R Tutorial to fit both DPMLE methods from the paper "Improved order selection method for hidden Markov models: a case study with movement data".

1. Setup and data preparation

In this R tutorial, we illustrate the implementation of the double penalized likelihood method to perform order selection described in the main text, using the source functions provided. We begin by first loading relevant packages and sourcing custom helper functions.

```r
library(momentuHMM) # Package for fitting Hidden Markov models (HMMs)
library(boot)       # Required to use the logit function
library(tidyverse)  # Collection of R packages for data manipulation and visualization
library(pracma)     # Functions for numerical analysis and mathematical computing
library(dirmult)    # Functions for working with Dirichlet-multinomial distributions
library(brms)       # Interface to fit Bayesian models using Stan
library(ggOceanMaps)# To get the land data
library(ggspatial)  # To plot spatial objects
library(geosphere)  # needed for ggOceanMaps
source("tutorial_source.R") # Custom functions, use the "sourcefunctionsNH.R" file that
# contains all the function used in the EM algorithm.
```

We load the dataset, which is a subset of the data used in the paper, focusing only on a single track (track ID = 10). Let us first have a look at the data.

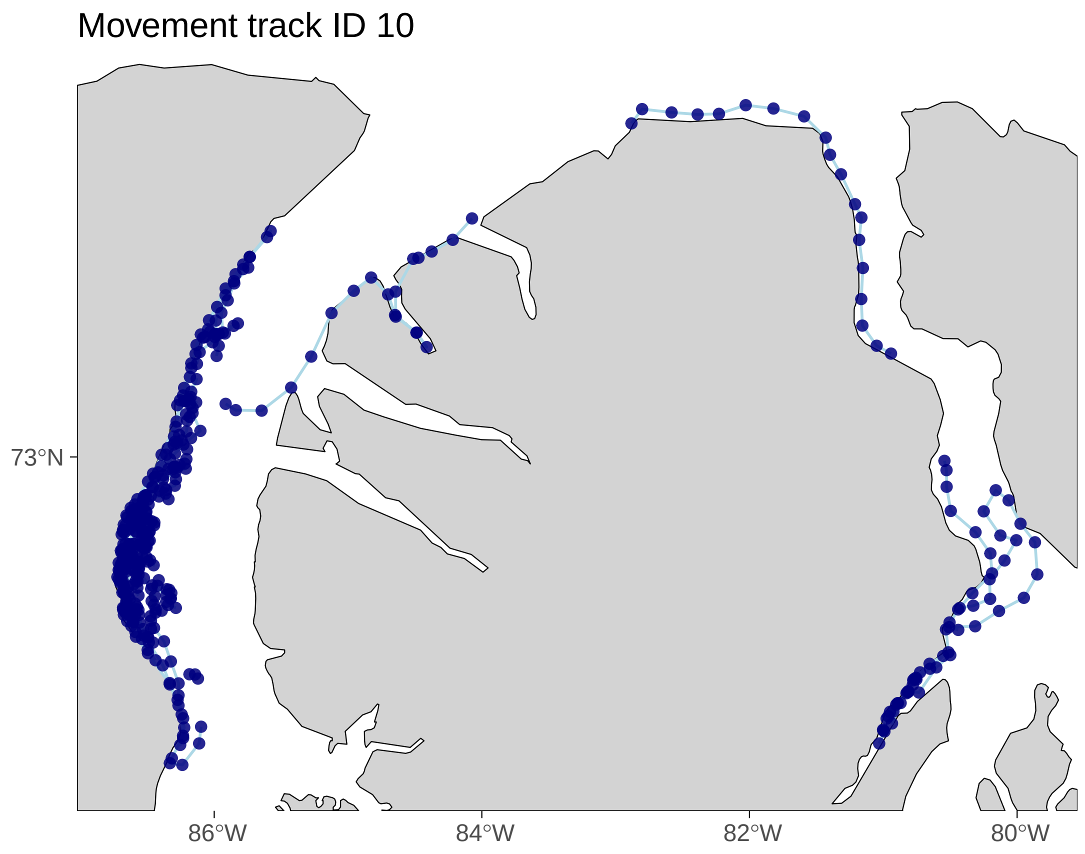
```r
setwd(your_directory)
NarwhalData <- readRDS("OneNarwhal_Tutorial.RData")
head(NarwhalData)
```

```
## # A tibble: 6 x 5
##      ID time                    x     y dtoshore
##   <int> <dttm>              <dbl> <dbl>    <dbl>
## 1    10 2017-08-03 22:00:00 -81.0  72.4    0.542
## 2    10 2017-08-03 23:00:00 -81.0  72.4    0.941
## 3    10 2017-08-04 00:00:00 -81.0  72.4    0.792
## 4    10 2017-08-04 01:00:00 -80.9  72.5    0.402
## 5    10 2017-08-04 02:00:00 -80.8  72.5    0.943
## 6    10 2017-08-04 03:00:00 -80.8  72.5    0.814
```

The dataset contains five columns: the `ID` of the track (in this example, it is always 10, but in the original dataset, it ranges from 1 to 18). `time` represents the time of measurement. The `x` column represents longitude, and `y` latitude. Locations were recorded at irregular time intervals, and were regularized before the analysis. Hidden Markov models (HMMs) assume that locations are recorded at regular time intervals and that there is negligible position error. This tutorial does not cover the process of regularizing location data, but it is well explained in this TUTORIAL. This tutorial focuses on order selection methods. The column `dtoshore` corresponds to the covariate `distance to shore`. Note that some of the values for `dtoshore` are 0, which is not possible for narwhals. We could correct it by setting these values to the minimum distance to shore detectable (here it is ~ 0.01). However 0.01 being close to 0, this should not really affect the inference and we will keep the dataset as such. However, this is something to keep in mind.

We can also plot the track.

```r
df <- as.data.frame(cbind(Lat = NarwhalData$y,Lon = NarwhalData$x))

# Get the land data within the box "bbox".
bbox <-c(min(na.omit(df$Lon))-0.3,max(na.omit(df$Lon))+0.3,
         min(na.omit(df$Lat))-0.1,max(na.omit(df$Lat))+0.1)


basemap(limits = c(bbox), crs = 4326,land.col="lightgrey") +
  ggspatial::geom_spatial_path(data=df, aes(x = Lon, y = Lat),colour="lightblue")+
  ggspatial::geom_spatial_point(data=df, aes(x = Lon, y = Lat),colour="navy",alpha=0.85)+
  ggtitle("Movement track ID 10") +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.title = element_blank()
  )
```



We use the function `prepData` from the package `momentuHMM` to compute step length (distance between two consecutive locations) and turning angle (angle between two steps) from the track. These are the data streams will we use to infer narwhal behavior. We are using WGS84 for lon/lat. We set the following arguments:

- `"type"`: whether it's easting/northing coordinate system (`type="UTM"`) or whether it's longitude/latitude (`type = "LL"`). We are using `"LL"`.
- `"coordNames"`: the names of the columns with the coordinates. So in our case `lon` and `lat`.
- `"covNames"`: "time","dtoshore".

The argument `covNames` distinguishes the model covariates from data streams.

If there are missing covariate values (which is the case in our dataset), the default is to fill the missing value with the nearest (in time) value. This can be reasonable for spatial covariates, but wouldn't be sensible for a covariate such as time of day.

```
exData <- NarwhalData
#Transform it into a data.frame
exData <- as.data.frame(exData)
#Compute the column step and angle
exData <- momentuHMM::prepData(exData,type=c("LL"),
                               coordNames = c("x", "y"),
                               covNames=c("time","dtoshore"))
```

Now we can have a look at the transformed data.

```
head(exData)
```

```
##   ID     step      angle        x       y                time  dtoshore
## 1 10 3.193875         NA -81.0305 72.3757 2017-08-03 22:00:00 0.5422048
## 2 10 3.187095  0.2428769 -80.9917 72.4018 2017-08-03 23:00:00 0.9408257
## 3 10 4.160123 -0.4370014 -80.9748 72.4299 2017-08-04 00:00:00 0.7921033
## 4 10 4.359697 -0.1616067 -80.9033 72.4603 2017-08-04 01:00:00 0.4019605
## 5 10 3.628045  0.1934962 -80.8122 72.4881 2017-08-04 02:00:00 0.9433606
## 6 10 4.292022 -0.3311030 -80.7525 72.5152 2017-08-04 03:00:00 0.8141269
```

We have two additional columns `step` and `angle` that the prepData function derived. When we provide longitude and latitude, steps are derived in kms. Now the data is ready to be fitted. We start by fitting the simplest model, that does not assume any covariate effect on the t.p.m: the stationary DPMLE.

