FASTER tutorial

Genshiro A Sunagawa

January 31, 2014

This tutorial shows how to use the *FASTER* method intorduced in the paper 'FASTER: an unsupervised fully automated sleep staging method for mice'. The latest source code can be downloaded from http://bitbucket.org/genshiro/faster/.

1 Setup

Load the packages used in this analysis. If you haven't installed the packages **pdfCluster** and **ggplot2** yet, run the first line without commenting them out.

```
# install.packages(c('pdfCluster', 'ggplot2'))
library(pdfCluster)
library(ggplot2)
sleep.stage.labels <- c("NREM", "REM", "Wake", "Unknown", "Outlier1", "Outlier2")
epoch.size <- 8  # seconds
epoch.count <- 10800  # how many epochs are going to be analyzed?
sampling.frequency <- 100  # Hz
freq <- (seq(sampling.frequency * epoch.size%/%2 + 1) - 1) * (1/epoch.size)
```

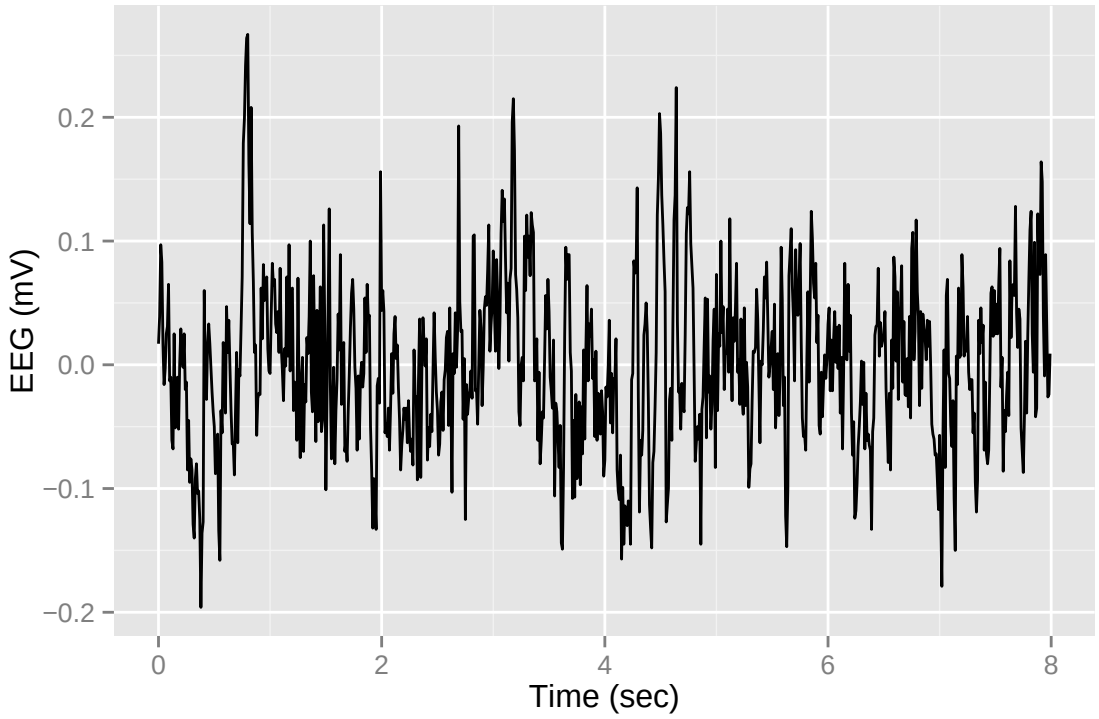
2 Loading the EEG/EMG data

Download the sample data. The eeg and emg should be a numeric vector which has same length.

```
load(url("http://goo.gl/MkqvLL"))
load(url("http://goo.gl/IoR5D7"))
eeg <- eeg[seq(epoch.count * epoch.size * sampling.frequency)]
emg <- emg[seq(epoch.count * epoch.size * sampling.frequency)]
```

