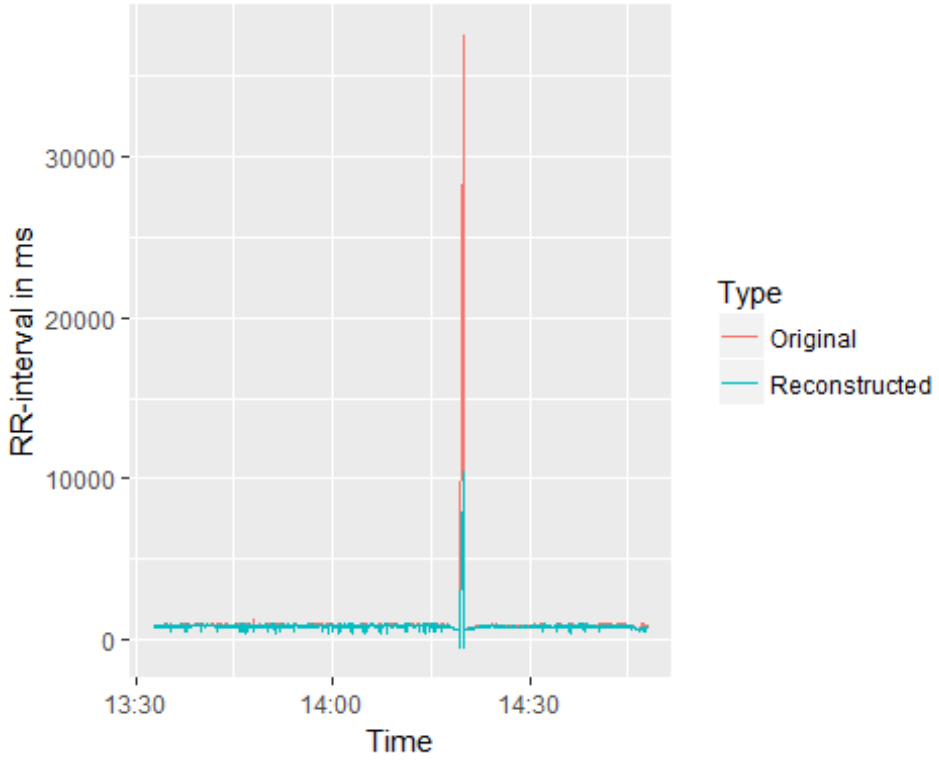
ECG/FirstBeat

```
filter_time_day1_fb.clean <- filter_time_day1_fb %>%
  mutate(fb_rr=rm_short_comp(fb_rr)) %>%
  filter(!is.na(fb_rr)) %>%
  mutate(fb_rr=window_impute_long(fb_rr))

fb.ssa <- ssa(L=15, x=filter_time_day1_fb.clean$fb_rr,  kind="toeplitz-ssa",
svd.method = "svd")
# plot(fb.ssa)
fb_res <- reconstruct(fb.ssa, group=list(Tr=c(1:4)))
res.trend.fb <- residuals(fb_res) # Extract seasonality from residuals, origi
nal values-predicted values
# spec.pgram(res.trend.fb, detrend=F, log="no")

# Apply formula that uses reconstructed value if residuals>0.5
fb_reconstructed <- reconstructed_rr(res.trend.fb, fb_res$Tr, filter_time_day
1_fb.clean$fb_rr)

# Original Plot
ggplot(filter_time_day1_fb.clean, aes(x=readable_timestamp, y=fb_rr)) + geom_
line(color="salmon") +
  ylab("RR interval in ms") + xlab("Time") +
  ggtitle("Plot of original rr-interval against time for first segment, after
removing outliers, \nmeasured by FirstBeat during the first day of study, \n2
016-11- 15, for participant 19.") +
```