2. Stationary DPMLE.

We start by setting the parameters for the algorithm. Both the stationary and the non-stationary versions of the DPMLE are set with the same parameters. As in the paper, λ and C_N are chosen randomly. Here we only explore *one* randomly chosen value, however, multiple values should be explored (e.g., 100 values are explored in the narwhal case study, 50 in the simulation study) and the model minimizing the Narwhal Information Criterion (NIC) criterion should be selected (an example of code is given in the last section of this tutorial). The upper bound N is not a parameter of the DPMLE function itself. Instead, it is carried in the dimension of the initial parameters of the state-dependent distributions. We set the following arguments:

- `"NbIter"`: maximum number of iterations for the EM algorithm.

- `"epsilon"`: threshold value to stop the EM algorithm i.e., the algorithm stops if $|\mathcal{L}^{i-1} - \mathcal{L}^{i}| < \epsilon$, where $\mathcal{L}^{i-1}$ is the joint negative double penalized log-likelihood at iteration $i-1$.
- `"nbSteps"`: length of the time series (number of steps).
- `"aInd"`: vector of time of first observation for each individual (vector of size NbAnimals).
- `"Obs"`: matrix of observations, with ncol = number of datastreams.
- `"Pi0"` : initial distribution (i.e., stationary probability in stationary HMM) or initial distributions (non-stationary HMM). It should be a vector of size nbStates.
- `"trMat0"`: initial t.p.m of dimension nbStates x nbStates, only required for stationary DPMLE, it does not allow for 0's.
- `"beta0"` : initial regression vector for the t.p.m, only required for non-stationary DPMLE.
- `"lmu0"`: logarithm of the initial mean values for gamma distributions.
- `"lsd0"`: logarithm of the initial standard deviation values for gamma distributions.
- `"lambda"`: hyperparameter for double penalized likelihood, for the SCAD function.
- `"a"`: a = 3.7, constant of the SCAD function.
- `"Cn"`: hyperparameter for double penalized likelihood, for the penalty applied on the stationary probabilities.
- `"type"`: list of two elements to describe which method to be applied: "multi-uni" dimensional and "NS-S" for non-stationary or stationary.

```r
# Number maximum of iterations for the EM algorithm
NbIter <- 100
# Threshold (stopping rule)
epsilon <- 1e-3
# Length of the time series
nbSteps <- length(exData$step)
# Vector of time of first observation for each individual (Vector of size NbAnimals)
aInd <- 1 # there is only one individual,
# for multiple individuals: c(1,cumsum(table(exData$ID))[1:(nbAnimals-1)]+1)
# Vector of observations
Obs <- cbind(exData$step,exData$angle)
# Hyperparameters chosen randomly
set.seed(456789)
lambda <- exp(runif(1,1,5))
Cn <- (runif(1,1,5))
a <- 3.7
typeS <- list("multi","S")   # multidimensional + stationary DPMLE
typeNS <- list("multi","NS") # multidimensional + non-stationary DPMLE
```

We use a seed to ensure the reproducibility of the results. We set the stopping rule at $1e-3$. Commonly, it is better to set it to an even smaller value (e.g., we used $1e-5$ in the paper) to get more accurate estimates but it is more time consuming. In this tutorial, we set it to $1e-3$ to ensure faster convergence.

To model turning angles, we need a circular distribution, which is a continuous probability distribution that is defined on the circle. The von Mises distribution is a unimodal circular distribution with a location parameter (mean direction) and a dispersion parameter. Other distributions can be chosen, such as the wrapped Cauchy that has fatter tails (equivalent to replacing the normal by a Student-t or Cauchy in non-circular statistics). To model step-lengths, we use a gamma distribution.

First, let us define initial values. The values we use were obtained from momentuHMM version 1.5.5, run on R 4.3.1. In the last section, we provide a complete example of how to obtain initial values using momentuHMM.

```r
# Upper bound for DPMLE
j <- 4

# Initial parameters for emission distribution
# step length
lmu0 <- log(c(6.05, 6.21, 4.18, 1.97)) # initial value of the means of gamma distribution
lsd0 <-  log(c(1.37, 2.07, 1.3, 1.19)) # initial value of the sds of gamma distribution
# Turning angle
lkappa0 <- log(c(1.84, 22.74, 7.4, 0.47)) # concentration parameter of von Mises

# Initial parameters for hidden process
delta0 <- c(0.14, 0.16, 0.25, 0.44) # initial distribution
trMat0 <- matrix(c(
  0.74, 0.26, 1e-15, 1e-15,
  1e-13, 0.71, 0.29, 1e-15,
  0.06, 0.04, 0.59, 0.31,
  0.05,1e-15,  0.12, 0.82
), nrow = 4, byrow = TRUE) # transition probability matrix.
```

We can now fit the stationary DPMLE to the data. It should take a little less than 3 minutes.

```r
stationaryRes <-  DPMLE(NbIter= NbIter ,epsilon = epsilon,
            Pi0 = delta0,trMat0 = trMat0,lmu0 = lmu0,lsd0 = lsd0,lkappa0 = lkappa0,
            nbSteps=nbSteps,aInd = aInd,
            Obs = Obs,
            lambda=lambda,a = 3.7,Cn = Cn,
            type=typeS)
```

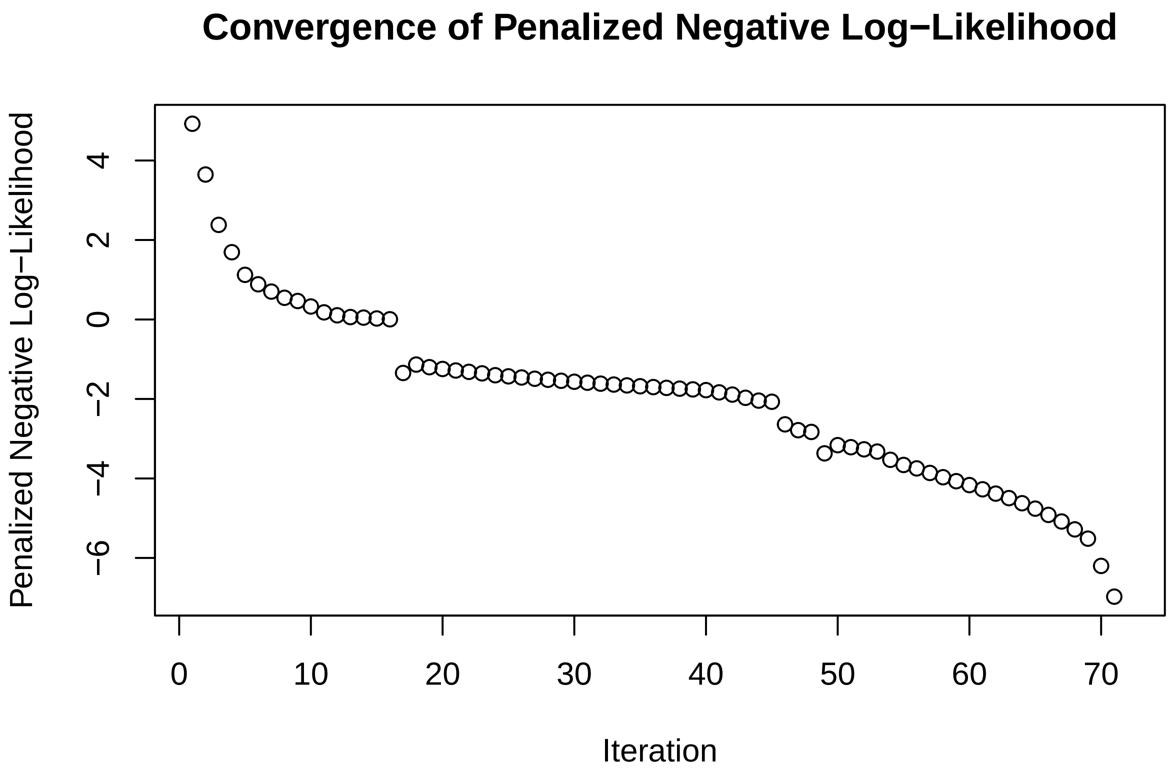
We can assess whether the model converged by looking at the iteration number at which the EM algorithm stopped. Here, it stopped before the total number of iterations allowed: `NbIter` = 100 Thus, we know that the model converged. However, it is a good habit to look at the `convergence` value returned by optimization algorithms. To check whether the model converged properly, we can look at the argument `convergence` that returns 1 if the iteration limit (`NbIter`) has been reached and 0 otherwise.

```r
stationaryRes$convergence
```

```
## [1] 0
```

The negative double penalized log-likelihood is also a good indicator of the behavior of the algorithm: it should be decreasing at each iteration and reaching a plateau as in the graph plotted. Note that if you plotted the negative likelihood (or log-likelihood) instead of the double penalized negative log-likelihood, you might observe an increase. This increase may surprise some as the negative likelihood should decrease during maximum likelihood inference. However, remember that to prevent from overfitting, we are minimizing the negative *double penalized* log-likelihood rather than the negative likelihood. For the DPMLE methods, it is thus more appropriate to look at the behavior of the double penalized likelihood to assess model convergence. Below we plot the negative log-likelihood on a log(NLL-min(NLL)) scale, to enhance sizes of improvement as the algorithm gets close to the MLE.

```r
plot(log(stationaryRes$nllpen-min(stationaryRes$nllpen)),
     xlab = "Iteration",
     ylab = "Penalized Negative Log-Likelihood",
     main = "Convergence of Penalized Negative Log-Likelihood")
```



Now let us have a look at the number of states selected by the non-stationary DPMLE. It is found in `K`.

```r
nbStates <- stationaryRes$K
nbStates
```

```
## [1] 2
```

The model selects *two* states. Let us have a better look at the states selected. From now, we refer to the `merged states` to describe the two resulting states, and `estimated states` to describe the N estimated states from the DPMLE. The `estimated states` will eventually be merged to form the `merged` states. The final estimates of interest are from the `merged` states, but first let's look at the original `estimated states`.

