Tutorial - Hidden Markov Models

Hidden Markov Models: tutorial goals and set up

The goal of this tutorial is to explore how to fit hidden Markov models (HMMs) to movement data. To do so, we will investigate a new R package, `momentuHMM`. This package builds on a slightly older package, `moveHMM`, that was developed by Théo Michelot , Roland Langrock, and Toby Patterson, see associated paper: https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.12578. `momentuHMM`, was developed by Brett McClintock and Théo Michelot. momentuHMM has new features such as allowing for more data streams, inclusion of covariates as raster layers, and much more, see associated paper: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.12995.

Setup and data preparation

First, let's load the packages that we will need to complete the analyses. Off course you need to have them installed first.

```
library(momentuHMM) # Package for fitting HMMs, builds on moveHMM
library(sp) # For GIS tasks (e.g., geographic transformations)
library(raster) # For raster spatial covariates
```

One of the main features of the data used with HMMs is that locations are taken at regular time steps and that there are negligible position error. For example, HMMs are adequate to use on GPS tags that take locations at a set temporal frequency (e.g., every 2 hrs). Without reprocessing, HMMs are not particularly good for irregular times series of locations or locations with large measurement error (e.g., you would need to preprocess Argos data before applying HMMs).

I work mainly on aquatic species, and unfortunately, movement of aquatic species is rarely taken at regular time intervals and without measurement error. For example, the data set we will work on, is the movement track of a grey seal tagged with a Fastlock GPS tag. This is a subset of the data set used in the paper from Whoriskey et al. 2017 (https://onlinelibrary.wiley.com/doi/abs/10.1002/ece3.2795), which applies a set of different HMMs to the movement data of various aquatic species. The original dataset is available in the online supplementary information.

Let's read the data file and peak at it.

```
# Make sure you are in the good directory!
seal <- read.csv("data/seals.csv", stringsAsFactors = FALSE)
# Look at the data
head(seal)
```

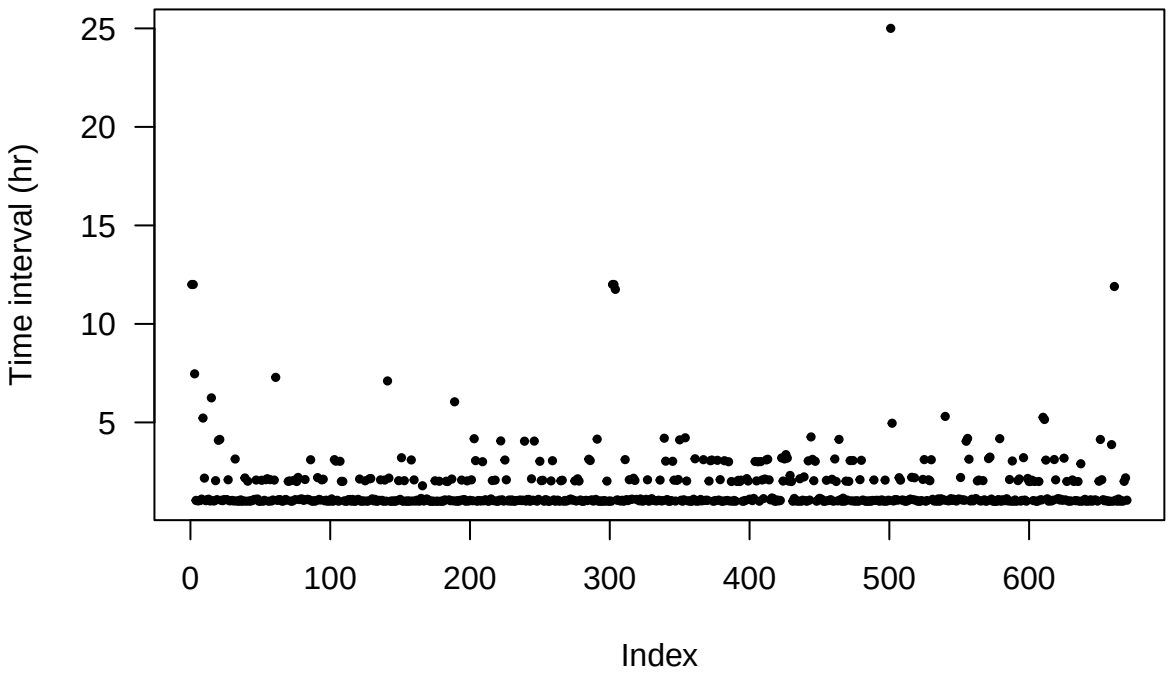
```
##                  date       lon      lat
## 1 2014-02-18 03:32:08 -60.16754 43.97577
## 2 2014-02-18 15:32:08 -60.16785 43.97579
## 3 2014-02-19 03:32:08 -60.16801 43.97568
## 4 2014-02-19 11:00:07 -60.17035 43.97532
## 5 2014-02-19 12:02:40 -60.20350 43.94928
## 6 2014-02-19 13:04:07 -60.22503 43.91563
```

The files has three columns: date with the date and time, lon with the longitude, and lat with latitude. It's clear from the date column that the locations are taken at irregular time intervals. So just like in Whoriskey et al. 2017, we will regularize it. Note, that `momentuHMM` has functions to preprocess irregular movement time series and/or time series with large measurement errors (e.g., `crawlWrap`). However, these functions are much more complex, they fit state-space models to the data, and we will not cover these in this tutorial. Some of the material covered in the state-space model tutorial will help understand these functions, but you can also see the vignette of `momentuHMM` to learn more about the preprocessing functions available. For now, as in Whoriskey et al. 2017, we will assume that the animal moves in a straight line and at constant speed between two locations and interpolate the location at regular time intervals based this assumption. Note that this kind of interpolation may not be adequate in many circumstances and see activities to investigate the effects of the choice of the time interval on the analysis.

```
# First, let's transform the date/time info into a
# proper time format
seal$date <- as.POSIXct(seal$date, format = "%Y-%m-%d %H:%M:%S",
tz = "GMT")
# Let's find the most appropriate time interval
# Calculate the time intervals
tint <- as.numeric(diff(seal$date), units = "hours")
# Let's look at the time intervals
summary(tint)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9997  1.0131  1.0579  1.6664  2.0333 25.0017
```

```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
plot(tint, ylab = "Time interval (hr)", pch = 19, cex = 0.5,
las = 1)
```




```
# Let's go for 2 hrs, look at the activities to see
# the effect of this choice
# Regularising Create a 2hr sequence (2*60*60) from
# the first time to the last time
ti <- seq(seal$date[1], seal$date[length(seal$date)],
by = 2 * 60 * 60)
head(ti)
```

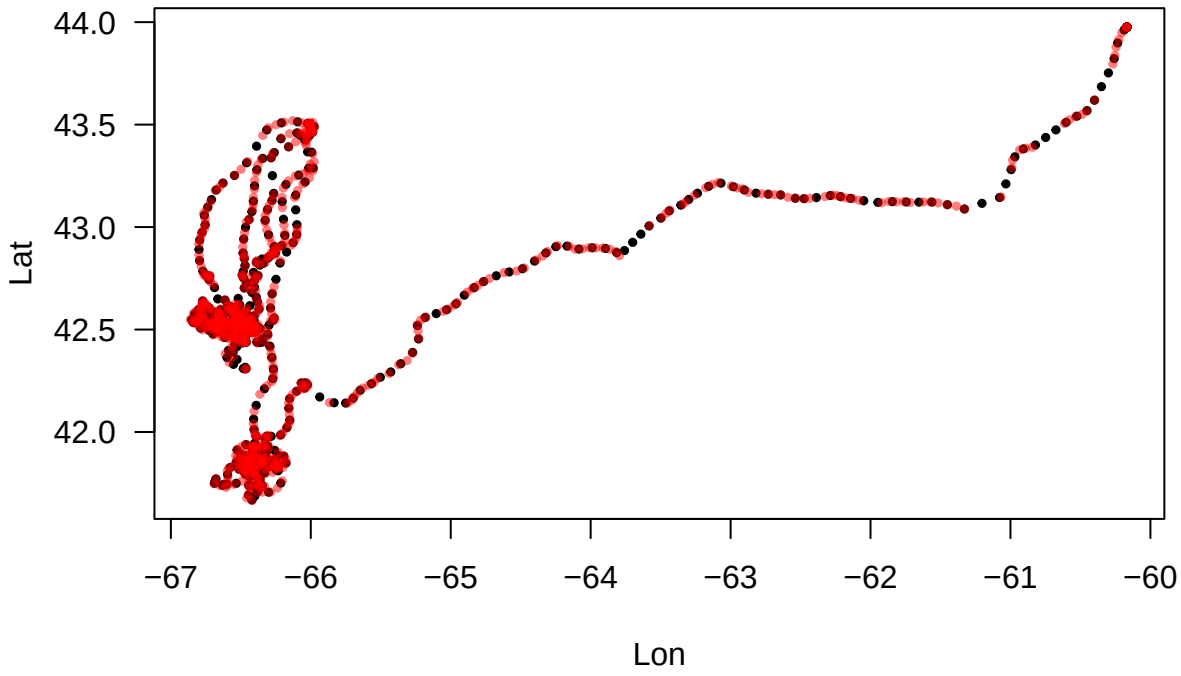
```
## [1] "2014-02-18 03:32:08 GMT" "2014-02-18 05:32:08 GMT"
## [3] "2014-02-18 07:32:08 GMT" "2014-02-18 09:32:08 GMT"
## [5] "2014-02-18 11:32:08 GMT" "2014-02-18 13:32:08 GMT"
```

```
# Interpolate the location at the times from the sequence
iLoc <- as.data.frame(cbind(lon = approx(seal$date, seal$lon,
xout = ti)$y, lat = approx(seal$date, seal$lat, xout = ti)$y))
# Create a new object that has the regular time and
# interpolated locations
sealreg <- cbind(date = ti, iLoc)
head(sealreg)
```

```
##                  date       lon      lat
## 1 2014-02-18 03:32:08 -60.16754 43.97577
## 2 2014-02-18 05:32:08 -60.16759 43.97577
## 3 2014-02-18 07:32:08 -60.16764 43.97578
## 4 2014-02-18 09:32:08 -60.16770 43.97578
```

```
## 5 2014-02-18 11:32:08 -60.16775 43.97578
## 6 2014-02-18 13:32:08 -60.16780 43.97578
```

```
# Quickly plot the regularized locations
plot(sealreg$lon, sealreg$lat, pch = 19, cex = 0.5, xlab = "Lon",
ylab = "Lat", las = 1)
# Add the original data
points(seal$lon, seal$lat, pch = 19, cex = 0.5, col = rgb(1,
0, 0, 0.5))
```



Ok, we have now a regular time series of locations. For the most basic movement HMMs, you need to transform the time series of locations into two time-series: one which represents the step lengths (distance between two locations) and one that represents the turning angle (angle between two steps). The package `momentuHMM` has a function that does that: `prepData`. The data is in latitude and longitude and I'm assuming using WGS84. You can use `prepData` on latitude and longitude data 1 or on projected data (e.g., UTM). `prepData` can do much more, and we will see some other features later, but for now the arguments that we need to think about are: - 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`.

```
# Preparing the data: here mainly calculating step
# lengths and turning angles
sealPrep <- prepData(sealreg, type = "LL", coordNames = c("lon",
"lat"))
# Let's peak at the data
head(sealPrep)
```

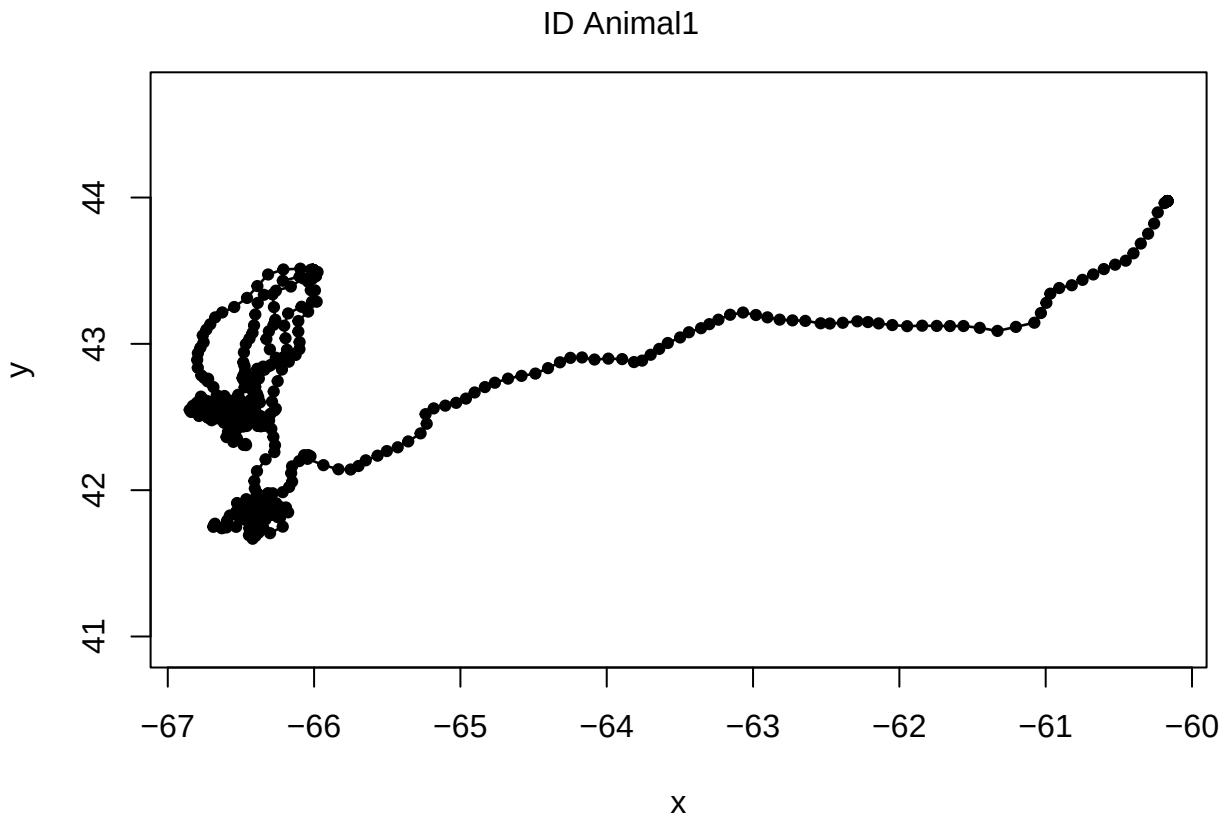
```
##        ID        step         angle                date         x        y
## 1 Animal1 0.004147236            NA 2014-02-18 03:32:08 -60.16754 43.97577
## 2 Animal1 0.004147236 -4.366461e-09 2014-02-18 05:32:08 -60.16759 43.97577
## 3 Animal1 0.004147236 -5.051761e-09 2014-02-18 07:32:08 -60.16764 43.97578
## 4 Animal1 0.004147235 -3.887481e-09 2014-02-18 09:32:08 -60.16770 43.97578
## 5 Animal1 0.004147235 -5.077021e-09 2014-02-18 11:32:08 -60.16775 43.97578
## 6 Animal1 0.004147235 -4.346000e-09 2014-02-18 13:32:08 -60.16780 43.97578
```

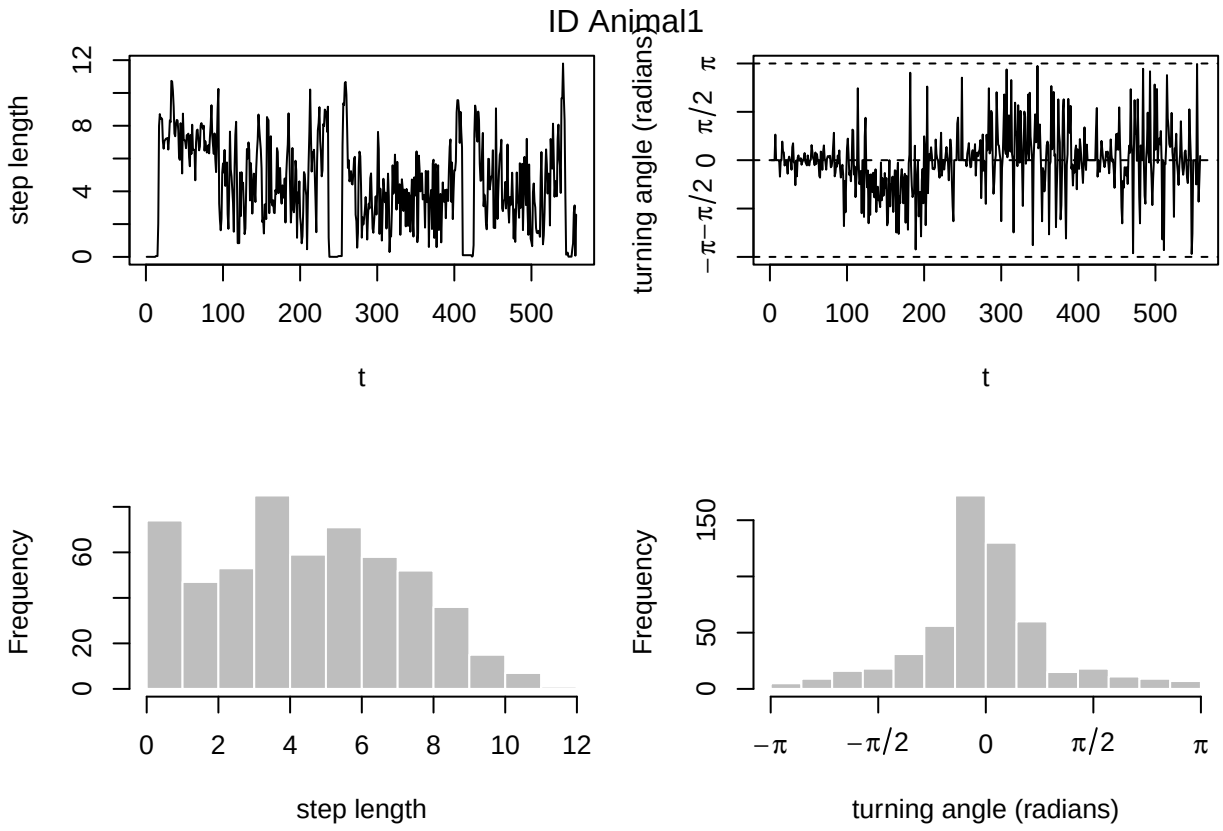
```
# What kind of object is this?
class(sealPrep)
```

```
## [1] "momentuHMMData" "data.frame"
```

If we take a quick look at the data, we can see that the step lengths (column step) and angles (column angle) have been calculated and that the names of the columns with the coordinates are now x and y, rather than `lon` and `lat`. We also have a new column: `ID`. In cases where you have multiple individuals, you would want your original data to have an `ID` column before being imputed in the `prepData` column. Note that the object it returns is from a specific S3 class defined in the `momentuHMM` package. This means that we can apply generic functions like `plot` to it and it will return specified outputs.

```
plot(sealPrep)
```





We can see the track of the animal and the times series of the step lengths and turning angles, as well as the histograms of the step lengths and turning angles. Can you already identify patterns?

