Analysing ecological data with HMMs in R

Timo Adam, Roland Langrock, Sina Mews, Théo Michelot

26 June 2022

1 Analysing tracking data with moveHMM

In this exercise, we analyse petrel tracking data using hidden Markov models (HMMs). The R package moveHMM allows us to model step lengths and turning angles using HMMs. We start by installing and loading moveHMM:

```r
## Install and load moveHMM
install.packages("moveHMM")
library(moveHMM)
```

1.1 Data pre-processing

The raw data are available on Movebank, click here for details. These raw data have been pre-processed by sub-setting relevant columns, splitting the tracks at gaps, and creating the covariate distance to the colony (we can skip the following code chunk and load the pre-processed data instead, see the last code chunk of this section):

```r
## Load data from Movebank
URL <- paste0("https://www.datarepository.movebank.org/bitstream/handle/",
              "10255/move.568/At-sea%20distribution%20Antarctic%20Petrel",
              "%2c%20Antarctica%202012%20%28data%20from%20Descamps%20et%",
              "20al.%202016%29-gps.csv")
raw <- read.csv(url(URL))

## Keep relevant columns: ID, time, lon, and lat
data_all <- raw[, c(13, 3, 4, 5)]
colnames(data_all) <- c("ID", "time", "lon", "lat")
```

```r
data_all$time <- as.POSIXct(data_all$time, tz = "MST")

## Keep first 10 tracks for this example
petrel_data <- subset(data_all, ID %in% unique(ID)[1:5])

## Split tracks at gaps using the function split_at_gaps
source(./R/split_at_gap.R)
petrel_data <- split_at_gap(data = petrel_data, max_gap = 60)

## Plot tracks
ggplot(petrel_data, aes(lon, lat, col = ID)) + geom_path() +
  geom_point(size = 0.3) + coord_map()

## Create distance to centre covariate
# Define centre for each track as first observation
i0 <- c(1, which(petrel_data$ID[-1] != petrel_data$ID[-nrow(petrel_data)])
        + 1)
centres <- petrel_data[i0, c("ID", "lon", "lat")]
petrel_data$centre_lon <- rep(centres$lon, rle(petrel_data$ID)$lengths)
petrel_data$centre_lat <- rep(centres$lat, rle(petrel_data$ID)$lengths)

# Add distance to centre covariate (based on sp for great circle distance)
petrel_data$d2c <- sapply(1:nrow(petrel_data), function(i) {
  spDistsN1(pts = matrix(as.numeric(petrel_data[i, c("lon", "lat")]),
            ncol = 2), pt = c(petrel_data$centre_lon[i],
            petrel_data$centre_lat[i]), longlat = TRUE)
})

# Divide by 1000 for numerical stability (i.e., unit = 1000 km)
petrel_data$d2c <- petrel_data$d2c / 1000
```

Instead of running the above code chunk, we can load the pre-processed data:

```r
## Load the data
petrel_data <- read.csv("./data/petrel_data.csv")
```

The data are in the proper format and can be processed using `prepData` to compute step lengths and turning angles. We choose the arguments carefully:

- `trackData` is a data frame of the tracking data, including at least coordinates (either easting/northing or longitude/latitude values), and optionally a field ID (identifiers for the observed individuals). Additional fields are considered as covariates.
- `type` specifies whether the coordinates are easting/northing (`type = "UTM"`) or longitude/latitude (`type = "LL"`) values.
- `coordNames` are the names of the coordinates in the input data frame. The default is `"x"` and `"y"`, so we need to call the function with the argument `coordNames = c("lon", "lat")`.

The call to this function for our data is:

```
## Compute step lengths and turning angles
hmm_data <- prepData(trackData = petrel_data, coordNames = c("lon", "lat"),
                     type = "LL")
```

Step lengths and turning angles are computed; the returned object is a data frame. The following commands can be used to get an overview of the data and to visualise the tracks:

```
## Get an overview of the data and to visualise the tracks
head(hmm_data)
plot(hmm_data)
```

1.2 Model specification

Before we can fit an HMM to our data, we need to specify initial values for the parameters of the state-dependent distributions. As a starting point, it helps to have a look at the histograms of step lengths and turning angles:

```
## Plot histograms of step length and turning angle
par(mfrow = c(1, 2))
hist(hmm_data$step)
hist(hmm_data$angle)
```

For an N-state HMM with gamma state-dependent distributions for the step lengths, we need N mean, standard deviation, and, if applicable, zero probability parameters, respectively. Furthermore, with wrapped Cauchy state-dependent distributions for the turning angles, we need N mean and concentration parameters, respectively.

```r
## Initial values for the parameters of the state-dependent distributions
# For step length
step_mean_par0 <- c(2, 8, 16) # means
step_SD_par0 <- c(4, 5, 6) # SDs
step_zeroprob_par0 <- c(0.01, 0.01, 0.01) # zero probabilities
step_par0 <- c(step_mean_par0, step_SD_par0, step_zeroprob_par0)
# For turning angle
angle_mean_par0 <- c(0, 0, 0) # means
angle_concentration_par0 <- c(0.5, 0.7, 0.9) # concentrations
angle_par0 <- c(angle_mean_par0, angle_concentration_par0)
```

The above initial values provide a good choice, but feel free to try other values as well!

1.3 Model fitting

To fit an HMM to data, we use `fitHMM`. The most important arguments of that function are:

- `data` is the data frame created by `prepData`.
- `nbStates` is the number of states.
- `stepDist` is the distribution assumed for step length.
- `angleDist` is the distribution assumed for turning angle.
- `stepPar0` is a vector of initial parameter values for step length.
- `anglePar0` is a vector of initial parameter values for turning angle.