The estimated emission parameters are provided in `theta`. This is a vector of size `Number of parameters x N`, where N corresponds to the upper bound (here 4) and `Number of parameters` corresponds to the number of parameters of the state-dependent distribution (e.g., 2 for gamma distribution). It provides the parameters of the gamma distributions first and then von Mises. The number corresponds to the state it is associated to. For example mu1 is the mean of the gamma distribution associated with state `1`.

```r
stationaryRes$theta
```

```
##       mu1       mu2       mu3       mu4       sd1       sd2       sd3       sd4
## 5.1427134 5.1427123 5.1427142 2.4252018 1.3023851 2.5120067 1.4993710 1.5231918
##    Kappa1    Kappa2    Kappa3    Kappa4
## 2.8940856 2.8940833 2.8940862 0.8664708
```

Do you see that it merges state 1, 2 and 3 ? The differences in standard deviation don't matter because we don't penalize for it. If it would be of interest, we would have to use the GSF method, to penalize both the mean and the sd.

As previously said, the model merges the three estimated states 1, 2 and 3 together. From it, we can compute the estimates of the merged states by taking the average. For example, the gamma mean of merged state 1 is the average of the means of estimated states 1,2 and 3.

```
# Take the average of the mean, sd and concentration parameters.
thetaS <- cbind(mu=c((stationaryRes$theta[1] +
                      stationaryRes$theta[2] +
                      stationaryRes$theta[3])/3,
                      stationaryRes$theta[4]),
               sd=c((stationaryRes$theta[1+4] +
                     stationaryRes$theta[2+4] + stationaryRes$theta[3+4])/3,
                     stationaryRes$theta[4+4]))
rownames(thetaS) <- c("Merge_state 1","Merge_state 2")

# From mean and sd, get shape and scale of gamma distribution
shapeS <- (thetaS[,1]  * thetaS[,1]  )/ (thetaS[,2]  * thetaS[,2] )
scaleS <- (thetaS[,2] * thetaS[,2]) / thetaS[,1]

# Same indices + 8, because there are 3 parameters to estimate for 4 states.
# The first four are the mean of the gamma distribution,
# then the sd of gamma distribution and then the concentration of the von Mise distribution.
kappaS = c((stationaryRes$theta[1+8] + stationaryRes$theta[2+8] + stationaryRes$theta[3+8])/3,
           stationaryRes$theta[4+8])
```

The estimated t.p.m. is provided by the `trMat` output. It is a matrix of dimension `N x N`. From it, we can compute the merged transition matrix of dimension `NbStates x NbStates`. To do so, observe that estimated state 1 transitioning to estimated state 2 corresponds to the merged state 1 transitioning to itself, and similarly for `state 1` transitioning to `state 3`. From that, we can derive the merged trMat.

```
trMatS <- array(dim=c(2,2))

# Merged state 1 = estimated states 1, estimated states 3, and estimated states 4.
# We take the average of the estimates, i.e., we divide by 3.

trMatS[1,1] <- (stationaryRes$trMat[1,1] +
            stationaryRes$trMat[1,2] +
            stationaryRes$trMat[1,3] + # from state 1 to the merged states 1
            stationaryRes$trMat[2,1] +
            stationaryRes$trMat[2,2] +
            stationaryRes$trMat[2,3] +  # from state 2 to the merged states 1
            stationaryRes$trMat[3,1] +
            stationaryRes$trMat[3,2] +
            stationaryRes$trMat[3,3])/3 # from state 3 to the merged states 1

trMatS[1,2] <- (stationaryRes$trMat[1,4]  + # from state 1 to the merged state
# = first merged state 1 (estimated state 1) to merged states 2 (here it is the estimated state 4)
                stationaryRes$trMat[2,4]  +  # from state 3 to the merged state
# = second merged state 1 (estimated state 2) to merged states 2 (estimated state 4)
            stationaryRes$trMat[3,4])/3      # from state 4 to the merged state
# = third merged state 1 (estimated state 3) to merged states 2 (estimated state 4)


# Merged state 2 = estimated state 2
trMatS[2,2] <- stationaryRes$trMat[4,4]   # from state 4 to the merged state 2
# (here it is the estimated state 4) = merged state 2 to merged states 2.

trMatS[2,1] <- stationaryRes$trMat[4,1] + # from state 2 to first element of the merged state 1
               stationaryRes$trMat[4,2] + # from state 2 to second element of the merged state 1
               stationaryRes$trMat[4,3]   # from state 2 to third element of the merged state 1

# Derive the stationary probabilities from the t.p.m, Eq (15).
piS <- base::solve(t(diag(nbStates)-trMatS+1), rep(1,nbStates))
```

Now we can have a look at the merged estimates.

```
thetaS
```

```
##                       mu       sd
## Merge_state 1 5.142713 1.771254
## Merge_state 2 2.425202 1.523192
```

```
kappaS
```

```
##    Kappa1    Kappa4
## 2.8940850 0.8664708
```

```
trMatS
```

```
##           [,1]       [,2]
## [1,] 0.9069349 0.09306513
## [2,] 0.1021671 0.89783290
```

High concentration parameter values ($\kappa \geq 2$) mean that the animal has a tendency to move in the same direction. Values close to 0 mean that the turning angle distribution is almost uniform (the animal turns in all directions). A very good tool to visualize the results is to get the decoded states sequence from the forward and backward probabilities and plot the points according to their assigned states. To do so, we first need to compute the diagonal matrix of emission distribution of the data depending on the parameter estimates.

```
# Initialize lnProbsS
lnProbsS <- array(0, dim=c(nbSteps,2)) #ncol = 2 because there are two merged states.
# Only steps at time t = 1 because angle needs two observations minimum.
lnProbsS[1,] <- apply(cbind(shapeS,scaleS), 1,
                      FUN=function(x)dgamma(Obs[1,1],shape=x[1],scale=x[2],log=TRUE))

# Diagonal matrix of log emission distributions for all observations.
```