You can take a look of the raw EEG data by inputing the following command. It will plot a line graph of the EEG of the very first 800 points stored in `eeg`.

```
one.epoch <- epoch.size * sampling.frequency
p <- ggplot(data.frame(eeg = eeg[seq(one.epoch)], time = (seq(one.epoch) - 1)/sampling.frequency))
p <- p + geom_line(aes(x = time, y = eeg))
p <- p + labs(x = "Time (sec)", y = "EEG (mV)")
print(p)
```



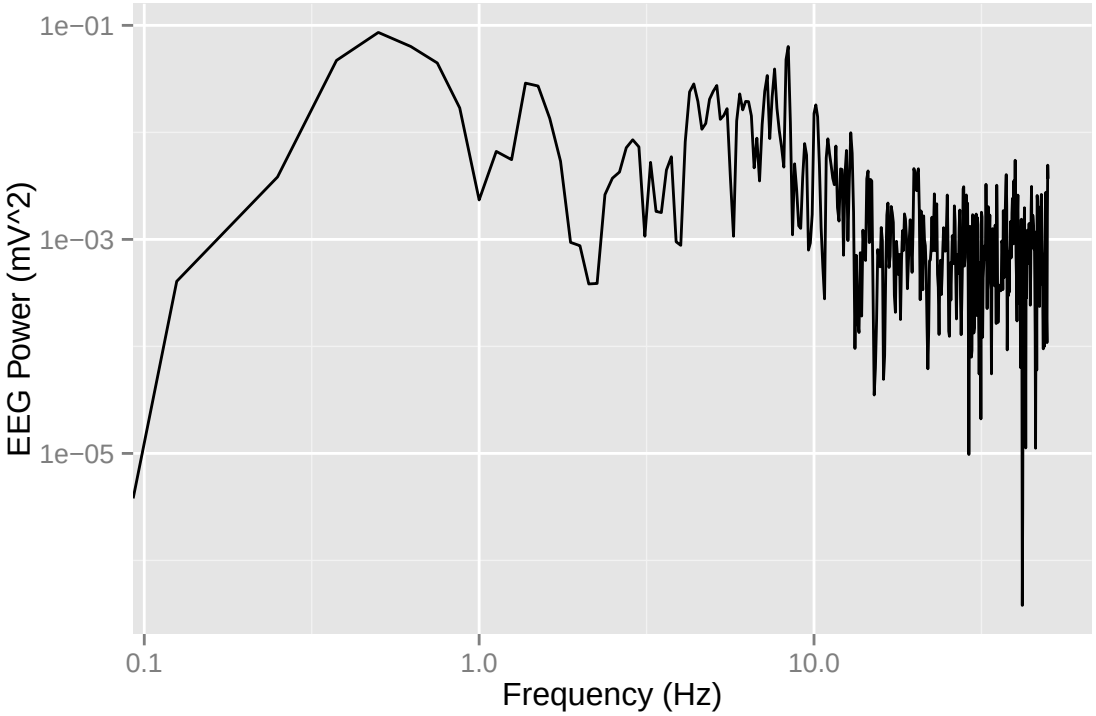
3 Calculation of power spectrum

The time domain data are transformed to frequency domain data as a power spectrum.

```
.GetPower <- function(vols) {

  dim(vols) <- c(epoch.size * sampling.frequency, epoch.count)

  # detrend
  .Detrend <- function(v) {
    time <- seq(sampling.frequency * epoch.size) - 1
    vol.lm <- lm(vol ~ time, data.frame(time, vol = v))
    return(v - (vol.lm$coefficients[2] * time + vol.lm$coefficients[1]))
  }
  vols <- apply(vols, 2, .Detrend)

  # Hann window
  n <- sampling.frequency * epoch.size - 1
  hw <- 0.5 - 0.5 * cos(2 * pi * (0:n)/n)
  .Hanning <- function(v) hw * v
  vols <- apply(vols, 2, .Hanning)

  # calc power
  .Pow <- function(v) (abs(fft(v)[seq(sampling.frequency * epoch.size%/%2 +
    1)]))^2/(sampling.frequency * epoch.size) * 2
  vols <- apply(vols, 2, .Pow)

  return(vols)
}

eeg.pow <- .GetPower(eeg)
emg.pow <- .GetPower(emg)
```