The call to the function for our data is:

```r
## Fit 3-state HMM
mod <- fitHMM(data = hmm_data, nbStates = 3, stepDist = "gamma",
                  angleDist = "wrpcauchy", stepPar0 = step_par0,
                  anglePar0 = angle_par0, verbose = 1)
```

If we want to watch `fitHMM` iteratively fitting our HMM, we can choose `verbose = 2`. The fitted model is stored as `mod`, which is used as an input to the following functions.

1.4 Results

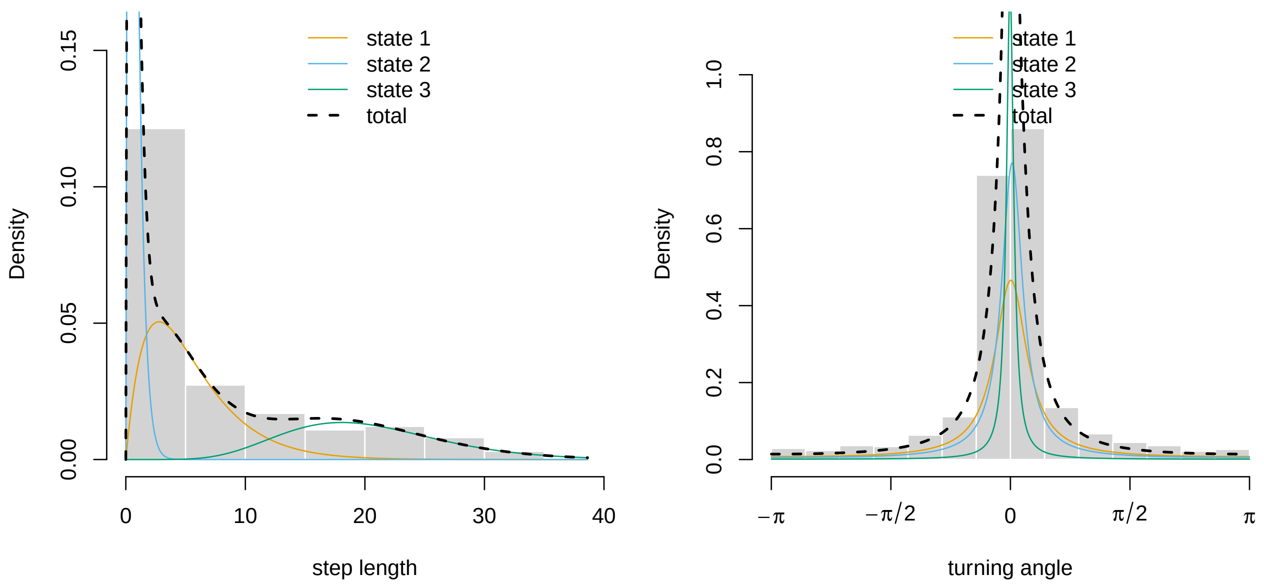
To obtain the estimated model parameters, we can call the fitted model object:

```r
## Print estimated model parameters
mod
```

```
Value of the maximum log-likelihood: -10635.39

Step length parameters:
----------------------
               state 1      state 2      state 3
mean      5.4768072082 7.349585e-01 2.035057e+01
sd        3.8453702342 5.090077e-01 6.790187e+00
zero-mass 0.0007077356 1.841818e-12 1.909972e-09

Turning angle parameters:
------------------------
                  state 1    state 2      state 3
mean          0.005705962 0.02210354 -0.004975031
concentration 0.772743630 0.84579270  0.944239037

Regression coeffs for the transition probabilities:
--------------------------------------------------
             1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1    3 -> 2
intercept -1.995085 -2.869465 -2.102824 -40.66451 -2.375585 -41.47468

Transition probability matrix:
-----------------------------
          [,1]         [,2]         [,3]
[1,] 0.8384117 1.140258e-01 4.756248e-02
[2,] 0.1088227 8.911773e-01 1.948005e-18
[3,] 0.0850535 8.895508e-19 9.149465e-01

Initial distribution:
--------------------
[1] 3.366961e-14 9.703801e-19 1.000000e+00
```

To plot the fitted state-dependent distributions, we can use `plot`:

```r
## Plot the fitted state-dependent distributions
par(mfrow = c(1, 2))
plot(mod, plotTracks = FALSE, ask = FALSE)
```



States 1 and 2 capture short and moderately long steps, which can be interpreted as resting and foraging behaviour, respectively. State 3 captures long, directed steps, which can be linked to travelling behaviour. To visualise the Viterbi-decoded tracks, choose `plotTracks = TRUE`.

To obtain the decoded state sequence, we can use `viterbi`, `stateProbs`, and `plotStates`:

```
## Global state decoding
viterbi(mod)
## Local state decoding
stateProbs(mod)
## Show both
plotStates(mod)
```

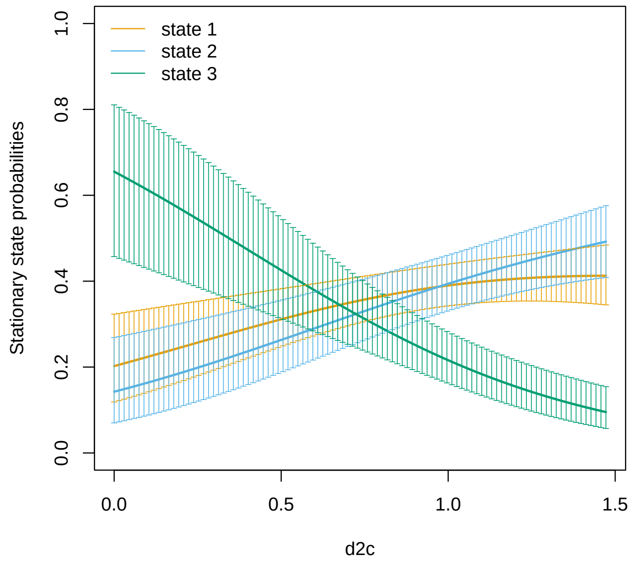
1.5 Adding covariates on the state transition probabilities

