---
title: "Supplemental material 1 - Processing GPS data and extracting environmental variables"
author: "Lucas Krüger"
date: "2024-02-08"
output: html_document

```{r}

library(terra)
library(track2KBA)
library(ggplot2)
library(adehabitatLT)
library(lubridate)
library(dplyr)
library(EMbC)
library(patchwork)

library(sf)
library(sp)
library(magrittr)
library(reshape2)

library(matrixStats)

library(raster)
library(sf)
library(ncdf4)




```

load and process GPS data

```{r}

# raw tracking data

setwd("C:/GiantSuperModel/SGP Tracking/") # working directory

hpdf<-readRDS("Tracking Data/tracking_data_2021_2022_2023.Rds") # harmony point data

### some birds are repeated in both season, so ID should consider that

hpdf$idY<-paste(hpdf$BirdId,hpdf$Season)

summary(as.factor(hpdf$Season))
summary(as.factor(hpdf$idY))

hpdf2<- hpdf[!duplicated(hpdf[, c("idY", "TS")]), ]


epsg4326<-CRS("+init=epsg:4326")

epsg6329<-CRS("+proj=ortho +lat_0=-90 +lon_0=0")


spdf<-SpatialPointsDataFrame(hpdf2[4:5],hpdf2,proj4string = epsg4326)
spdfP<-spTransform(spdf,CRSobj = epsg6329)
```

```
lt<-as.ltraj(xy=hpdf2[4:5],date=hpdf2$TS,id=hpdf2$idY,slsp = "remove",
               infolocs=hpdf2,typeII=T,proj4string = epsg6329)

rtd<-ld(lt)
head(rtd)
summary(rtd$abs.angle)
rtd$abs.dir<-((rtd$abs.angle*180)/max(na.omit(rtd$abs.angle)))+180
summary(rtd$abs.dir)


# some issues with these trips below, let's eliminate them

#rtd<-subset(rtd, burst!="TT00672 22/23" & burst!="TT00692 22/23" &
#              burst!="V00629 22/23" & burst!="V00648 21/22" &
#              burst!="V00648 22/23" & burst!="V00649 22/23")




### lets classify trips and also eliminate points in the colony using track2kba

tkd<-data.frame(track_id=rtd$burst,
                date_gmt=paste(year(rtd$date),month(rtd$date),day(rtd$date),sep="-"),
                time_gmt=paste(hour(rtd$date),minute(rtd$date),second(rtd$date),sep=":"),
                lon=rtd$x,lat=rtd$y,abs.dir=rtd$abs.dir)

dataGroup <- formatFields(
  dataGroup = tkd,
  fieldID   = "track_id",
  fieldDate = "date_gmt",
  fieldTime = "time_gmt",
  fieldLon  = "lon",
  fieldLat  = "lat"
)

colony <- dataGroup %>%
  summarise(
    Longitude = (-59.196),
    Latitude  = (-62.307)
  )

trips <- tripSplit(
  dataGroup  = dataGroup,
  colony     = colony,
  innerBuff  = 0.3,      # kilometers
  returnBuff = 0.3,
  duration   = 1,      # hours
  rmNonTrip  = TRUE
)


plot(trips)

mapTrips(trips = trips, colony = colony)

head(trips)
summary(trips$ColDist)



trips2 <- subset(trips, ColDist>500)

summary(trips2)
length(unique(trips2$tripID))
```

```
length(unique(trips2$ID))
length(unique(trips2$date_gmt))

sumTrips <- tripSummary(trips = trips, colony = colony)

sumTrips

head(trips2)

summary(sumTrips)
summary(sumTrips$duration/24)

tdf<-merge(trips2,sumTrips,by=c("ID","tripID"))
head(tdf)
summary(tdf$DateTime)

#tdf<-subset(tdf,DateTime<'2022-06-06 00:00:00')


summary(tdf$total_dist)


length(unique(tdf$ID)) # 40 animals
length(unique(tdf$tripID)) # 172 trips

summary(tdf$DateTime)

df<-data.frame(tdf)

head(df)