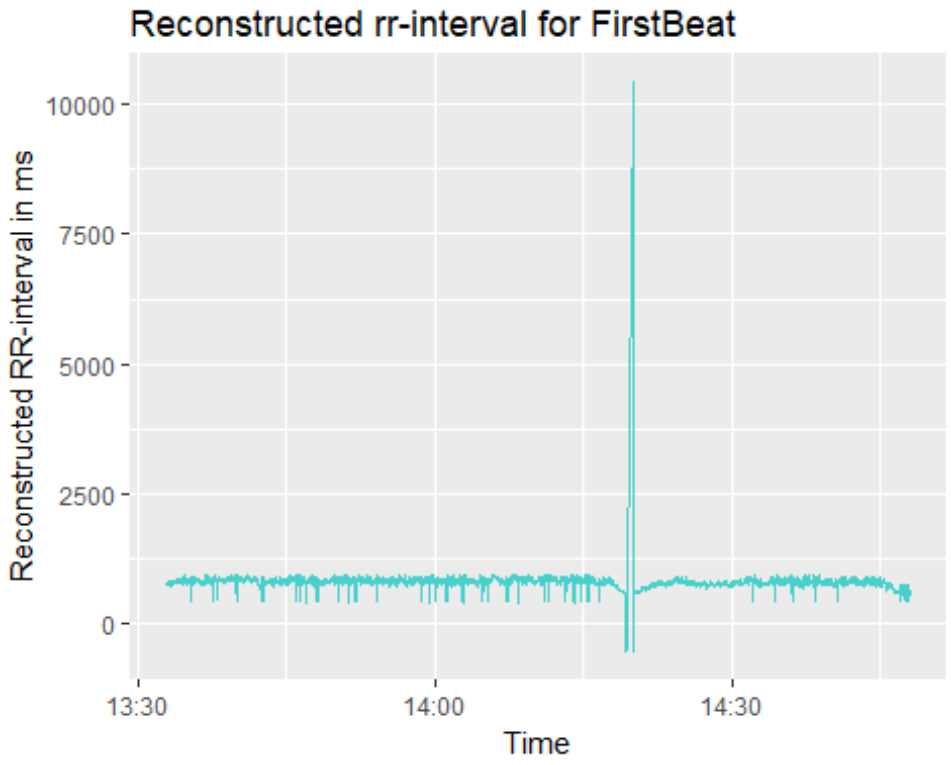
```
  ylab("RR interval in milliseconds") +
  xlab("time")
```




```
df_fb <- data.frame(Reconstructed=fb_reconstructed$RR, Original=filter_time_d
ay1_fb.clean$fb_rr, Time=filter_time_day1_fb.clean$readable_timestamp)

# Reconstructed and original plot combined
df_fb %>%
  gather(Reconstructed, Original, key="Type", value="RR-interval") %>%
  ggplot(aes(x=Time, y=`RR-interval`)) +
  geom_line(aes(colour=Type)) +
  ylab("RR-interval in ms")
```

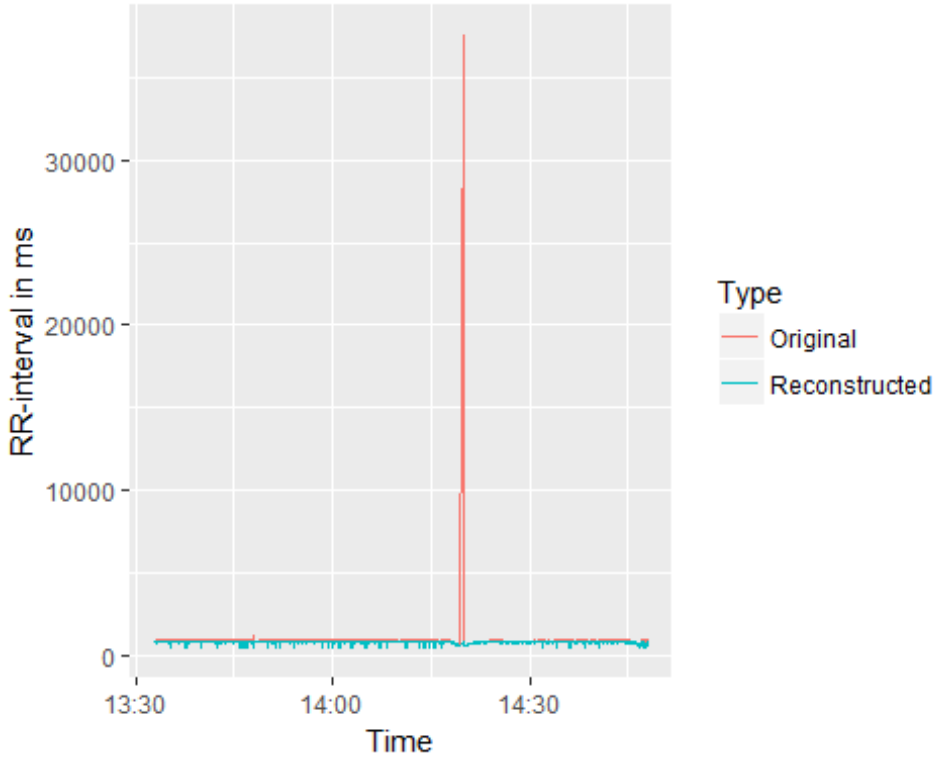



```
# Huge spike of ECG at, 14:19:58, >37,000ms. This is due to removal of previo
us short rr-intervals (rows 4081-4177) and adding it to the rr-interval at ro
w 4081.

# Reconstructed plot
ggplot(df_fb, aes(x=Time, y=Reconstructed)) + geom_line(colour="mediumturquoi
se") + ylab("Reconstructed RR-interval in ms") + xlab("Time") + ggtitle("Reco
nstructed rr-interval for FirstBeat")
```