Let's take a look to the power spectrum of the first epoch. Compare with the time domain version.

```
p <- ggplot(data.frame(freq, eeg.pow = eeg.pow[, 1]))
p <- p + geom_line(aes(x = freq, y = eeg.pow))
p <- p + scale_y_log10("EEG Power (mV^2)")
p <- p + scale_x_log10("Frequency (Hz)")
print(p)
```



4 Character extraction

The power spectrum shares the same amount of data with the time domain data expect the phase information which is lost by `abs` function. For clustering effeciently, we are reducing the dimension of the data using the principal component analysis (PCA). In *FASTER*, the components derived from the PCA are called characters.

4.1 Preps for PCA

Reshaping the EEG and EMG power spectrum before performaing the PCA.

```
# EEG characters
pow.eeg.delta <- colSums(eeg.pow[(freq >= 0.5) & (freq <= 4), ])
pow.eeg.theta <- colSums(eeg.pow[(freq >= 6) & (freq <= 10), ])

# EMG characters
pow.emg.total <- colSums(emg.pow)

# calc the amplitude
.amplitude <- function(v) (Mod(v))^2/(sampling.frequency * epoch.size) * 2
.calc.mat <- function(v, vol.type) {
  v.mat <- apply(v, 2, .amplitude)
  v.mat <- t(log10(v.mat))
  v.mat[is.infinite(v.mat)] <- 0
  v.mat <- (v.mat - mean(v.mat))/sd(v.mat)
  dimnames(v.mat) <- list(epoch = seq(dim(v.mat)[1]), signal = sprintf("%s_%03d",
    vol.type, seq(dim(v.mat)[2])))
  return(v.mat)
}

eeg.mat <- .calc.mat(eeg.pow, "EEG_AMP")
emg.mat <- .calc.mat(emg.pow, "EMG_AMP")
```

4.2 Principal component analysis (PCA)

Running the PCA. Since we have centered and scaled the dataset in the previous code, `center` and `scale.` option are set to `FALSE`.

```
pca.results <- prcomp(cbind(eeg.mat, emg.mat), center = FALSE, scale. = FALSE)
```

This is how the PCA results look like:

```
str(pca.results)

## List of 5
##  $ sdev    : num [1:802] 16.2 13.01 3.76 2.38 1.44 ...
##  $ rotation: num [1:802, 1:802] 0.0142 -0.00388 -0.0162 -0.02528 -0.02951 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:802] "EEG_AMP_001" "EEG_AMP_002" "EEG_AMP_003" "EEG_AMP_004" ...
##   .. ..$ : chr [1:802] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : logi FALSE
##  $ scale   : logi FALSE
##  $ x       : num [1:10800, 1:802] 4.332 4.096 15.303 7.245 0.719 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:10800] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:802] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

5 Clustering and staging

Using the principal components, we are going to group the dataset into several clusters. Every clusters will be annotated to proper stages of sleep and wake, which is usually called 'staging'.

5.1 Preps

Preparation for clustering. We use 'Nonparametric density estimation clustering' which utilize no model for clutering. The method is available as **pdfCluster** pacakge in R. How the parameters were optimized are described in the paper.

```
group.length <- 5400
hmult <- 0.7
pcn <- 4
n.grid <- group.length%/%10
group.epochs <- t(matrix(seq(epoch.count), ncol = group.length))
stages <- rep(NA, epoch.count)
P.emg <- 0.5
P.delta <- 0.1
```

5.2 Looping the clustering and the staging code

The **pdfCluster** pacakge uses enourmus size of memory when the source data are high-dimensional. In *FASTER*, therefore, the original data are resampled to the size of `group.length` and each groups are clustered and staged individually.