```

use EMbC package to classify behavioral modes

```{r}
head(df)

em<-data.frame(timeStamp=df$DateTime,lon=df$Longitude,
               lat=df$Latitude,id=df$tripID,bursted=TRUE)

mybc<-stbc(em)

EMbC::stts(mybc)  # behavioral states classification

expth<-data.frame(mybc@pth,mybc@X)

head(expth)

embc.out<-embc(mybc@X,U=mybc@U)
embc.out
stts(embc.out)

summary(as.factor(embc.out@A))/65890

expth$embc[embc.out@A=="1"]<-"LL"
expth$embc[embc.out@A=="2"]<-"LH"
expth$embc[embc.out@A=="3"]<-"HL"
expth$embc[embc.out@A=="4"]<-"HH"
expth$embc[embc.out@A=="5"]<-NA


summary(as.factor(expth$embc))/65890
```

```
expth$BehMode[expth$embc=="HH" |expth$embc=="HL"  ] <-"Transit"
expth$BehMode[expth$embc=="LL"] <-"Foraging"
expth$BehMode[expth$embc=="LH"] <-"Scavenging/Resting"

(summary(as.factor(expth$BehMode))/65890)*100

exp<-data.frame(tripID=expth$id,speed.ms=expth$velocity..m.s.,
                turn=expth$turn..rad.,beh.state=expth$BehMode,
                Longitude=expth$lon,Latitude=expth$lat,
                DateTime=expth$dTm)



edf<-merge(df,exp,by=c("DateTime","tripID","Longitude","Latitude"))



## now lets calculate heading direction

lt<-as.ltraj(xy=edf[3:4],date=edf$DateTime,id=edf$ID,slsp = "remove",
             infolocs=edf,typeII=T,proj4string = epsg4326)

rtd<-ld(lt)
head(rtd)
summary(rtd$abs.angle)

edf$abs.dir<-((rtd$abs.angle*180)/max(na.omit(rtd$abs.angle)))+180
edf$dist<-rtd$dist   # distance between consecutive points

summary(edf$abs.dir)


length(rtd$x)
length(df$X)
gc()

### wind timestamp is in hours, so:

edf$hourstamp<-as.POSIXct(strptime(paste(paste(year(edf$DateTime),
                                               month(edf$DateTime),
                                               day(edf$DateTime),

                                               sep="-"),
                                         paste(hour(edf$DateTime),c("00"),c("00"))),
format="%Y-%m-%d %H"))


### OBS: the date format needs to match the format of the Copernicus netcdf files!

head(edf)

# sea ice and CHL timestamps are in days

edf$daystamp<-as.POSIXct(strptime(paste(year(edf$DateTime),
                                        month(edf$DateTime),
                                        day(edf$DateTime),sep="-"),
                                  format="%Y-%m-%d", tz="GMT"))



### use this bounding box to download data from copernicus
(https://data.marine.copernicus.eu/):
```

```
summary(edf$Longitude)
summary(edf$Latitude)

summary(edf$DateTime)

# subset seasons and check time range to dowload data
dfS1<-subset(edf,DateTime<'2022-06-06 00:00:00') # 2021/22
dfS2<-subset(edf,DateTime>'2022-06-06 00:00:00') # 2022/23

summary(dfS1$DateTime)
summary(dfS2$DateTime)


```

```
# load, process and extract environemtnal variables into tracking data
# wind is matched by the hour, sea ice cover and chlorophyll by the day
# this can be time consuming depending on the capacity of your computer

```{r}

#### given the resolution of wind (in hours) two netcdf files needed to be
# downloaded separately, one for each season. Each file has over 600mb! XO