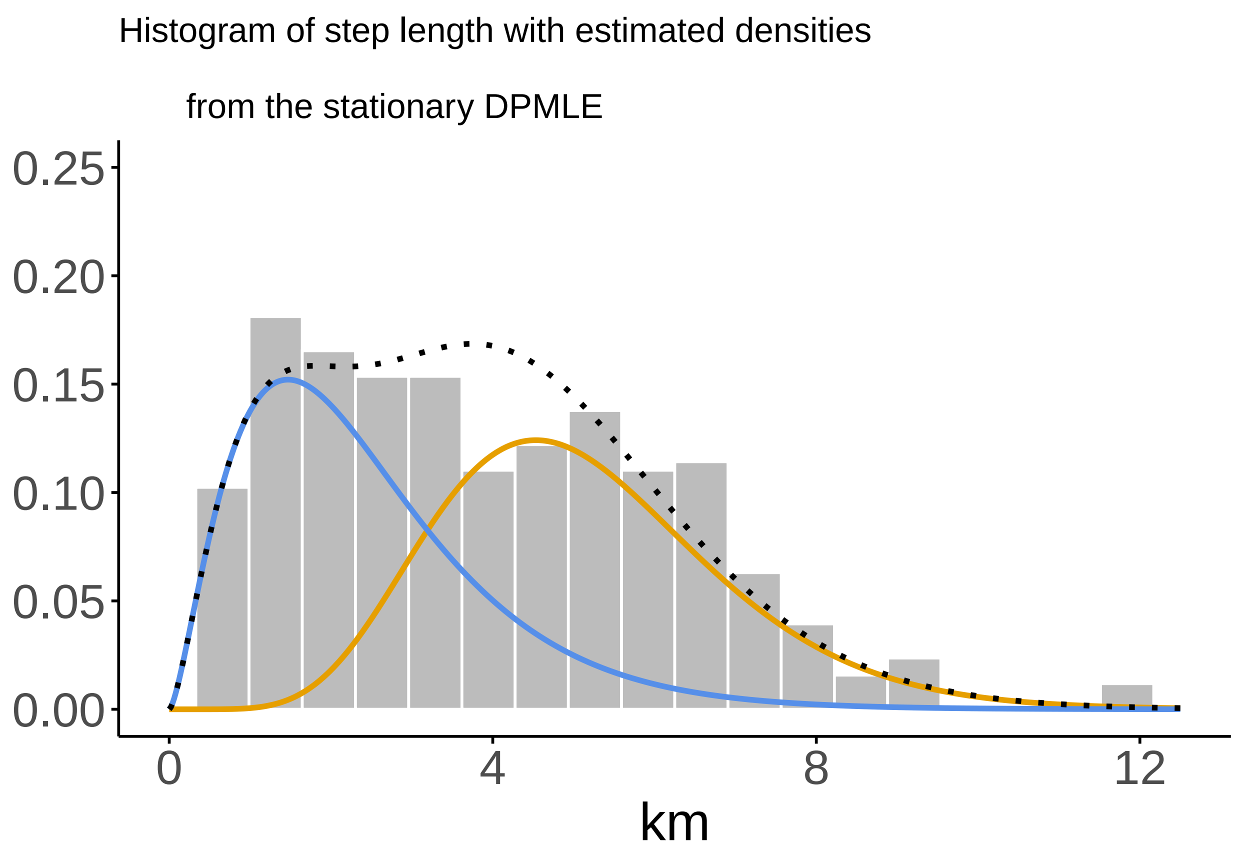
```
for(t in (2:nbSteps)){
  if(!is.na(Obs[t,1])){lnProbsS[t,] <- apply(cbind(shapeS,scaleS),1,
                                          FUN=function(x)dgamma(Obs[t,1],
                                                                shape=x[1],
                                                                scale=x[2],
                                                                log=TRUE))}
  if(!is.na(Obs[t,2])){lnProbsS[t,] <- lnProbsS[t,]+sapply(kappaS,
                                          FUN=function(x)dvon_mises(Obs[t,2],
                                                                   mu=0,
                                                                   kappa=x,
                                                                   log=TRUE))}
}
```

The functions `logAlpha` and `logBeta` are in the `sourcefunctionsNH.R` file. They compute the forward and backward log-probabilities.

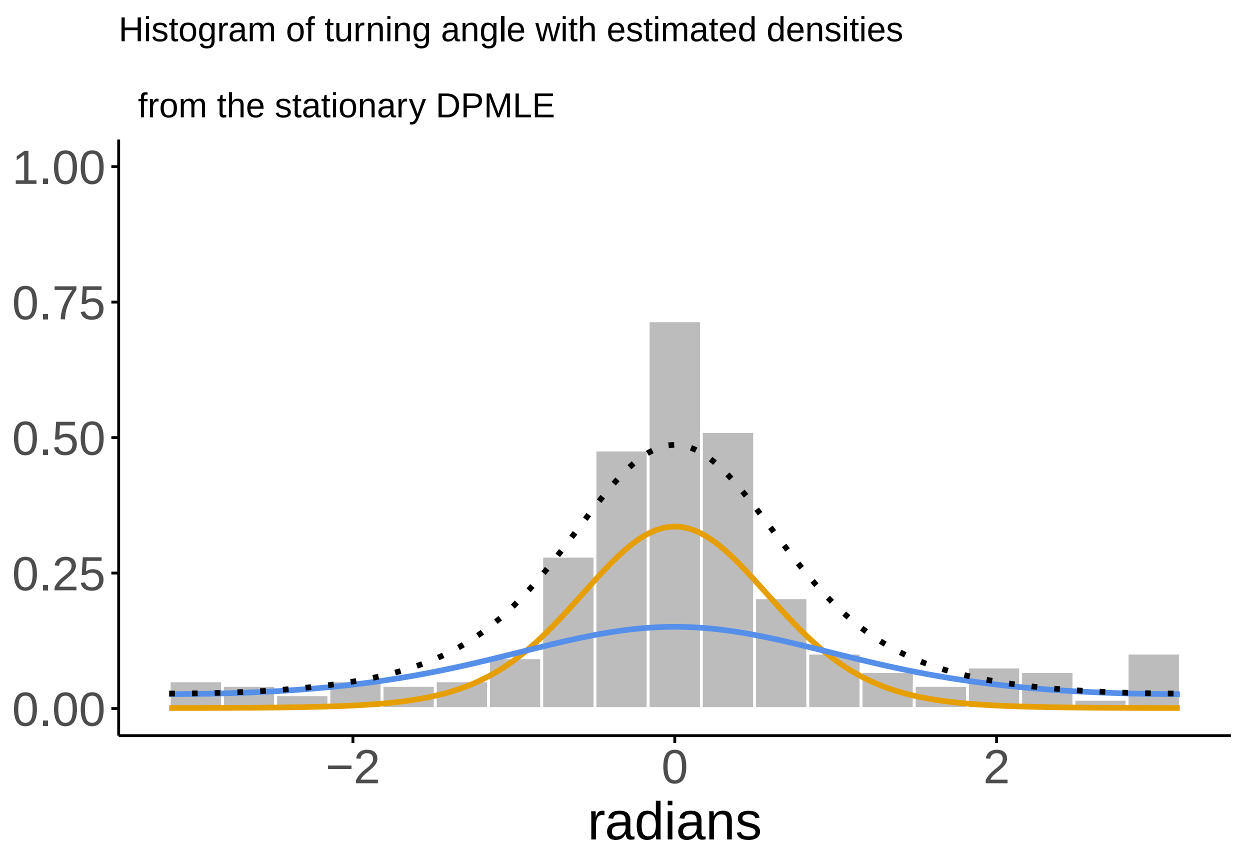
```
# Compute the forward and backward log-probabilities for stationary HMM
lalphaS <- logAlpha(piS, trMatS, lnProbsS, nbSteps, aInd)
lbetaS <- logBeta(piS, trMatS, lnProbsS, nbSteps, aInd)

# Compute the negative log-likelihood
negllk <- forward_alg(piS,trMatS,lnProbsS,nrow(lnProbsS),1)

# Initialize vector of state probabilities
ProbaState <- array(dim = c(nbSteps,2))
for (t in (1:length(exData$step))){
  for(j in (1:nbStates)){
    ProbaState[t,j] <- exp(lalphaS[t,j] + lbetaS[t,j]  + negllk)
  }
}

# Compute the vector of the most likely state (i.e., the one with the highest probability)
# at each time (i.e., for each row).
states <- apply(ProbaState, 1, which.max)

# Add the column of states to the data.
data <- mutate(exData,states=states)
```

The Viterbi algorithm can also be used and R code can be found in Zucchini et al. (2017). Now, we can the observed step length and turning angle frequencies, colored according to their assigned states and overlay the estimated distribution for each state on top of it.

```
density <- data.frame(x = seq(0, 70, by = 0.02),
                      y1 = dgamma(seq(0, 70, by = 0.02),
                                  shape = shapeS[1], scale = scaleS[1])*piS[1],
                      y2 = dgamma(seq(0, 70, by = 0.02),
                                  shape = shapeS[2], scale = scaleS[2])*piS[2])
density$sum <- rowSums(density[, -1])
density$ytotal <- density$y1 + density$y2
ggplot() +
  geom_histogram(data = data.frame(x = exData),
                 aes(x = x.step, y = ..density..),
                 bins = 20,
                 alpha=0.4,
                 color="white") +
  geom_line(data = density, aes(x = x, y = y1),
            color ="#E69F00",
            size = 1) +
  geom_line(data = density, aes(x = x, y = y2),
            color = "#568fe9",
            size = 1) +
  geom_line(data = density, aes(x = x, y = ytotal),
            color ="black",
            linetype="dotted",
            size = 1) +
  labs(title ="",
       y = "Density",
       x = NULL) +
  ylim(0,0.25) +
  xlim(0, 12.5) +
  labs(x = "km",
       title ="Histogram of step length with estimated densities \n
       from the stationary DPMLE") +
    theme(
    panel.background = element_blank(),
    plot.background = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.line = element_line(color = "black"),
    axis.title.y = element_blank(),
    legend.axis.line = element_blank(),
    axis.ticks = element_line(color = "black", size = 0.5),
    axis.text = element_text(size = 18),
    axis.title = element_text(size = 20))
```



```r
density <- data.frame(x = seq(-pi,pi,by=0.02),
                     y1 = dvon_mises(seq(-pi,pi,by=0.02),
                                     mu = 0, kappa = kappaS[1])*piS[1],
                     y2 = dvon_mises(seq(-pi,pi,by=0.02),
                                     mu = 0, kappa = kappaS[2])*piS[2])
density$sum <- rowSums(density[, -1])
density$ytotal <- density$y1 + density$y2

ggplot() +
  geom_histogram(data = data.frame(x = exData),
                 aes(x = x.angle, y = ..density..),
                 bins = 20,
                 alpha=0.4,
                 color="white") +
  geom_line(data = density, aes(x = x, y = y1),
            color ="#E69F00",
            size = 1) +
  geom_line(data = density, aes(x = x, y = y2),
            color = "#568fe9",
            size = 1) +
  geom_line(data = density, aes(x = x, y = ytotal),
            color ="black",
            linetype="dotted",
            size = 1) +
  labs(title ="Histogram of turning angle with estimated densities \n
  from the stationary DPMLE",
       y = "Density",
       x = NULL) +
  ylim(0,1) +
  xlim(-pi, pi) +
  labs(x = "radians") +
    theme(
    panel.background = element_blank(),
    plot.background = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.line = element_line(color = "black"),
    axis.title.y = element_blank(),
    legend.axis.line = element_blank(),
    axis.ticks = element_line(color = "black", size = 0.5),
    axis.text = element_text(size = 18),
    axis.title = element_text(size = 20))
```



