Analysing ecological data with HMMs in R

Jan-Ole Koslik, Roland Langrock

3 March 2024

1 Analysing tracking data with moveHMM

In this exercise, we analyse elephant 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 required packages
# install.packages("moveHMM")
# install.packages("ggplot2")
# install.packages("sp")
library(moveHMM)
library(ggplot2)
library(dplyr)
library(sp)
```

The `R` package `moveHMM` has extensive documentation in several vignettes:

```r
help(package = "moveHMM")
```

1.1 Data pre-processing

Originally, 15 elephants from the Etosha National Park where fitted with GPS collars to collect hourly movement data (Tsalyuk et al. (2019)). For simplicity, the raw data have already been preprocessed and we consider the track of only one individual. However, we still have a little preprocessing work to do:

```r
## Loading the data
elephant_data = read.csv("http://www.rolandlangrock.com/elephant_data.csv")

## Defining time of day variable (could also be done with lubridate)
elephant_data$timestamp = strptime(elephant_data$timestamp,
                                   "%Y-%m-%d %H:%M:%S", tz = "GMT")

hours = as.numeric(format(elephant_data$timestamp, "%H"))
minutes = as.numeric(format(elephant_data$timestamp, "%M"))

elephant_data$timeOfDay = hours + (minutes/60)
```

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

- `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("location.long", "location.lat")`.

For our data, we hence use:

```r
## Compute step lengths and turning angles
hmm_data <- prepData(trackData = elephant_data,
                     coordNames = c("location.long", "location.lat"),
                     type = "LL")
```

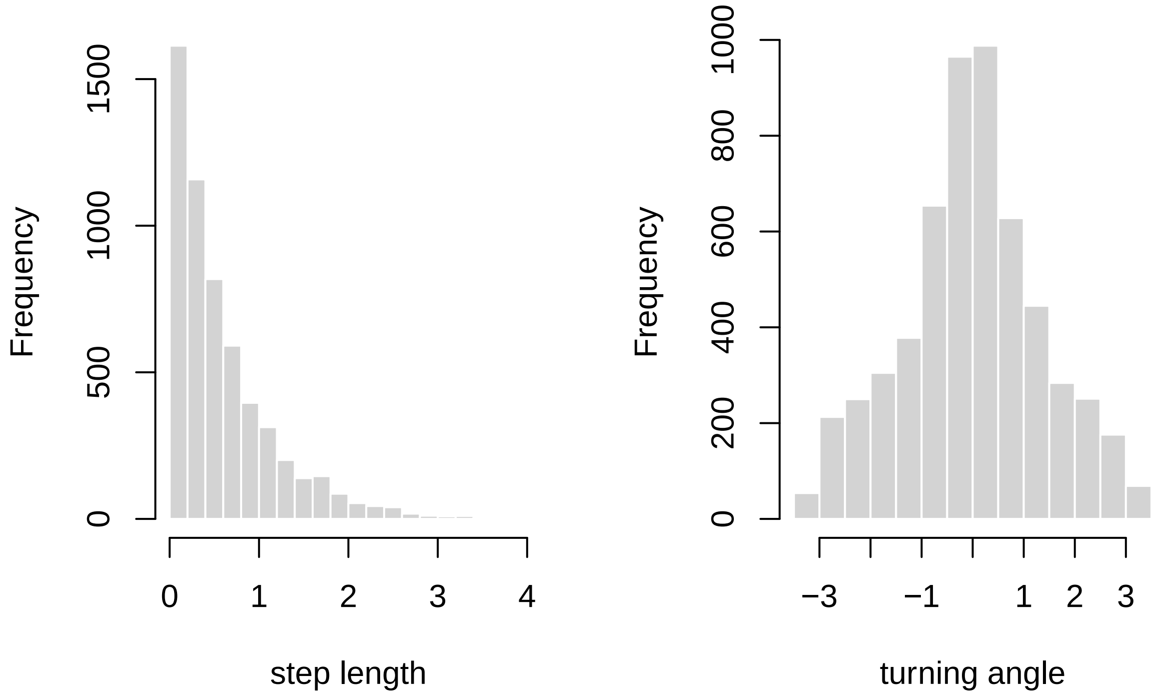
Step lengths and turning angles are computed; the returned object is a data frame. The following commands can be used to get an overview of the data and to visualise the tracks:

```
## Get an overview of the data and to visualise the first 2 tracks
head(hmm_data)
plot(hmm_data, ask = FALSE)
```

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$step, xlim = c(0, 4), breaks = 30, main = "",
     xlab = "step length", border = "white")
hist(hmm_data$angle, main = "",
     xlab = "turning angle", border = "white")
```



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 von Mises state-dependent distributions for the turning angles, we need N mean and concentration parameters, respectively.

```
## Initial values for the parameters of the state-dependent distributions
# For step length
step_mean_par0 <- c(0.1, 0.4, 1) # means
step_SD_par0 <- c(0.1, 0.3, 0.6) # 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(0.1, 0.9, 2) # 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 this 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 function call to fit a simple 3-state HMM to our data is:

```
## Fit HMM
mod <- fitHMM(data = hmm_data,
              nbStates = 3,
              stepPar0 = step_par0,
              anglePar0 = angle_par0,
              stationary = TRUE, # we assume a stationary Markov chain
              verbose = 0)
```

To watch the iterations of the numerical likelihood optimisation – which can be both meditative but also helpful – we need to specify `verbose = 2`. The fitted model is stored as `mod`, which is used as an input to the following functions.

1.4 Results

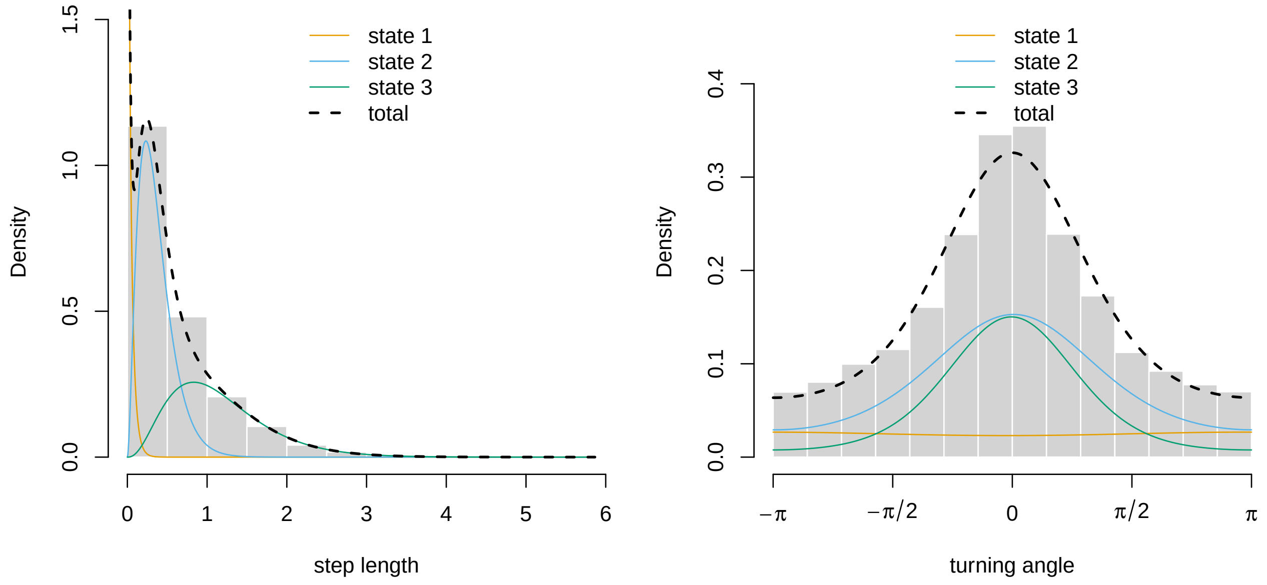
To obtain the estimated model parameters, we can call the fitted model object:

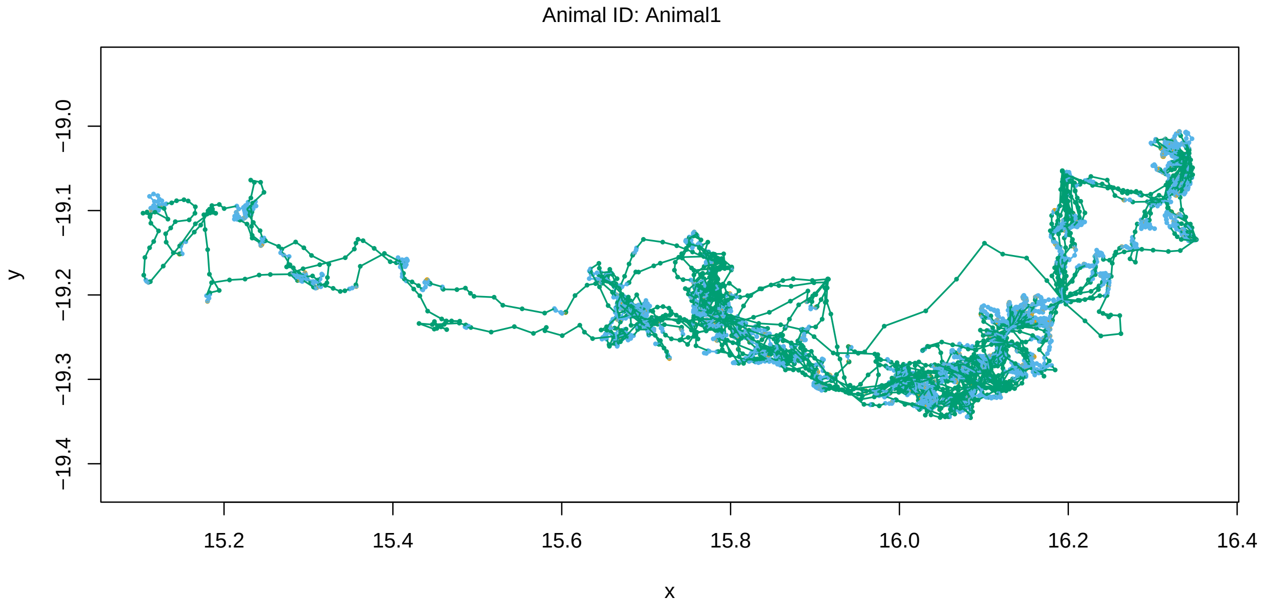
```
## Print estimated model parameters
mod
```

```
Value of the maximum log-likelihood: -10605.32

Step length parameters:
----------------------
             state 1            state 2             state 3
mean      0.03334228 0.3623886821862348 1.16298300312458247
sd        0.04210630 0.2173691778371893 0.62010211793242820
zero-mass 0.00216311 0.0000000006920838 0.00000000001319144

Turning angle parameters:
------------------------
                 state 1    state 2     state 3
mean          3.06892525 0.01612904 -0.01245977
concentration 0.07409659 0.82729330  1.48640393

Regression coeffs for the transition probabilities:
--------------------------------------------------
              1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1    3 -> 2
intercept -0.6081662 -2.442193 -1.780627 -1.618818 -4.210936 -1.332954

Transition probability matrix:
-----------------------------
          [,1]      [,2]       [,3]
[1,] 0.6130012 0.3336861 0.05331268
[2,] 0.1233166 0.7317081 0.14497535
[3,] 0.0116012 0.2062504 0.78214845

Initial distribution:
--------------------
[1] 0.1632017 0.4784570 0.3583412
```

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

```
## Plot the fitted state-dependent distributions
par(mfrow = c(1, 2))
plot(mod, plotTracks = TRUE, 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, we specify `plotTracks = TRUE`.

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

```
## Global state decoding
(states = viterbi(mod))
## Local state decoding
(stateprobs = stateProbs(mod))
## Show both
plotStates(mod, ask = FALSE, animals = 1)
```

1.5 Adding covariates on the state transition probabilities

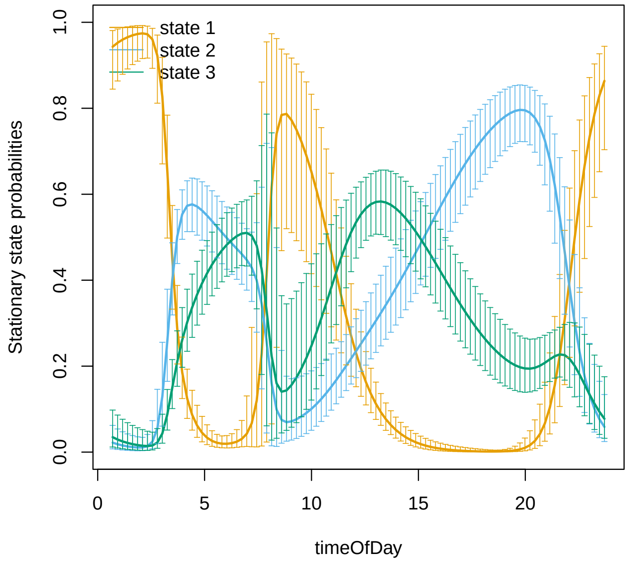
Covariates can be included in the hidden state model to capture the effect of environmental or individual-specific variables on the animal's behaviour. In `moveHMM`, covariates can be included by specifying a `formula`. Here, we are interested in modelling the diurnal variation present in the elphant's behaviour. 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), where in the simplest case we include one sine and one cosine frequency.

```
## Fit HMM with trigonometric effect of time of day
mod_tod <- fitHMM(data = hmm_data,
                  nbStates = 3,
                  stepPar0 = step_par0,
                  anglePar0 = angle_par0,
                  formula = ~ sin(2*pi*timeOfDay/24) + cos(2*pi*timeOfDay/24),
                  verbose = 0)
```

```
mod_tod
```