### ------- eastward wind component -----------


ncpath <- "C:/GiantSuperModel/WindUse/GiantsOnIce/Copernicus/"
ncname1 <- "cmems_obs-wind_glo_phy_nrt_l4_0.125deg_PT1H_1707309523210"
ncname2 <- "cmems_obs-wind_glo_phy_nrt_l4_0.125deg_PT1H_1707309891816"
ncfname1 <- paste(ncpath, ncname1, ".nc", sep="")
ncfname2 <- paste(ncpath, ncname2, ".nc", sep="")
dname <- "eastward_wind"
tmp_raster1 <- brick(ncfname1, varname=dname)
tmp_raster1 <- brick(ncfname2, varname=dname)
tmp_brick1 <- brick(ncfname1, varname=dname)
tmp_brick2 <- brick(ncfname2, varname=dname)

plot(tmp_brick1)

# save each raster to a file, just in case
# Loop through each layer (hour) in the brick
for (i in 1:nlayers(tmp_brick1)) {
  # Extract the raster for the current day
  tmp_raster1 <- tmp_brick1[[i]]


  # Save raster to file
  writeRaster(tmp_raster1, filename =
paste0("C:/GiantSuperModel/WindUse/WindNetCdfs/Ewind_output_hours_2021_22", i, ".tif"),
format = "GTiff", overwrite = TRUE)
}

for (i in 1:nlayers(tmp_brick2)) {
  # Extract the raster for the current day
  tmp_raster2 <- tmp_brick2[[i]]