Covariates can be included in the hidden state model to capture the effect of environmental or individual-specific variables on the animal's behaviour. In moveHMM, covariates can be included by specifying a `formula`. Here, we are interested in modelling a linear, quadratic, and cubic effect of the distance to the colony (`d2c`).

```
# Fit 3-state model with linear effect of d2c
mod_d2c <- fitHMM(data = hmm_data, nbStates = 3, stepDist = "gamma",
                  angleDist = "wrpcauchy", stepPar0 = step_par0,
                  anglePar0 = angle_par0, formula = ~ d2c, verbose = 1)

# Fit 3-state model with squared effect of d2c
mod_d2c2 <- fitHMM(data = hmm_data, nbStates = 3, stepDist = "gamma",
                   angleDist = "wrpcauchy", stepPar0 = step_par0,
                   anglePar0 = angle_par0, formula = ~ d2c + I(d2c ^ 2),
                   verbose = 1)

# Fit 3-state model cubic effect of d2c
mod_d2c3 <- fitHMM(data = hmm_data, nbStates = 3, stepDist = "gamma",
                   angleDist = "wrpcauchy", stepPar0 = step_par0,
                   anglePar0 = angle_par0, formula = ~ d2c + I(d2c ^ 2)
                   + I(d2c ^ 3), verbose = 1)
```

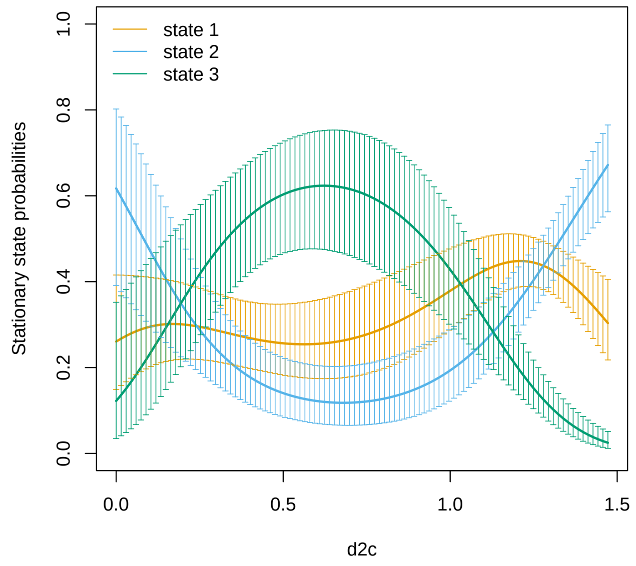
To visualise the probability of being in either of the 3 states as functions of the distance to the colony, we can plot the stationary state probabilities using `plotStationary`.

```
## Plot stationary state probabilities as function of d2c
plotStationary(mod_d2c, plotCI = TRUE)
```



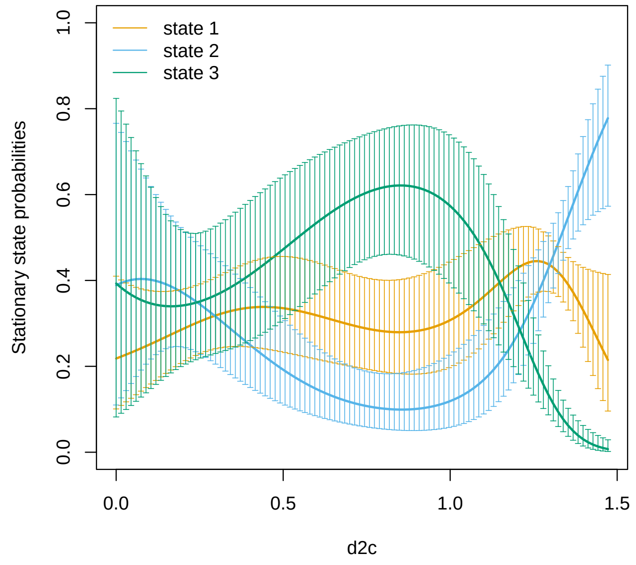
It looks like there is evidence of an effect of the distance to the colony: the further away from the colony, the less likely petrels are to be in state 3 (i.e., the travelling state), and the more likely they are to be in states 1 and 2 (i.e., the resting and foraging states, respectively).

```
## Plot stationary state probabilities as function of d2c
plotStationary(mod_d2c2, plotCI = TRUE)
```



The quadratic effect suggests that the effect actually is more complex.

```
## Plot stationary state probabilities as function of d2c
plotStationary(mod_d2c3, plotCI = TRUE)
```



The cubic effect does not differ much from the quadratic effect. Therefore, the question arises if the additional complexity added to the model is justified by the data? This question can be answered using model selection criteria.

