COMPUTATION OF ACCELERATION VARIABLES FOR BEHAVIOURAL IDENTIFICATION

Set up parameters

The script uses the following libraries, make sure you install them beforehand.

```
library(chron)
```

Mov.Av. is a function to compute the running mean of a variable *obj*, used in the present work to extract the static acceleration and smooth VeDBA

```
Mov.Av.=function(obj){
  ma = obj
  for(i in AccHz:length(obj)){ma[i] = mean(obj[(i-AccHz):(i+AccHz)],na.rm=T)}
  # First points set to NA
  for(i in 1:AccHz){ ma[i] = NA }
  # Last points set to NA as well
  for(i in (length(obj)-AccHz):length(obj)){ ma[i] = NA }
  ma
}
```

Set the sampling frequency and timestep of the running mean.

Hz states for the sampling frequency of acceleration (in our case 40Hz)
Duration states for the timestep in seconds to be used in the running mean (in our case 3 seconds)

```
Hz=40
Duration=3
AccHz=Hz*Duration/2
```

Load and set up the file

```
DD=read.table(file.choose(),header=T, sep=",")

   # Your file should be composed of the following column.
   # Behav.Lab  is populated with behavioural observations required for
   # teaching the model
```

```
 str(DD)
```

```
## 'data.frame':    1048575 obs. of  6 variables:
##  $ Date     : Factor w/ 1 level "27/05/2015": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Time     : Factor w/ 26373 levels "07:00:21","07:00:22",..: 1 1 1 1 1 1
 1 1 1 1 ...
##  $ AccX     : num  0.42 0.408 0.384 0.384 0.396 ...
##  $ AccY     : num  -0.192 -0.18 -0.168 -0.192 -0.192 ...
##  $ AccZ     : num  0.864 0.912 0.888 0.888 0.864 ...
##  $ Behav.Lab: Factor w/ 5 levels "0","FOR","GROOM",..: 1 1 1 1 1 1 1 3 3 3
 ...
```

```
  DD$Date=as.Date(DD$Date,format="%d/%m/%Y")
  DD$Time <- chron(times=DD$Time)
  DD$Timestamp=as.POSIXct(paste(DD$Date,DD$Time),format="%Y-%m-%d %H:%M:%S")
```

Compute the raw variables

First step is to compute the raw variables with the recorded sampling frequency.

Static Acceleration (St)

```
    DD$stX=Mov.Av.(DD$AccX)
    DD$stY=Mov.Av.(DD$AccY)
    DD$stZ=Mov.Av.(DD$AccZ)
```

Dynamic Acceleration (Dy)

```
    DD$dyX=DD$AccX-DD$stX
    DD$dyY=DD$AccY-DD$stY
    DD$dyZ=DD$AccZ-DD$stZ
```

VeDBA

```
    DD$VeDBA=sqrt(DD$dyX^2+DD$dyY^2+DD$dyZ^2)
    DD$VeDBAs=Mov.Av.(DD$VeDBA) # smoothed VeDBA
```

Angles in radians

```
    DD$Pitch=asin(DD$stX)
    DD$Sway=asin(DD$stY)
```

A few NaNs can be produced when stX and stY are above 1 due to a drastic change in posture and high level of activity. This may happen more often if the collar is turning on the animal's neck or if the Running Mean timestep (*Duration* for the function *Mov.Av.*) is too small.

PDBA

```
    DD$PDBAX=abs(DD$dyX)
    DD$PDBAY=abs(DD$dyY)
    DD$PDBAZ=abs(DD$dyZ)
```

Ratio VeDBA-Dy (Dynamic acceleration)

```
    DD$RatioX=DD$VeDBA/DD$PDBAX
    DD$RatioY=DD$VeDBA/DD$PDBAY
    DD$RatioZ=DD$VeDBA/DD$PDBAZ
```

Compute the *by-second* file

Create an empty DataFrame

```
  ColDD.sec=c("Date","Time","Timestamp","ID","Behav.Lab",
              apply(
                expand.grid(
                  "mean",colnames(DD)[which(colnames(DD)=="stX"):ncol(DD)])
                  , 1, paste, collapse="."),
              "PSD1X","Freq1X","PSD2X","Freq2X","PSD1Y","Freq1Y","PSD2Y",
              "Freq2Y","PSD1Z","Freq1Z","PSD2Z","Freq2Z")

  DD.sec=as.data.frame(matrix(data=NA, nrow=length(unique(DD$Timestamp)),
                      ncol=length(ColDD.sec),
                      dimnames=list(1:length(unique(DD$Timestamp)),ColDD.sec)))
```