  # Save raster to file
  writeRaster(tmp_raster2, filename =
paste0("C:/GiantSuperModel/WindUse/WindNetCdfs/Ewind_output_hours_2022_23", i, ".tif"),
format = "GTiff", overwrite = TRUE)
}
```
```

```
#tmp_brick<-brick(tmp_brick1,tmp_brick2) # join the two bricks didnt work, so lets move
each season separately


# test to see if the comand works

which(as.character(tmp_brick1@z$`Date/time`) == as.character(dfS1$hourstamp[1]))

first(tmp_brick1@z$`Date/time`)
last(tmp_brick1@z$`Date/time`)

summary(dfS1$hourstamp)

extracted_values <- list()
# extract data for season 2021/22
# OBS: with 32 gb RAM and a 13th Gen Intel(R) Core(TM) i9-13900H 2.60 GHz,
# it took roughly 20 to 30 minutes.
# R is searching the 1105 rasters for the one timestamp that matches each of the 20
thousand
# geographical fixes, then extracting the geographical
# information into that fix. Patience is a virtue required at this point.

# while you wait, try some ASCII art for relaxation

#_       `-._       `-.      `.   \       :       /   .'      .-'       _.-'       _
# `--._      `-._      `-.      `.  `.     :     .'  .'     .-'     _.-'     _.--'
#`--._     `-._     `-.     `.  \   :   /  .'    .-'    _.-'     _.--'
#`--.__      `--._     `-._    `-.  `. `. : .' .'   .-'    _.-'     _.--'      __.--'
#__      `--.__      `--._     `-._   `-.  `. \:/ .' .-'    _.-'     _.--'      __.--'      __
#  `--..__      `--.__      `--._     `-._`-._`-.`_=_'.-'_.-'     _.--'      __.--'      __..--'
#--..__      `--..__      `--.__      `--._`-q(-_-)p-'_.--'      __.--'      __..--'      __..--
#       ``--..__      `--..__      `--.__      `-'_) (_`-'      __.--'      __..--'      __..--''
#...___          ``--..__      `--..__      `--/__/  \--'      __..--'      __..--''          ___...
#      ```---...___      ``--..__      `_(<_    _/)_'      __..--''      ___...---'''
#```-----.....______```---...___(__\_\_|_/__)___...---'''_____.....-----'''
# ___    __   _________   ________   _          _   ________     ___   __   ________
#|| \\   ||      ||      ||______))  \\       //  ||______||   || \\   ||   ||______||
#||  \\_||   ___||___   ||       \\   \\___//   ||      ||   ||   \\_||   ||      ||



for (i in 1:nrow(dfS1)) {
  # Extract the timestamp, lat, and lon for the current row
  current_timestamp <- as.character(dfS1$hourstamp[i] )
  current_lat <- dfS1$Latitude[i]
  current_lon <- dfS1$Longitude[i]

  # Find the index where the timestamp matches
  timestamp_index <- which(as.character(tmp_brick1@z$`Date/time`) == current_timestamp)

  # Use the index to get the raster layer
  current_raster <- tmp_brick1[[timestamp_index]]

  # Create a SpatialPoints object
  spatial.point<-SpatialPoints(matrix(c(current_lon, current_lat), ncol = 2))

  # Extract the value at the specified coordinates, removing missing values
  extracted_value <- raster::extract(current_raster, spatial.point,
                                     method="bilinear",fun="mean")


  # Store the extracted value in the list
  extracted_values[[i]] <- c(timestamp = current_timestamp, value = extracted_value)
}
```

```
# Convert the list to a data frame
result_EwindS1 <- data.frame(do.call(rbind, extracted_values))
summary(result_EwindS1$value)
write.csv(result_EwindS1,"result_EwindS1.csv")

dfS1$Ewind<-as.numeric(result_EwindS1$value)

# another one

#
#                            _
#                         _ooOoo_
#                        o8888888o
#                        88" . "88
#                        (| -_- |)
#                        O\  =  /O
#                     ____/`---'\____
#                   .'  \\|     |//  `.
#                  /  \\|||  :  |||//  \
#                 /  _||||| -:- |||||_  \
#                 |   | \\\  -  /'| |   |
#                 | \_|  `\`---'//  |_/ |
#                 \  .-\__ `-. -'__/-.  /
#               ___`. .'  /--.--\  `. .'___
#            ."" '<  `.___\_<|>_/___.' _> \"".
#           | | :  `- \`. ;`. _/; .'/ /  .' ; |
#           \  \ `-.   \_\_`. _.'_/_/  -' _.' /
#===========`-.`___`-.__\ \___  /__.-'_.'_.-'================