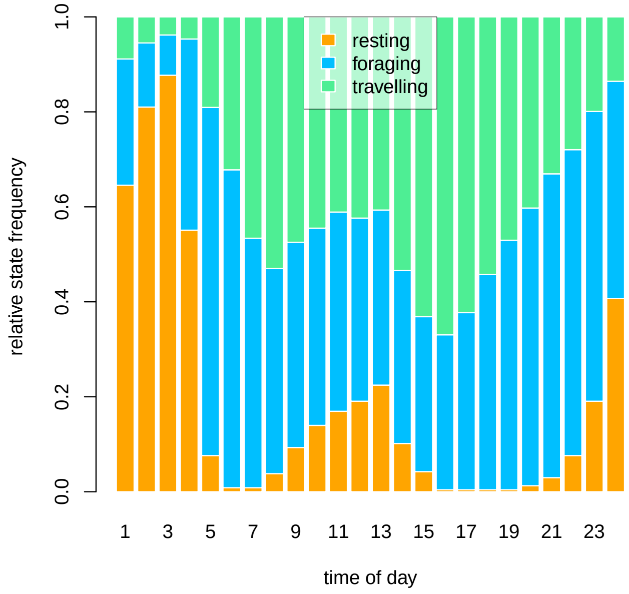
To visualise the probability of being in either of the 3 states as functions of the time of day, we can plot the stationary state probabilities using `plotStationary`.

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



With the current implementation, these probabilities are only approximately correct - they are essentially shifted by about 1-3 hours - but we will inform the package maintainers how to fix this, so you can expect this functionality to work properly in the near future. At the moment, we propose to look at the decoded states as a function of the time of day to interpret the diurnal variation in activity budgets:

```r
# summarise relative frequencies by time of day
hmm_data$timeOfDay2 = round(hmm_data$timeOfDay)
hmm_data$states = viterbi(mod_tod)

statefreq = hmm_data %>%
  group_by(timeOfDay2) %>%
  summarise(n = n(), freq1 = sum(states==1)/n,
            freq2 = sum(states==2)/n, freq3 = sum(states==3)/n)

# make a bar plot
color = c("orange", "deepskyblue", "seagreen2")
barplot(cbind(freq1, freq2, freq3) ~ timeOfDay2,
        data = statefreq, col = color, border = "white",
        xlab = "time of day", ylab = "relative state frequency")
legend("top", fill = color, legend = c("resting", "foraging", "travelling"),
       box.lwd = 0, bg = "#ffffff98", border = "white")
```



We can increase model flexibility by including higher sine and cosine frequencies, which might be necessary when transition probabilities peak more than once in a day.

```
# Fit HMM with trigonometric effect of time of day
mod_tod2 <- fitHMM(data = hmm_data,
                  nbStates = 3,
                  stepPar0 = step_par0,
                  anglePar0 = angle_par0,
                  formula = ~ sin(2*pi*timeOfDay/24)+cos(2*pi*timeOfDay/24)+
                    # aditionally double the frequency
                    sin(2*pi*timeOfDay/12)+cos(2*pi*timeOfDay/12),
                  verbose = 0)
```

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_tod, mod_tod2)
```

```
     Model       AIC
```

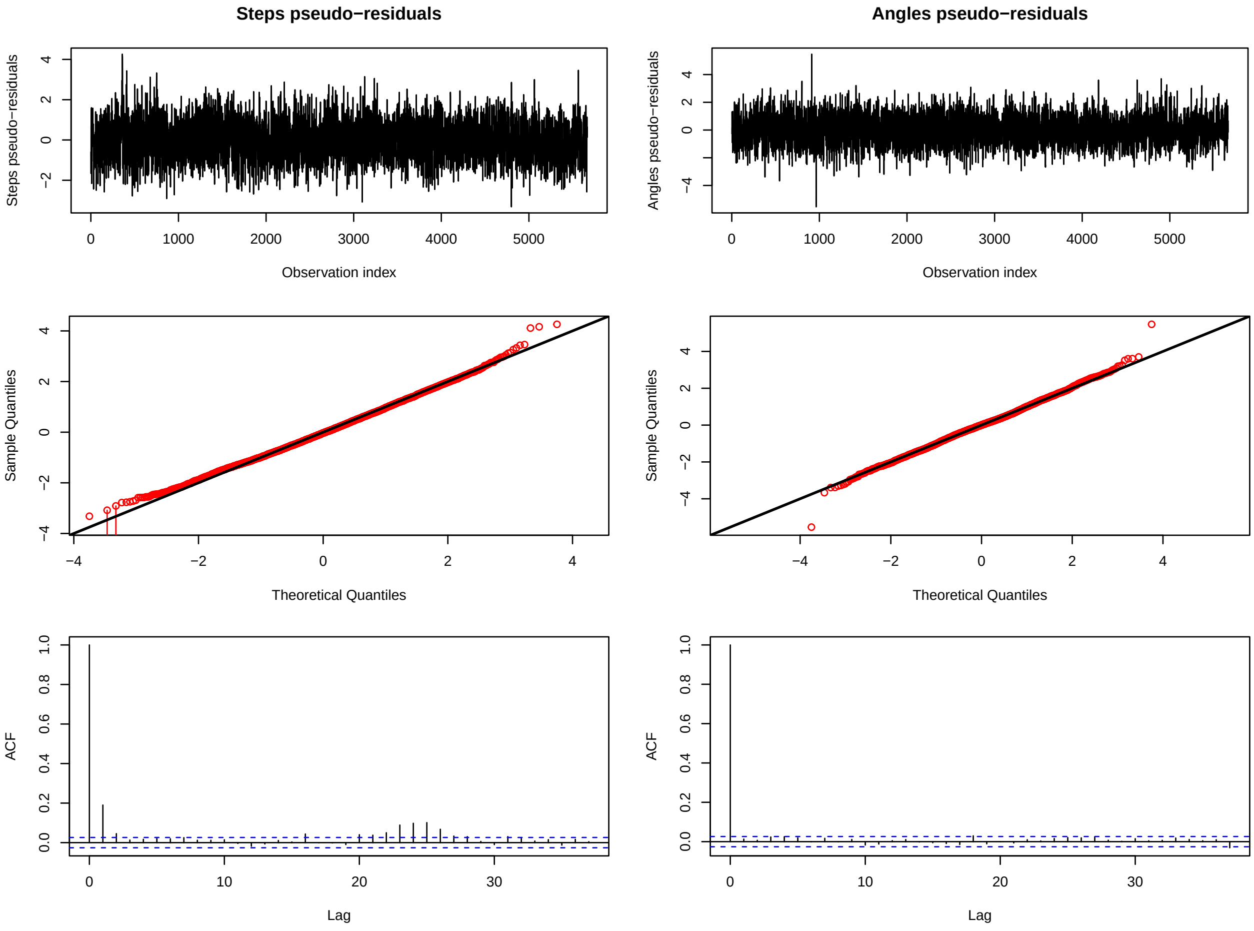
```
1 mod_tod2 20275.50
2  mod_tod 20548.84
3      mod 21252.64
```

In this case, we find that the most complicated model shows the best fit in terms of AIC. However, one should not only rely on the AIC for HMM model selection.

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.