Fitting the HMM

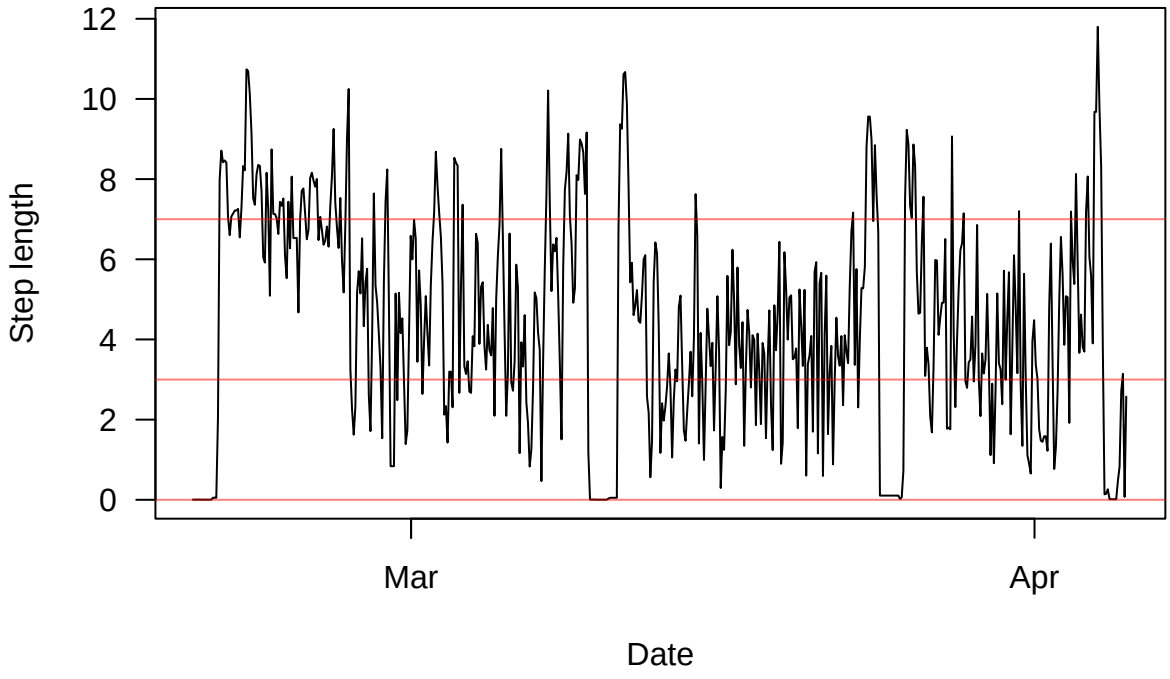
Now that our data is ready to use, we can fit the HMM. For this we use the function `fitHMM`. This is where we need to make many of our modelling decisions and most of these decisions will be associated with one of the arguments of the function. First, when we fit a HMM, we need to decide the number of behavioural states we are going to model. To start simple, we will only use two behavioural states. These could be, for example, representing one behaviour with fast and directed movement (e.g., travelling) and one behaviour with a more slow and tortuous movement (e.g., foraging). This mean that the argument `nbState` will be set to 2. Second, we need to decide whether we make the transition probabilities between the behavioural states dependent on covariates. Here again, we will start simple and we will not use covariates. That means that our formula argument will be `~1`. Third, we need to select the distribution we will use to model the step lengths and the one we will use to model the turning angles. For now, we will use the gamma distribution for the step length and the von Mises for the turning angles. This means that the argument dist will be set to: `list(step="gamma", angle="vm")`. Note that `dist` should be a list with an item for each data stream columns in the data that we are modelling (so here the column `step` and `angle`). The gamma distribution is strictly positive (i.e., it does not allow for 0s). If you have step lengths that are exactly zero in your data set, you need to use zero-inflated distributions. But in this case, we have no zeros. Let's check

```
sum(sealPrep$step == 0, na.rm = TRUE)
```

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

We are ok. We can ignore the zero-inflation. By default, `fitHMM` will set the argument `estAngleMean` to `NULL`, which means that we assume that the mean angle is 0 for both behaviours (i.e., the animal has a tendency to continue in the same direction) and that only the angle concentration parameters differ. The concentration parameters control the extent to which the animal continues forward versus turn. Doing so reduces the number of parameters to estimate. These are all very important decisions that you must make when you construct your model. In addition, we need to specify initial values for the parameters to estimate. The HMMs are fitted using maximum likelihood estimate (MLE) with a numerical optimizer. An unfortunate aspect of fitting models using numerical optimizers, is that, to be able to explore the parameter space and find the best parameter values for the model, the optimizer needs to start somewhere. You need to decide where it starts. Unfortunately, choosing bad starting values can result in parameter estimates that are not the MLE, but just a local maximum. To choose your initial parameter you can take a quick peak at the data (e.g., using the plots above) and use some general biological information. For example, it's common for animals to have one behaviour with long step lengths and small turning angles and one behaviour with short step lengths and larger turning angles. From the plots above, it looks like the animal has step lengths that are close to 0, 3, 7. As I see it, there are probably three behaviours with different mean step lengths around these values.

```
plot(sealPrep$step ~
sealPrep$date, ty = "l", ylab = "Step length",
xlab = "Date", las = 1)
abline(h = 0, col = rgb(1, 0, 0, 0.5))
abline(h = 3, col = rgb(1, 0, 0, 0.5))
abline(h = 7, col = rgb(1, 0, 0, 0.5))
```



So let's choose 3 and 7 for the means step lengths and use the same values for the standard deviations. The turning angles are either very close to 0 or pretty spread from $-\pi$ to π. High concentration parameter values

(κ, said kappa) 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). Note that κ cannot be exactly 0, so let's choose 0.1 and 1.

```
# Setting up the starting values
mu0 <- c(3, 7) # Mean step length
sigma0 <- c(3, 7) # Sd of the step length
kappa0 <- c(0.1, 1) # Turning angle concentration parameter (kappa > 0)
```

Ok, were are ready. Let's fit the HMM2

```
# Fit a 2 state HMM
sealHMM2s <- fitHMM(sealPrep, nbState = 2, dist = list(step = "gamma",
angle = "vm"), Par0 = list(step = c(mu0, sigma0), angle = kappa0), formula = ~ 1)
```

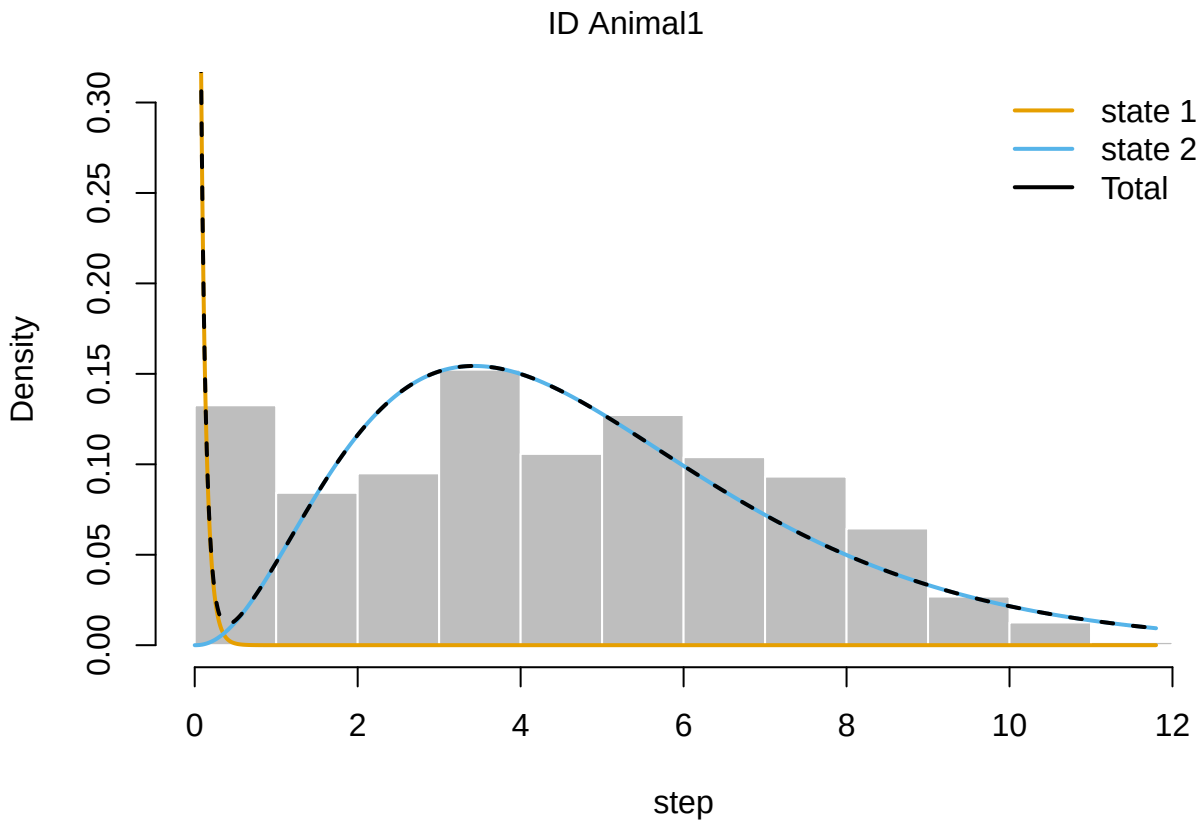
Let's explore the results.

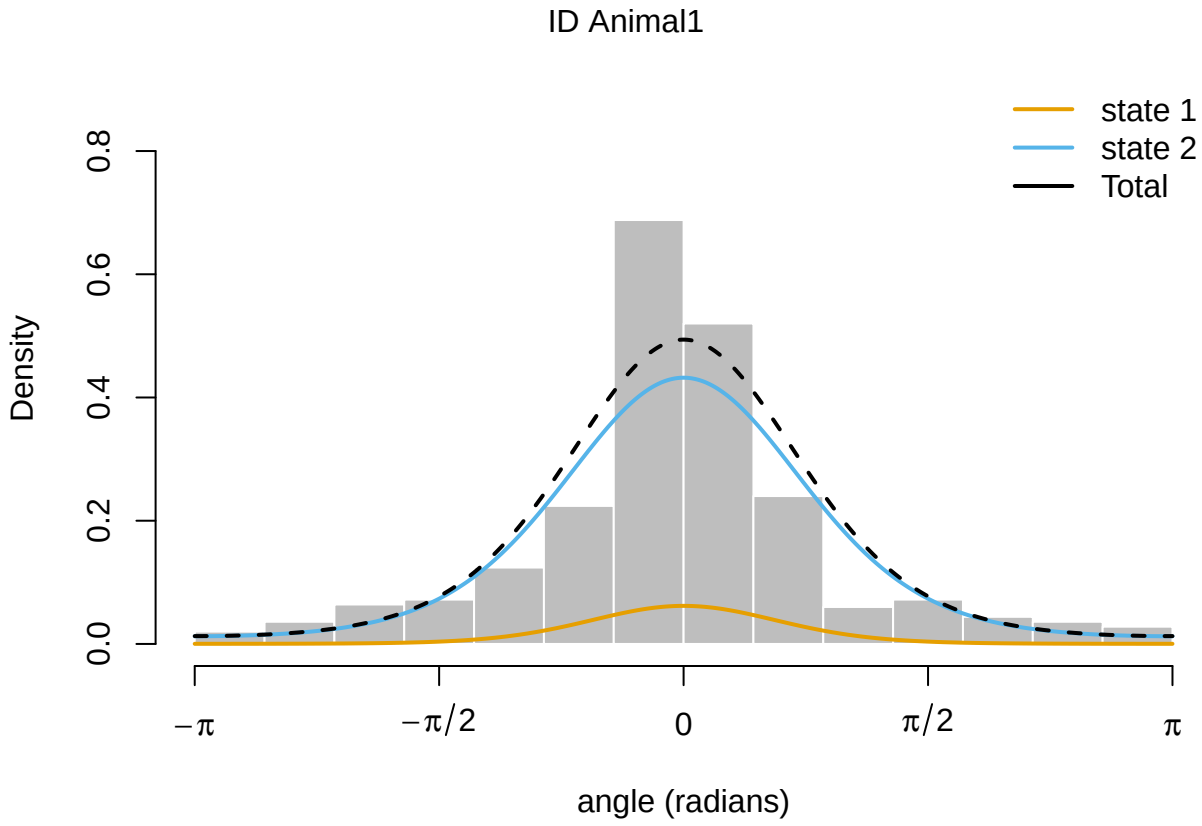
```
# Let's look at parameter estimates
sealHMM2s
```