1.6 Model selection

To compare and select among candidate models using information criteria, we can use `AIC`. Here, we select among HMMs with different covariate effects (for model selection with regard to the number of states, see Section 2):

```
## Compute AIC
AIC(mod, mod_d2c, mod_d2c2, mod_d2c3)
```

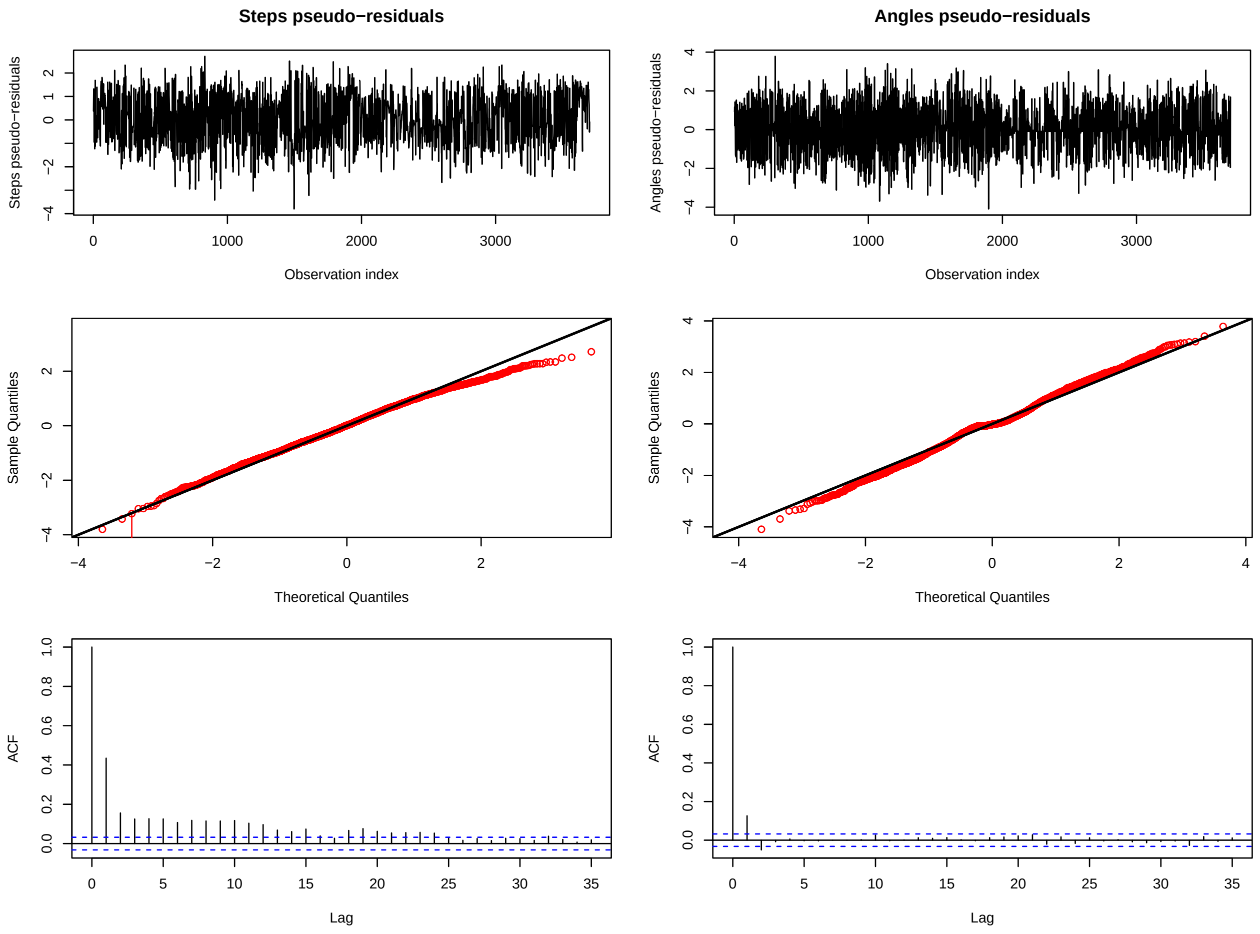
```
     Model      AIC
1 mod_d2c2 21261.70
2 mod_d2c3 21264.43
3  mod_d2c 21279.65
4      mod 21316.77
```

The AIC prefers the HMM with a quadratic effect of the distance to the colony: while the HMMs without covariate effects and with a linear effect are too simplistic, the HMM with a cubic effect is too complex, i.e. the additional complexity added to the model is not justified by the data.

1.7 Model checking

Pseudo-residuals are the most common way to check whether an HMM fits the data well. Similarly to linear model residuals, pseudo-residuals should be independent and normally distributed if the model assumptions are satisfied. The function `plotPR` produces a quantile-quantile (QQ) plot against the normal distribution, and an autocorrelation function (ACF) plot of the pseudo-residuals.

```
## Plot QQ and ACF plots
plotPR(mod_d2c2)
```



The QQ plot is useful to identify deviations from normality, which suggest that the estimated state-dependent distributions do not capture the empirical distribution of the data perfectly. However, there is no indication for any major lack of fit. The ACF plot suggests that there is only little correlation that is not captured by our HMM.

2 Analysing accelerometer data with momentuHMM

The R package momentuHMM extends on moveHMM to allow for more general HMM formulations. Some of the main extensions are:

- possibility to include any number of observed variables, rather than just step length and turning angle, with many possible probability distributions (including normal, Poisson, beta, gamma, etc.);
- possibility to include covariate effects on the observation parameters, rather than only on the transition probabilities;

- integration with the R package crawl to regularise or filter tracking data, including multiple imputation;
- biased random walks models, for movement with centres of attraction.

Here, we will mainly illustrate the first point, with the analysis of accelerometer data. The analysis of acceleration data using hidden Markov models is discussed in detail by Leos-Barajas et al. (2017).