#                         `=---='                    hjw # ASCII art author
#



for (i in 1:nrow(dfS2)) {
  # Extract the timestamp, lat, and lon for the current row
  current_timestamp <- dfS2$hourstamp[i]
  current_lat <- dfS2$Latitude[i]
  current_lon <- dfS2$Longitude[i]

  # Find the index where the timestamp matches
  timestamp_index <- which(as.character(tmp_brick2@z$`Date/time`) ==
as.character(current_timestamp))

  # Use the index to get the raster layer
  current_raster <- tmp_brick2[[timestamp_index]]

  # Create a SpatialPoints object
  spatial.point<-SpatialPoints(matrix(c(current_lon, current_lat), ncol = 2))

  # Extract the value at the specified coordinates, removing missing values
  extracted_value <- raster::extract(current_raster, spatial.point,
                                     method="bilinear",fun="mean")


  # Store the extracted value in the list
  extracted_values[[i]] <- c(timestamp = current_timestamp, value = extracted_value)
}

# Convert the list to a data frame
result_EwindS2 <- data.frame(do.call(rbind, extracted_values))

#summary(as.numeric(result_EwindS1))

write.csv(result_EwindS2,"result_EwindS2.csv")
```

```
dfS2$Ewind<-result_EwindS2$value
summary(dfS2$Ewind)
summary(dfS1$Ewind)

# if we got here without issues, I recommend saving the data we have so far, just in case
saveRDS(dfS1, "dfS1.Rds")
saveRDS(dfS2, "dfS2.Rds")


###-----------Northward wind component--------------------

#### now the same stuff for the northward wind component X0


ncpath <- "C:/GiantSuperModel/WindUse/GiantsOnIce/Copernicus/"
ncname1 <- "cmems_obs-wind_glo_phy_nrt_l4_0.125deg_PT1H_1707309523210"
ncname2 <- "cmems_obs-wind_glo_phy_nrt_l4_0.125deg_PT1H_1707309891816"
ncfname1 <- paste(ncpath, ncname1, ".nc", sep="")
ncfname2 <- paste(ncpath, ncname2, ".nc", sep="")
dname <- "northward_wind"
tmp_raster1 <- brick(ncfname1, varname=dname)
tmp_raster1 <- brick(ncfname2, varname=dname)
tmp_brick1 <- brick(ncfname1, varname=dname)
tmp_brick2 <- brick(ncfname2, varname=dname)

plot(tmp_brick1)

# save each raster to a file, just in case
# Loop through each layer (hour) in the brick
for (i in 1:nlayers(tmp_brick1)) {
  # Extract the raster for the current day
  tmp_raster1 <- tmp_brick1[[i]]


  # Save raster to file
  writeRaster(tmp_raster1, filename =
paste0("C:/GiantSuperModel/WindUse/WindNetCdfs/Nwind_output_hours_2021_22", i, ".tif"),
format = "GTiff", overwrite = TRUE)
}

for (i in 1:nlayers(tmp_brick2)) {
  # Extract the raster for the current day
  tmp_raster2 <- tmp_brick2[[i]]


  # Save raster to file
  writeRaster(tmp_raster2, filename =
paste0("C:/GiantSuperModel/WindUse/WindNetCdfs/Nwind_output_hours_2022_23", i, ".tif"),
format = "GTiff", overwrite = TRUE)
}

#tmp_brick<-brick(tmp_brick1,tmp_brick2) # join the two bricks didnt work, so lets move
each season separately


extracted_values <- list()

# extract data for season 2021/22

for (i in 1:nrow(dfS1)) {
  # Extract the timestamp, lat, and lon for the current row
  current_timestamp <- as.character(dfS1$hourstamp[i] )
  current_lat <- dfS1$Latitude[i]
  current_lon <- dfS1$Longitude[i]
```

```
  # Find the index where the timestamp matches
  timestamp_index <- which(as.character(tmp_brick1@z$`Date/time`) == current_timestamp)

  # Use the index to get the raster layer
  current_raster <- tmp_brick1[[timestamp_index]]

  # Create a SpatialPoints object
  spatial.point<-SpatialPoints(matrix(c(current_lon, current_lat), ncol = 2))

  # Extract the value at the specified coordinates, removing missing values
  extracted_value <- raster::extract(current_raster, spatial.point,
                                   method="bilinear",fun="mean")


  # Store the extracted value in the list
  extracted_values[[i]] <- c(timestamp = current_timestamp, value = extracted_value)
}