The estimated distributions for the step length appear to represent the data properly. The turning angle histogram exhibits some extreme turning angles values. Such extreme turn angles can often be attributed to location errors (Hurford, 2009). Based on the mean step length parameters, it looks like movement of type `2` (merged state `2`) is slower than movement of type `1`. This is particularly easy to see in the step length histogram (first figure). The concentration parameter of state `1` is larger than that of state `2`, which indicates that movement of type `2` is slower and with lower directional persistence than movement of type `1`.

Now we can fit the non-stationary HMM with the covariate `distance to shore` and assess whether it is significant or not.

3. Non-stationary DPMLE with `distance to shore`

Now we will investigate whether `distance to shore` better explains the movement of our narwhal, while also estimating the model parameters and the number of states. To do so, we will fit the non-stationary DPMLE. We use the same values for λ and C_N and we chose the initial values randomly. We start by first fitting a standard non-stationary HMM with covariate `distance to shore` in the t.p.m. with `fitHMM` from the package momentuHMM. We provide the estimated parameters from the standard HMM and define them as the initial values of the DPMLE function. The complete code to use momentuHMM is provided in the last section.

Remember that the non-stationary DPMLE takes `beta0` for argument unlike the stationary DPMLE that takes `trMat0`.

We have to define the vector of covariates. We call it `cov` and we also specify the first column of intercept. If `cov` is only a column of `1` then a non-stationary HMM with no covariate in the t.p.m is fitted to the data.

```
# ncol = number of covariates + 1
cov <- matrix(c(rep(1,nbSteps), exData$dtoshore), ncol=2)
```

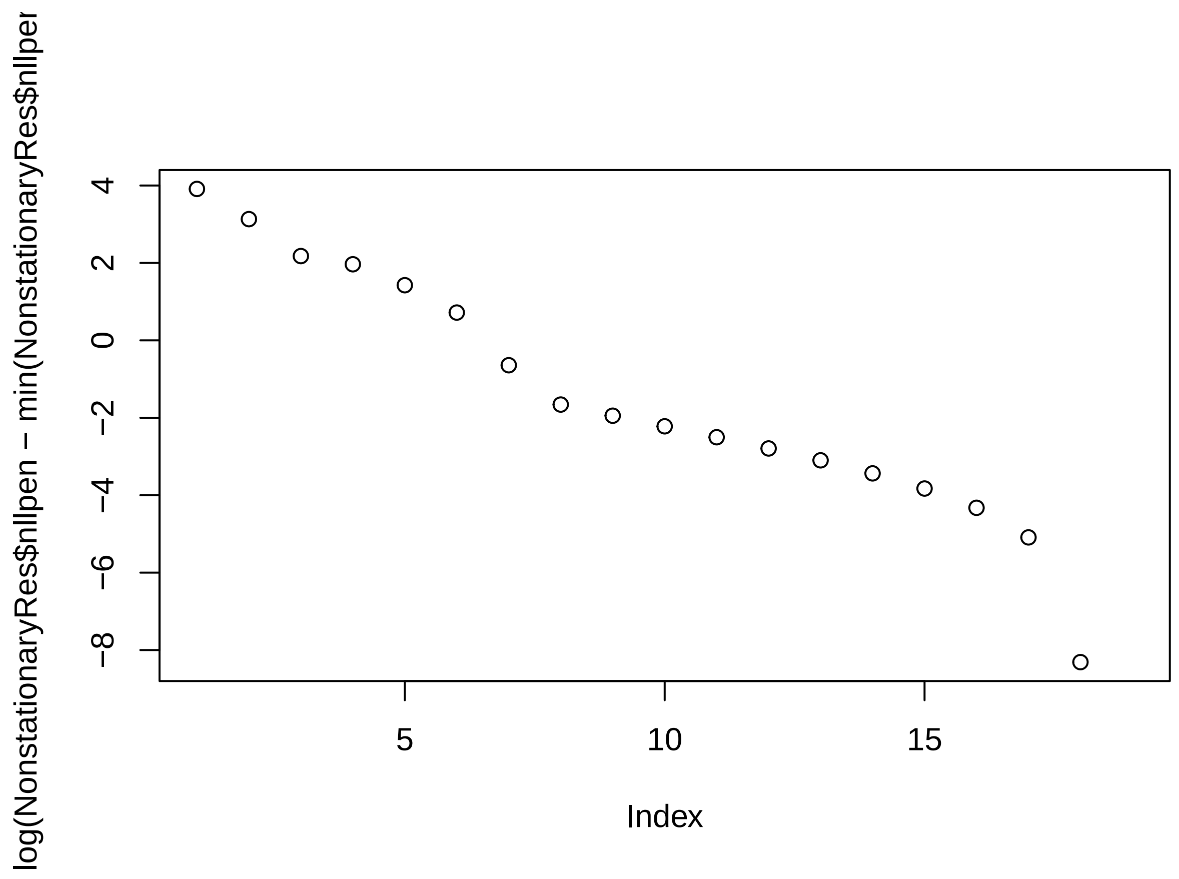
Now we can fit the non-stationary DPMLE. This will take a few minutes to run.

```
NonstationaryRes <-  DPMLE(NbIter = NbIter , epsilon = epsilon,
            Pi0 = Pi0, beta0 = beta0, X = cov, lmu0 = lmu0, lsd0 = lsd0, lkappa0 = lkappa0,
            nbSteps = nbSteps, aInd = aInd,
            Obs = Obs,
            lambda = lambda, a = 3.7, Cn = Cn,
            type = typeNS)
```

```
NonstationaryRes$convergence
```

```
## [1] 0
```

```
plot(log(NonstationaryRes$nllpen-min(NonstationaryRes$nllpen)))
```



Let us now have a look at the number of states selected.

In the paper, we set the maximum number of iterations to 200 and lower the stopping criterion to $1e-5$; the non-stationary method converged in 142 iterations, while the stationary method did not converge for many runs, despite a convergence in 114 iterations in the final model (selected by NIC). As per the stationary DPMLE, we first look at the model convergence. The non-stationary DPMLE function has the same output as the stationary DPMLE function, allowing us to apply all the same procedures as before. Now let us look at the estimates obtained with the non-stationary DPMLE.

```
NonstationaryRes$K
```

```
## [1] 2
```

```
NonstationaryRes$theta
```

```
##       mu1       mu2       mu3       mu4       sd1       sd2       sd3       sd4 
## 4.3625832 4.3625838 4.3625836 3.1004789 2.6115987 2.6549255 1.8716318 2.3469573 
##    Kappa1    Kappa2    Kappa3    Kappa4 
## 2.6913839 2.6913843 2.6913844 0.4032598
```

The model selected *two* states: estimates states `1`, `2` and `3` are merged together, and estimated states `4` corresponds to a merged state. These coincidentally match the merged states of the stationary DPMLE. Now, let's get the merged estimates from the model.

```r
thetaNS <- cbind(mu = c((NonstationaryRes$theta[1] +
                         NonstationaryRes$theta[2] +
                        NonstationaryRes$theta[3])/3,
                        NonstationaryRes$theta[4]),
                 sd = c((NonstationaryRes$theta[1+4] +
                         NonstationaryRes$theta[2+4] +
                        NonstationaryRes$theta[3+4])/3,
                     NonstationaryRes$theta[4+4]))
rownames(thetaNS) <- c("Merge_state 1","Merge_state 2")

# From mean and sd, get shape and scale of gamma distribution
shapeNS <- (thetaNS[,1] * thetaNS[,1])/(thetaNS[,2] * thetaNS[,2] )
scaleNS <- (thetaNS[,2] * thetaNS[,2])/ thetaNS[,1]

# Same indices + 8, because there are 3 parameters to estimate for 4 states.
# The first four are the mean of the gamma distribution,
# then the sd of gamma distribution and then the concentration of the von Mises distribution.
kappaNS <- c((NonstationaryRes$theta[1+8] +
              NonstationaryRes$theta[2+8] +
              NonstationaryRes$theta[3+8])/3,
            NonstationaryRes$theta[4+8])

deltaNS <- c((NonstationaryRes$pi[1] + NonstationaryRes$pi[2] + NonstationaryRes$pi[3]),
             NonstationaryRes$pi[4])
```