We start by detaching moveHMM if necessary (because some functions have the same names, which could cause conflicts), then loading momentuHMM, as well as ggplot2 and a colour palette to create plots later on.

```
# # Detach moveHMM if loaded
# detach("package:moveHMM", unload = TRUE)
library(momentuHMM)
library(ggplot2)
theme_set(theme_bw())
pal <- c("#E69F00", "#56B4E9", "#009E73")
```

2.1 Data

We consider accelerometer data collected on a whitetip shark over 4 days, provided by Yannis Papastamatiou (Florida International University, USA) and Yuuki Watanabe (National Institute of Polar Research, Japan). The raw data were at a very high sub-second resolution, but we thinned them to a 30-sec resolution for this analysis, for computational speed. The observed variable is the overall dynamic body acceleration (ODBA), which is some measure of the magnitude of the three-dimensional acceleration of the animal, and has sometimes been proposed as a proxy for energy expenditure. We also have another variable, the time of day, which we could consider including as a covariate.

```
# Load whitetip shark accelerometer data
data <- read.csv("./data/whitetip_data.csv")
data$Time <- as.POSIXct(data$Time)
head(data)
```

```
                 Time       ODBA TimeOfDay
1 2013-05-08 10:25:45 0.05538590  10.42896
2 2013-05-08 10:26:45 0.36452689  10.44563
3 2013-05-08 10:27:45 0.89695752  10.46230
```

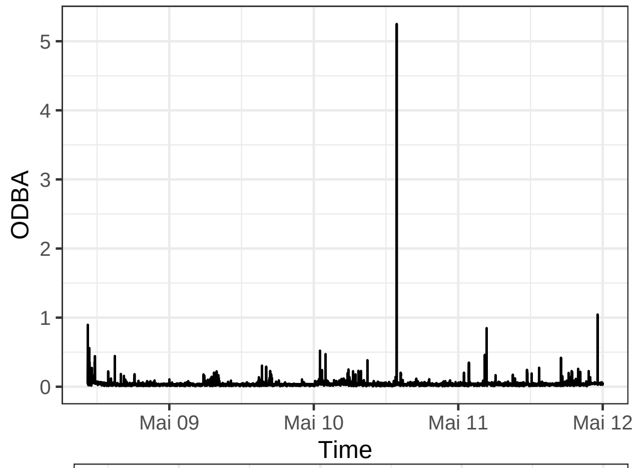
```
4 2013-05-08 10:28:45 0.04144502  10.47896
5 2013-05-08 10:29:45 0.06610186  10.49563
6 2013-05-08 10:30:45 0.03380349  10.51230
```

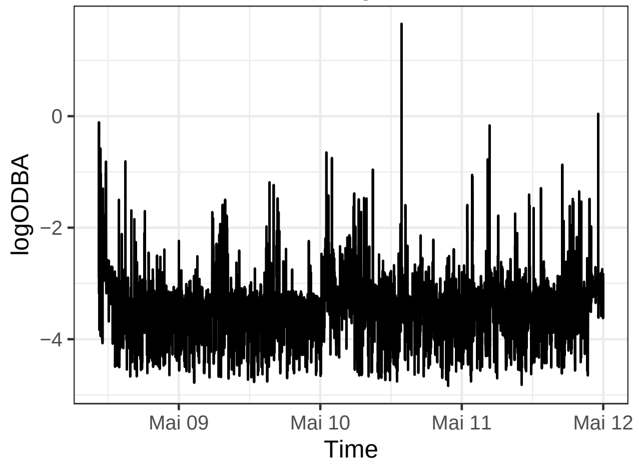
```
nrow(data)
```

```
[1] 5135
```

The ODBA variable is strictly positive, and it is difficult to identify different states when plotting it, because most values are very small in contrast with a few large observations. To better distinguish between the different types of movement with small ODBA, we use the logarithm of the ODBA in the HMM analysis instead.

```
# Get logarithm of ODBA for HMM analysis
data$logODBA <- log(data$ODBA)

# Time series plots of ODBA and log-ODBA
ggplot(data, aes(Time, ODBA)) + geom_line()
ggplot(data, aes(Time, logODBA)) + geom_line()
```





Like in the tracking data example, we need to use the function `prepData` to check the format of the data set, and create a data object that will be recognised by momentuHMM. In this case, there is no need to derive step lengths and turning angles, so the function just copies the data frame passed as input. One difference from moveHMM is that we specify `coordNames = NULL` to indicate that there are no coordinates in the data (because we are not analysing location data), and the argument `covNames` to let momentuHMM know which variables are covariates (rather than response variables).

```r
data_hmm <- prepData(data = data, coordNames = NULL, covNames = "TimeOfDay")
```

2.2 Model 1: two states

We don't know a priori how many states would best capture the patterns in the accelerometer data. This is in general a difficult problem, and this choice typically requires combining biological expertise and model checking to find a trade-off between model complexity and interpretation ((**pohle2017?**)). It is often helpful to start from a simple model with only two states, and to build up complexity, and this is what we do here.

2.2.1 Model formulation

Before fitting a model, we need to specify a distribution for each observed variable. In the example, the log-ODBA variable is continuous and unbounded, so we propose modelling it using normal distributions. We also need to specify initial parameter values, from which to start the likelihood optimisation. Although there is no general guidelines that apply in every situation, the idea is to pick values that are plausible given the data and our expectations of the states. Most observed values of the variable are between -5 to -1, so we might for example choose initial means of -4 and -2 (expecting that the first state will capture the lower values, and the second state the higher values). Similarly, we can choose initial standard deviations based on how much we expect the distributions to spread. Both the distributions and the initial parameters are stored in named lists.

```r
# List of observation distributions
dist <- list(logODBA = "norm")

# List of initial parameters
mean0 <- c(-4, -2)
sd0 <- c(0.5, 0.5)
Par0_2s <- list(logODBA = c(mean0, sd0))
```

2.2.2 Model fitting

We can now pass these arguments, as well as the dataset and the number of states, to `fitHMM` for model fitting. Note that, although some of the argument names are slightly different because more general, this is very similar syntax to moveHMM.