```r
## Plot QQ and ACF plots
plotPR(mod_tod2)
```



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 – in fact, a fit as good as the one shown here is quite rare in real-data applications. The ACF plot suggests that there is only little correlation that is not captured by our HMM. We have some difficulties in capturing the periodical structure caused by autocorrelation for lags ~ 24 hours.

2 Analysing accelerometer data with momentuHMM

The R package momentuHMM extends 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 and showcase 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.

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

The R package momentuHMM also has extensive documentation:

```
help(package = "momentuHMM")
```

2.1 Data

We consider accelerometer data collected for 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, such that we thinned them to a 1-min resolution for this analysis, for computational speed. The observed variable is the overall dynamic body acceleration (ODBA), which is a measure of the magnitude of the three-dimensional acceleration of the animal.

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

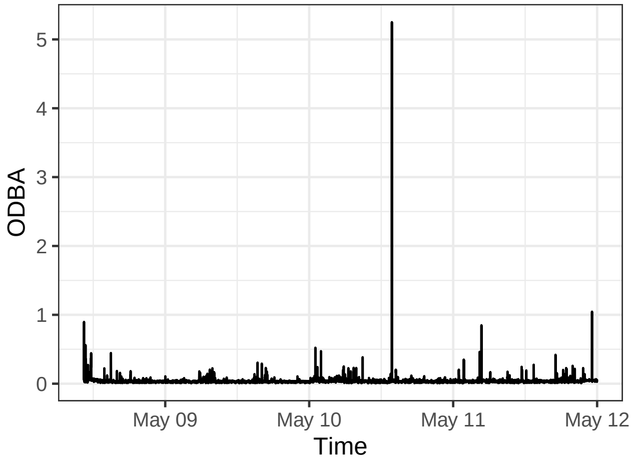
```
                 Time       ODBA TimeOfDay
1 2013-05-08 10:25:45 0.05538590  10.42896
2 2013-05-08 10:26:45 0.36452689  10.44563
3 2013-05-08 10:27:45 0.89695752  10.46230
4 2013-05-08 10:28:45 0.04144502  10.47896
5 2013-05-08 10:29:45 0.06610186  10.49563
6 2013-05-08 10:30:45 0.03380349  10.51230
```

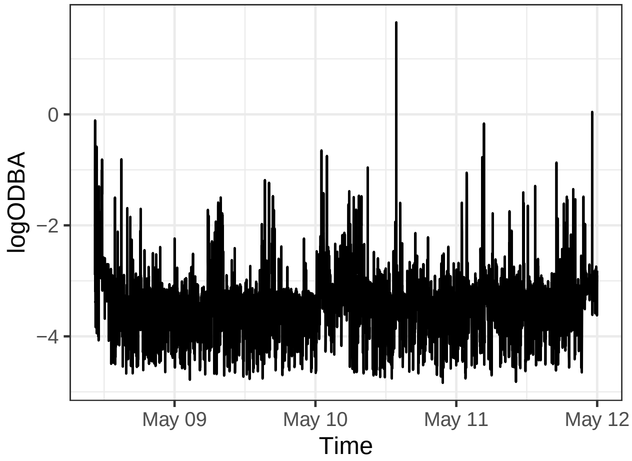
```
nrow(data)
```

```
[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 prepare 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. A difference to `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 a challenging problem in general, and this choice typically requires combining biological expertise and model checking to find a trade-off between model complexity and interpretation (Pohle et al. (2017)). It is often helpful to start from a simple model with only two states, then building up complexity if necessary.

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. The idea is to pick values that are plausible given the data and our expectations of the states – from such sensible initial guesses, the numerical optimiser should usually be able to identify the best parameter values (i.e. the global maximum of the likelihood). 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. While some of the argument names are slightly different to `moveHMM`, the syntax is still very similar.

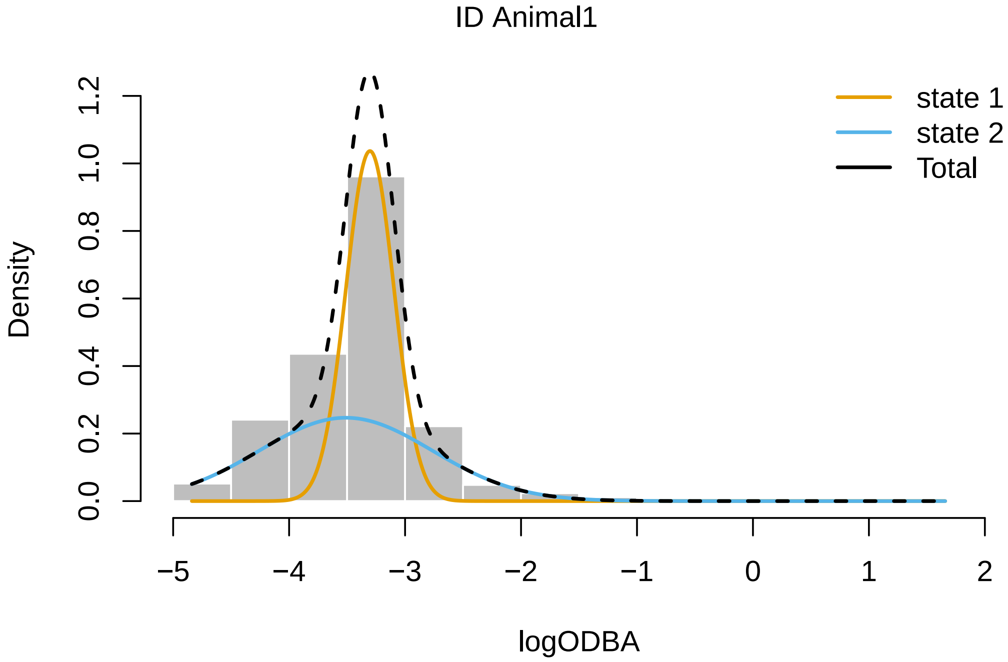
```
hmm_2s <- fitHMM(data = data_hmm, nbStates = 2, dist = dist, Par0 = Par0_2s)
```

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.