# wait

#           _________________________
#         ,'        _____            `.
#       ,'       _.'_____`._          `.
#      :       .'.-'  12 `-.`.           \
#      |      /,' 11  .   1 `.\           :
#      ;     // 10    |     2 \\          |
#    ,'     ::        |        ::         |
#  ,'       || 9   ---O      3 ||         |
# /         ::                 ;;         |
#:           \\ 8           4 //          |
#|            \`. 7       5 ,'/           |
#|             '.`-.__6__.-'.'            |
#:              ((-._____.-))             ;
# \             _))       ((_            /
#  `.          '--'       '--'         ,'
#    `.______________________________,'
#        ,-.
#        `-'
#           O
#            o
#             .     ____________
#            ,('`)./____________`-.-,|
#           |'-----\\--------------| |
#           |_______^______________|,|
#           |                      |   SSt
#


# Convert the list to a data frame
result_NwindS1 <- data.frame(do.call(rbind, extracted_values))
summary(result_NwindS1$value)
write.csv(result_NwindS1,"result_NwindS1.csv")

dfS1$Nwind<-as.numeric(result_NwindS1$value)
summary(dfS1$Nwind)


for (i in 1:nrow(dfS2)) {
  # Extract the timestamp, lat, and lon for the current row
  current_timestamp <- dfS2$hourstamp[i]
  current_lat <- dfS2$Latitude[i]
  current_lon <- dfS2$Longitude[i]

  # Find the index where the timestamp matches
```

```
  timestamp_index <- which(as.character(tmp_brick2@z$`Date/time`) ==
as.character(current_timestamp))

  # Use the index to get the raster layer
  current_raster <- tmp_brick2[[timestamp_index]]

  # Create a SpatialPoints object
  spatial.point<-SpatialPoints(matrix(c(current_lon, current_lat), ncol = 2))

  # Extract the value at the specified coordinates, removing missing values
  extracted_value <- raster::extract(current_raster, spatial.point,
                                     method="bilinear",fun="mean")


  # Store the extracted value in the list
  extracted_values[[i]] <- c(timestamp = current_timestamp, value = extracted_value)
}

# wait a bit longer
#      )  (
#     (   ) )
#      ) ( (
#    _______)_
# .-'---------|
#( C|/\/\/\/\/|
# '-./\/\/\/\/|
#   '_________'
#    '-------'
#


# Convert the list to a data frame
result_NwindS2 <- data.frame(do.call(rbind, extracted_values))
summary(result_NwindS2)
write.csv(result_NwindS2,"result_NwindS2.csv")

dfS2$Nwind<-result_NwindS2$value

dfS<-rbind(dfS1,dfS2)


# calcualte wind speed

dfS$windspeed<-sqrt((dfS$Ewind*dfS$Ewind)+(dfS$Nwind*dfS$Nwind))

# calcualte wind direction

dfS$wind.atan<-atan(dfS$Ewind/dfS$Nwind)
dfS$wind.dir<-((dfS$wind.atan*180)/1.58)+180


# calculate tail and head winds



# Convert wind direction to radians
dfS$wind_direction_rad <-dfS$wind.dir * (pi / 180)

# Calculate tail and head winds in bird flight direction
dfS$tail_wind <- cos(dfS$abs.dir) - cos(dfS$wind_direction_rad)*dfS$windspeed
dfS$head.tails<-ifelse(dfS$tail_wind<0,"Head","Tail")

dfS$heaedtailN<-sqrt(dfS$tail_wind*dfS$tail_wind)

head(dfS)
```

```
# now that we achieved illumination through suffering, we can move on



###--------sea ice---------------



ncname <- "cmems_mod_glo_phy_anfc_0.083deg_P1D-m_1707308192914"
ncfname <- paste(ncpath, ncname, ".nc", sep="")
dname <- "siconc"
tmp_raster <- brick(ncfname, varname=dname)
tmp_brick <- brick(ncfname, varname=dname)
plot(tmp_brick)


# Loop through each layer (hour) in the brick
for (i in 1:nlayers(tmp_brick)) {
  # Extract the raster for the current day
  tmp_raster <- tmp_brick[[i]]


  # Save raster to file
  writeRaster(tmp_raster, filename =
paste0("C:/GiantSuperModel/WindUse/SICNetCdfs/Siconc_output_days_", i, ".tif"), format =
"GTiff", overwrite = TRUE)
}

tmp_brick@z$`Date/time`

plot(tmp_brick)

extracted_values <- list()


for (i in 1:nrow(dfS)) {
  # Extract the timestamp, lat, and lon for the current row
  current_timestamp <- dfS$daystamp[i]
  current_lat <- dfS$Latitude[i]
  current_lon <- dfS$Longitude[i]

  # Find the index where the timestamp matches
  timestamp_index <- which(as.character(tmp_brick@z$`Date/time`) ==
as.character(current_timestamp))

  # Use the index to get the raster layer
  current_raster <- tmp_brick[[timestamp_index]]