Now let us have a look at the estimates for regression coefficient `beta`. The non-stationary DPMLE output for beta is a matrix with `nbStates*(nbStates-1)` rows and `Nb of covariates + 1` columns. The first column corresponds regression coefficient associated to the intercept. The name of each row corresponds to the associated transition: `1 -> 4` corresponds to transition from state one to state four.

```r
betaNS <- array(dim=c(2,2))

betaNS[1,1] <-  (NonstationaryRes$beta[3,1]  + #1 -> 4
                NonstationaryRes$beta[6,1]   + #2 -> 4
                NonstationaryRes$beta[9,1])/3  #3 -> 4


#Same indices + j * (j-1)
betaNS[1,2] <-   (NonstationaryRes$beta[3,2] + #1 -> 4
                NonstationaryRes$beta[6,2]   + #2 -> 4
                NonstationaryRes$beta[9,2])/3  #3 -> 4





betaNS[2,1] <- (NonstationaryRes$beta[10,1]  +  #4 -> 1
                NonstationaryRes$beta[11,1]  + #4 -> 2
                NonstationaryRes$beta[12,1])/3   #4 -> 3


#Same indices + j * (j-1)
betaNS[2,2] <-  (NonstationaryRes$beta[10,2]  + #4 -> 1
                NonstationaryRes$beta[11,2]   + #4 -> 2
                NonstationaryRes$beta[12,2])/3  #4 -> 3

colnames(betaNS) <- c("intercept","distance to shore")
rownames(betaNS) <- c("1 -> 2","2 -> 1")
betaNS
```

```
##           intercept distance to shore
## 1 -> 2 -872.1933333       -969.590000
## 2 -> 1    0.5733333         -5.706667
```

The second column represents the effects of `distance to shore` on the transition probabilities. A negative value means that increasing values of the covariate decreases the switching probability. From beta, we can get the merged transition probability matrix as follows:

```
trMatNS <- HMM.beta2tpm(2, betaNS, cov, length(exData$ID))
trMatNS <- trMatNS$tpm
```

Now let us compute the negative log-likelihood and compare it to the stationary model. To do so, we first need to compute the diagonal matrix of emission distribution, with the parameter estimates.

```
nbSteps <- length(exData$ID)

lnProbsNS <- array(0, dim=c(nbSteps,2))
lnProbsNS[1,] <- apply(cbind(shapeNS,scaleNS),1,FUN=function(x)dgamma(Obs[1,1],
                                                                      shape=x[1],
                                                                      scale=x[2],
                                                                      log=TRUE))

for(t in (2:nbSteps)){
 if(!is.na(Obs[t,1])){lnProbsNS[t,] <- apply(cbind(shapeNS,scaleNS),1,
                                            FUN=function(x)dgamma(Obs[t,1],
                                                                  shape=x[1],
                                                                  scale=x[2],
                                                                  log=TRUE))}
  if(!is.na(Obs[t,2])){lnProbsNS[t,] <- lnProbsNS[t,]+sapply(kappaNS,
                                            FUN=function(x)dvon_mises(Obs[t,2],
                                                                      mu=0,
                                                                      kappa=x,
                                                                      log=TRUE))}
}
```

We use BIC to select the best performing method, between DPMLE and non-stationary DPMLE. As mentioned in the paper, information criteria are useful tools to select covariates.

```
negllkNS <- forward_algNH(deltaNS,trMatNS,lnProbsNS,nbSteps,aInd)
negllkS <- forward_alg(piS,trMatS,lnProbsS,nbSteps,aInd)

# Derive BIC for nonstationary model
BICnonstat <- 2 * negllkNS + log(nbSteps) * (ncol(cov)*2 * (2- 1) + 2 - 1 + 2 * 3)
# Derive BIC for stationary model
BICstat <- 2 * negllkS + log(nbSteps) * (2 *(2 - 1)  + 2 * 3)
```

BICnonstat (BIC for the fitted model from the non-stationary DPMLE) is computed differently than BICstat because of the difference in the number of parameters for the two models. From Eq (6), there are `nbStates x (nbStates-1)\times` regression coefficients estimated per covariate (including the intercept), which we write $ncol(cov) \times 2 \times (2-1)$. The initial distribution $\boldsymbol{\delta}$ is estimated along with the parameters of the emission distributions, which include three parameters (μ, σ, κ) for each state. In the stationary model, the initial distribution is set equal to the stationary distribution, which is derived from the transition probability matrix. Therefore, it does not need to be estimated separately, resulting in one (nbStates-1) fewer parameters compared to the non-stationary version.

```
BICnonstat
```

```
## [1] 2779.086
```

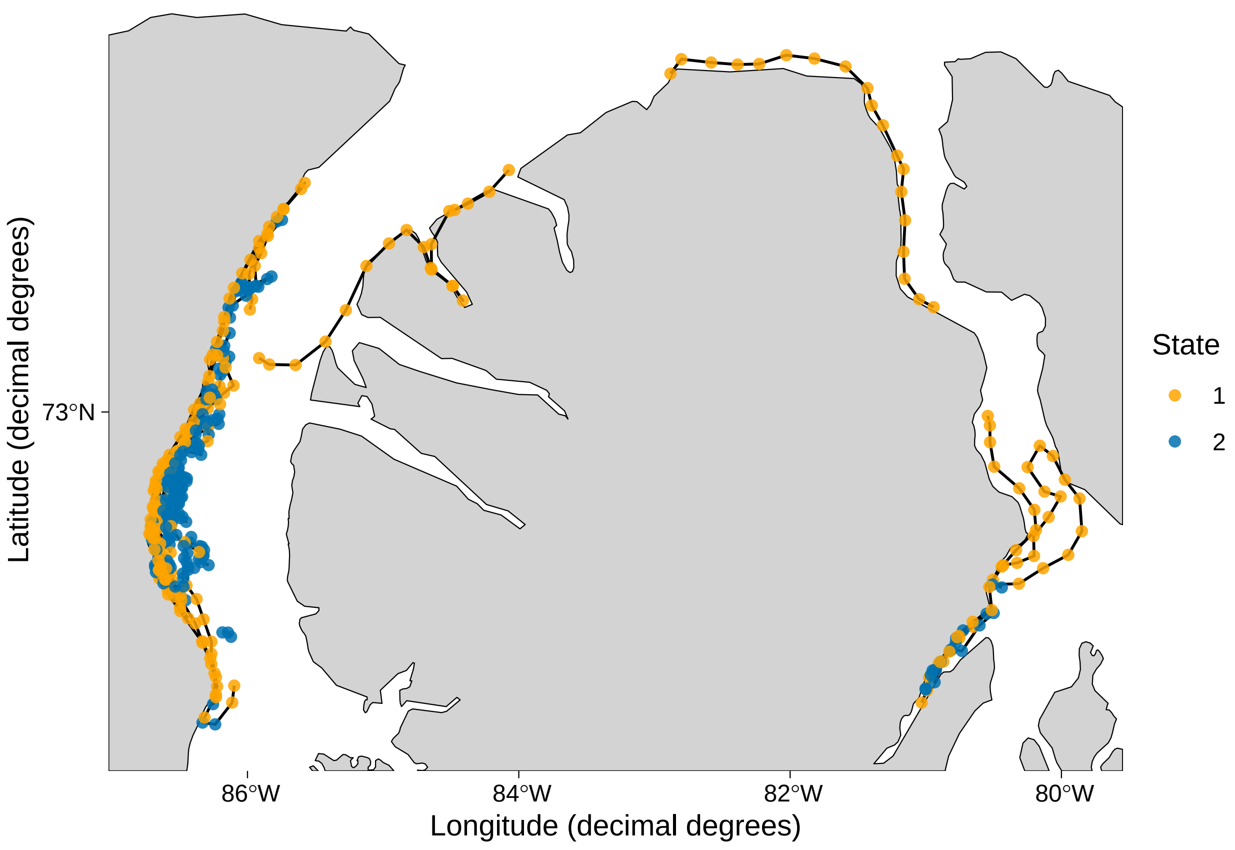
```
BICstat
```

```
## [1] 2524.571
```

The stationary model is selected by BIC. This result differs from the paper. **In the paper**, the non-stationary model is selected. The result from the tutorial is meaningless since we have not explored multiple hyper-parameters and initial values. It is important to remember that for any analysis to be meaningful, multiple random initial values and pairs of hyperparameters should be explored. An example of how to implement this is provided in Section 4 of this tutorial.