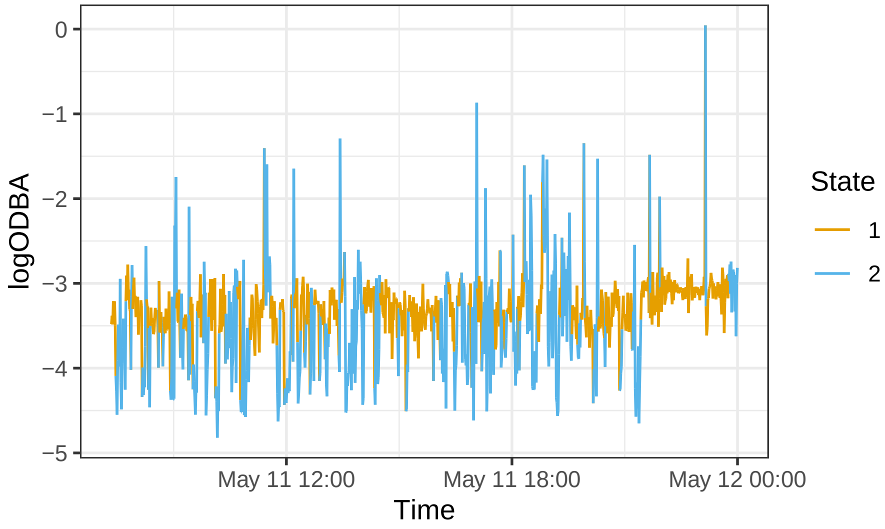
```
plot(hmm_2s, ask = FALSE)
```

```
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.

```
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")
```



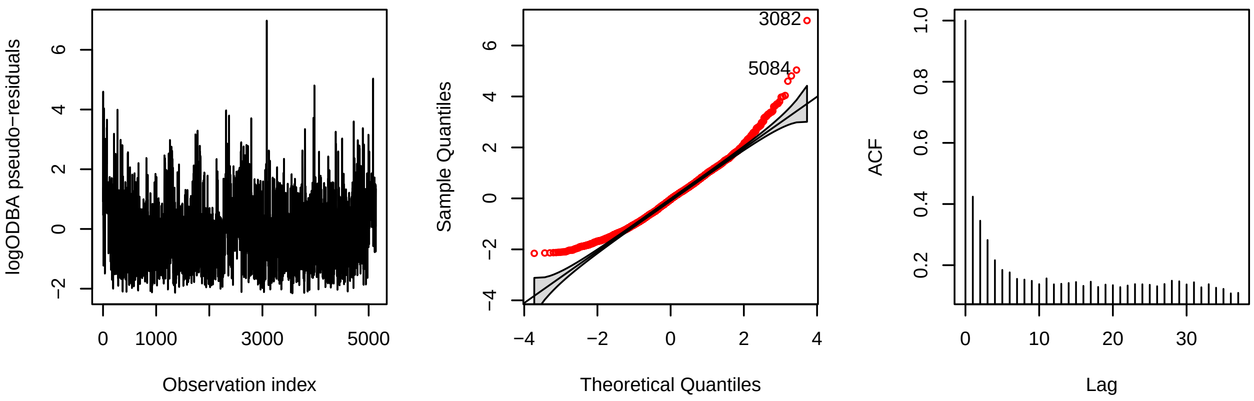
These plots suggest that state 2 captures both the lower tail and higher tail of the distribution of log-ODBA, whereas state 1 captures intermediate values. This makes biological interpretation a little tricky, because it looks like state 2 includes several behaviours (either no activity or high activity).

2.2.4 Model checking

We again utilise Pseudo-residuals for checking the goodness of fit of our HMM.

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

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



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.

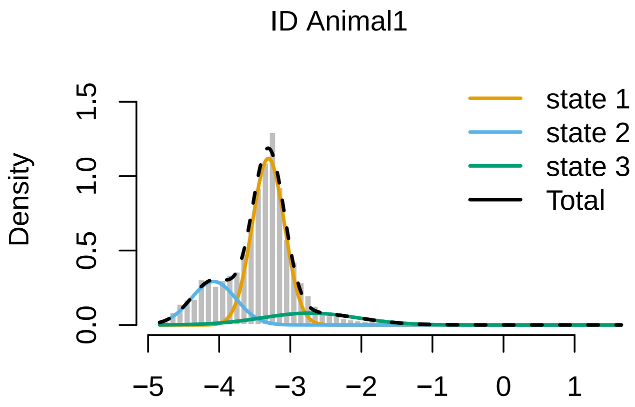
2.3 Model 2: three states

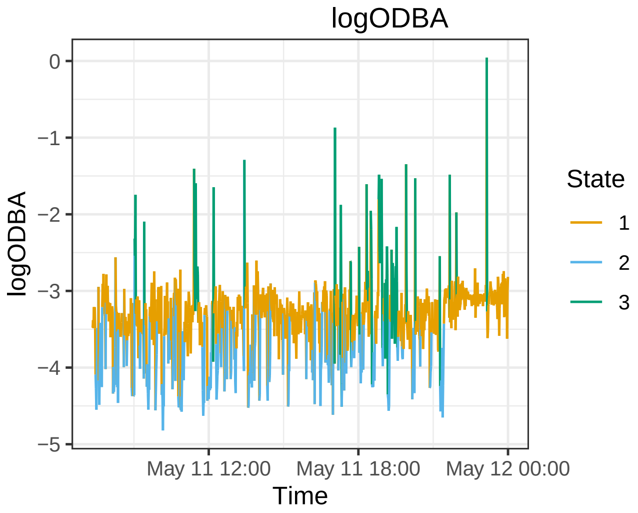
The code required to create a 3-state model is very similar. The only thing we need to change is the `nbStates` argument in `fitHMM`, and the initial parameter values. We now need three values for each parameter (mean, standard deviation).