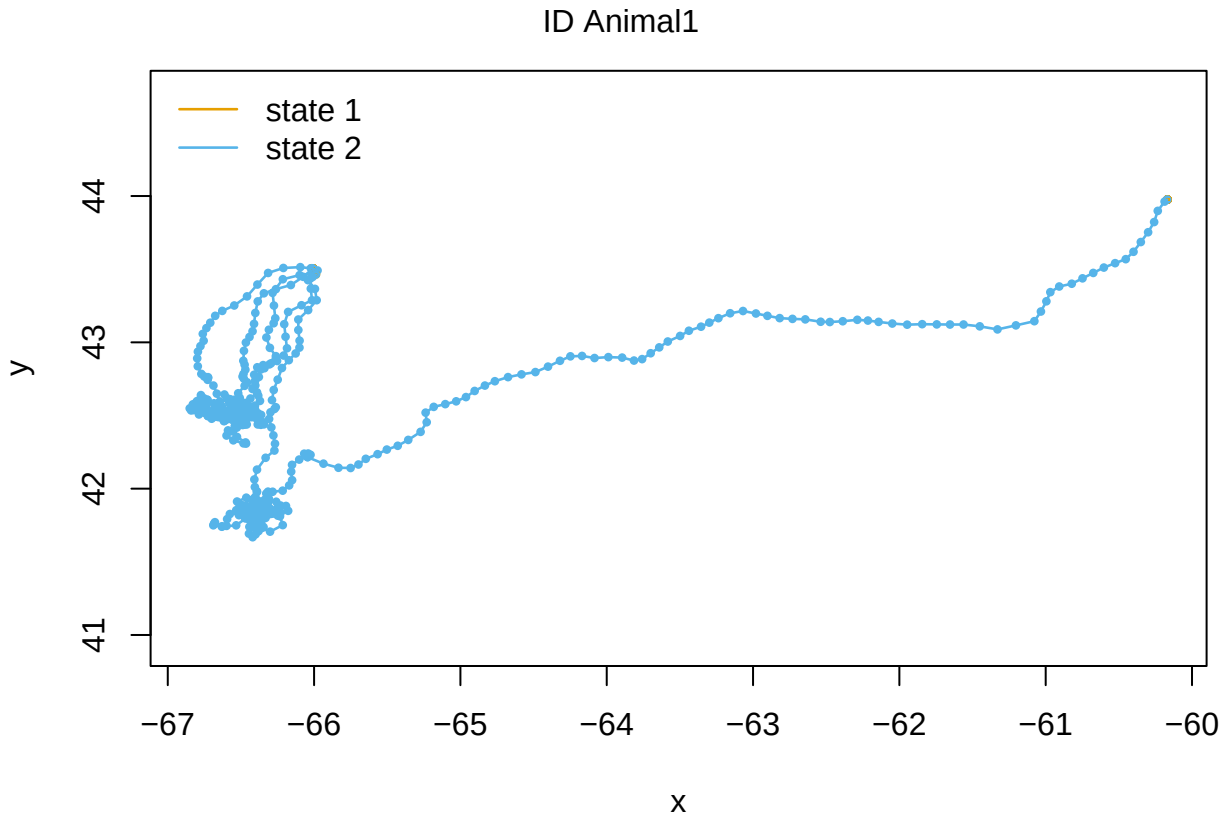
```
## Value of the maximum log-likelihood: -1810.643
##
##
## step parameters:
## ----------------
##         state 1  state 2
## mean 0.05014676 4.902540
## sd   0.06428478 2.689235
##
## angle parameters:
## -----------------
##                state 1  state 2
## mean          0.000000 0.000000
## concentration 2.789103 1.771595
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                1 -> 2    2 -> 1
## (Intercept) -2.346323 -4.840844
##
## Transition probability matrix:
## ------------------------------
##             state 1    state 2
## state 1 0.912641513 0.08735849
## state 2 0.007838455 0.99216155
##
## Initial distribution:
## ---------------------
##      state 1      state 2
## 9.999995e-01 5.007276e-07
```

```
# Let's plot it
plot(sealHMM2s)
```

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




ID Animal1
state 1
state 2
Total
Density
0.0
0.2
0.4
0.6
0.8
−π
−π/2
0
π/2
π
angle (radians)



Based on the mean step parameters, it looks like the first behavioural state (state 1) has really small step lengths compare to state 2. This is particularly easy to see in the step histogram, where the estimated distribution for each state is overlaid on top of the observed step length frequencies. Surprisingly, the turning angle distributions of the two states both indicate directed movement. In fact, the concentration parameter of state 1 is bigger than that of state 2. In the figure with the track, it looks like if we only have state 2. This is strange, since the transition probabilities estimated indicate that the animal remain in each behaviour for multiple steps (diagonal values of the transition probability matrix are > 0.5). What's happening?

Identifying behavioural states

Maybe looking at the state sequence in more detail will help us understand. Actually, we are interested in identifying when an animal is in each of the behavioural states (here when the animal is in state 1 vs state 2), something we call state decoding. To do so, we can use the function `viterbi`, which uses the Viterbi algorithm to produce the most likely sequence of states according to your fitted model and data.

```
# Apply the Viterbi algorithm using your fited model object
sealStates <- viterbi(sealHMM2s)
# Let's look at predicted states of the first 20 time steps
head(sealStates, 20)
```

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

```r
# How many locations in each state do we have?
table(sealStates)
```

```
## sealStates
##   1   2
##  55 504
```

So they are some state 1, we are just not seeing them! In many cases, it is more interesting to get the probability of being in each state rather than the most likely state. To do so, you can use the function `stateProbs`, which returns a matrix of the probability of being in each state for each time step.

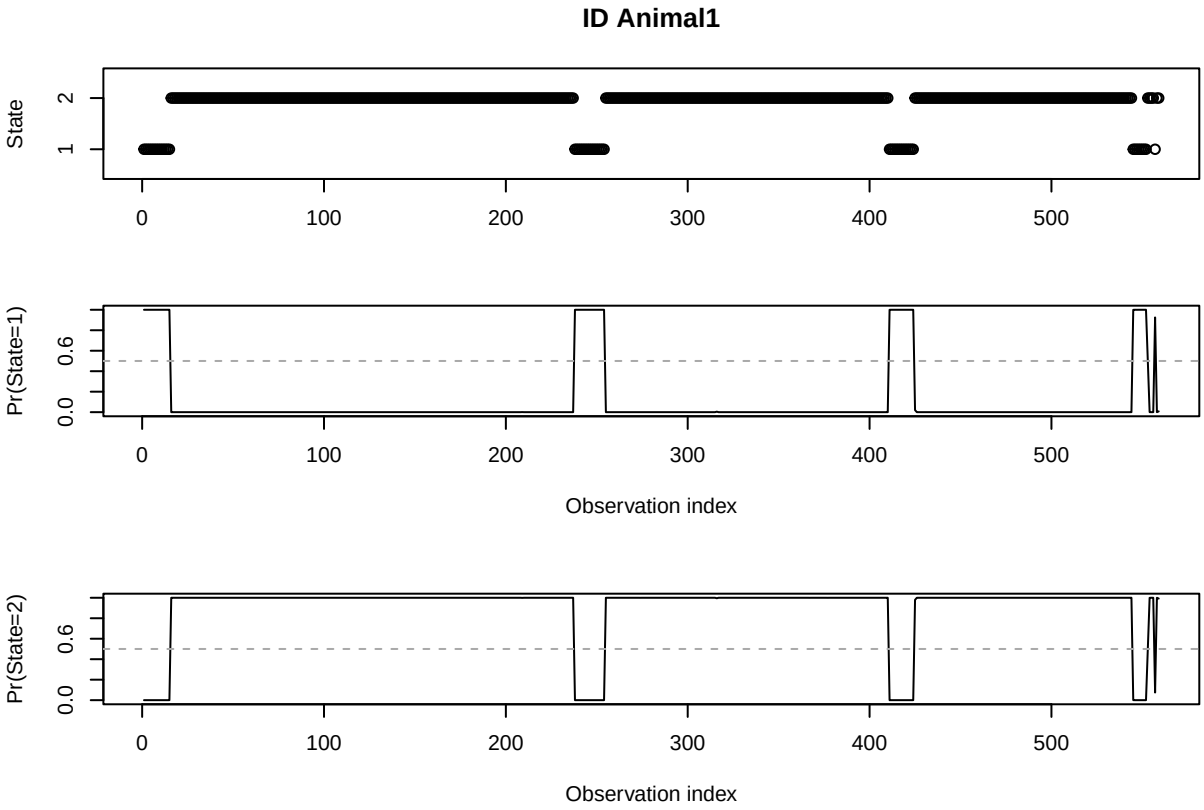
```r
# Calculate the probability of being in each state
sealProbs <- stateProbs(sealHMM2s)
# Let's look at the state probability matrix
head(sealProbs)
```

```
##      state 1      state 2
## [1,]       1 9.447526e-12
## [2,]       1 7.208566e-12
## [3,]       1 7.208565e-12
## [4,]       1 7.208564e-12
## [5,]       1 7.208563e-12
## [6,]       1 7.208561e-12
```

We can see here the probability of being in both states for the first 6 time steps. Here, the probability of being in state 1 is really high for these steps, but that may not always be the case. Sometimes you might have values closer to 0.5, which would indicate that for that time step, it's hard to identify which state you are in (i.e., which step length and turning angle distributions fit best). You can plot the results from both of these functions using the function `plotState`

```r
plotStates(sealHMM2s)
```

```
## Decoding state sequence... DONE
## Computing state probabilities... DONE
```



We clearly have two behavioural states. The steps from `state` 1 are just too small for us to see them in the track.

But do we have a good model?

This is all great, but is this a good model for our data?

Starting values: should we care?

The first thing we need to look into is whether the parameter estimates are reliable. As I mentioned above, the initial parameter values can affect the estimating procedures. So it's always good to check if you get similar results with different starting values. Here, we will make the starting values of the two behavioural states closer to one another.

```
# Setting up the starting values (more similar values)
mu02 <- c(5, 7) # Mean step length
sigma02 <- c(5, 7) # Sd of the step length
kappa02 <- c(1, 1) # Turning angle concentration parameter (kappa > 0)


# Fit the same 2 state HMM
sealHMM2s2 <- fitHMM(sealPrep,
                     nbState = 2,
                     dist=list(step="gamma", angle="vm"),
                     Par0 = list(step=c(mu02, sigma02), angle=kappa02))
```

Let's compare the two results. First let's look at which of the two has the lowest negative log likelihood (equivalent of highest log likelihood, so closer to the real MLE). Let's look also at the parameter estimates they each returned.

```
# Negative log likelihood
c(original = sealHMM2s$mod$minimum, new = sealHMM2s2$mod$minimum)
```

```
## original      new
## 1810.643 1810.643
```

```
# Parameter estimates
cbind(sealHMM2s$mle$step, sealHMM2s2$mle$step)
```

```
##         state 1  state 2    state 1  state 2
## mean 0.05014676 4.902540 0.05014667 4.902536
## sd   0.06428478 2.689235 0.06428472 2.689231
```

```
cbind(sealHMM2s$mle$angle, sealHMM2s2$mle$angle)
```

```
##                 state 1  state 2  state 1  state 2
## mean          0.000000 0.000000 0.000000 0.000000
## concentration 2.789103 1.771595 2.789099 1.771601
```

```
cbind(sealHMM2s$mle$gamma, sealHMM2s2$mle$gamma)
```

```
##             state 1    state 2     state 1   state 2
## state 1 0.912641513 0.08735849 0.912643603 0.0873564
## state 2 0.007838455 0.99216155 0.007838449 0.9921616
```

Looks like they both returned very close values for everything. So that's good! The function `fitHMM` also has the argument `retryFits` which perturbs the parameter estimates and retry fitting the model. The argument is used to indicate the number of times you want perturb the parameters and retry fitting the model (you can choose the size of the perturbation by setting the argument `retrySD`).

Let's try (this will take a few minutes).

```
# Fit the same 2-state HMM with retryFits
# This is a random pertubation, so setting the seed to get the same results
set.seed(1234)
sealHMM2sRF <- fitHMM(sealPrep,
                      nbState = 2,
                      dist=list(step="gamma", angle="vm"),
                      Par0 = list(step=c(mu02, sigma02), angle=kappa02),
                      retryFits=10)
```

```
## Attempting to improve fit using random perturbation. Press 'esc' to force exit from 'fitHMM'
##     Attempt 1 of 10 -- current log-likelihood value: -1810.643             Attempt 2 of 10 -- curren
```

Let's compare the results again.

```
# Negative log likelihood
c(original = sealHMM2s$mod$minimum, new = sealHMM2s2$mod$minimum,
  retryFits = sealHMM2sRF$mod$minimum)
```

```
##  original       new retryFits
##  1810.643  1810.643  1810.643
```

```
# Parameter estimates
cbind(sealHMM2s$mle$step, sealHMM2s2$mle$step, sealHMM2sRF$mle$step)
```

```
##         state 1  state 2    state 1  state 2    state 1  state 2
## mean 0.05014676 4.902540 0.05014667 4.902536 0.05014667 4.902544
## sd   0.06428478 2.689235 0.06428472 2.689231 0.06428477 2.689237
```

```
cbind(sealHMM2s$mle$angle, sealHMM2s2$mle$angle, sealHMM2sRF$mle$angle)
```

```
##                state 1  state 2  state 1  state 2  state 1  state 2
## mean          0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## concentration 2.789103 1.771595 2.789099 1.771601 2.789111 1.771598
```

```
cbind(sealHMM2s$mle$gamma, sealHMM2s2$mle$gamma, sealHMM2sRF$mle$gamma)
```

```
##             state 1    state 2     state 1   state 2    state 1    state 2
## state 1 0.912641513 0.08735849 0.912643603 0.0873564 0.91264264 0.08735736
## state 2 0.007838455 0.99216155 0.007838449 0.9921616 0.00783852 0.99216148
```

Still looking very similar!

But let's choose some wild starting values to see the type of problems we might run into.

```
# Setting wild starting values
mu0W <- c(5, 100) # Mean step length
sigma0W <- c(5, 0.1) # Sd of the step length
kappa0W <- c(10, 0.01) # Turning angle concentration parameter (kappa > 0)
sealHMM2sW <- fitHMM(sealPrep, nbState = 2,
                     dist=list(step="gamma", angle="vm"),
                     Par0 = list(step=c(mu0W,sigma0W), angle=kappa0W))
```

Let's compare the negative log likelihood (note that we want the minimum value here to get the MLE) and parameter estimates.

```
# Negative log likelihood
c(original = sealHMM2s$mod$minimum, wild=sealHMM2sW$mod$minimum)
```

```
## original     wild
## 1810.643 2122.390
```

```
# Parameter estimates
cbind(sealHMM2s$mle$step, sealHMM2sW$mle$step)
```

```
##         state 1  state 2  state 1 state 2
## mean 0.05014676 4.902540 4.420324   100.0
## sd   0.06428478 2.689235 4.479956     0.1
```

```
cbind(sealHMM2s$mle$angle, sealHMM2sW$mle$angle)
```

```
##                state 1  state 2  state 1 state 2
## mean          0.000000 0.000000 0.000000    0.00
## concentration 2.789103 1.771595 1.840826    0.01
```

```
cbind(sealHMM2s$mle$gamma, sealHMM2sW$mle$gamma)
```

```
##             state 1    state 2   state 1      state 2
## state 1 0.912641513 0.08735849 1.0000000 0.000000e+00
## state 2 0.007838455 0.99216155 0.9999985 1.485142e-06
```

The negative log likelihood here is much larger than the negative log likelihood of the model fitted with the sensible starting values. Note also that some of the parameter estimates are exactly the same values as the initial values, which is always a sign of optimization problems. These are all signs that the model fitted with this new set of starting values is not good. Sometimes, you will get error messages saying that `nlm` or `optim`, the functions that minimize the negative log likelihood, did not converged or that non-finite values were supplied to them. Sometimes, you might not be able to plot the data or get the states. These are also signs that the starting values are poor or that the model is not good for your data.

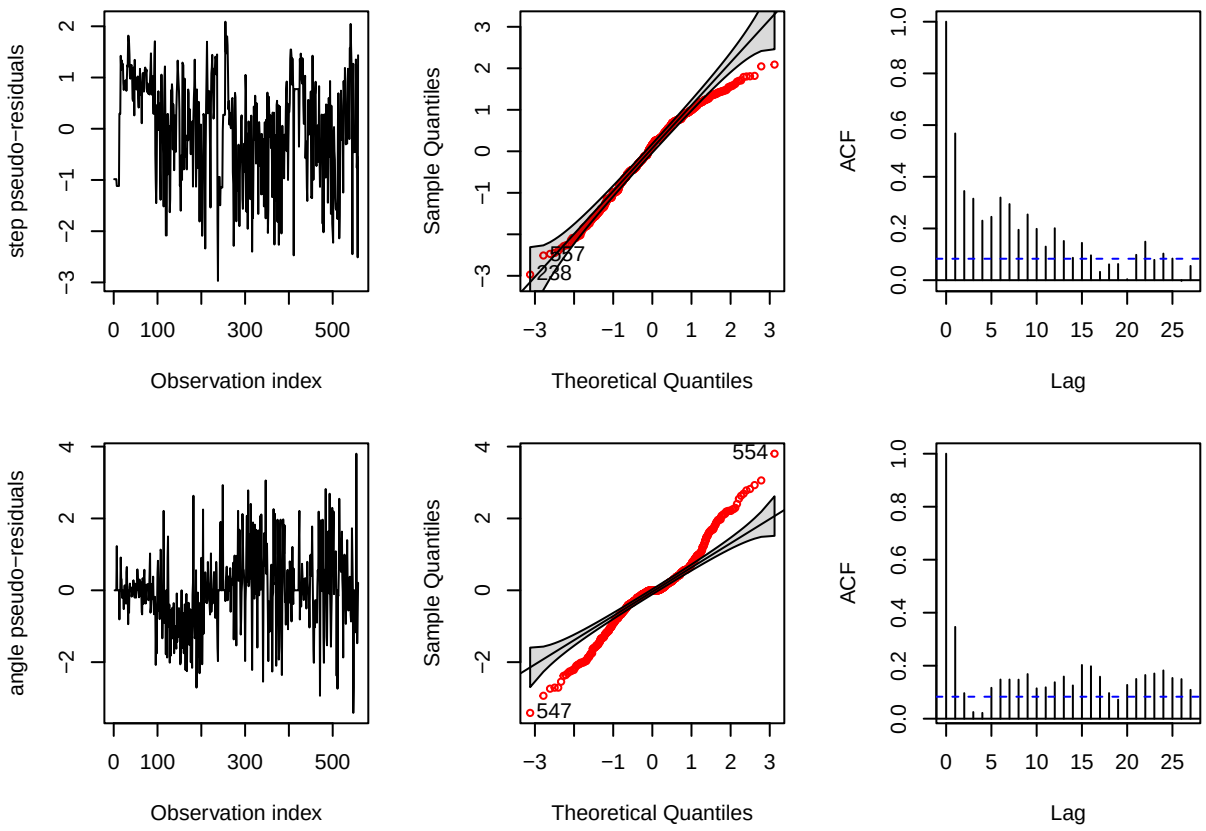
We will see later, that the package `momentuHMM` has functions to help selecting initial parameters for complex models. However, these functions rely on the user choosing initial parameter values for a simple model like the one we just explored.