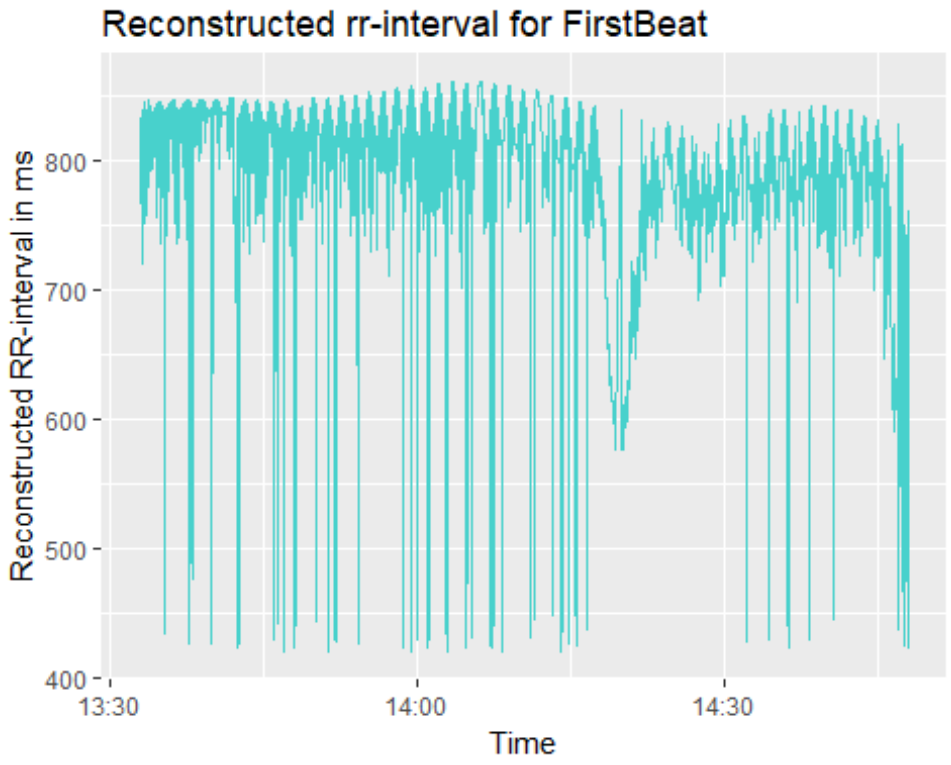


Instead of L=15, use L=N/2, to get rid of huge upward spike for FirstBeat.

```
fb.ssa1 <- ssa(x=filter_time_day1_fb.clean$fb_rr,  kind="toeplitz-ssa", svd.m
ethod = "svd")

fb_res1 <- reconstruct(fb.ssa1, group=list(Tr=1:4))
res.fb <- residuals(fb_res1)

# Apply formula that uses reconstructed value if residuals>0.5
fb_reconstructed1 <- reconstructed_rr(res.fb, fb_res1$Tr, filter_time_day1_fb
.clean$fb_rr)

df_fb1 <- data.frame(Reconstructed=fb_reconstructed1$RR, Original=filter_time
_day1_fb.clean$fb_rr, Time=filter_time_day1_fb.clean$readable_timestamp)

# Reconstructed and original plot combined
df_fb1 %>%
  gather(Reconstructed, Original, key="Type", value="RR-interval") %>%
  ggplot(aes(x=Time, y=`RR-interval`)) +
  geom_line(aes(colour=Type)) +
  ylab("RR-interval in ms")
```

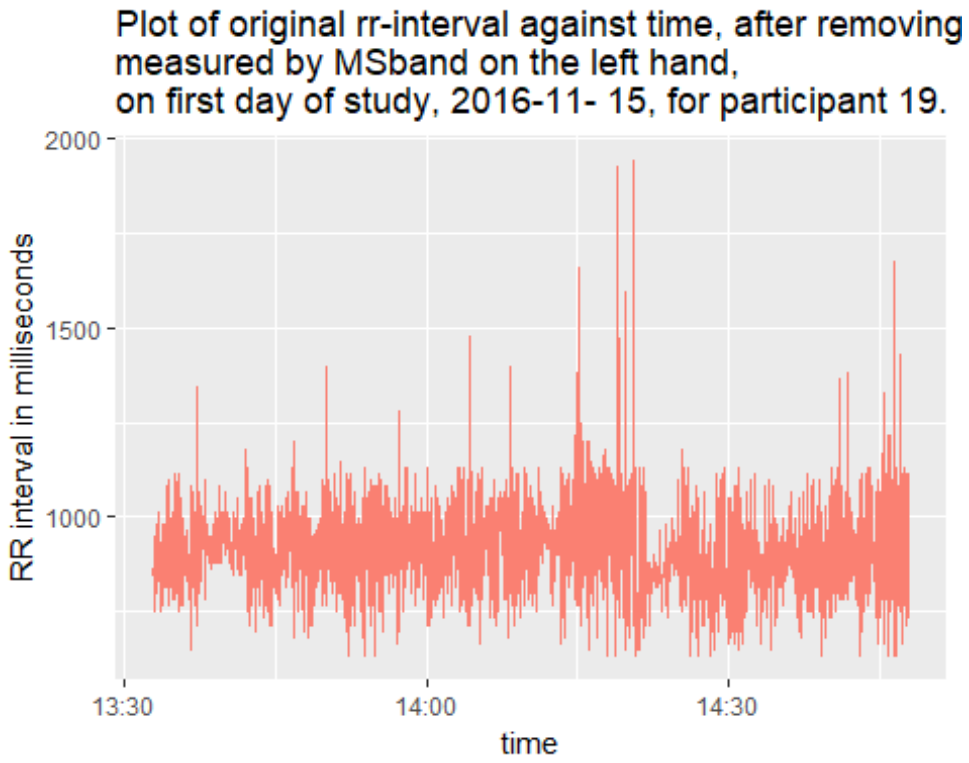



```
# Reconstructed plot
ggplot(df_fb1, aes(x=Time, y=Reconstructed)) + geom_line(colour="mediumturquo
ise") + ylab("Reconstructed RR-interval in ms") + xlab("Time") + ggtitle("Rec
onstructed rr-interval for FirstBeat")
```