```
# 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, breaks = 50)
```

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

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

# Plot log-ODBA coloured by states
ggplot(tail(data_hmm, 1000),
       aes(Time, logODBA, col = viterbi_3s, group = NA)) +
  geom_line() +
  scale_color_manual(values = pal, name = "State")
```

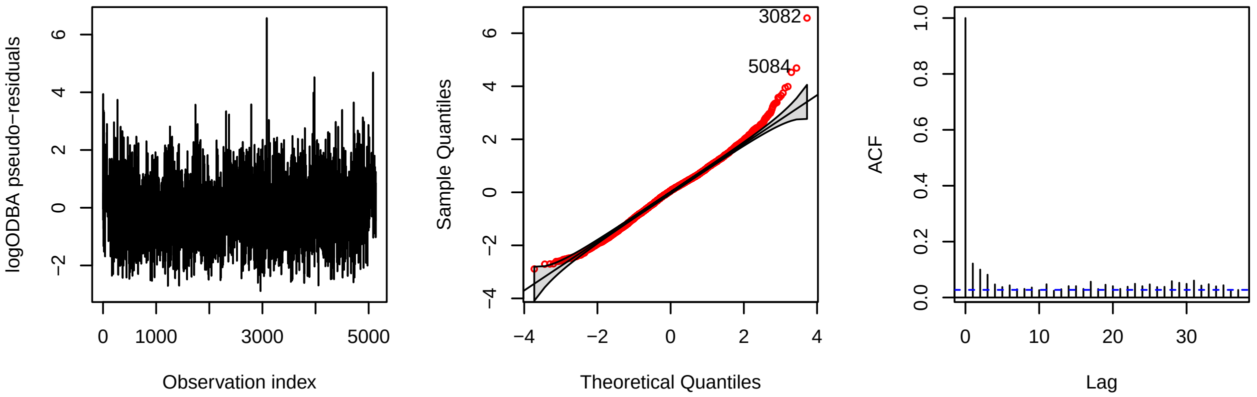




The three states are easier to interpret: state 1 seems to indicate normal swimming, state 2 may be some sort of gliding (?) and state 3 is linked to activity bursts. We can also look at the pseudo-residuals, which suggest that the fit is much better than that of the 2-state model.

```
plotPR(hmm_3s)
```

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

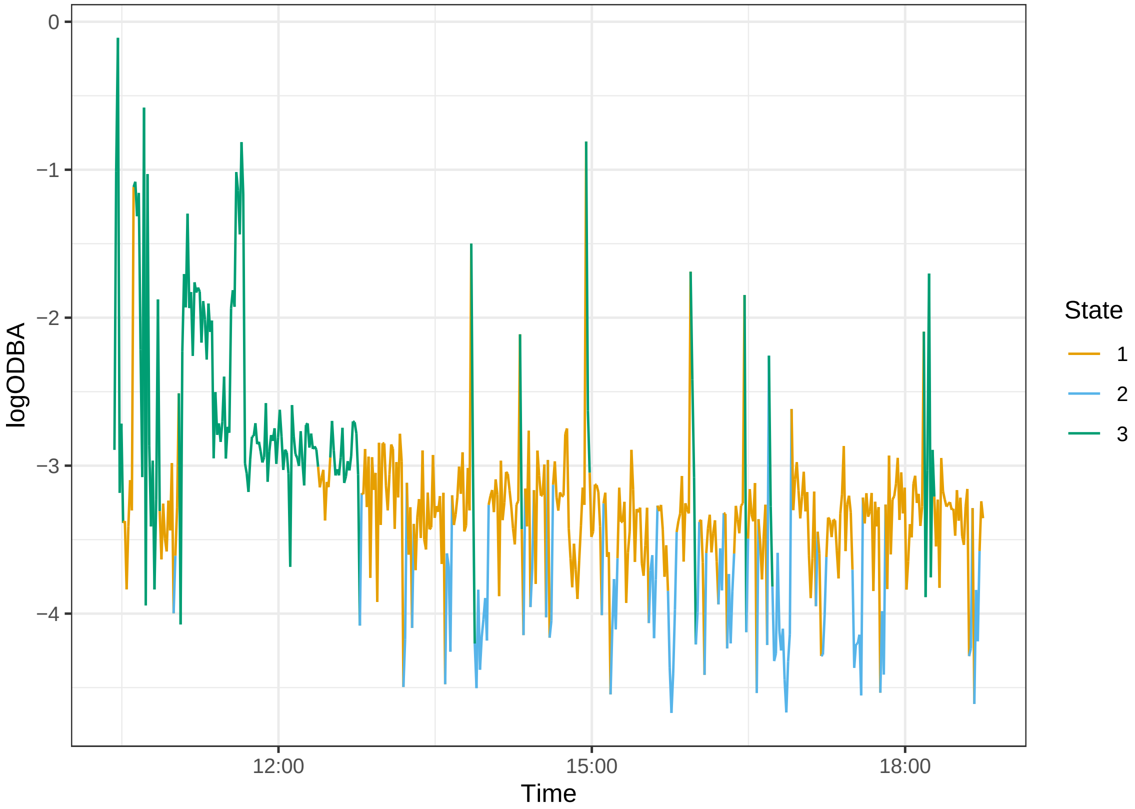


While still not perfect (notable residual autocorrelation, slight lack of fit in the marginal distribution's tails), these residuals look much better than the 2-state model's residuals. To obtain a better fit, it would probably make sense to include autoregressive structures in the state-dependent process (Slide 58), as the temporal resolution of the data is high.