This produces the following data.frame to be populated:

```
  str(DD.sec)

## 'data.frame':    26373 obs. of  31 variables:
##  $ Date       : logi  NA NA NA NA NA NA ...
##  $ Time       : logi  NA NA NA NA NA NA ...
##  $ Timestamp  : logi  NA NA NA NA NA NA ...
##  $ ID         : logi  NA NA NA NA NA NA ...
##  $ Behav.Lab  : logi  NA NA NA NA NA NA ...
##  $ mean.stX   : logi  NA NA NA NA NA NA ...
##  $ mean.stY   : logi  NA NA NA NA NA NA ...
##  $ mean.stZ   : logi  NA NA NA NA NA NA ...
##  $ mean.dyX   : logi  NA NA NA NA NA NA ...
##  $ mean.dyY   : logi  NA NA NA NA NA NA ...
##  $ mean.dyZ   : logi  NA NA NA NA NA NA ...
##  $ mean.VeDBA : logi  NA NA NA NA NA NA ...
##  $ mean.VeDBAs: logi  NA NA NA NA NA NA ...
##  $ mean.PDBAX : logi  NA NA NA NA NA NA ...
##  $ mean.PDBAY : logi  NA NA NA NA NA NA ...
##  $ mean.PDBAZ : logi  NA NA NA NA NA NA ...
##  $ mean.RatioX: logi  NA NA NA NA NA NA ...
##  $ mean.RatioY: logi  NA NA NA NA NA NA ...
##  $ mean.RatioZ: logi  NA NA NA NA NA NA ...
##  $ PSD1X      : logi  NA NA NA NA NA NA ...
##  $ Freq1X     : logi  NA NA NA NA NA NA ...
##  $ PSD2X      : logi  NA NA NA NA NA NA ...
##  $ Freq2X     : logi  NA NA NA NA NA NA ...
##  $ PSD1Y      : logi  NA NA NA NA NA NA ...
##  $ Freq1Y     : logi  NA NA NA NA NA NA ...
##  $ PSD2Y      : logi  NA NA NA NA NA NA ...
##  $ Freq2Y     : logi  NA NA NA NA NA NA ...
##  $ PSD1Z      : logi  NA NA NA NA NA NA ...
##  $ Freq1Z     : logi  NA NA NA NA NA NA ...
##  $ PSD2Z      : logi  NA NA NA NA NA NA ...
##  $ Freq2Z     : logi  NA NA NA NA NA NA ...
```

Basic informations

Before proceeding, fill the *IDNo* accordingly to your Individual code number

```
  IDNo="M2"
  DD.sec$Date=as.Date(unique(DD$Date),format="%Y-%m-%d")
  DD.sec$Time <- chron(times=unique(DD$Time))
  DD.sec$Timestamp=unique(DD$Timestamp)
  DD.sec$ID=IDNo
  DD.sec$Behav.Lab=tapply(DD$Behav.Lab,DD$Time,function(x)
                          names(which.max(table(x))))

# This line assigns a behaviour to each second according to the most frequent
 behaviour label in that second
```

Select only identified behaviour lines for the Learning Sample. Code for no behaviour observed is *0*. Sub-setting the data can save a considerable amount of processing time.

```
  DD.sec=DD.sec[which(DD.sec$Behav.Lab!=0),]
```

All descriptive stats

```
   a=which(ColDD.sec=="mean.stX")
    for(C in which(colnames(DD)=="stX"):ncol(DD)){
      DD.sec[,a]=tapply(DD[,C],DD$Time,FUN=mean,na.rm=T)
      a=a+1
    }
```

Max Power Spectrum Density and frequencies

WARNING! This part requires a long processing time.
Compute the PSDs and frequency for each axis for each line

```
  a=which(ColDD.sec=="PSD1X")
  for (C in which(colnames(DD)=="dyX"):which(colnames(DD)=="dyZ")){
    for (n in 3:(nrow(DD.sec)-3)){
# Can't compute it for the first and for the last 3 seconds

# Extract the data needed for each seconds
# To compute a frequency, we need to take a larger sample than one second
# We choose to use a 3 second windows to run our Fast Fourier Analysis.

sampling=(80+length(which(DD.sec$Time[n]==DD$Time)))/3

# Compute the exact sampling frequency for these 3 seconds. 1 second before
# and after at 40 Hz=80 + how many points were sampled at that second
# (recording frequency range between 39 and 41 points per second)
```

```
    trajectory = DD[(which(DD$Time==DD.sec$Time[n])[1]-40):
                        (which(DD$Time==DD.sec$Time[n])
                            [length(which(DD.sec$Time[n]==DD$Time))]+40),C]

# The data needs to be detrended before processing.
# This enhances the quality of the Fast Fourier Analysis

      alpha=1:length(trajectory)
      detrended.trajectory <- lm(trajectory ~ alpha)$residuals
      FFT=fft(detrended.trajectory)

# Fast Fourier Analysis results in a vector of complex numbers.
# The following line extracts the Power Spectral Density and frequency

      PSD=(sqrt(Re(FFT)^2+Im(FFT)^2)*2/length(trajectory))^2

# The following lines store the maximum PSD and its associated frequency for
# each second
# The PSD is extracted in frequencies ranging between 1 and 60 because we are
# not interested in behaviours happening at more than 60/3=20Hz

     DD.sec[n,a]=max(PSD[1:60])
     DD.sec[n,a+1]=(which.max(PSD[1:60]))/length(trajectory)*sampling  #freq1

# The second maximum PSD and its associated frequency are computed.
# We take out any frequencies and PSD in the second around the maximum PSD

      DD.sec[n,a+2]=max(PSD
                        [c(if(which.max(PSD[1:60])>3)
                          1:(which.max(PSD[1:60])-3)
                          else(NA),
                          if(which.max(PSD[1:60])<58)
                            seq(which.max(PSD[1:60])+3,60)
                          else(NA)
                        )],na.rm=T)
      DD.sec[n,a+3]=which(PSD[1:60]==DD.sec[n,a+2])[1]
                         /length(trajectory)*sampling
    }
    a=a+4
  }
```