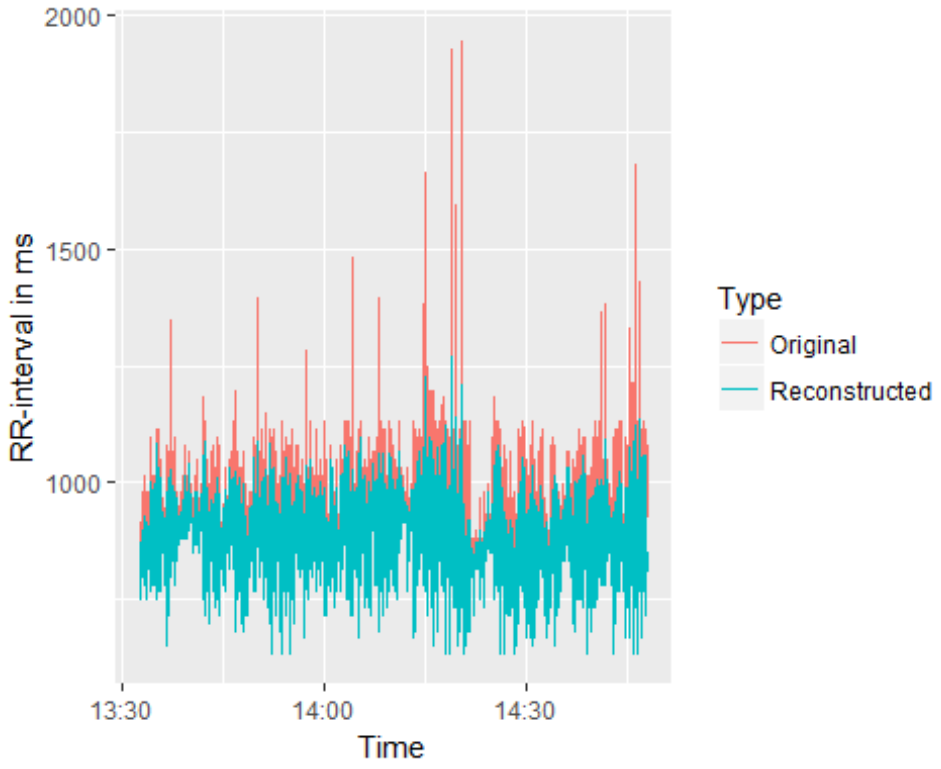


MSband

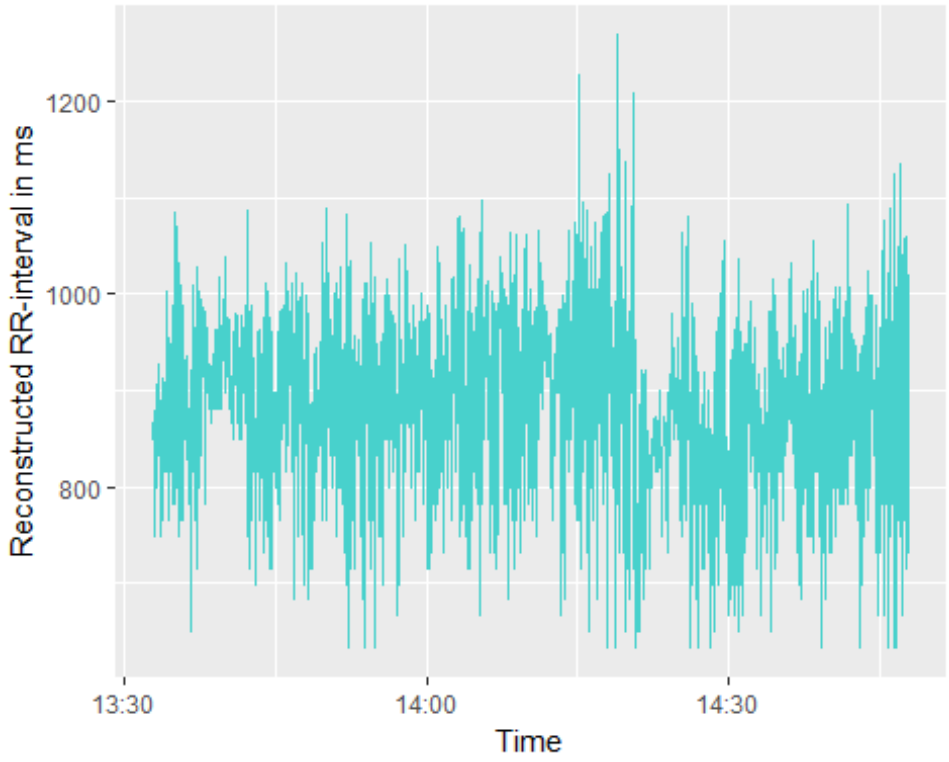
```
filter_time_day1_mb.clean <- filter_time_day1_mb %>%
  mutate(mb_rr=rm_short_comp(mb_rr)) %>%
  filter(!is.na(mb_rr)) %>%
  mutate(mb_rr=window_impute_long(mb_rr))

mb.ssa <- ssa(L=15, x=filter_time_day1_mb.clean$mb_rr,  kind="toeplitz-ssa",
svd.method = "svd")
# plot(mb.ssa)
mb_res <- reconstruct(mb.ssa, group=list(Tr=c(1:4)))
res.trend.mb <- residuals(mb_res) # Extract seasonality from residuals, origi
nal values-predicted values
# spec.pgram(res.trend.mb, detrend=F, log="no")

# Apply formula that uses reconstructed value if residuals>0.5
mb_reconstructed <- reconstructed_rr(res.trend.mb, mb_res$Tr, filter_time_day
1_mb.clean$mb_rr)

# Original Plot
ggplot(filter_time_day1_mb.clean, aes(x=readable_timestamp, y=mb_rr)) + geom_
line(color="salmon") +
  ggtitle("Plot of original rr-interval against time, after removing outliers
, \nmeasured by MSband on the left hand, \non first day of study, 2016-11- 15
, for participant 19.") +
  ylab("RR interval in milliseconds") +
  xlab("time")
```




```
df_mb <- data.frame(Reconstructed=mb_reconstructed$RR, Original=filter_time_d
ay1_mb.clean$mb_rr, Time=filter_time_day1_mb.clean$readable_timestamp)

# Reconstructed and original plot combined
df_mb %>%
  gather(Reconstructed, Original, key="Type", value="RR-interval") %>%
  ggplot(aes(x=Time, y=`RR-interval`)) +
  geom_line(aes(colour=Type)) +
  ylab("RR-interval in ms")
```




```r
# Reconstructed plot
ggplot(df_mb, aes(x=Time, y=Reconstructed)) + geom_line(colour="mediumturquoise") + ylab("Reconstructed RR-interval in ms")
```