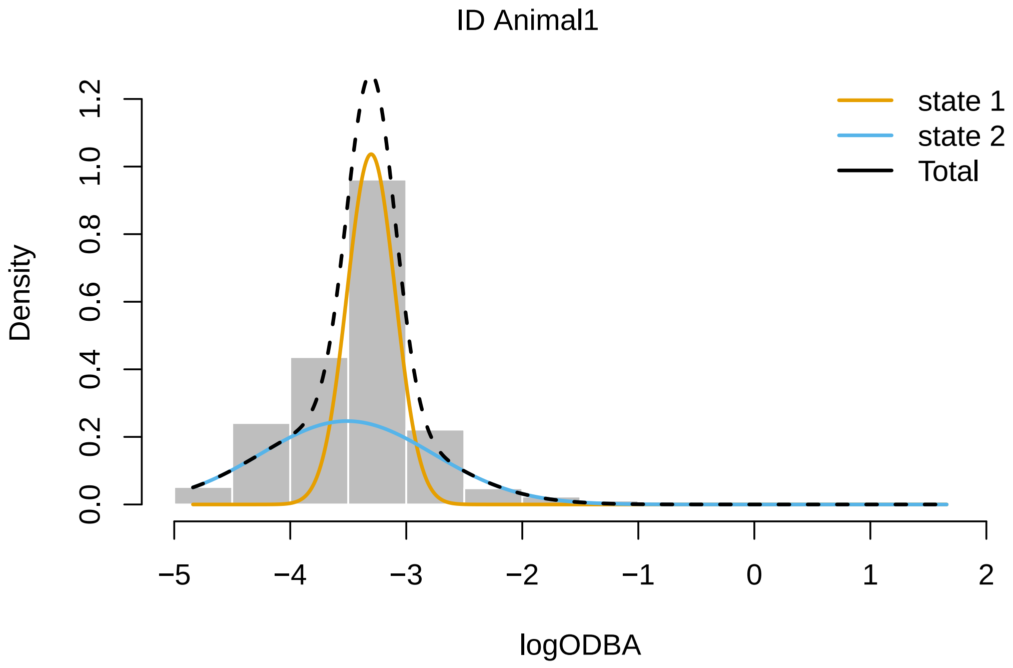
```
hmm_2s <- fitHMM(data = data_hmm, nbStates = 2, dist = dist, Par0 = Par0_2s)
```

2.2.3 Model visualisation

We can visualise the state-dependent distributions of log-ODBA, on top of a histogram of observed values, using the `plot` function.

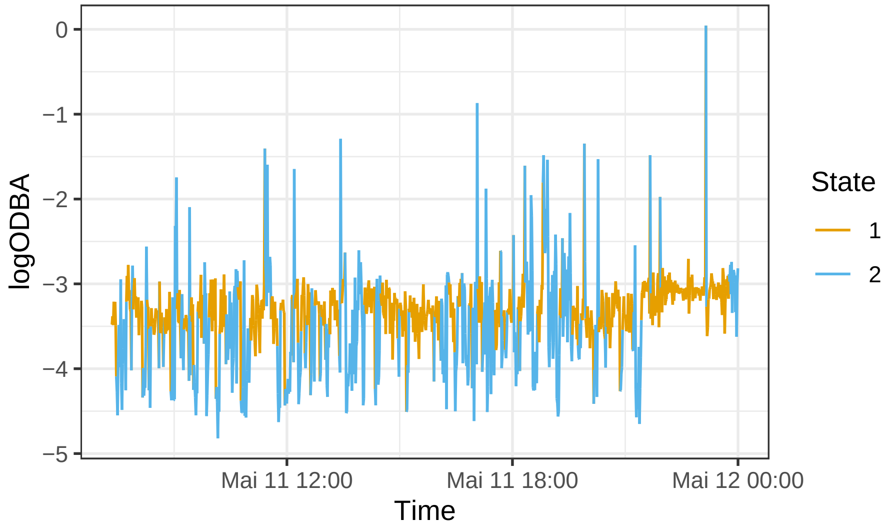
```
plot(hmm_2s, ask = FALSE)
```

```
Decoding state sequence... DONE
```



We can use the function `viterbi` to obtain the most likely state sequence, and use it to colour a time series plot of the observations. To help with visualising patterns in each state, we only show the last 1000 observations, covering a little less than a day.

```r
data_hmm$viterbi_2s <- factor(viterbi(hmm_2s))

ggplot(tail(data_hmm, 1000), aes(Time, logODBA, col = viterbi_2s, group = NA)) +
  geom_line() +
  scale_color_manual(values = pal, name = "State")
```