Pseudo-residuals

Ok, we fitted a model to data, looks like the parameter estimates are reliable, and we can get state probabilities, but is this a good model for our data? Can we use the model results? The best way to investigate model fit is through pseudo-residuals. Pseudo-residuals are a type of model residuals that account for the interdependence of observations. They are calculated for each of the time series (e.g., you will have pseudo-residuals for the step length time series and for the turning angle time series). If the fitted model is appropriate, the pseudo-residuals produced by the functions `pseudoRes` should follow a standard normal distribution. You can look at pseudo-residuals directly via the function `plotPR`, which plots the pseudo-residual times-series, the qq-plots, and the autocorrelation functions (ACF).

```
plotPR(sealHMM2s)
```

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



Both the qq-plot and the ACF plot indicate that the model does not fit the step length time series particularly well. The ACF indicates that there is severe autocorrelation remaining in the time series, even after accounting for the persistence in the underlying behavioural states.

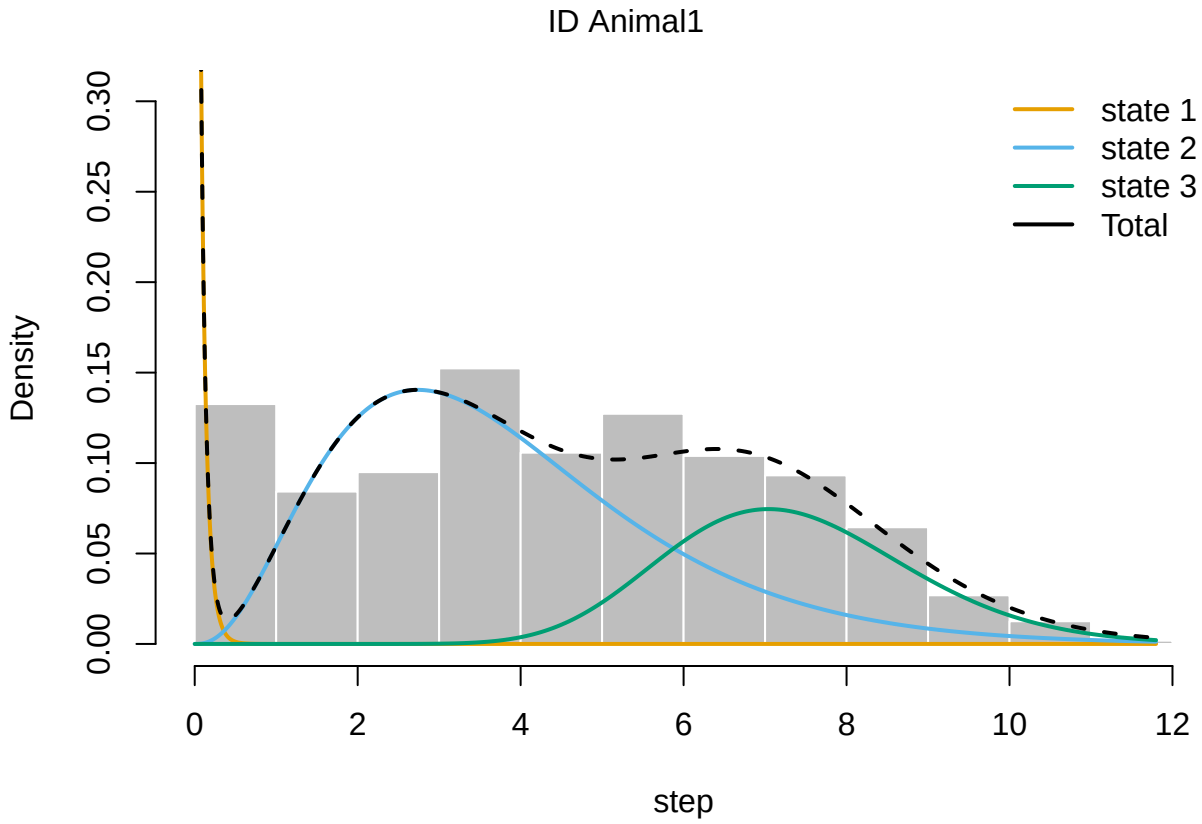
This could indicate that there is more hidden behavioural states. Let's try a 3-state HMMs.

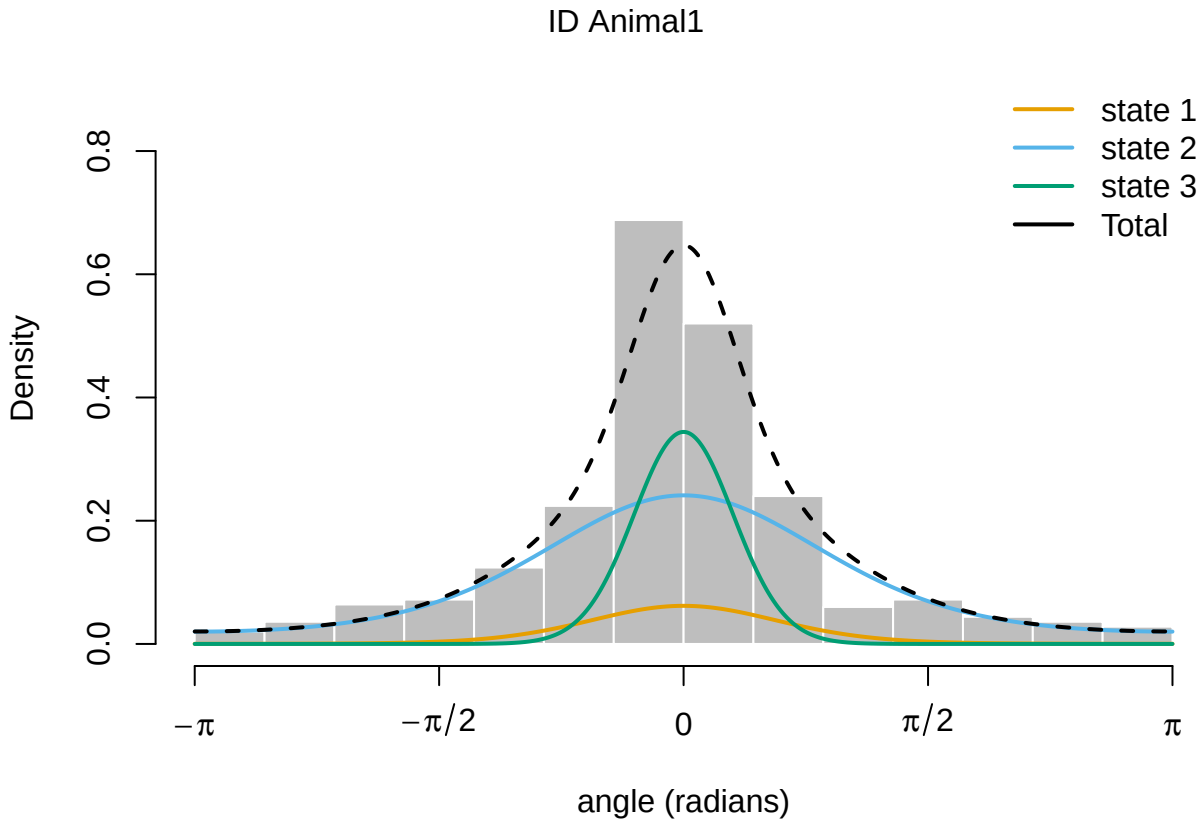
```
# Setting up the starting values (we need 3 now, because 3 states)
mu03s <- c(0.1, 3, 7) # Mean step length
sigma03s <- c(0.1, 3, 7) # Sd of the step length
kappa03s <- c(0.1, 1, 1) # Turning angle concentration parameter

# Fit a 3 state HMM
sealHMM3s <- fitHMM(sealPrep,
                    nbState = 3,
                    dist=list(step="gamma", angle="vm"),
                    Par0 = list(step=c(mu03s, sigma03s), angle=kappa03s))
```

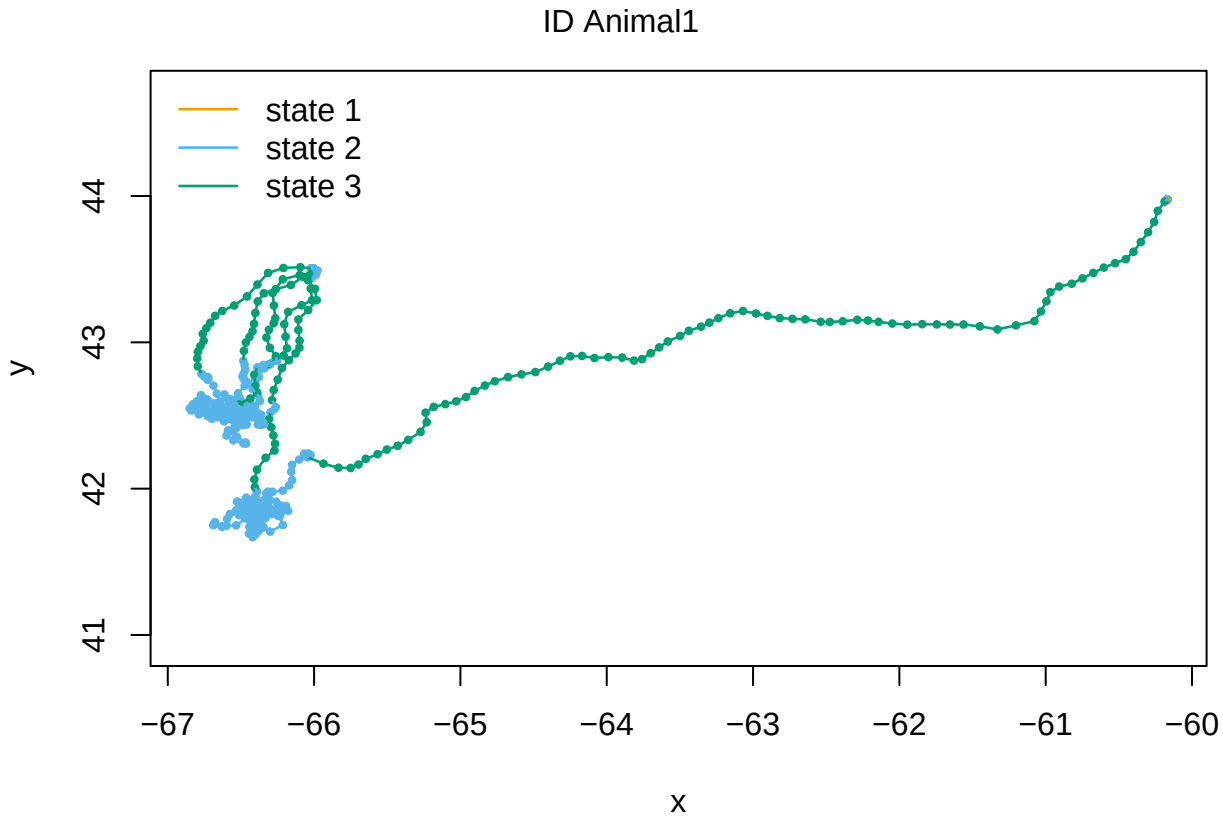
Let's look at it and at the pseudo-residuals.

```
plot(sealHMM3s)
```

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

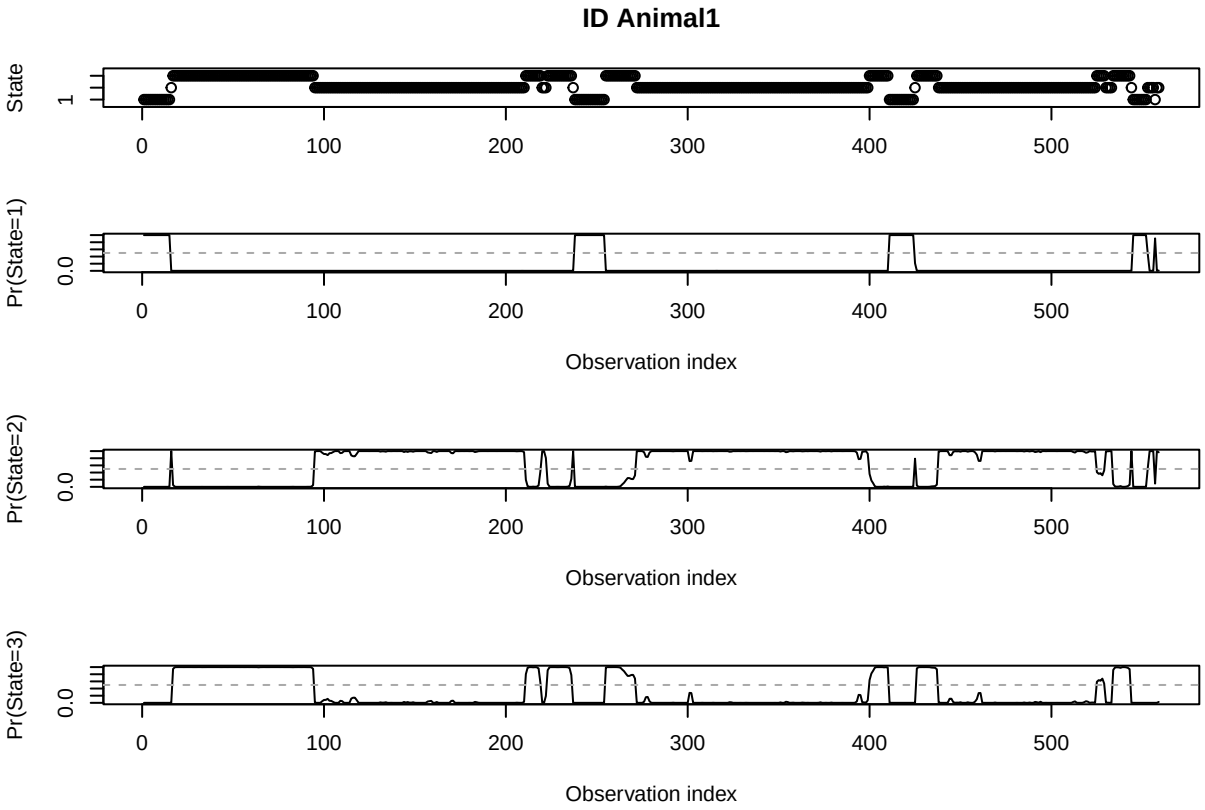

ID Animal1
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
2
4
6
8
10
12
step
state 1
state 2
state 3
Total


ID Animal1
state 1
state 2
state 3
Total
Density
0.0
0.2
0.4
0.6
0.8
−π
−π/2
0
π/2
π
angle (radians)