Analyze HRV via the time domain, RMSSD, most relevant and accurate measure of autonomic nervous system (ANS) over the short-term

RMSSD is a time series measure of HRV, associated with short-term rapid changes in heart rate, and correlated with vagus-mediated (parasympathetic) components of HRV (DeGiorgio et al 2011). Low RMSSD values indicate poor vagus mediated HRV. Reduced HRV is a biomarker for mortality and sudden death in heart disease, and is correlated with stress.

```
# x should be the column containing rr-interval, units in millisecond
HRV <- function(x){
  total <- 0
  for (i in 1:(length(x)-1)){
     total <- sum(total, (x[i+1] -x[i])^2, na.rm=T)
  }
  return (sqrt(total/(length(x)-1)))
}
```

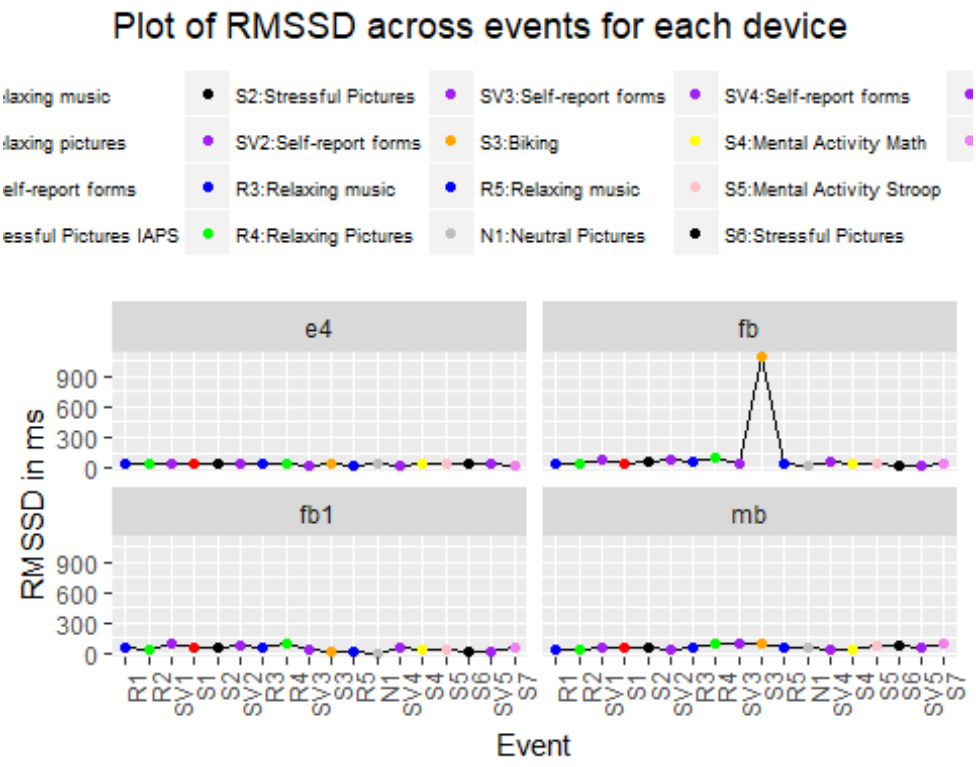
Select each period and calculate RMSSD for E4, fb, and MSband. Fb has 2 versions, one with L=15, and one with L=N/2.

```
for (i in 3:(n-1)){
  # select time for rr-interval
  rr_int_e4 <- df_e4 %>%
    filter(strftime(Time, "%H:%M")>=df$Time[i] & strftime(Time, "%H:%M") <df$
Time[i+1])
  rr_int_fb <- df_fb %>%
    filter(strftime(Time, "%H:%M")>=df$Time[i] & strftime(Time, "%H:%M") < df
$Time[i+1])
  rr_int_fb1 <- df_fb1 %>%
    filter(strftime(Time, "%H:%M")>=df$Time[i] & strftime(Time, "%H:%M") < df
$Time[i+1])
  rr_int_mb <- df_mb %>%
    filter(strftime(Time, "%H:%M")>=df$Time[i] & strftime(Time, "%H:%M") < df
$Time[i+1])
  # calculate RMSSD for each rr interval according to time
  df$RMSSD_e4[i] <- HRV(rr_int_e4$Reconstructed)
  df$RMSSD_fb[i] <- HRV(rr_int_fb$Reconstructed)
  df$RMSSD_fb1[i] <- HRV(rr_int_fb1$Reconstructed)
  df$RMSSD_mb[i] <- HRV(rr_int_mb$Reconstructed)
}
df$RMSSD_e4[n] <- NA
df$RMSSD_fb[n] <- NA
df$RMSSD_fb1[n] <- NA
df$RMSSD_mb[n] <- NA

df #output
```

```
## # A tibble: 21 x 7
##              Event       Time                 Activity RMSSD_e4 RMSSD_fb
##              <chr>      <chr>                    <chr>    <dbl>    <dbl>
##  1            Date 11/15/2016                     <NA>       NA       NA
##  2 Session 1 Start   13:32:52                     <NA>       NA       NA
##  3              R1      13:35          Relaxing music 35.70953 49.25316
##  4              R2      13:44       Relaxing pictures 40.42683 39.56516
##  5             SV1      13:46       Self-report forms 49.45418 81.87692
##  6              S1      13:48 Stressful Pictures IAPS 41.11288 52.78726
##  7              S2      14:00      Stressful Pictures 42.41722 58.25464
##  8             SV2      14:02       Self-report forms 42.96767 88.43546
##  9              R3      14:03          Relaxing music 40.77023 56.96728
## 10              R4      14:14       Relaxing Pictures 50.69409 95.98889
## # ... with 11 more rows, and 2 more variables: RMSSD_fb1 <dbl>,
## #   RMSSD_mb <dbl>
```