Let us plot the movement of the narwhal according states associated with the stationary model.

```
colors <- c("1" = "orange","2"="#0072B2")

df <- as.data.frame(cbind(Lat =data$y,Lon = data$x))

    basemap(limits = c(bbox), crs = 4326,land.col="lightgrey")+
    ggspatial::geom_spatial_path(data=df, aes(x = Lon, y = Lat))+
    ggspatial::geom_spatial_point(data=df, aes(x = Lon,
                                               y = Lat,
                                               colour=factor(states)),
                                                  alpha=0.85)+
    scale_color_manual(values = colors,name = "State")+
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(colour = "black"),
    axis.ticks = element_line(colour = "black"),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black")
  )
```



From it, we see that the directed movement is associated to locations where the individual is travelling (leaving Baffin Bay to go to Arctic Bay) and movement of type `2` is associated with more local behavior.

We can do more to assess the goodness of it of our model. For example, we can look at the pseudo-residuals of the model. The code to derive the pseudo residuals for gamma and von Mises distributions is in `tutorial_source.R`. It was derived from section 6.2 in Zucchini et al.

The pseudo-residual u_t of an observation x_t from a continuous random variable X_t is defined as follows:

$$u_t = \mathbb{P}(X_t \leq x_t)$$

where $\mathbb{P}$ is the probability under the fitted model. If the model is correct, we have that the normal pseudo-residuals defined as follows:

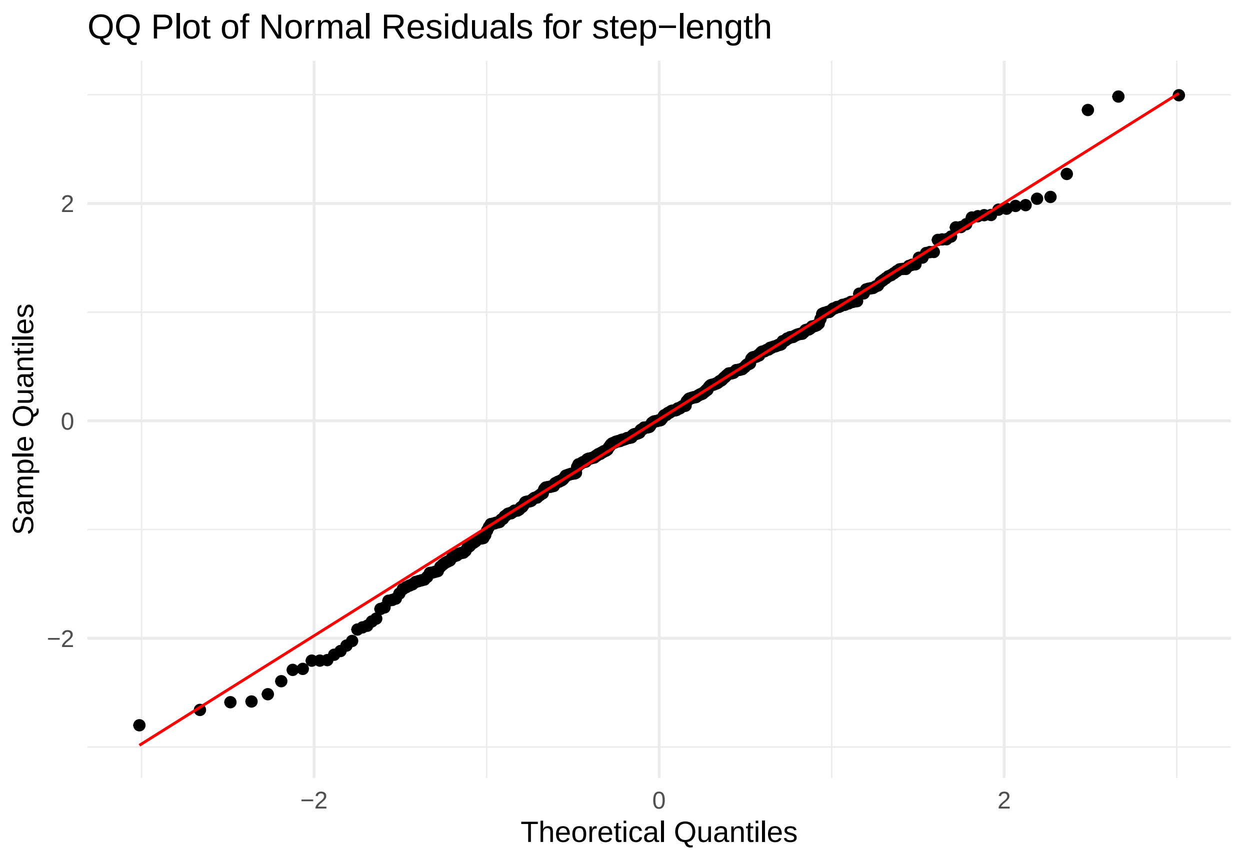
$$z_t = \phi^{-1}(u_t)$$

are standard normal distributed. The pseudo_residuals function takes the following arguments:

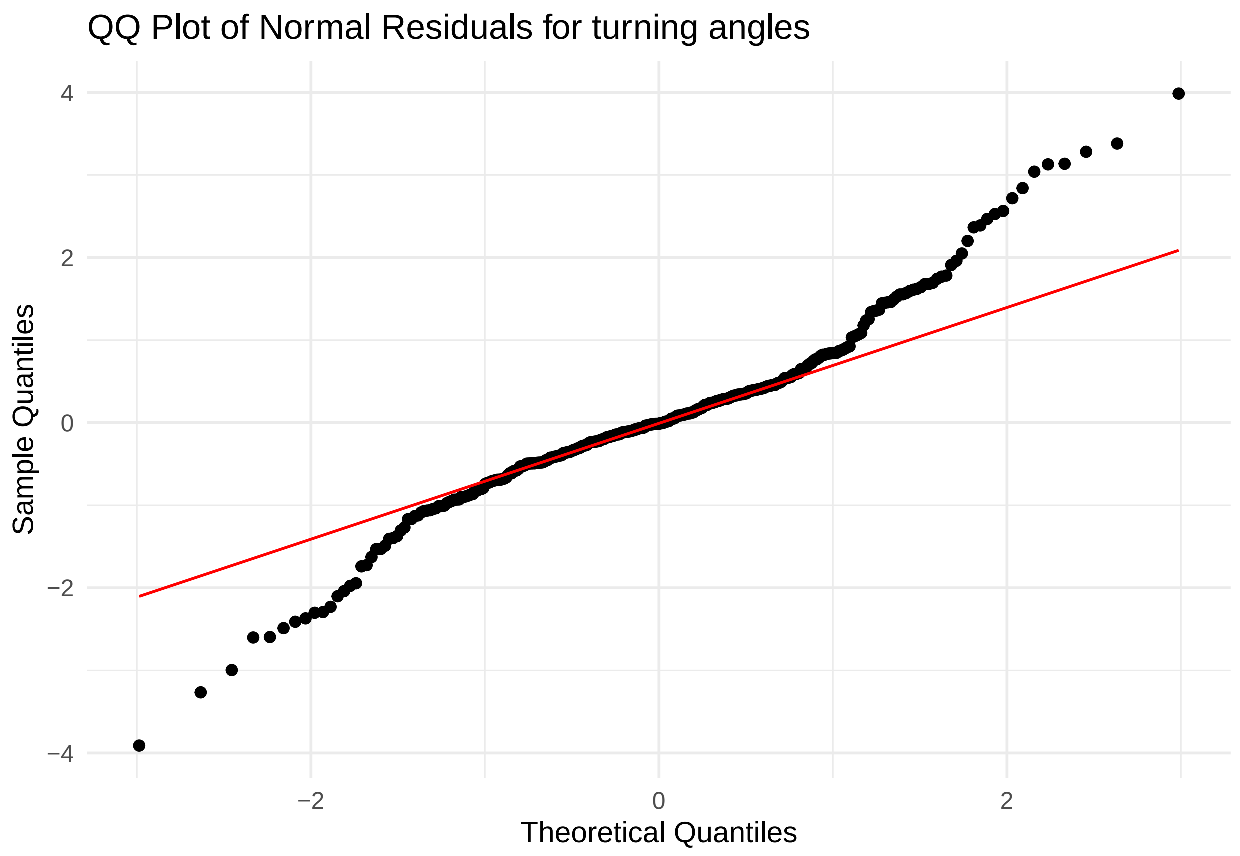
- `"Obs"` : data stream of interest (e.g., step-length time series).
- `"type"` : list of two elements: (1) type of pseudo-residuals either "ordinary" or "forecast" see section 6.2 of Zucchini et al. for more information (2) type of HMM "H" or "NH" (homogeneous or non homogeneous).
- `"dist"` : distributions associated with each data stream, which is gamma for step length and von Mises for turning angle.
- `"lalpha"`: logarithm of the forward probabilities of the fitted model.