```
for (j in seq(dim(group.epochs)[2])) {

  target.epochs <- group.epochs[, j]
  outliers1 <- rep(FALSE, length(target.epochs))
  outliers2 <- rep(FALSE, length(target.epochs))

  # labeling outliers if any scores exceed 10SD
  sds <- rep(0, pcn)
  x <- pca.results$x[target.epochs, seq(pcn)]
  for (i in seq(pcn)) {
    sds[i] <- sd(x[, i])
    outliers1[which(abs(x[, i] - mean(x[, i])) > 10 * sds[i])] <- TRUE
  }

  # PDF clustering
  cl <- new("NULL")
  cl.noc <- NA
  hmult.k <- (-1)
  while (is.null(cl)) {
    hmult.k <- hmult.k + 1
    cl <- pdfCluster(x[!outliers1, ], n.grid = n.grid, hmult = hmult + 0.05 *
      hmult.k)
  }
  outliers2[which(!outliers1)[which(cl@clusters == 0)]] <- TRUE

  if (cl@noc == 1) {
    # only one cluster
    stages[target.epochs] <- "Unknown"
    stages[target.epochs[outliers1]] <- "Outlier1"
  } else {
    # cl.noc>1
    deltas <- rep(0, cl@noc)
    thetas <- rep(0, cl@noc)
    emgs <- rep(0, cl@noc)

    for (i in seq(cl@noc)) {
      # calc delta and emg medians for each stages
      ids <- target.epochs[which(!outliers1)[which(cl@clusters == i)]]
      deltas[i] <- median(log10(pow.eeg.delta[ids]))
      emgs[i] <- median(log10(pow.emg.total[ids]))
      thetas[i] <- median(log10(pow.eeg.theta[ids]))
    }

    anot <- rep("", cl@noc)  # core number to annotation

    med.emg.pow <- median(log10(pow.emg.total[target.epochs[!outliers1]]))
    for (i in seq(cl@noc)) if (emgs[i] > med.emg.pow)
      anot[i] <- "Wake"

    not.wake.epochs <- target.epochs[which(!outliers1)[!cl@clusters %in%
      which(anot == "Wake")]]

    for (i in which(anot == "")) {
      if (deltas[i] < quantile(log10(pow.eeg.delta[not.wake.epochs]),
        0.1))
        anot[i] <- "REM" else anot[i] <- "NREM"
    }

    stages[target.epochs[!(outliers1 | outliers2)]] <- anot[cl@clusters[which(cl@clusters !=
      0)]]
    stages[target.epochs[outliers1]] <- "Outlier1"
    stages[target.epochs[outliers2]] <- "Outlier2"
  }
}