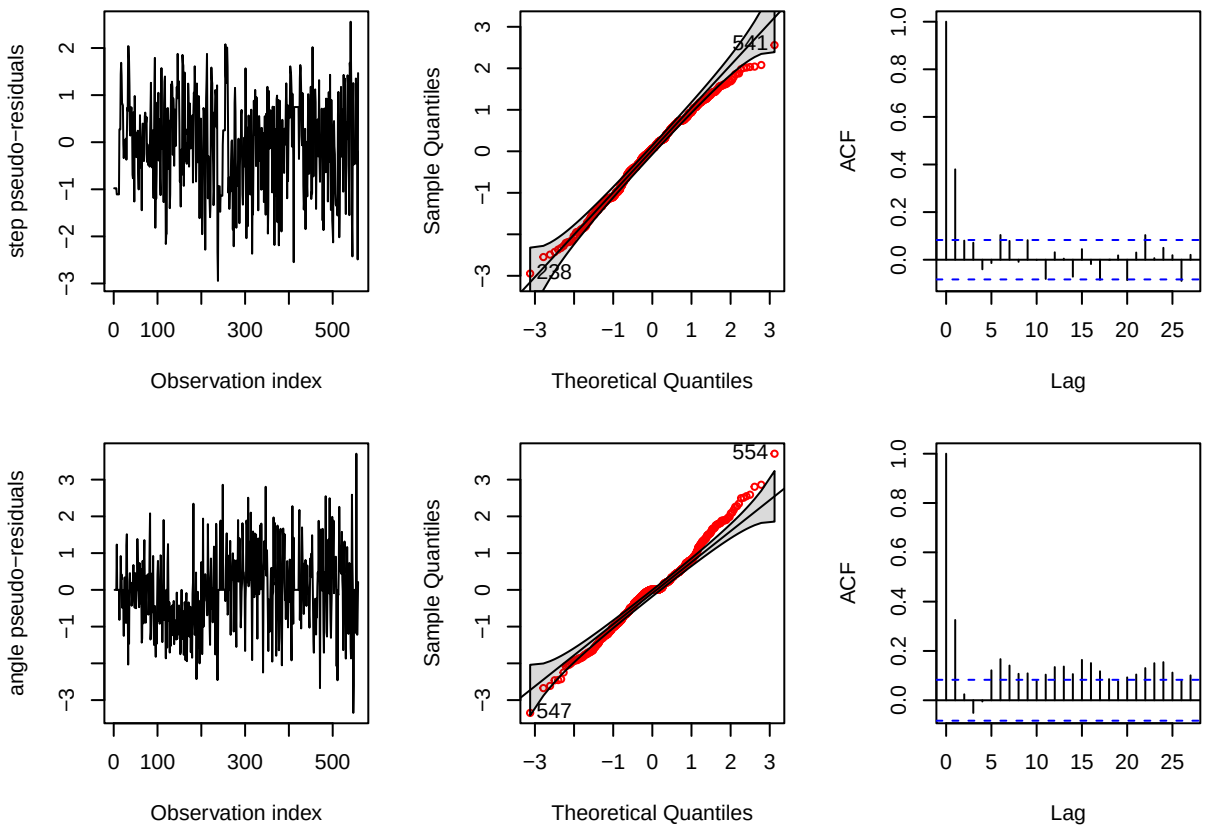

```
plotStates(sealHMM3s)
```

```
## Decoding state sequence... DONE
## Computing state probabilities... DONE
```




```
plotPR(sealHMM3s)
```

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



Ah, that's better. Not perfect, but less unexplained autocorrelation, especially in the step lengths. Some of the unexplained autocorrelation in the turning angles is related to the method we used to get location estimates at regular time intervals, see activities to explore this a bit more in depth. If we look at the step length distribution, we can see that we have still our state with really small steps (maybe resting?), a state with steps of medium size (maybe foraging?), and a state with large steps (maybe travelling?). Looking at the track, it does look like the movement in state 2 is more tortuous and in focal areas, while the movement in state 3 is very directed and span larger areas.

Simulating data

We can also use the function `simData` to look at whether the movement produce by the models is similar to the observed movement.

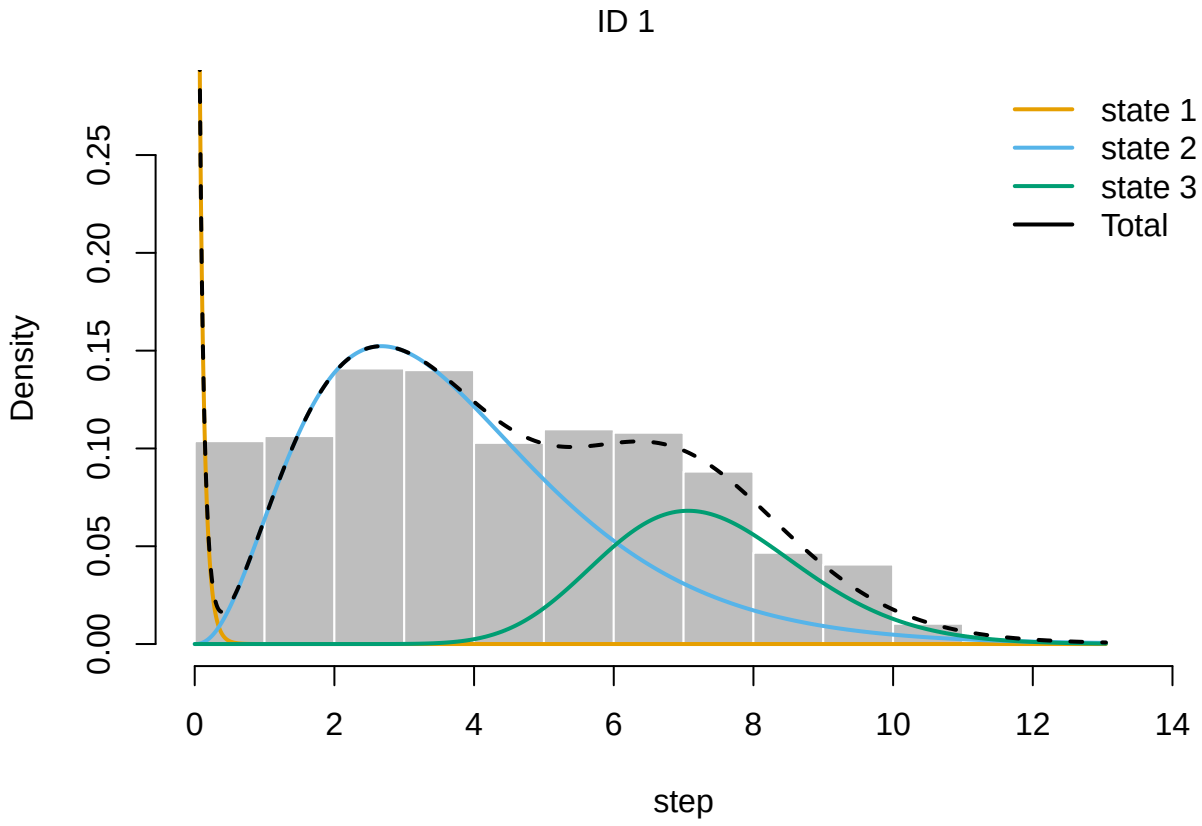
Let's simulate the 3-state HMMs, and then fit it to be able to identify the behavioural states. Note that we could just plot directly the simulated data, rather than estimating the fitting the model and predicting the states. This might require a bit of fiddling (A good thing to know is that the colour paletter used by the packages is: `c("\#E69F00", "\#56B4E9", "\#009E73", "\#F0E442", "\#0072B2", "\#D55E00", "\#CC79A7")`). One way fitting the model to the simulated data may be useful is in identifying whether you have enough data to estimate your parameters accurately.

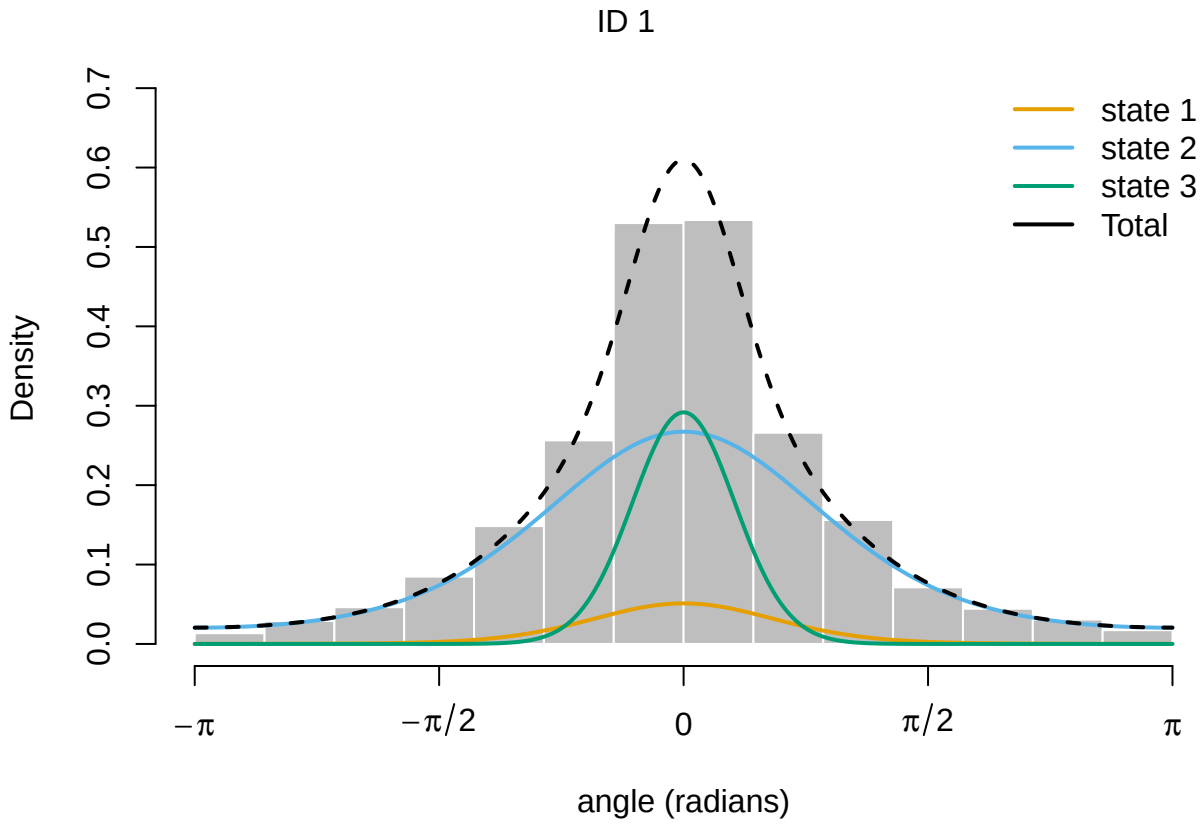
```
# Simulate the model
sealSim <- simData(model=sealHMM3s, nbAnimals =1, states = TRUE)
# Fit the model to the simulated data
sealSimFit <- fitHMM(sealSim,
                     nbState = 3,
```

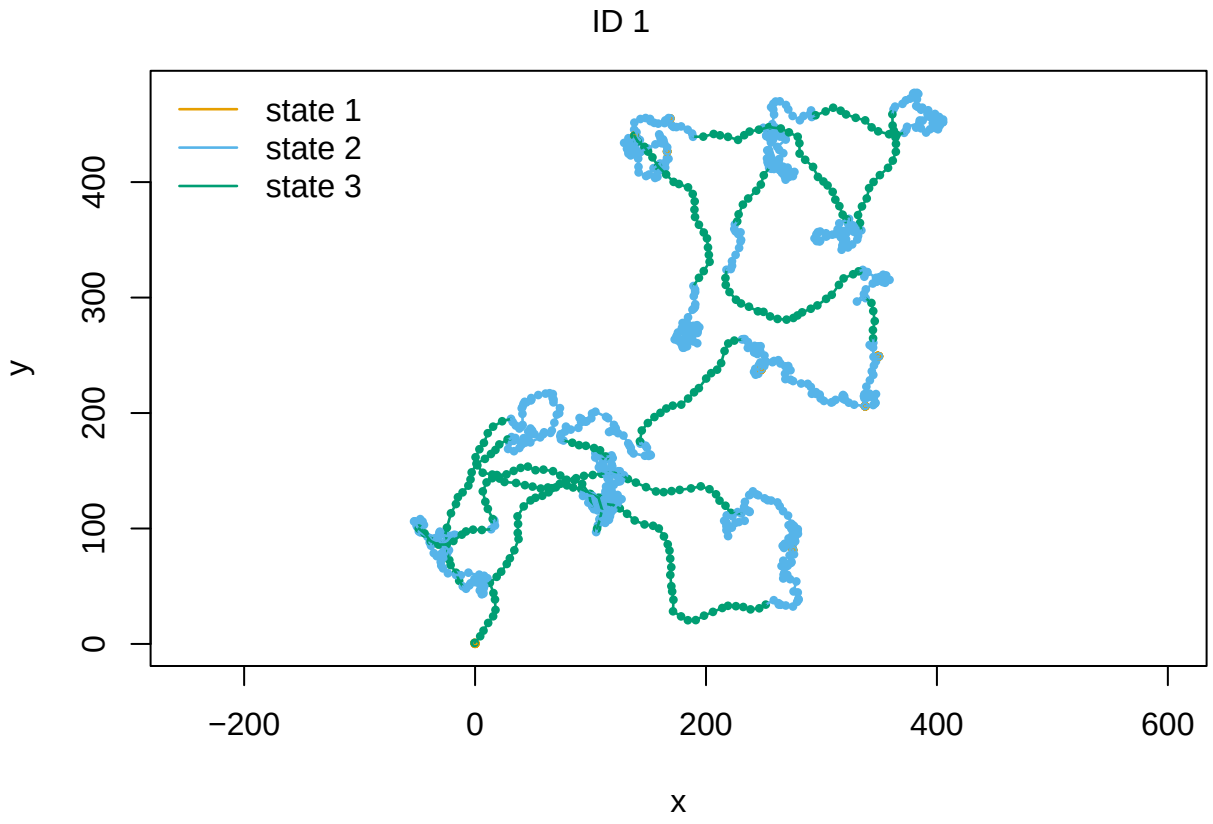
```
                    dist=list(step="gamma", angle="vm"),
                    Par0 = list(step=c(mu03s, sigma03s), angle=kappa03s))
```

```
# Plot the results
plot(sealSimFit)
```

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




ID 1
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
state 1
state 2
state 3
Total
−π
−π/2
0
π/2
π
angle (radians)




```
# Compared the parameter estimates
sealSimFit
```

```
## Value of the maximum log-likelihood: -3474.566
##
##
## step parameters:
## ----------------
##         state 1  state 2  state 3
## mean 0.05783532 3.772843 7.352621
## sd   0.07358709 2.036328 1.455070
##
## angle parameters:
## -----------------
##                state 1  state 2  state 3
## mean          0.000000 0.000000 0.000000
## concentration 2.959238 1.285337 9.212642
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1   3 -> 2
## (Intercept) -2.644709 -3.716303 -4.678915 -3.844702 -44.02857 -2.69548
##
## Transition probability matrix:
## ------------------------------
##              state 1    state 2    state 3
```

```
## state 1 9.129504e-01 0.06484323 0.02220636
## state 2 9.012562e-03 0.97023149 0.02075595
## state 3 7.083742e-20 0.06324060 0.93675940
##
## Initial distribution:
## ---------------------
##      state 1      state 2      state 3
## 9.999993e-01 3.963521e-07 3.136509e-07
```

```
sealHMM3s
```

```
## Value of the maximum log-likelihood: -1605.31
##
##
## step parameters:
## ----------------
##         state 1  state 2  state 3
## mean 0.05215805 3.809156 7.357247
## sd   0.06769966 2.018804 1.520866
##
## angle parameters:
## -----------------
##                state 1  state 2  state 3
## mean          0.000000 0.000000 0.000000
## concentration 2.793543 1.244964 9.814968
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1    3 -> 2
## (Intercept) -2.634604 -3.758437 -4.740937 -3.638665 -4.986308 -2.777883
##
## Transition probability matrix:
## ------------------------------
##             state 1    state 2    state 3
## state 1 0.913185716 0.06551865 0.02129563
## state 2 0.008435082 0.96616689 0.02539803
## state 3 0.006389930 0.05815707 0.93545300
##
## Initial distribution:
## ---------------------
##      state 1      state 2      state 3
## 9.999991e-01 4.949611e-07 4.549331e-07
```

Looks similar, except maybe that the real seal goes back and forth to some of the same locations, something not captured in the simulation. The parameter estimates are closed to those used to simulate the data, which is good.

Confidence intervals

We can look at the confidence intervals for the parameter estimates. This does not necessarily give us an idea of whether the model is good or not, but it helps us understand whether the behavioural states have different movement characteristics. To compute the confidence intervals, we can use the function `CIreal`.

For example, let's look the step length mean parameters.

```
# Caculate the CIs
sealCI <- CIreal(sealHMM3s)
# Let's look at the step lenth CIs
rbind(lower=sealCI$step$lower["mean",],
      upper = sealCI$step$upper["mean",])
```

```
##          state 1  state 2  state 3
## lower 0.01900435 3.572183 7.081672
## upper 0.08531176 4.046129 7.632822
```

We can see that the 95% confidence intervals of mean step length does not overlap for the three states, indicating that, in term of step lengths, they vary quite a bit.