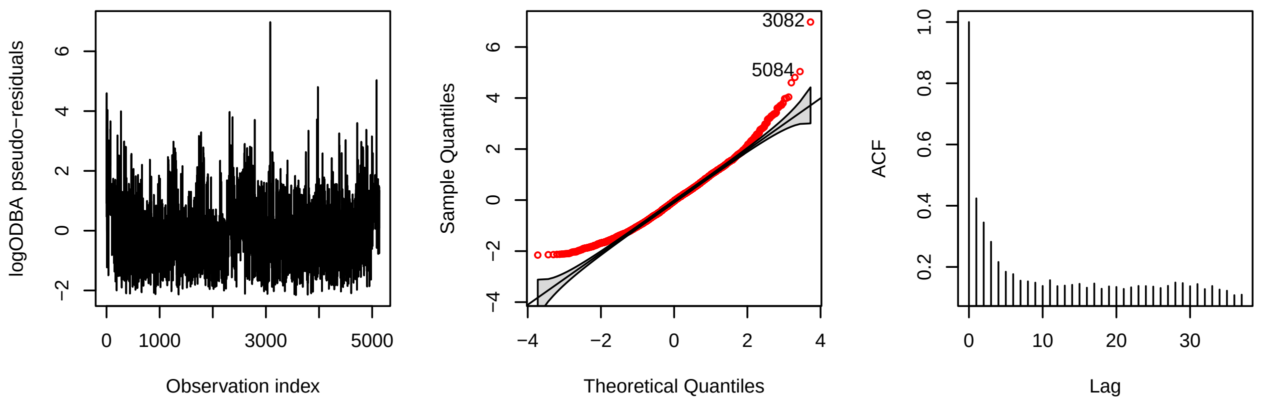
These plots suggest that state 2 captures both the lower tail and higher tail of the distribution of log-ODBA, whereas state 1 captures intermediate values. This makes biological interpretation a little tricky, because it looks like state 2 includes several behaviours (either no activity or high activity).

2.2.4 Model checking

Pseudo-residuals are a common tool to check whether an HMM fits the data well. Similarly to linear model residuals, pseudo-residuals should be independent and normally distributed if the model assumptions are satisfied. The function `plotPR` produces a quantile-quantile (QQ) plot against the normal distribution, and an autocorrelation function (ACF) plot of the pseudo-residuals.

```r
# Plots of pseudo-residuals
plotPR(hmm_2s)
```

```
Computing pseudo-residuals...
```



The QQ plot is useful to identify deviations from normality, which suggest that the state-dependent observation distributions don't capture the empirical distribution of the data well. Here, the QQ plot strongly deviates from the 1:1 diagonal in both tails, showing lack of fit. Potential improvements might include trying different distributions, or a model with more states. In the next section, we explore this latter option.

2.3 Model 2: three states

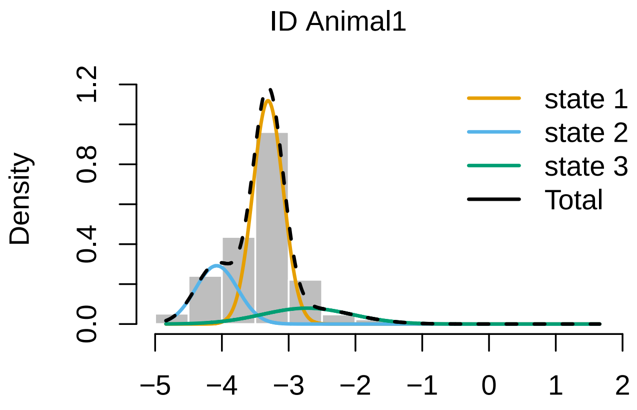
The code required to create a 3-state model is very similar. The only thing we need to change is the `nbStates` argument in `fitHMM`, and the initial parameter values. We now need three values for each parameters: one for each state.

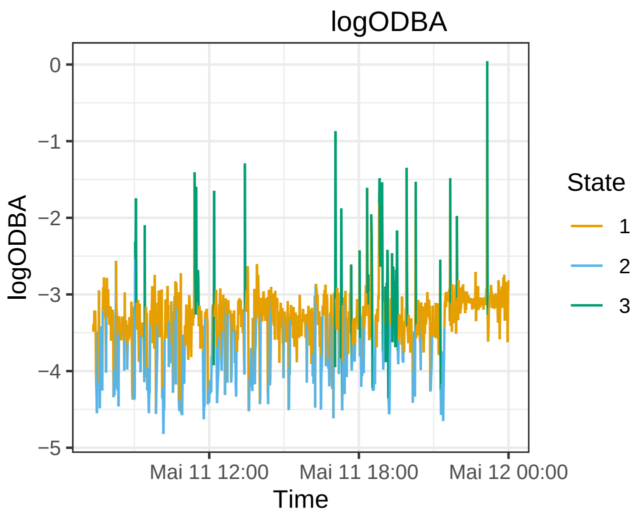
```r
# List of initial parameters
mean0 <- c(-4, -3, -2)
sd0 <- c(0.5, 0.5, 0.5)
Par0_3s <- list(logODBA = c(mean0, sd0))

# Fit HMM
hmm_3s <- fitHMM(data = data_hmm, nbStates = 3, dist = dist, Par0 = Par0_3s)

# Plot state-dependent distributions
plot(hmm_3s, ask = FALSE)
```

```
Decoding state sequence... DONE
```

```r
# Get most likely state sequence
data_hmm$viterbi_3s <- factor(viterbi(hmm_3s))

# Plot log-ODBA coloured by states
```

```r
ggplot(tail(data_hmm, 1000), aes(Time, logODBA, col = viterbi_3s, group = NA)) +
  geom_line() +
  scale_color_manual(values = pal, name = "State")
```