Plot HRV over different periods of stress and relaxation, during the study, for each device

```
hrv_df <- df %>%
  slice(3:20) %>%
  gather(`RMSSD_e4`, `RMSSD_fb`, `RMSSD_fb1`,`RMSSD_mb`, key="Device", value="RMSSD") %>%
  mutate(Device=str_replace(Device, "^.*_", ""),
         Event=factor(Event, levels=df$Event[3:20]))

clr <- c("blue", "green", "purple", "red", "black", "purple", "blue", "green", "purple", "orange", "blue", "gray", "purple", "yellow", "pink", "black", "purple", "violet")
ggplot(hrv_df, aes(x=Event, y=RMSSD)) +
  geom_path(aes(group=Device)) +
  geom_point(aes(colour=Event)) +
  facet_wrap(~Device) +
  scale_colour_manual(values=clr,
                      labels=paste(hrv_df$Event,hrv_df$Activity, sep=":"),
                      name=NULL) +
  ggtitle("Plot of RMSSD across events for each device") +
  ylab("RMSSD in ms") +
  theme(legend.position="top",
        legend.text=element_text(size=7),
        axis.text.x = element_text(angle = 90, hjust = 1))
```



Implement timing, 3 min + 30s

```
time <- hms(start) + minutes(3)
timings <- c(as.character(hms(start)),vector())
while (as.character(time)<(hms(end)-seconds(30))){
  timings <- c(timings, as.character(seconds_to_period(period_to_seconds(time
)+seconds(30))))
  time <- seconds_to_period(period_to_seconds(time)+seconds(30))
}
timings <- c(timings, as.character(hms(end)))
for (i in 1:length(timings)){
  timings[i]<- ifelse(nchar(timings)[i]<11,
                      paste(substr(timings[i],1,3)," 0", substr(timings[i],5,
nchar(timings[i])), sep=""),
                      timings[i])
  # Add zeros in between minutes for some timings
}
timings <- substr(str_replace_all(timings, "\\D\\s", ":"),1,8) # Replace HMS
with semicolon for comparison of timing
timings
```

```
##   [1] "13:32:52" "13:36:22" "13:36:52" "13:37:22" "13:37:52" "13:38:22"
##   [7] "13:38:52" "13:39:22" "13:39:52" "13:40:22" "13:40:52" "13:41:22"
##  [13] "13:41:52" "13:42:22" "13:42:52" "13:43:22" "13:43:52" "13:44:22"
##  [19] "13:44:52" "13:45:22" "13:45:52" "13:46:22" "13:46:52" "13:47:22"
##  [25] "13:47:52" "13:48:22" "13:48:52" "13:49:22" "13:49:52" "13:50:22"
```

```
##  [31] "13:50:52" "13:51:22" "13:51:52" "13:52:22" "13:52:52" "13:53:22"
##  [37] "13:53:52" "13:54:22" "13:54:52" "13:55:22" "13:55:52" "13:56:22"
##  [43] "13:56:52" "13:57:22" "13:57:52" "13:58:22" "13:58:52" "13:59:22"
##  [49] "13:59:52" "14:00:22" "14:00:52" "14:01:22" "14:01:52" "14:02:22"
##  [55] "14:02:52" "14:03:22" "14:03:52" "14:04:22" "14:04:52" "14:05:22"
##  [61] "14:05:52" "14:06:22" "14:06:52" "14:07:22" "14:07:52" "14:08:22"
##  [67] "14:08:52" "14:09:22" "14:09:52" "14:10:22" "14:10:52" "14:11:22"
##  [73] "14:11:52" "14:12:22" "14:12:52" "14:13:22" "14:13:52" "14:14:22"
##  [79] "14:14:52" "14:15:22" "14:15:52" "14:16:22" "14:16:52" "14:17:22"
##  [85] "14:17:52" "14:18:22" "14:18:52" "14:19:22" "14:19:52" "14:20:22"
##  [91] "14:20:52" "14:21:22" "14:21:52" "14:22:22" "14:22:52" "14:23:22"
##  [97] "14:23:52" "14:24:22" "14:24:52" "14:25:22" "14:25:52" "14:26:22"
## [103] "14:26:52" "14:27:22" "14:27:52" "14:28:22" "14:28:52" "14:29:22"
## [109] "14:29:52" "14:30:22" "14:30:52" "14:31:22" "14:31:52" "14:32:22"
## [115] "14:32:52" "14:33:22" "14:33:52" "14:34:22" "14:34:52" "14:35:22"
## [121] "14:35:52" "14:36:22" "14:36:52" "14:37:22" "14:37:52" "14:38:22"
## [127] "14:38:52" "14:39:22" "14:39:52" "14:40:22" "14:40:52" "14:41:22"
## [133] "14:41:52" "14:42:22" "14:42:52" "14:43:22" "14:43:52" "14:44:22"
## [139] "14:44:52" "14:45:22" "14:45:52" "14:46:22" "14:46:52" "14:47:22"
## [145] "14:47:52" "14:47:57"
```

```
df_window <- data.frame(time=timings) %>%
  mutate(time=as.character(time))
for (i in 1:(nrow(df_window)-1)){
  # select time
  e4 <- df_e4 %>%
    filter(strftime(Time, "%H:%M:%S")>=df_window$time[i] & strftime(Time, "%H
:%M:%S") <df_window$time[i+1])
  fb <- df_fb %>%
  filter(strftime(Time, "%H:%M:%S")>=df_window$time[i] & strftime(Time, "%H:%
M:%S") < df_window$time[i+1])
  fb1 <- df_fb1 %>%
  filter(strftime(Time, "%H:%M:%S")>=df_window$time[i] & strftime(Time, "%H:%
M:%S") < df_window$time[i+1])
  mb <- df_mb %>%
  filter(strftime(Time, "%H:%M:%S")>=df_window$time[i] & strftime(Time, "%H:%
M:%S") < df_window$time[i+1])
  # calculate RMSSD for each time interval
  df_window$RMSSD_e4[i] <- HRV(e4$Reconstructed)
  df_window$RMSSD_fb[i] <- HRV(fb$Reconstructed)
  df_window$RMSSD_fb1[i] <- HRV(fb1$Reconstructed)
  df_window$RMSSD_mb[i] <- HRV(mb$Reconstructed)
}

head(df_window)
```