Understanding the factors that affect movement: including covariates

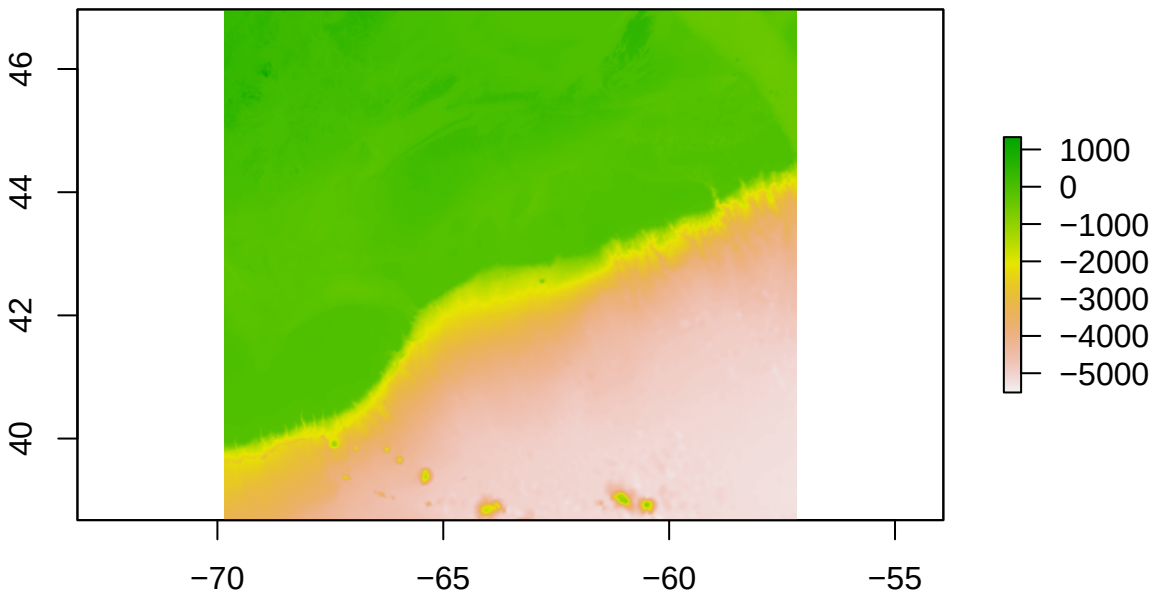
momentuHMM allows you to incorporate various covariates in the model in many different ways, which allows you to understand how covariates affect the behaviour and movement of the animal. There are so many different options (including activity centers, spatial covariates, etc), that it's impossible to look at them all here. Here are a few examples.

Spatial covariates

One of the most exciting aspect of momentuHMM is that you can incorporate spatial covariates that are kept in a raster file. Here we will use a raster that contains the bathymetry in the area of the seal. I created this file using the package marmap. To read the file you need the function raster from the package with the same name.

Let's read the file and plot it.

```
# Read the raster
bathy <- raster("data/bathy.grd")
# Let's plot the bathymethy raster
plot(bathy)
```



This represents the bathymetry in meters. Any value above 0 is on land.

Here we will use the function `prepData` again, but this time including the raster as a spatial covariate. The function will extract the raster values at the seal locations. Off course for this to work the spatial covariate raster needs to be in the same projection as the movement data. It's the case here.

```
sealPrep <- prepData(sealreg, type="LL", coordNames = c("lon", "lat"),
                     spatialCovs = list(bathy=bathy))
head(sealPrep)
```

```
##        ID        step         angle                date         x        y
## 1 Animal1 0.004147236            NA 2014-02-18 03:32:08 -60.16754 43.97577
## 2 Animal1 0.004147236 -4.366461e-09 2014-02-18 05:32:08 -60.16759 43.97577
## 3 Animal1 0.004147236 -5.051761e-09 2014-02-18 07:32:08 -60.16764 43.97578
## 4 Animal1 0.004147235 -3.887481e-09 2014-02-18 09:32:08 -60.16770 43.97578
## 5 Animal1 0.004147235 -5.077021e-09 2014-02-18 11:32:08 -60.16775 43.97578
## 6 Animal1 0.004147235 -4.346000e-09 2014-02-18 13:32:08 -60.16780 43.97578
##   bathy
## 1   -18
## 2   -18
## 3   -18
## 4   -18
## 5   -18
## 6   -18
```

Now our `sealPrep` object has a new column with the bathymetry value at that location.

There are two ways in which the bathymetry can influence the movement of the animal. It can influence the behavioural state the animal is in or it can influence the movement characteristics in some of the state (e.g. change the mean step length value in one of the behavioural states).

Let's explore the first option. So here we are assuming that the probability to switch between behavioural states is affected by the bathymetry. Because of the way the HMMs are constructed, we have to estimate one more parameter per switching probability (γ_{ij} said gamma i j), excluding the probability of remaining in the same state, which can be deducted based on the switching probabilities. Please see the mathy section below to understand the details. So if N represents the number of states, it means you need to estimate $N(N-1)$ new parameters. For an HMM with 3 states, we would estimate 6 new parameters.

The main argument of the `fitHMM` function that we need to change are the `formula` and the starting values (because we have 6 new parameters to estimate). The default value for the `formula` is ~ 1, which means that the transition probability are not affected by covariates. Here, we want to indicate that the transition probability are affected by the bathymetry. We can use the function `getPar0` to help us find starting values based on our simpler 3-state HMMs. This might be a bit of an overkill for this model, as `getPar0` will set the starting values for the covariate effects on the transition probabilities to 0. But it can make a small difference to use good starting values for the other parameters (might be worth looking if it makes indeed a difference). Note that to be able to use `getPar0` in this case, we have to refit the simpler model with the new data that has the bathymetry data.

```
# New formula
bathyForm <- ~bathy

# Refit the simple 3-states HMM - so it's associated with the data with bathymetry
sealHMM3s <- fitHMM(sealPrep,
                    nbState = 3,
                    dist=list(step="gamma", angle="vm"),
                    Par0 = list(step=c(mu03s, sigma03s), angle=kappa03s))

# Get new starting values based on simpler model, here the 3-states HMM
par0b <- getPar0(model=sealHMM3s, formula=bathyForm)
```

```
# Look at the starting values
par0b
```

```
## $Par
## $Par$step
##     mean_1     mean_2     mean_3       sd_1       sd_2       sd_3
## 0.05215805 3.80915581 7.35724680 0.06769966 2.01880445 1.52086562
##
## $Par$angle
## concentration_1 concentration_2 concentration_3
##        2.793543        1.244964        9.814968
##
##
## $beta
##                1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1    3 -> 2
## (Intercept) -2.634604 -3.758437 -4.740937 -3.638665 -4.986308 -2.777883
## bathy        0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
##
## $delta
## [1] 9.999991e-01 4.949611e-07 4.549331e-07
```

```r
# Now we fit the new model with bathymetry
sealHMMb <- fitHMM(sealPrep, nbState = 3,
                   dist=list(step="gamma", angle="vm"),
                   Par0 = par0b$Par, beta0=par0b$beta,
                   formula=bathyForm)
```

```r
sealHMMb
```

```
## Value of the maximum log-likelihood: -1586.374
##
##
## step parameters:
## ----------------
##         state 1  state 2  state 3
## mean 0.05048474 3.839012 7.381275
## sd   0.06478268 2.032187 1.535276
##
## angle parameters:
## -----------------
##                 state 1  state 2  state 3
## mean          0.000000 0.000000 0.000000
## concentration 2.795512 1.260602 9.851771
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                   1 -> 2   1 -> 3    2 -> 1     2 -> 3   3 -> 1        3 -> 2
## (Intercept) -2.20106434 -3.88486 0.5581421 -1.1189611 28.13538 -2.7257072728
## bathy        0.02381034 17.63829 0.4211997  0.0202352 28.03142  0.0009176153
##
## Transition probability matrix (based on mean covariate values):
## ---------------------------------------------------------------
##              state 1     state 2    state 3
## state 1 9.970341e-01 0.002965928 0.00000000
## state 2 2.828888e-28 0.985115923 0.01488408
## state 3 0.000000e+00 0.053906873 0.94609313
##
## Initial distribution:
## ---------------------
##      state 1      state 2      state 3
## 9.999974e-01 9.657186e-07 1.651298e-06
```

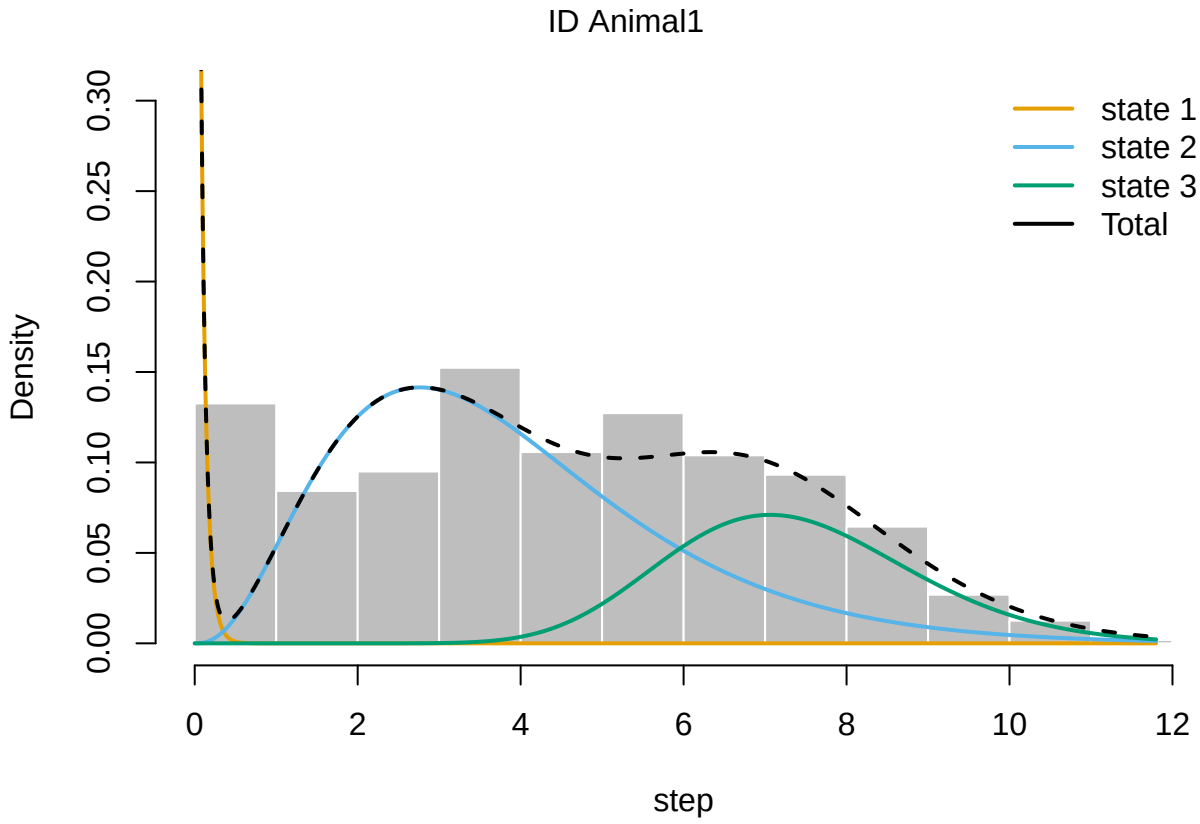
The first thing to note is that the model now has a new row of parameters in the section `Regression coeffs for the transition probabilities`. This row represents the effects of bathymetry on the transition probabilities. A positive value means that increasing values of the covariate increase the switching probability, while a negative value means the reverse relationship. So for example, as bathymetry increases, the probability of switching from state 1 to state 2 only slightly increases, but the probability of switching from state 3 to state 1 increases much more. Note that the relationship between these coefficients and the final transition probabilities are dependent on each other and on the intercept, see math section for details. Keep in mind here that bathymetry is represented in negative values, so as bathymetry value increases the seal is in shallower water.

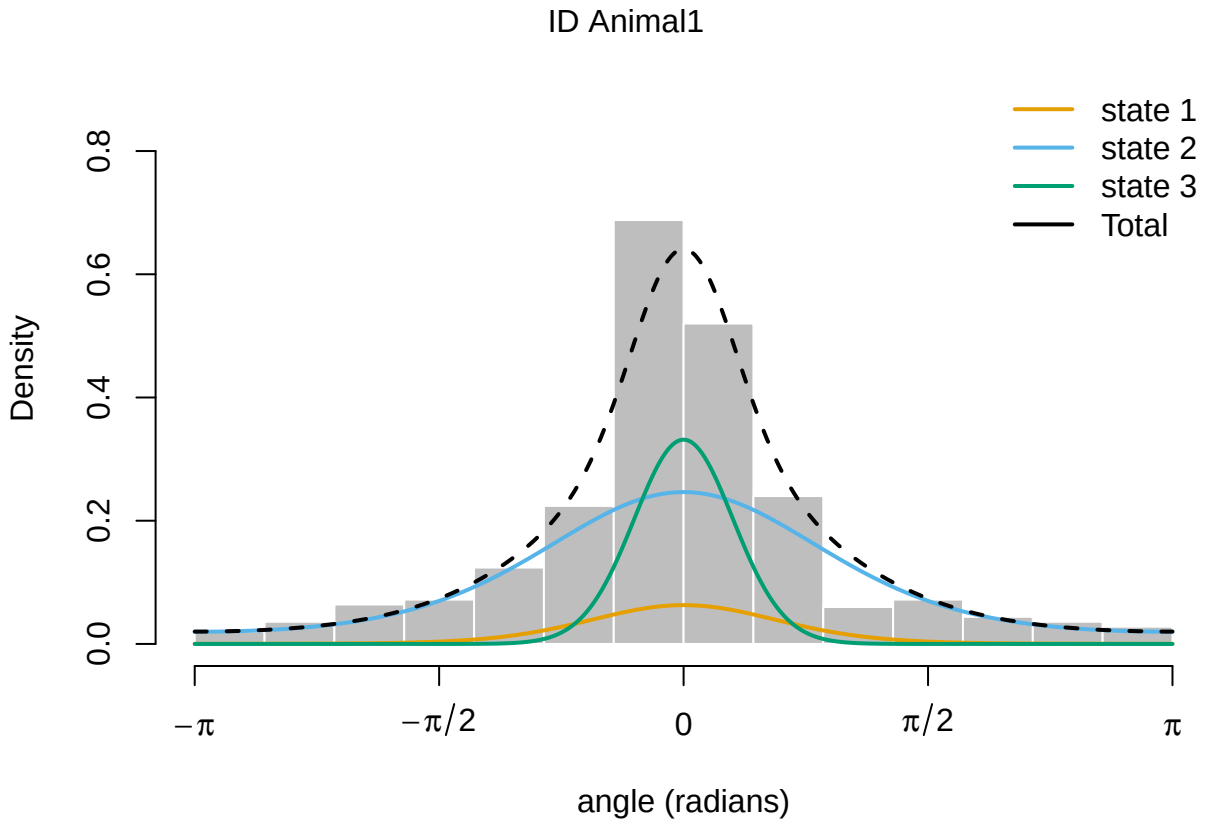
The row with the intercept represents the base probabilities before we incorporate the bathymetry effects. In model without covariates, you would only get the intercept values and would use them alone to calculate the transition probabilities, see the math section below for more details.

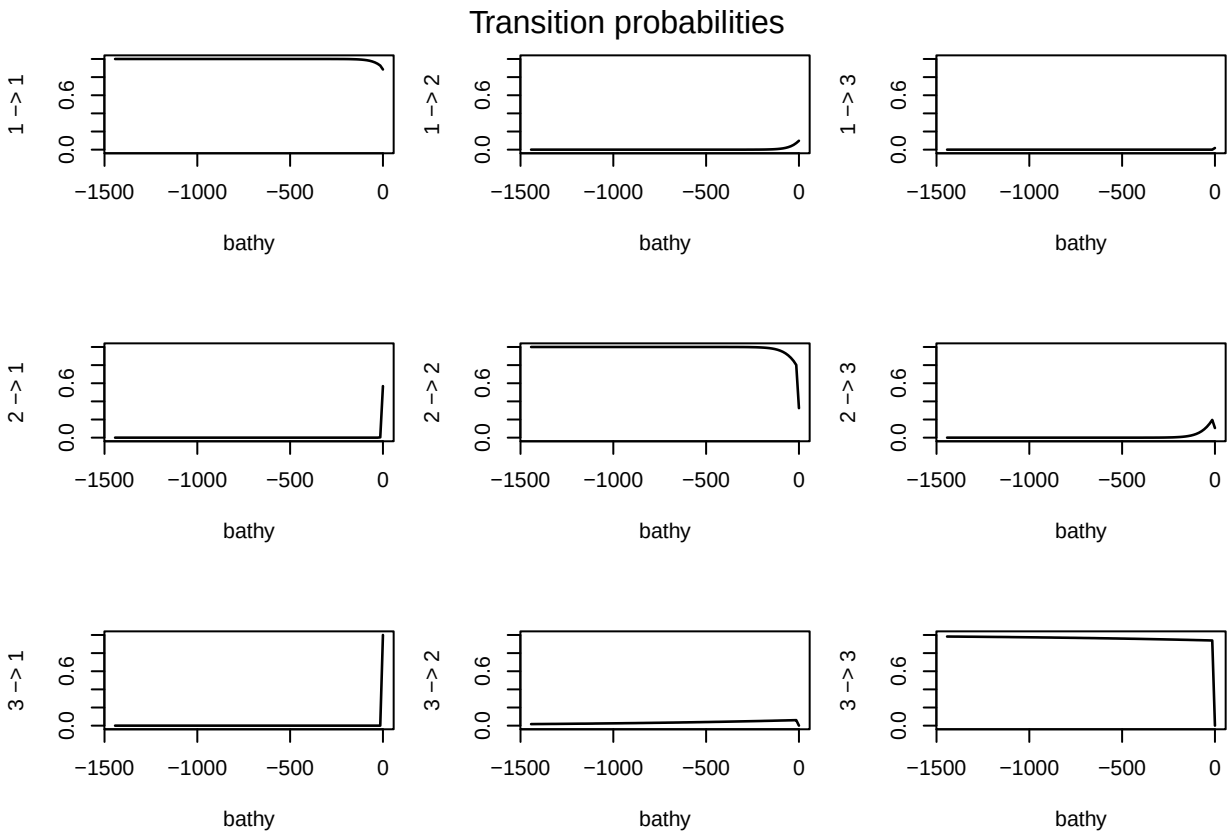
Let's look at it visually. First by using the generic `plot` function and, then, by using the function `plotStationary`, which plots the stationary state probabilities.