- `"lbeta"` : logarithm of the backward probabilities of the fitted model.
- `"delta"` : vector of initial probabilities of the fitted model.
- `"kappa"`: vector of estimated concentration parameters of the von Mises distribution.
- `"shape"`: vector of estimated shape parameters of the gamma distribution.
- `"scale"`: vector of estimated scale parameters of the gamma distribution.
- `"trMat"` : t.p.m of fitted model.

```
# For step length
npsr.step <- pseudo_residuals(exData$step,
                              type=list("ordinary","H"), dist=c("gamma"),
                              lalpha=lalphaS, lbeta=lbetaS, delta=piS,
                              shape=shapeS, scale=scaleS, trMat=trMatS)
npsr.step <- data.frame(npsr.step = npsr.step)

# Generate QQ plot
ggplot(npsr.step, aes(sample = npsr.step)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(title = "QQ Plot of Normal Residuals for step-length",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme_minimal()
```



```
#for turning angle
npsr.angle <- pseudo_residuals(exData$angle,type=list("ordinary","H"),dist=c("vonmises"),
                              lalpha=lalphaS,lbeta=lbetaS,delta=piS,kappa=kappaS,trMat=trMatS)
npsr.angle <- data.frame(npsr.angle = npsr.angle)


# Generate QQ plot
ggplot(npsr.angle, aes(sample = npsr.angle)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(title = "QQ Plot of Normal Residuals for turning angles",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme_minimal()
```



The QQ plot for the step-length time series shows a good fit. The QQ plot for the turning angle indicates that the turning angles exhibit heavier tails compared to the model, as also reflected in the histograms. This behavior was attributed to measurement error (Hurford, 2009)

4. Narwhal Information Criterion

In this section, we provide the code to fit the stationary model with multiple random initial values and hyperparameters, and then to select the best model with NIC. This part is more time consuming, since it requires fitting the model multiple times. We define two parameters: `"Ntest"` that corresponds to the number of explorations of tuning parameters (λ and C_N) and "Nrep" which is the number of random initial values explored.

As mentioned in the paper, maximum likelihood estimates (MLE) obtained from fitting a standard HMM with the `fitHMM` function in the `momentuHMM` R package are used as initial values for the DPMLE. Therefore, we have to set the initial parameters for fitting a standard HMM with the momentuHMM package. We need to define:

- "dist": distributions associated with each data stream, which is gamma for step length and von Mises for turning angle.
- "nbStates": number of states, which is the upper bound used in the double-penalized maximum likelihood method. Here we choose $N = 4$.
- "Par0": initial parameters for the emission distribution: the initial means and standard deviations $(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0)$ of the gamma distributions, the initial concentration parameter $\boldsymbol{\kappa}_0$ of the von Mises distribution.
- "beta0": the initial parameters for t.p.m..

```r
# Set parameters for momentuHMM
dist <- list(step = "gamma", angle= "vm")
```

There are various methods for initializing the emission distribution and transition probability matrix parameters. Here, I reproduce the method used by Pohle et al. (2017). Another, and often recommended, way to initialize the mean parameters of the gamma distribution involves a closer look at the data. Théo Michelot and Roland Langrock provide a concise guide on how to choose initial values for HMMs in this moveHMM vignette. In general however, this does not have much impact if a sufficiently large number of random initial values are explored. **In the paper**, we explored 100 values for the narwhal case study, and 50 in the simulation study. **In the tutorial**, I am only using one set of initial values to minimize the computational burden.

Once the initial parameters are defined, we can fit the standard HMM to the data with `fitHMM` from `momentuHMM`, to obtain initial values for the stationary DPMLE. In the ideal setting, this step would be repeated multiple times to obtain the best fit from the standard HMM. This can be done by repeating this step with different starting values and by adjusting the argument `retryFits` in the function `fitHMM`, that perturbs the parameter estimates and retry fitting the model. Here, we set it to `1`.

By default, fitHMM will set the argument `estAngleMean` to NULL, which means that we assume that the mean turning angle is 0 for both behaviors (i.e., the animal has a tendency to continue in the same direction) and that only the angle concentration parameters differ. The concentration parameter controls the extent to which the animal continues forward versus turn. We use the same set up in the DPMLE method, where we only consider the concentration parameters and set the means to 0.

```r
exData$ID <- as.numeric(as.character(exData$ID)) #transform to numeric for fitHMM


# number of explorations of tuning parameters pairs
Ntest <- 10
# number of random initial values.
Nrep  <- 5

#Number of maximum iterations in the EM algorithm
NbIter <- 100

#Initialize parameters, from Pohle et al (2017).
```

```r
null <- list()         # model selected after random initial values explorations
NICbetter <- array()   # NIC criterion for each of the Ntest model selected
for(j in (1:Ntest)){
  set.seed(2*j)
  lambda = exp(runif(1,1,5))
  Cn = runif(1,1,5)
    for (i in (1:Nrep)){
      #Initialize parameters
      inter <- list()        # intermediate model, for each of the Nrep runs.
      nllpen <- array()     # Double penalized negative log-likelihood for each of the Nrep models.
      # step parameters
      j <- 4

      skip_to_next <- FALSE
      mu0<-c(runif(j,min=min(exData$step,na.rm=TRUE),max=max(exData$step,na.rm=TRUE)))
      shape0<-c(rgamma(j,shape=1,scale=2.5))
      sigma0<- mu0/sqrt(shape0)
      beta<-NULL

      # Stationary distribution and t.p.m
      Pi0 <- array(dim=c(j))
      trMat0 <-  array(dim=c(j,j))

      beta<-array(NA,dim=c(j,j))
      for(l in 1:j){
        alpha<-rep(0.2/(j-1),j)
        alpha[l]<-0.8
        alpha<-alpha*10
        beta[l,]<-rdirichlet(1,alpha)
      }
      beta <- t(beta)
      trMat0 <- beta

      beta <- logit(beta)
      beta <- matrix(beta[col(beta) != row(beta)],ncol=j*(j-1),nrow=1)

      Pi0 <- rep(1/j, j)
      Kappa0 <- rgamma(j,shape=1,scale=3)

      Par0 <- list(step=c(mu0,sigma0),angle=c(Kappa0))
      tryCatch(b <- fitHMM(exData, nbState = j,
                          dist = dist, Par0 = Par0, beta0=beta,estAngleMean=NULL,
                          stationary=TRUE,retryFits = 1),
          error = function(e){skip_to_next<<- TRUE}) #If there is an error, the code keeps on running.
           if(skip_to_next){ next }
      skip_to_next <- FALSE

      # Set the estimates of the best model as starting parameter for EM run with 100 iterations.
      Pi0 <- as.numeric(b$mle$delta[1,])
      tryCatch( bb <- DPMLE(NbIter= NbIter ,epsilon = epsilon,
            Pi0 =  Pi0,trMat0 =  b$mle$gamma ,lmu0 =  log(b$mle$step[c(1,3,5,7)]),
            lsd0 = log(b$mle$step[c(2,4,6,8)]),lkappa0 = log(b$mle$angle[2,]) ,
            nbSteps=nbSteps,aInd = aInd,
            Obs = Obs,
            lambda=lambda,a = 3.7,Cn = Cn,
            type=typeS),error = function(e){skip_to_next<<- TRUE})
      if(skip_to_next){ next }
      inter[[i]]<- bb
      nllpen[i] <- inter[[i]]$nllpen
    }

    idx <- which.min(nllpen)
    null[[j]] <- inter[[idx]]
    NICbetter[j] <- 2*null[[j]]$nllk + log(length(Obs[,1]))*(null[[j]]$K*(null[[j]]$K-1)+null[[j]]$K*2)
  }
```

For each of paired hyperparameter value, we now have the best model over multiple fit with different initial values `null` and its associated NIC. The code for the non-stationary version is very similar, except that `beta0` of dimension `ncol(cov) x N x (N-1)` is used instead of `trMat0` in the `DPMLE()` function. Additional object `formula ~ dtoshore` must be included in `fitHMM`. Importantly, this process can be parallelized over the hyperparameters (C_N and λ) and initial values to improve computational efficiency.

5. References

- Hurford, A. (2009). GPS Measurement Error Gives Rise to Spurious 180° Turning Angles and Strong Directional Biases in Animal Movement Data. PLoS ONE 4(5), e5632.
- Pohle, J., Langrock, R., Van Beest, F. M., & Schmidt, N. M. (2017). Selecting the number of states in hidden Markov models: pragmatic solutions illustrated using animal movement. Journal of Agricultural, Biological and Environmental Statistics, 22, 270-293.
- Zucchini, W., I. L. MacDonald, and R. Langrock (2017). Hidden Markov models for time series: an introduction using R. CRC press.