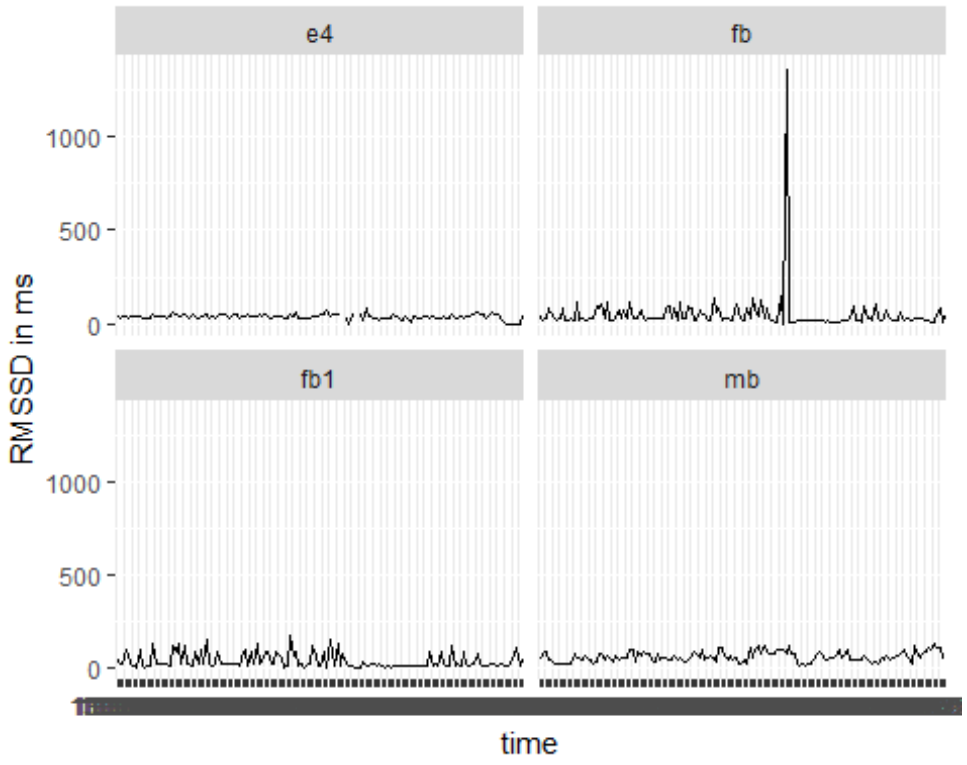
```
##       time RMSSD_e4 RMSSD_fb RMSSD_fb1 RMSSD_mb
## 1 13:32:52 35.35419 36.22986  39.25579 45.90531
## 2 13:36:22 29.33599 19.45375  19.18113 74.40244
## 3 13:36:52 36.72371 26.51069  17.17777 84.55981
```

```
## 4 13:37:22 34.64609 85.22152  92.43649 47.86705
## 5 13:37:52 31.31101 65.14340  79.43071 48.45938
## 6 13:38:22 37.60952 23.81902  17.74642 16.64338

tail(df_window)

##         time RMSSD_e4 RMSSD_fb RMSSD_fb1  RMSSD_mb
## 141 14:45:52  0.00000 11.39918  15.44252  99.47055
## 142 14:46:22  0.00000 11.77657  17.70395 126.14013
## 143 14:46:52  0.00000 51.39032  74.38258 111.50107
## 144 14:47:22  0.00000 86.79267 109.45519 113.01484
## 145 14:47:52  0.00000 10.37041  10.88577  64.25495
## 146 14:47:57 35.35419 36.22986  39.25579  45.90531
```

Zeros or NaN in between indicate that there isn't any value for certain timings.

Plot RMSSD across continuous timings

```
df_window1 <- df_window %>%
   gather(`RMSSD_e4`, `RMSSD_fb`,`RMSSD_fb1`, `RMSSD_mb`, key="Device", value
="RMSSD") %>%
   mutate(Device=str_replace(Device, "^.*_", ""))
head(df_window1)

##       time Device    RMSSD
## 1 13:32:52     e4 35.35419
## 2 13:36:22     e4 29.33599
## 3 13:36:52     e4 36.72371
## 4 13:37:22     e4 34.64609
## 5 13:37:52     e4 31.31101
## 6 13:38:22     e4 37.60952

ggplot(df_window1, aes(x=time,y=RMSSD)) +
  geom_path(aes(group=Device)) +
  facet_wrap(~Device) +
  ylab("RMSSD in ms")
```



Take a segment of rr-interval for 20s between 500-1000ms from MB, for inspection of algorithm

```
(test.data.mb <- filter_time_day1_mb %>%
  filter(strftime(readable_timestamp, "%H:%M:%S") >= "13:41:00" & strftime(re
adable_timestamp, "%H:%M:%S") <= "13:41:20") %>%
  select(readable_timestamp,mb_rr))

##     readable_timestamp   mb_rr
## 1  2016-11-15 13:41:00 779.824
## 2  2016-11-15 13:41:01 746.640
## 3  2016-11-15 13:41:01 730.048
## 4  2016-11-15 13:41:02 746.640
## 5  2016-11-15 13:41:03 696.864
## 6  2016-11-15 13:41:03 696.864
## 7  2016-11-15 13:41:04 746.640
## 8  2016-11-15 13:41:05 746.640
## 9  2016-11-15 13:41:06 746.640
## 10 2016-11-15 13:41:06 763.232
## 11 2016-11-15 13:41:08 647.088
## 12 2016-11-15 13:41:08 713.456
## 13 2016-11-15 13:41:09 746.640
## 14 2016-11-15 13:41:10 829.600
## 15 2016-11-15 13:41:11 796.416
## 16 2016-11-15 13:41:12 713.456
## 17 2016-11-15 13:41:12 696.864
```

```
## 18 2016-11-15 13:41:13 746.640
## 19 2016-11-15 13:41:14 779.824
## 20 2016-11-15 13:41:15 746.640
## 21 2016-11-15 13:41:15 696.864
## 22 2016-11-15 13:41:16 713.456
## 23 2016-11-15 13:41:17 713.456
## 24 2016-11-15 13:41:17 713.456
## 25 2016-11-15 13:41:18 696.864
## 26 2016-11-15 13:41:19 713.456
## 27 2016-11-15 13:41:19 696.864
## 28 2016-11-15 13:41:20 730.048
```

```
write_csv(test.data.mb, "test_data_mb.csv")
```

Analyze HRV via the frequency domain

HRV, especially the high frequency components, is regulated by parasympathetic activity from the vagus nerve; only parasympathetic action can mediate the rapid changes accompanying such high frequency variation. Plan to use RHRV package for analysis, to be continued.

```
library(RHRV)
```