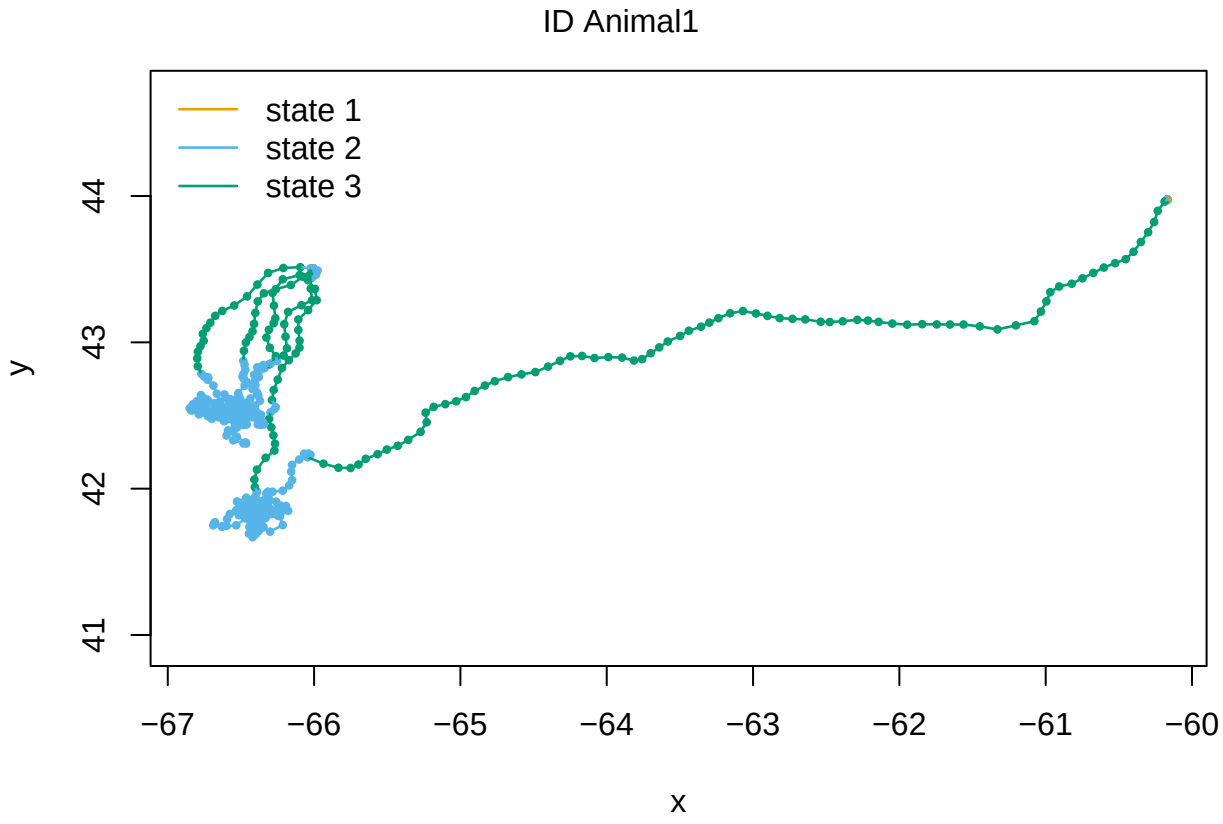
```
plot(sealHMMb)
```

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




ID Animal1
state 1
state 2
state 3
Total
Density
0.0
0.2
0.4
0.6
0.8
−π
−π/2
0
π/2
π
angle (radians)

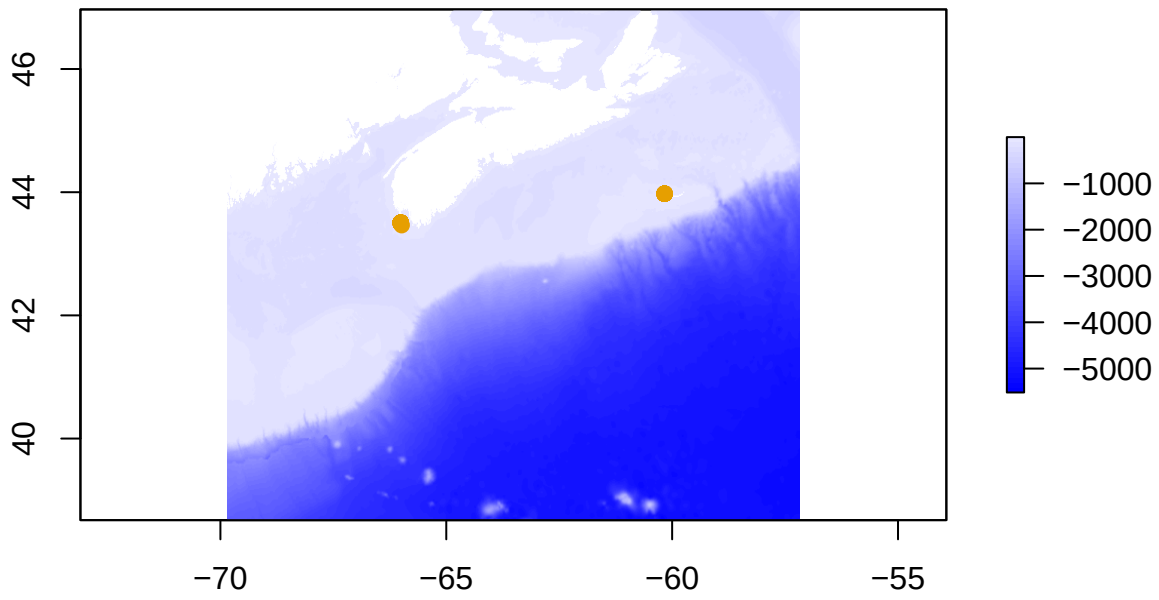

Transition probabilities
1 -> 1
1 -> 2
1 -> 3
2 -> 1
2 -> 2
2 -> 3
3 -> 1
3 -> 2
3 -> 3
0.0
0.6
−1500
−1000
−500
0
bathy



We can see that the bathymetry mainly affects the transition probability in high bathymetry values (so, close/on land). One of the most obvious patterns is that it increases the probability of switching to state 1 from both states 2 and 3 and decreases the probability of staying in state 2 and staying in state 3. Remember, state 1 is associated with the very small step lengths. This is likely seal hauling out on land or resting in shallow waters! In fact, one of the places where we have a lot of state 1 is on small islands, where I'm pretty sure they are known to haul out.

Here a quick plot to convince you.

```
# Create a new file, so we can see land better
water <- bathy
water[water >= 0] <- NA
# Create color ramp to show bathymetry in blue gradients
bluePal <- colorRampPalette(c(rgb(0,0,1), rgb(0.9,0.9,1)))
plot(water, col=bluePal(50))

# Let's plot only the state 1
points(sealPrep[viterbi(sealHMMb) == 1, c("x","y")],
       col="#E69F00",
       pch=19)
```



Ok, but it is a better model?

```
AIC(sealHMM3s, sealHMMb)
```

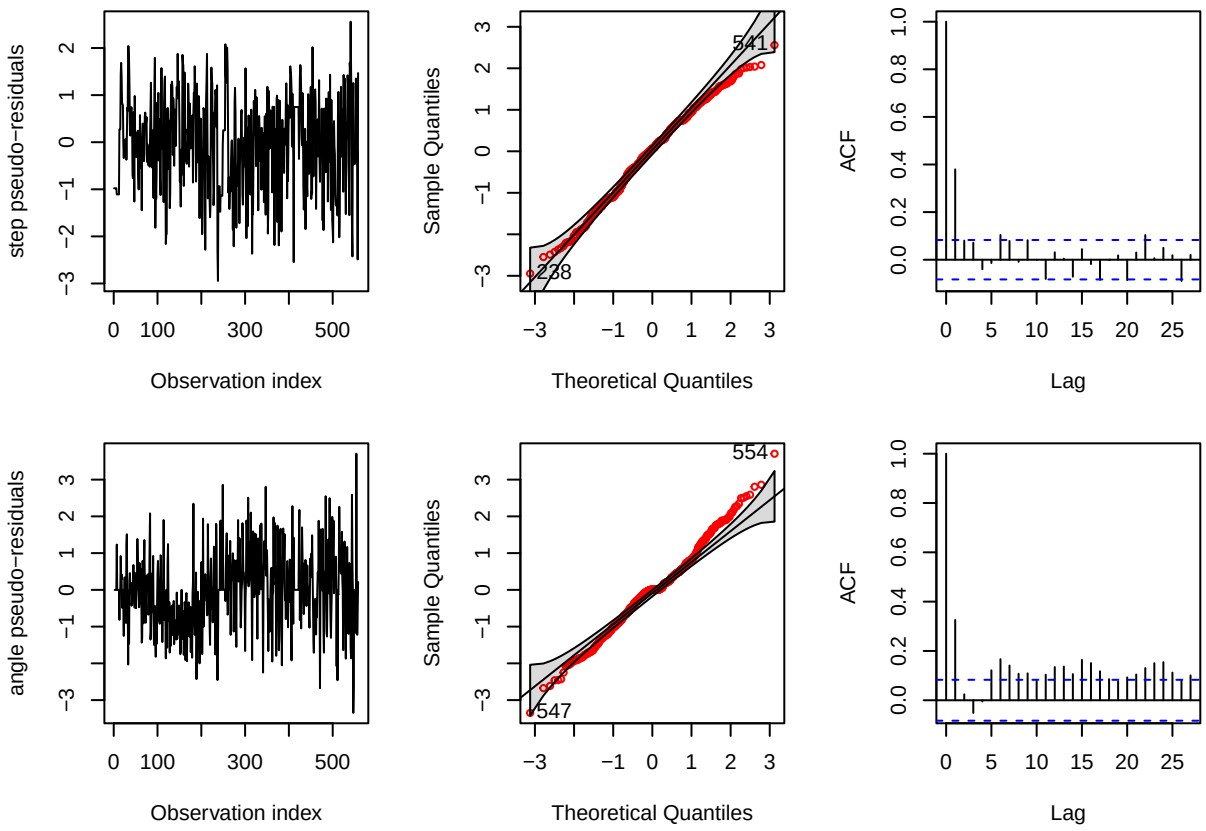
```
##       Model      AIC
## 1  sealHMMb 3218.748
## 2 sealHMM3s 3244.619
```

Yes.

What about the pseudo-residuals?

```
plotPR(sealHMM3s)
```

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



Still some issues, but not too too bad.

Non spatial covariates

Could there be diurnal patterns explaining the autocorrelation in turning angles? We are not limited to incorporate spatial covariates. We can incorporate any covariates as long as we can relate each of the locations to a covariate value (that's in essence what the `prepData` function does for the spatial covariates).

So here we are going to create a new column that keeps track of the hour of the day for each observation. We are going to set it in the time zone where the seals are.

```
sealPrep$hour <- as.integer(strftime(sealPrep$date, format = "%H", tz="Etc/GMT+4"))
```

Here again we can make the covariate affect the transition probabilities. The difference, is that we have to use a more complex function that represents the cyclical nature of time of day. This is the `cosinor` function. For more information on the `cosinor` function, see mathy section below.

```
# New formula that says that the transition probabilties are affected by the time of day
diurnForm <- ~ cosinor(hour, period = 24)

# Refit model with new data
sealHMM3s <- fitHMM(sealPrep,
                    nbState = 3,
                    dist = list(step = "gamma", angle = "vm"),
                    Par0 = list(step = c(mu03s, sigma03s), angle = kappa03s))
```

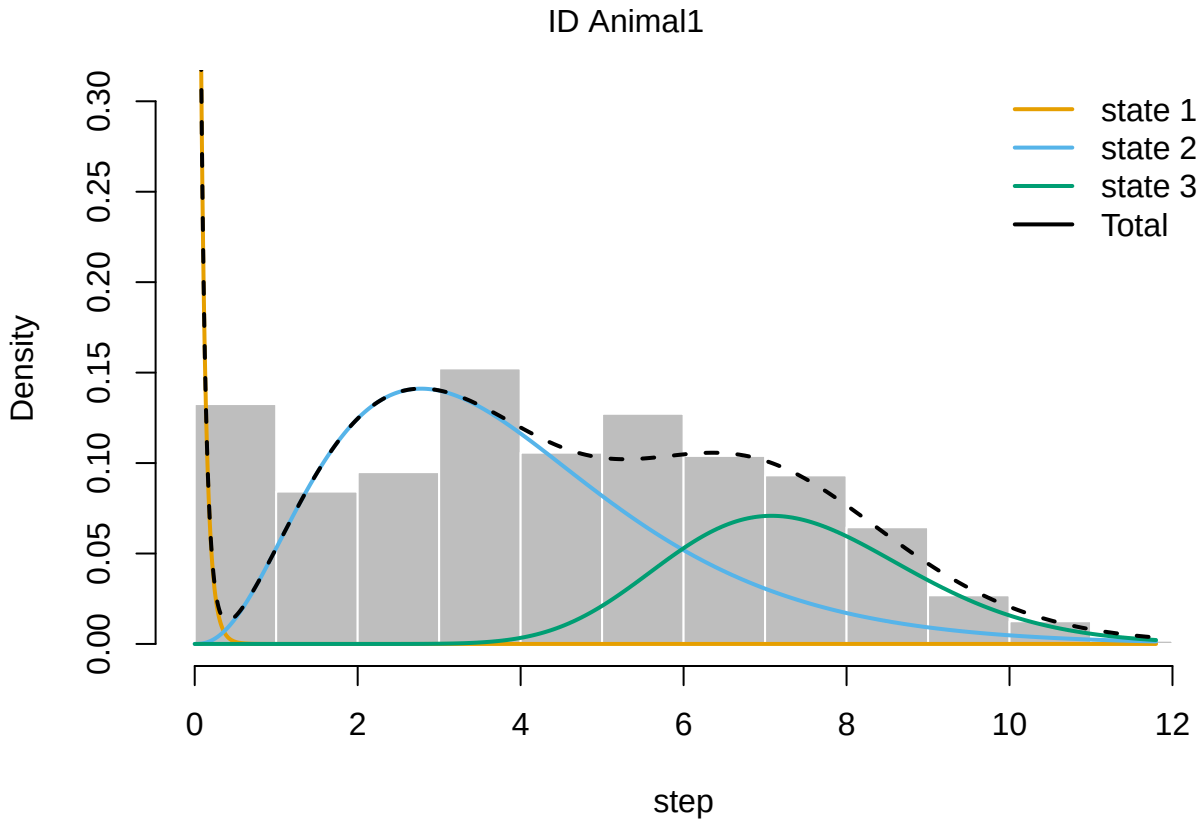
```r
# Get initial parameters
par0diurn <- getPar0(model = sealHMM3s, formula = diurnForm)

# Now we fit the diurnal model with the starting values returned by getPar0
sealHMMdiurn <- fitHMM(sealPrep,nbState = 3,
                  dist = list(step = "gamma", angle = "vm"),
                  Par0 = par0diurn$Par, beta0 = par0diurn$beta,
                  formula = diurnForm)
```