  # Create a SpatialPoints object
  spatial.point<-SpatialPoints(matrix(c(current_lon, current_lat), ncol = 2))

  # Extract the value at the specified coordinates, removing missing values
  extracted_value <- raster::extract(current_raster, spatial.point,
                                     method="bilinear",fun="mean",na.rm=T)


  # Store the extracted value in the list
  extracted_values[[i]] <- c(timestamp = current_timestamp, value = extracted_value)
}

# Convert the list to a data frame
result_SIC <-data.frame(do.call(rbind, extracted_values))
summary(result_SIC)
```

```
write.csv(result_SIC,"result_SIC.csv")

dfS$SIC<-result_SIC$value



### ---------Chlorophyll-a concentration --------------

ncname <- "cmems_mod_glo_bgc-pft_anfc_0.25deg_P1D-m_1707308783337"
ncfname <- paste(ncpath, ncname, ".nc", sep="")
dname <- "chl"
tmp_raster <- brick(ncfname, varname=dname)
tmp_brick <- brick(ncfname, varname=dname)
plot(tmp_brick)


# Loop through each layer (hour) in the brick
for (i in 1:nlayers(tmp_brick)) {
  # Extract the raster for the current day
  tmp_raster <- tmp_brick[[i]]


  # Save raster to file
  writeRaster(tmp_raster, filename =
paste0("C:/GiantSuperModel/WindUse/CHLNetCdfs/CHL_output_days_", i, ".tif"), format =
"GTiff", overwrite = TRUE)
}


extracted_values <- list()


for (i in 1:nrow(dfS)) {
  # Extract the timestamp, lat, and lon for the current row
  current_timestamp <- dfS$daystamp[i]
  current_lat <- dfS$Latitude[i]
  current_lon <- dfS$Longitude[i]

  # Find the index where the timestamp matches
  timestamp_index <- which(as.character(tmp_brick@z$`Date/time`) ==
as.character(current_timestamp))

  # Use the index to get the raster layer
  current_raster <- tmp_brick[[timestamp_index]]

  # Create a SpatialPoints object
  spatial.point<-SpatialPoints(matrix(c(current_lon, current_lat), ncol = 2))

  # Extract the value at the specified coordinates, removing missing values
  extracted_value <- raster::extract(current_raster, spatial.point,
                                     method="bilinear",fun="mean",na.rm=T)


  # Store the extracted value in the list
  extracted_values[[i]] <- c(timestamp = current_timestamp, value = extracted_value)
}

# Convert the list to a data frame
result_CHL <-data.frame(do.call(rbind, extracted_values))
summary(result_CHL)
write.csv(result_CHL,"result_CHL.csv")

dfS$CHL<-result_CHL$value
```

```
write.csv(dfS,"C:/GiantSuperModel/WindUse/tracking_data_filtered.csv")


#### we finally have the data to start the analysis!



# some last editing and it's done

dfS$Trip<-stringr::str_sub(dfS$tripID,-2,-1) # Trip number from trip ID

dfS$Ring<-substring(dfS$ID,first=1,last=6)   # ring number from trip ID


head(dfS)

saveRDS(dfS,"C:/GiantSuperModel/WindUse/GiantsOnIce/GiantPetrels_processed_trips.Rds")


# next code is for  figure 2:


head(sumTrips)
sumTrips$days<-as.numeric(sumTrips$return-sumTrips$departure)/24

ggplot(subset(df,DateTime<'2022-06-06 00:00:00'),
       aes(DateTime,ColDist/1000,colour=ID))+geom_line(linewidth=1,alpha=0.5)+
  theme_bw()+ylab("Straight line distance from colony (km)")+

  ggplot(subset(df,DateTime>'2022-06-06 00:00:00'),
         aes(DateTime,ColDist/1000,colour=ID))+geom_line(linewidth=1,alpha=0.5)+
  theme_bw()+ylab("Straight line distance from colony (km)")+


ggplot(subset(sumTrips,departure<'2022-06-06 00:00:00'),
       aes(direction,total_dist+1/1000,colour=days,size=days))+geom_point()+
  scale_y_log10(limits=c(1,10000))+theme_bw()+
  scale_colour_distiller(palette="Spectral",name="Duration (days)")+
  coord_polar()+
  scale_x_continuous(limits=c(0, 360),breaks = c(0,90,180,270))+
  theme_bw()+ylab("Total trip distance (km)")+


ggplot(subset(sumTrips,departure>'2022-06-06 00:00:00'),
       aes(direction,total_dist+1/1000,colour=days,size=days))+geom_point()+
  scale_y_log10(limits=c(1,10000))+theme_bw()+
  scale_colour_distiller(palette="Spectral",name="Duration (days)")+
  coord_polar()+
  scale_x_continuous(limits=c(0, 360),breaks = c(0,90,180,270))+
  theme_bw()+ylab("Total trip distance (km)")



```