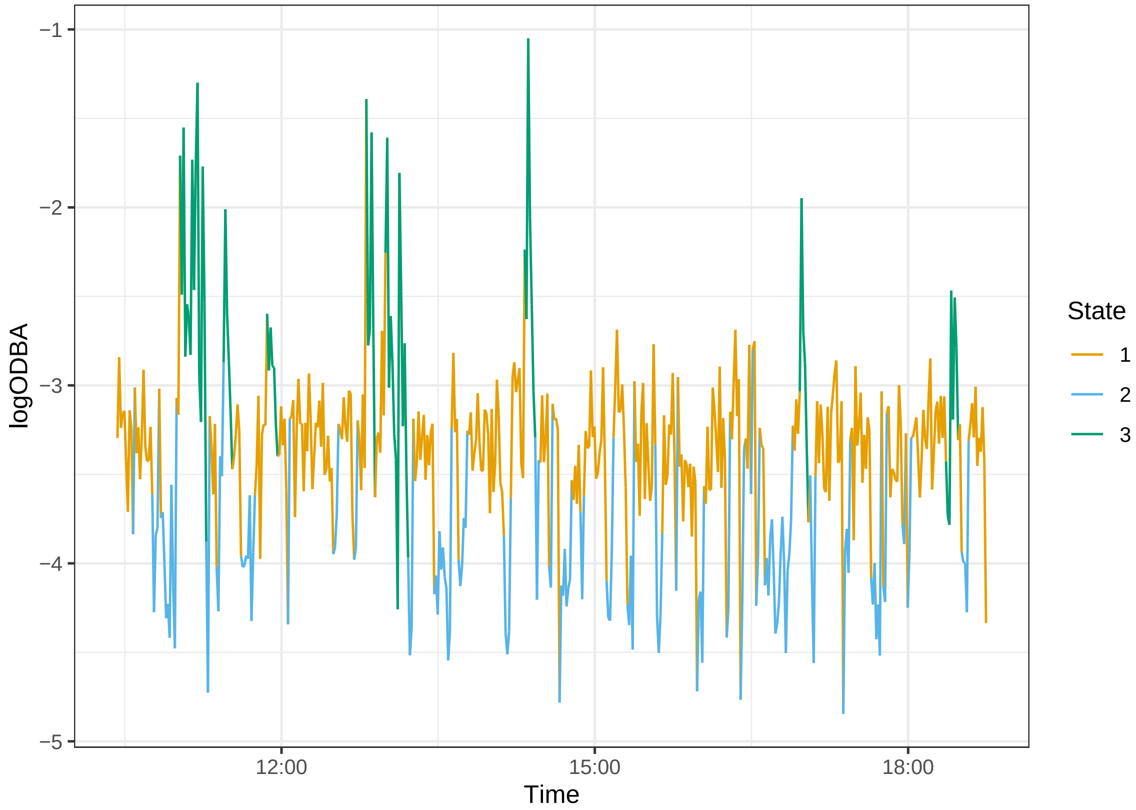
2.4 Simulating from the estimated model

Another option for checking the model fit is to simulate data from the estimated model and compare it to the observed data. If the model describes the true data well, these two should not deviate substantially.

```
simdata = simData(model = hmm_3s,
                  states = TRUE,
                  obsPerAnimal = nrow(data_hmm))
simdata$Time = data_hmm$Time

# Plotting original data coloured by decoded states
ggplot(head(data_hmm, 500),
       aes(Time, logODBA, col = viterbi_3s, group = NA)) +
  geom_line() +
  scale_color_manual(values = pal, name = "State")
```



```
# Plotting simulated data coloured by states
ggplot(head(simdata, 500),
       aes(Time, logODBA, col = factor(states), group = NA)) +
  geom_line() +
  scale_color_manual(values = pal, name = "State")
```

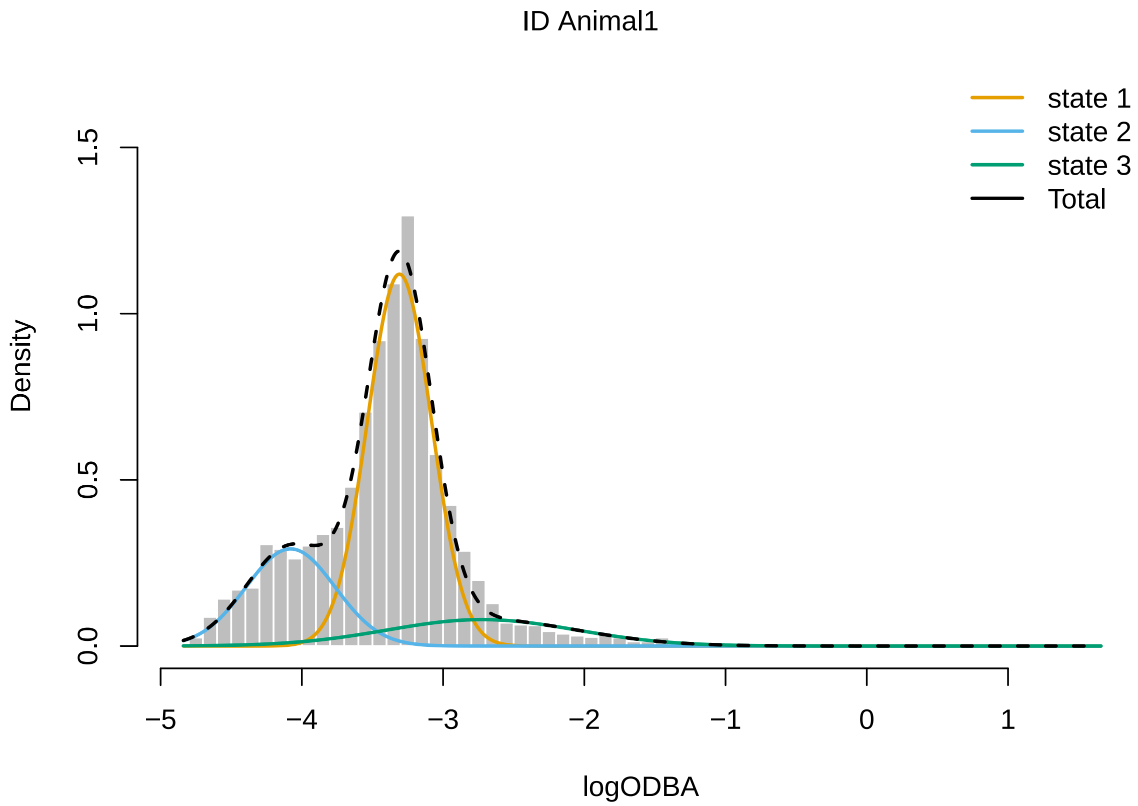


Overall, our model looks good. We only see minor issues with state 3, as in the real data some sojourn times are very long while most are very short and in the simulated data this is more homogeneous.