stages[is.na(stages)] <- "Unknown"
s <- factor(stages, levels = sleep.stage.labels)
```

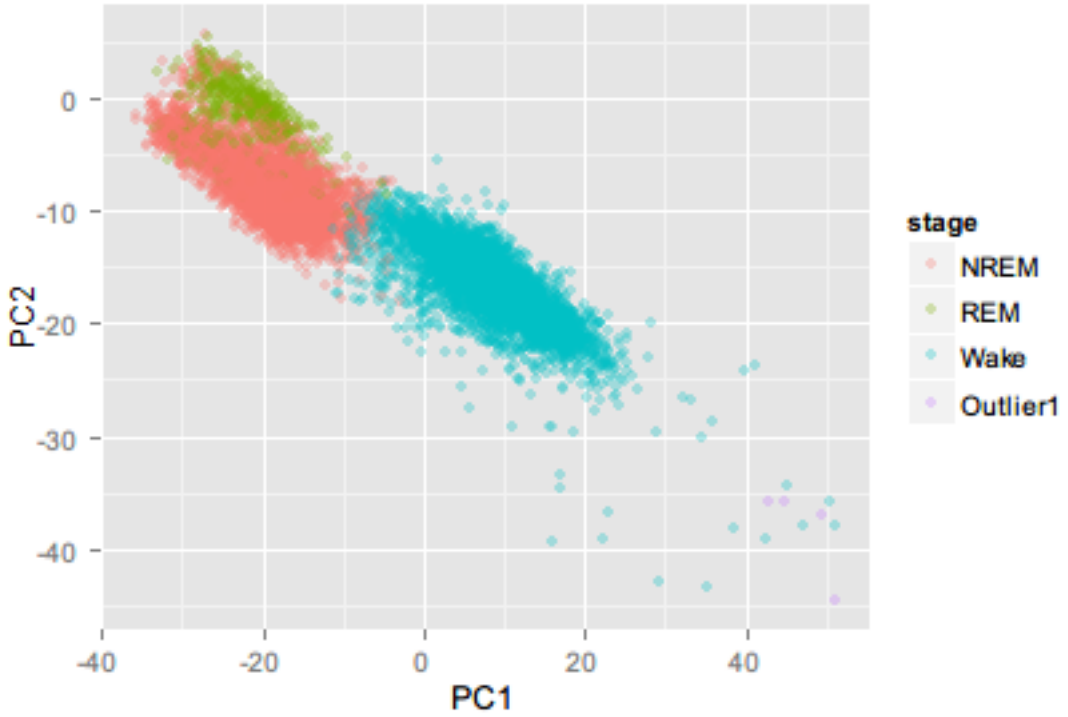
6 Results

Let's take a look to the results. The number of sleep and wake counts can be seen by using the `summary` function.

```
summary(s)

##      NREM       REM      Wake   Unknown Outlier1 Outlier2
##      5194       571      5031         0        4        0
```

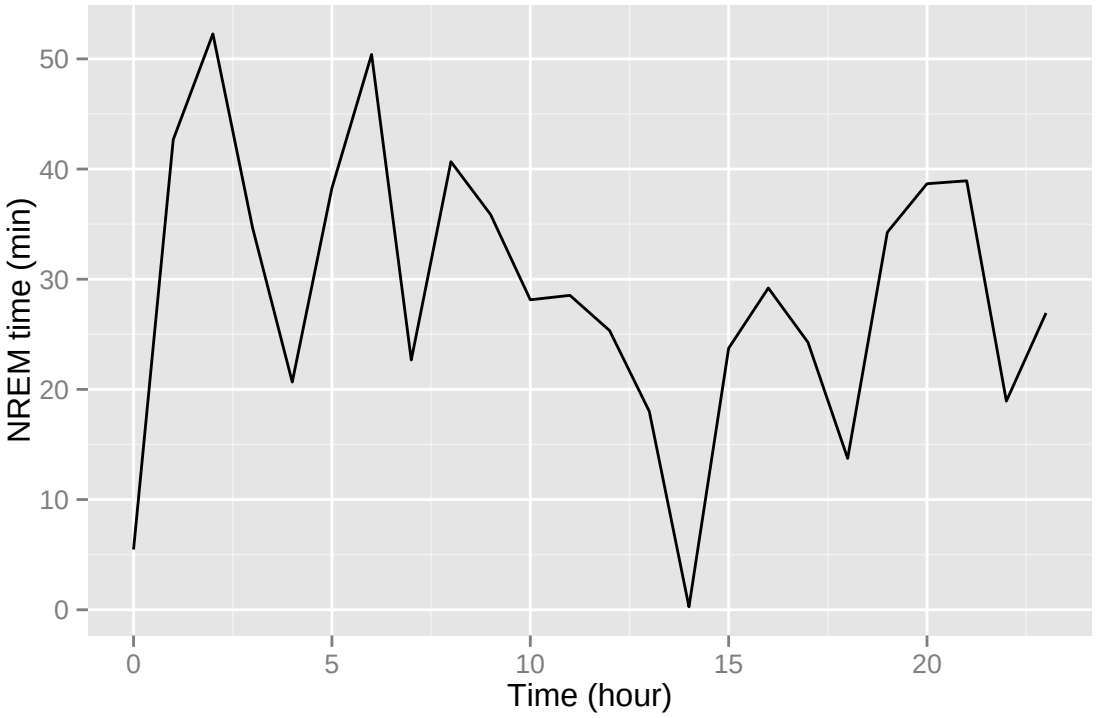
Scatter plot of the first two principal components will give us the overview of the distribution of the dataset. Coloring the results by stages is not difficult.