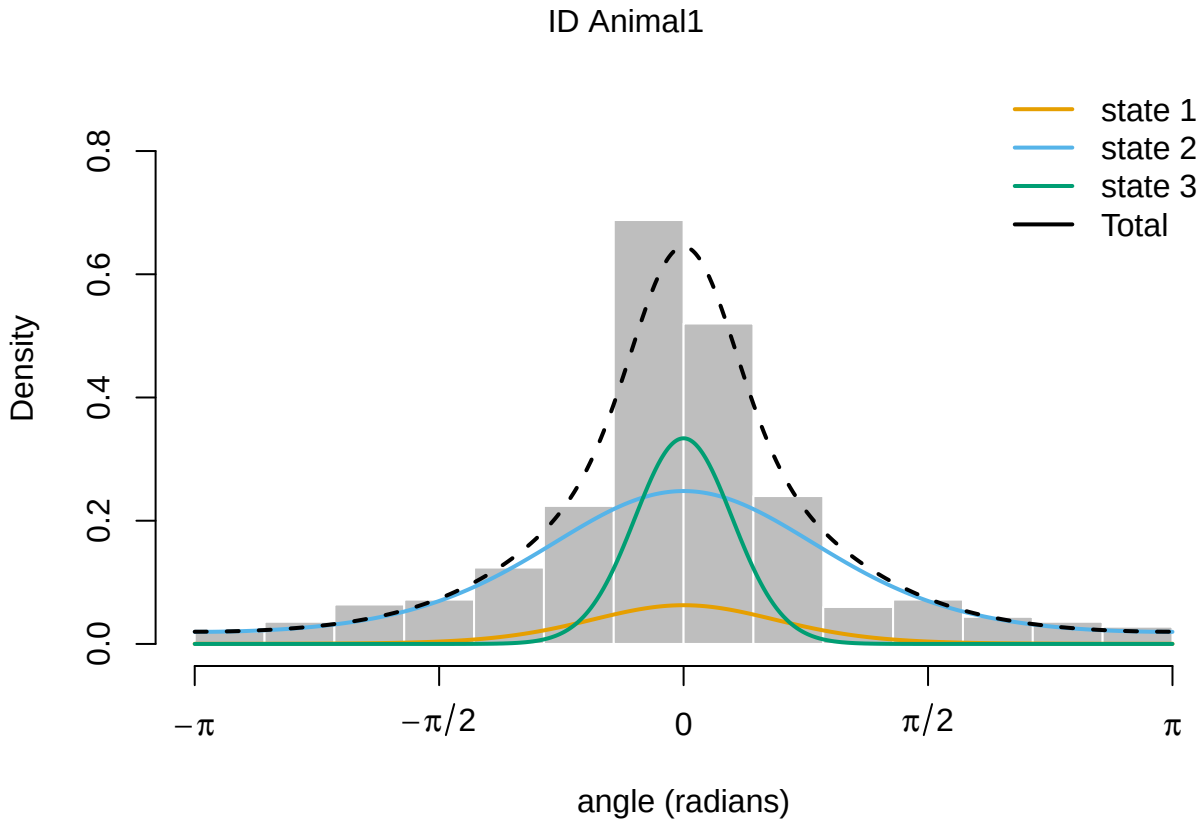
```r
# Let's look at the results
sealHMMdiurn
```

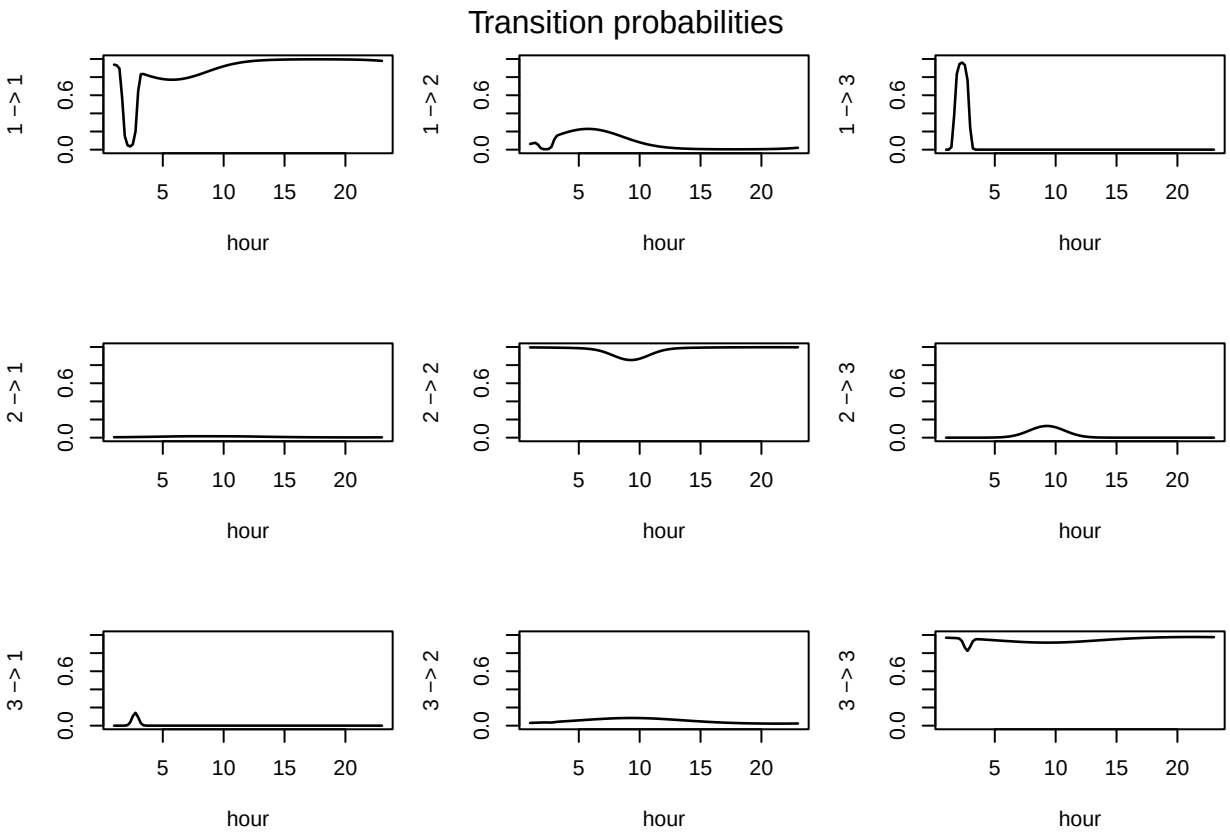
```
## Value of the maximum log-likelihood: -1591.808
##
##
## step parameters:
## ----------------
##         state 1  state 2  state 3
## mean 0.05161132 3.859728 7.396961
## sd   0.06660842 2.043112 1.528737
##
## angle parameters:
## -----------------
##                state 1  state 2  state 3
## mean          0.000000 0.000000  0.00000
## concentration 2.784346 1.269179 10.11164
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                                    1 -> 2    1 -> 3     2 -> 1    2 -> 3
## (Intercept)                    -3.4102734 -266.3777 -4.9320961 -9.726483
## cosinorCos(hour, period = 24)   0.1381921  222.3857 -0.6234936 -5.935069
## cosinorSin(hour, period = 24)   2.1911354  152.5816  0.6596653  5.112554
##                                    3 -> 1     3 -> 2
## (Intercept)                    -252.7053 -3.0804835
## cosinorCos(hour, period = 24)   189.0561 -0.5317083
## cosinorSin(hour, period = 24)   165.0249  0.4483214
##
## Transition probability matrix (based on mean covariate values):
## ---------------------------------------------------------------
##               state 1    state 2       state 3
## state 1  9.713296e-01 0.02867036 3.181031e-212
## state 2  1.306606e-02 0.96385369  2.308026e-02
## state 3 9.064525e-192 0.07286015  9.271398e-01
##
## Initial distribution:
## ---------------------
##      state 1      state 2      state 3
## 1.000000e+00 3.444088e-62 8.242713e-62
```

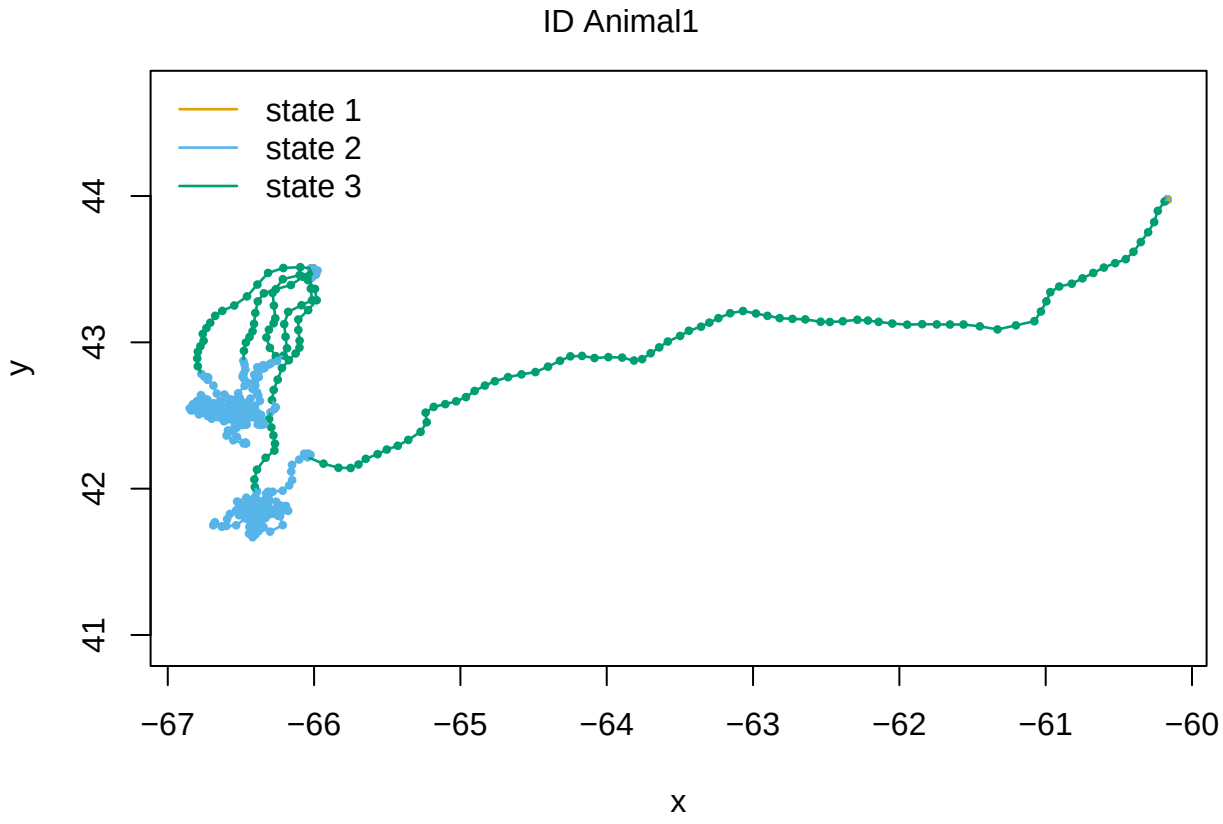
```r
plot(sealHMMdiurn)
```

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


ID Animal1
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
2
4
6
8
10
12
step
state 1
state 2
state 3
Total


ID Animal1
state 1
state 2
state 3
Total
Density
0.0
0.2
0.4
0.6
0.8
−π
−π/2
0
π/2
π
angle (radians)


Transition probabilities
1 -> 1
1 -> 2
1 -> 3
2 -> 1
2 -> 2
2 -> 3
3 -> 1
3 -> 2
3 -> 3
0.0
0.6
5
10
15
20
hour



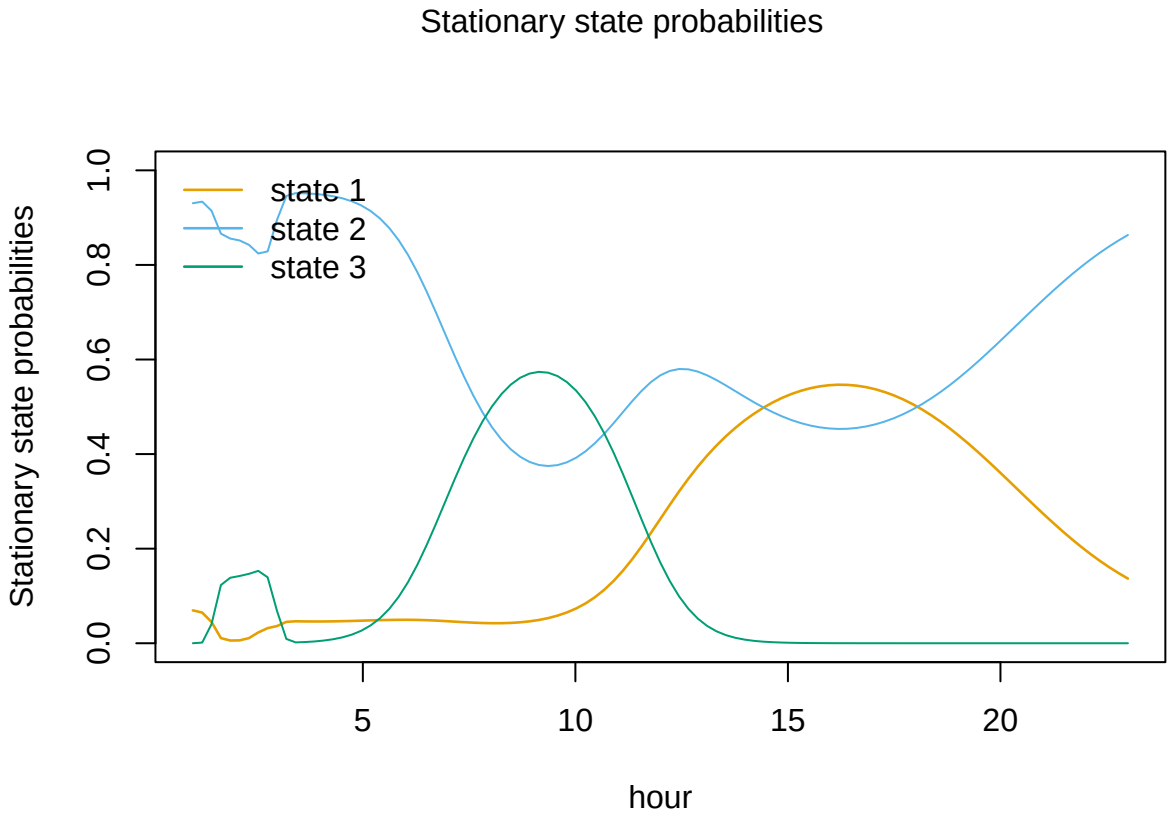

```
AIC(sealHMMb, sealHMMdiurn)
```

```
##          Model      AIC
## 1     sealHMMb 3218.748
## 2 sealHMMdiurn 3241.615
```

I'm not a 100% sure how to interpret the results biologically. But again it appears to be affecting state 1, which is likely resting. According to AIC it's less good than the bathymetry model.

To help understand the relationship further we can also use the function `plotStationary`, which display the relationship between the covariate and the stationary probability of being in each state rather than the relationship between the covariates and the switching probability.

```
plotStationary(sealHMMdiurn)
```



So it looks like there is a higher probability of being in state 2 (the foraging type of behaviour) at night and a peak in each of the other behaviours during different period of the day.

Covariates can affect the state-dependent movement, by which I mean the characteristics of the movement in a given state. So for example, the animal might make all the behaviours at night and during the day, but at night it might do them with smaller steps. Here, we are going to look at whether the seal is more or less directed at night. To do this, we are going to change the argument `DM`. For this kind of model, it's particular useful to use `getPar0`, because it helps set starting values for the parameters associated with the effects of covariates on the movement parameters.

```
# Use the formula to describe the changes in the design matrix
diurnDM <- list(angle = list(concentration = ~ cosinor(hour, period = 24)))

# Get parameter starting values
par0diurnDM <- getPar0(model=sealHMM3s, DM=diurnDM)
# Look at starting values
par0diurnDM
```

```
## $Par
## $Par$step
##     mean_1     mean_2     mean_3       sd_1       sd_2       sd_3
## 0.05215805 3.80915581 7.35724680 0.06769966 2.01880445 1.52086562
##
## $Par$angle
##                   concentration_1:(Intercept)
##                                     1.0273107
```

```
## concentration_1:cosinorCos(hour, period = 24)
##                                     0.0000000
## concentration_1:cosinorSin(hour, period = 24)
##                                     0.0000000
##                   concentration_2:(Intercept)
##                                     0.2191064
## concentration_2:cosinorCos(hour, period = 24)
##                                     0.0000000
## concentration_2:cosinorSin(hour, period = 24)
##                                     0.0000000
##                   concentration_3:(Intercept)
##                                     2.2839086
## concentration_3:cosinorCos(hour, period = 24)
##                                     0.0000000
## concentration_3:cosinorSin(hour, period = 24)
##                                     0.0000000
##
##
## $beta
##                1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1    3 -> 2
## (Intercept) -2.634604 -3.758437 -4.740937 -3.638665 -4.986308 -2.777883
##
## $delta
## [1] 9.999991e-01 4.949611e-07 4.549331e-07
```

->

-> -> -> ->