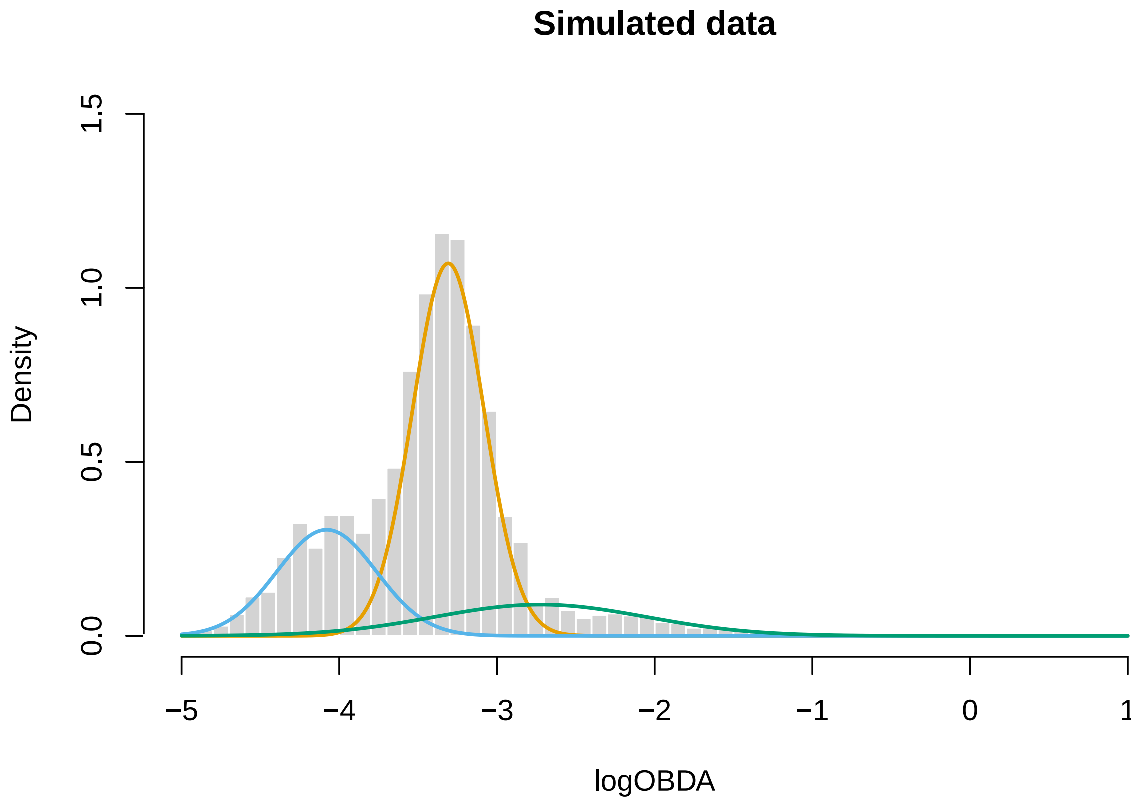
We can also visualize the marginal distribution implied by our model:

```r
# getting state-dependent parameters from fitted model
mu = hmm_3s$mle$logODBA[1,] # 3 means
sigma = hmm_3s$mle$logODBA[2,] # 3 standard deviations
delta = stationary(hmm_3s)[[1]] # stationary distribution

# histogram of original data
plot(hmm_3s, ask = FALSE, breaks = 50)
```

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



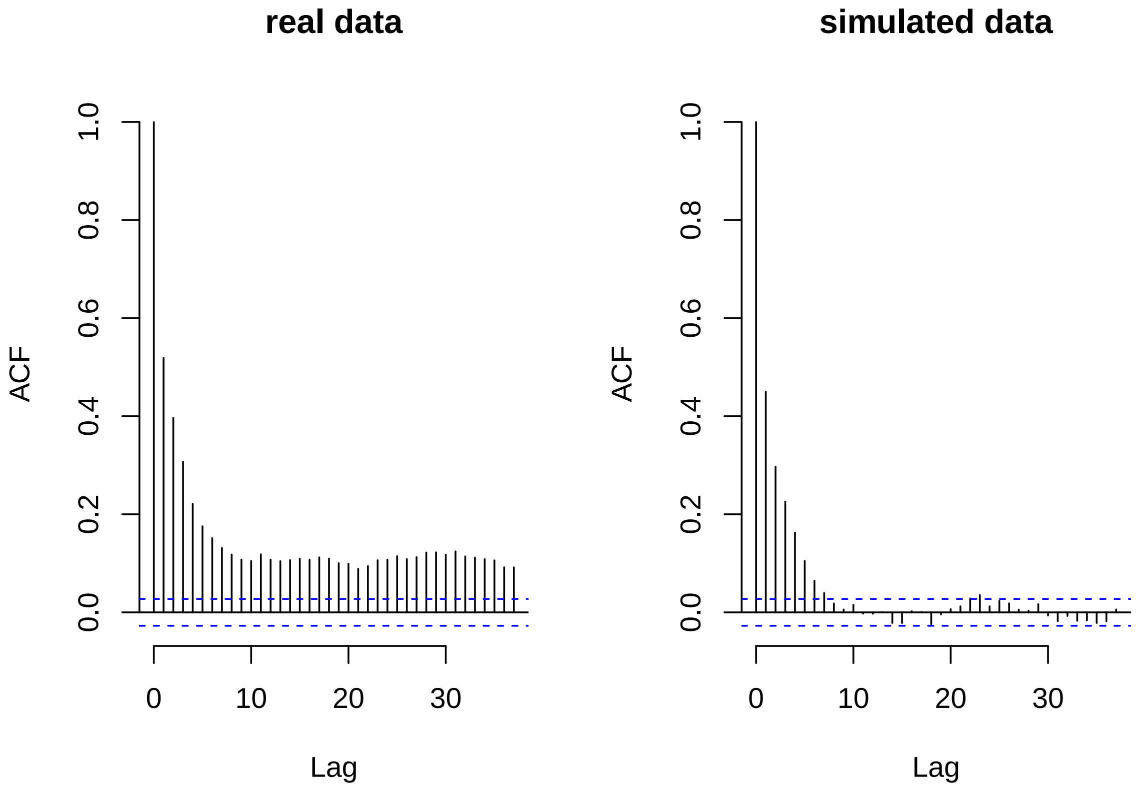
```r
# histogram of simulated data
hist(simdata$logODBA, prob = TRUE, breaks = 50, border = "white",
     xlim = c(-5,1), ylim = c(0,1.5), main = "Simulated data", xlab = "logOBDA")
curve(delta[1]*dnorm(x, mu[1], sigma[1]), add = TRUE, lwd=2, col=pal[1], n=500)
curve(delta[2]*dnorm(x, mu[2], sigma[2]), add = TRUE, lwd=2, col=pal[2], n=500)
curve(delta[3]*dnorm(x, mu[3], sigma[3]), add = TRUE, lwd=2, col=pal[3], n=500)
```



The marginal distribution looks really good!

Lastly, we can investigate the autocorrelation function of the real and simulated data.

```
# autocorrelation function
par(mfrow = c(1,2))
acf(data_hmm$logODBA, main = "real data", bty = "n")
acf(na.omit(simdata$logODBA), main = "simulated data", bty = "n")
```



In this case, we find a minor lack of fit regarding larger lags which is really typical for HMM analyses due to the conditional independece assumption. This can be improved by including autoregressive structures in the state-dependent process (Slide 60), but as the inferential focus typically lies on the state process, it might not be worth the extra effort.

2.5 In-house advertising

If you want to delve deeper into building HMMs and their extensions by writing custom code, you should check out the vignettes of my `R` package Lcpp. They demonstrate how to simulate and estimate a wide range of models.

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.

Pohle, Jennifer, Roland Langrock, Floris M Van Beest, and Niels Martin Schmidt. 2017. "Selecting the Number of States in Hidden Markov Models: Pragmatic Solutions Illustrated Using Animal Movement." *Journal of Agricultural, Biological and Environmental Statistics* 22: 270–93.

Tsalyuk, Miriam, Werner Kilian, Björn Reineking, and Wayne Marcus Getz. 2019. "Temporal Variation in Resource Selection of African Elephants Follows Long-Term Variability in Resource Availability." *Ecological Monographs* 89 (2): e01348.