```
p <- ggplot(data = data.frame(pca.results$x, stage = s))
p <- p + geom_point(aes(x = PC1, y = PC2, color = stage), alpha = 0.3)
print(p)
```



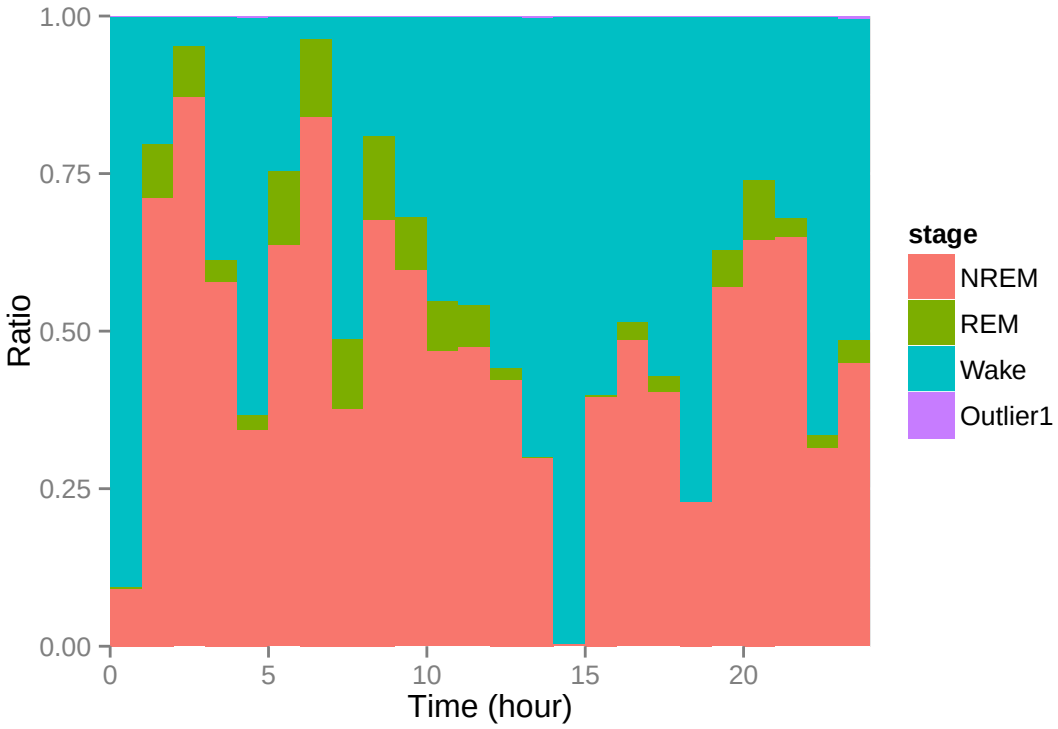
The hourly difference of NREM time in minutes.

```
.count <- function(y, st) length(which(y == st))
.calc.binned.stage <- function(x, bin, st) apply(matrix(x, nrow = bin), 2, .count,
  st)
p <- ggplot(data.frame(NREM = .calc.binned.stage(s, 450, "NREM")/450 * 60, time = 0:23))
p <- p + geom_line(aes(x = time, y = NREM))
p <- p + labs(x = "Time (hour)", y = "NREM time (min)")
print(p)
```



Another figure showing hourly difference of each stages.