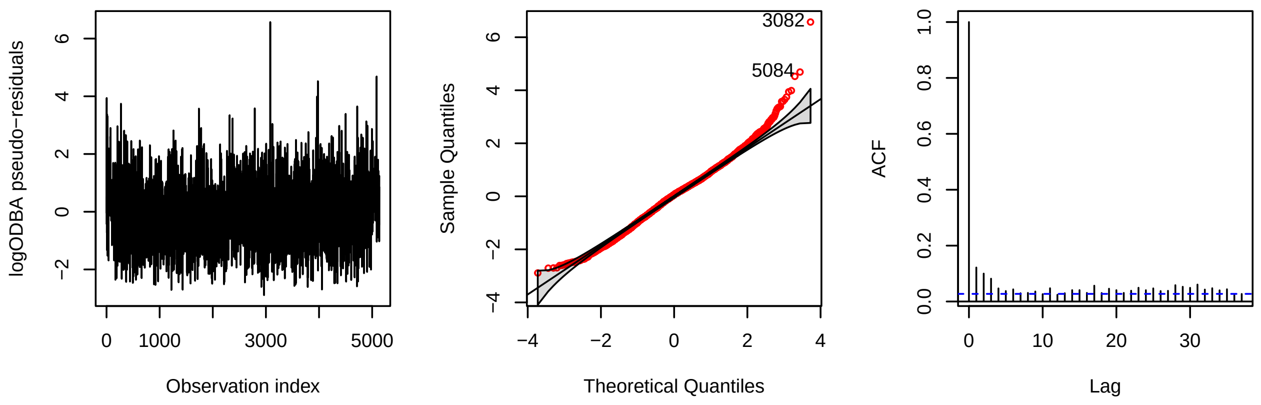




The three states look like they might have a nicer interpretation: state 1 captures little to no movement, state 3 captures high activity, and state 2 is intermediate activity. We can also look at the pseudo-residuals, which suggest that the fit is much better than in the 2-state model.

```r
plotPR(hmm_3s)
```

```
Computing pseudo-residuals...
```

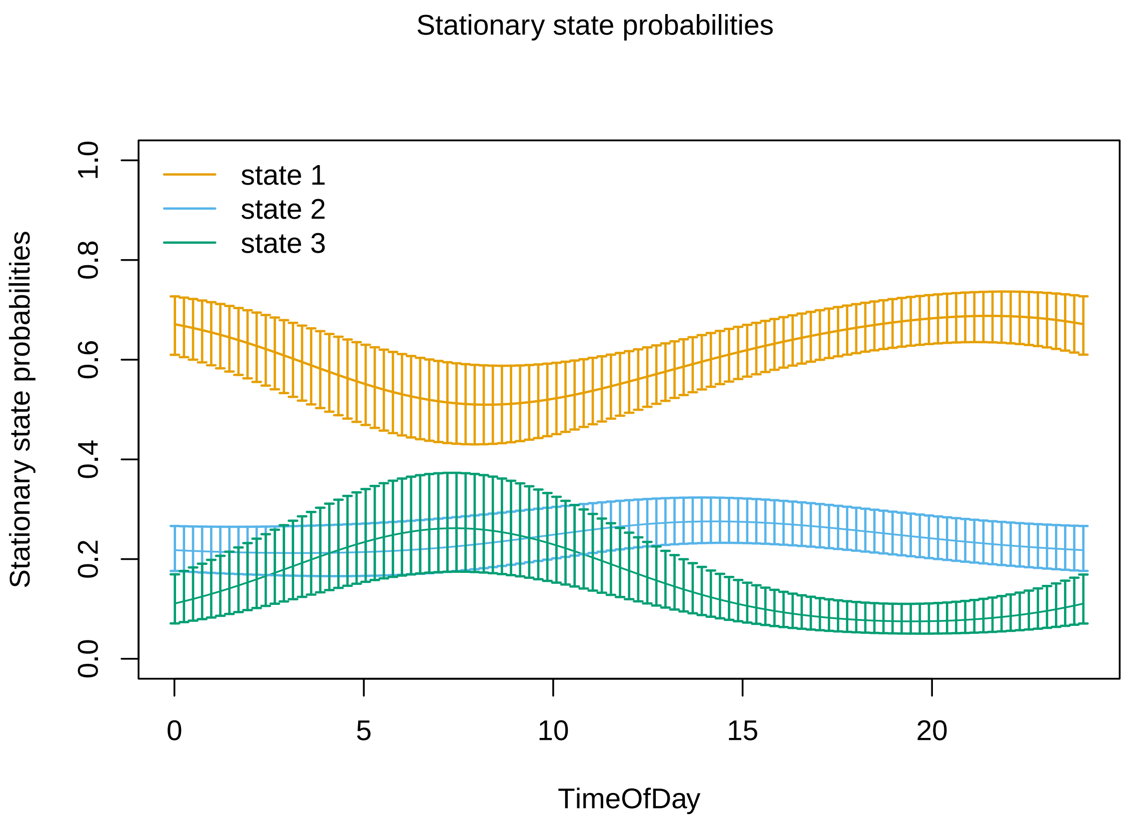


2.4 Model 3: covariates on the transition probabilities

Just like for tracking data, we can include covariate in the hidden state model, to capture the effect of environmental or individual-specific variables on the animal's behaviour. In the following, we include the effect of time of day, to investigate possible diurnal patterns in the shark's activity.

Time of day is defined between 0 and 24, and it should have a cyclical effect, i.e., the transition probabilities at 24:00 should match those at 00:00. One way to do this is to use trigonometric functions, as described by Leos-Barajas et al. (2017). In momentuHMM, the `cosinor` function can be used in the model formula to create a cyclical effect with some specified period (here, 24).

```
# Fit model with time of day covariate
hmm_3s_tod <- fitHMM(data = data_hmm, nbStates = 3, dist = dist, Par0 = Par0_3s,
                     formula = ~ cosinor(TimeOfDay, period = 24))

# Plot stationary state probabilities as function of time of day
plotStationary(hmm_3s_tod, plotCI = TRUE)
```



There is some evidence of a time of day effect, such that the high-activity state is most likely to occur in the morning between 5:00 and 10:00.

References

Leos-Barajas, Vianey, Theoni Photopoulou, Roland Langrock, Toby A Patterson, Yuuki Y Watanabe, Megan Murgatroyd, and Yannis P Papastamatiou. 2017. "Analysis of Animal Accelerometer Data Using Hidden Markov Models." *Methods in Ecology and Evolution* 8 (2): 161–73.