```
p <- ggplot(data.frame(stage = s, hour = (seq(10800) - 1)/10800 * 24))
p <- p + geom_bar(aes(x = hour, fill = stage), binwidth = 1, position = "fill")
p <- p + scale_x_continuous("Time (hour)", limits = c(0, 24), expand = c(0,
  0))
p <- p + scale_y_continuous("Ratio", expand = c(0, 0))
print(p)
```

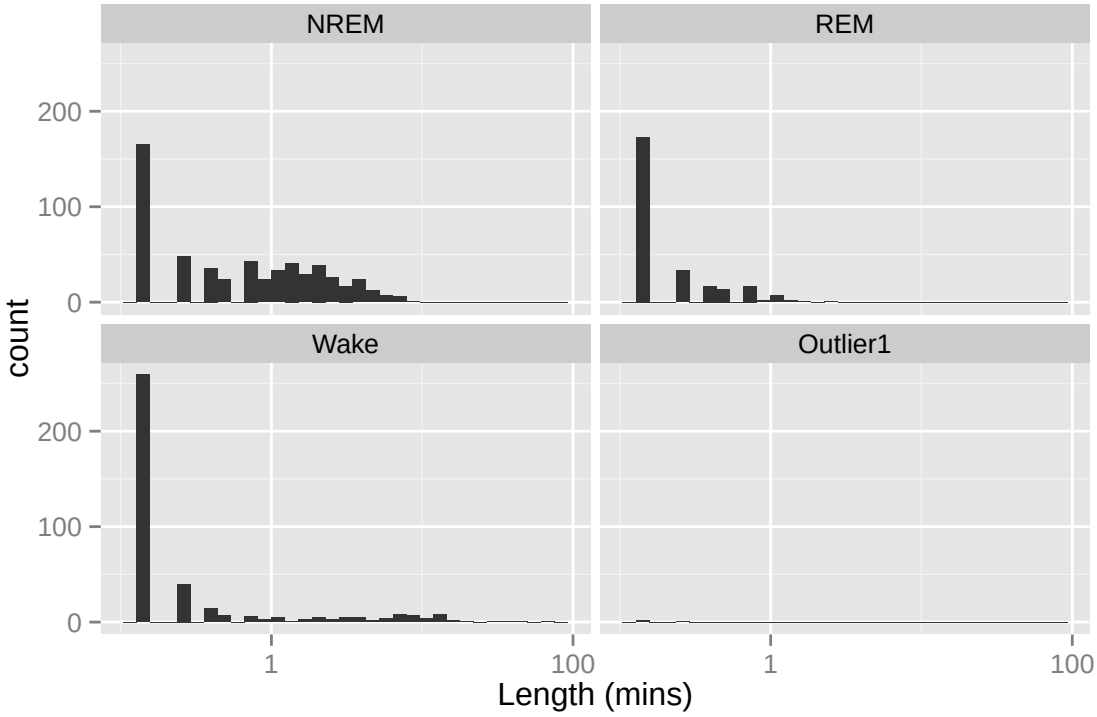


The following code will calculate the bouts of each stages in minutes.

```
bout.length <- rep(0, length(s))
prev.s <- s[1]
y <- 1
for (i in 2:length(bout.length)) {
  if (prev.s != s[i]) {
    bout.length[i - 1] <- y
    y <- 1
  } else {
    y <- y + 1
  }
  prev.s <- s[i]
}
bout.df <- data.frame(length = bout.length * 8/60, stage = s)
bout.df <- subset(bout.df, bout.length != 0)
```

The histogram of bout length of each stages.

```
p <- ggplot(bout.df)
p <- p + geom_histogram(aes(x = length))
p <- p + scale_x_log10("Length (mins)")
p <- p + facet_wrap(~stage)
print(p)

## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
```



Showing the number of transitions from one stage to another.

```
trans.df <- data.frame(s1 = s[1:10799], s2 = s[2:10800])
p <- ggplot(trans.df)
p <- p + geom_jitter(aes(x = s1, y = s2), size = 1, alpha = 0.3)
p <- p + labs(x = "Previous stage", y = "Stage")
print(p)
```