moveHMM workflow: wild haggis analysis

Théo Michelot

2023-05-08

This vignette briefly describes the workflow of a typical analysis in moveHMM, including data preparation, model specification, model fitting, and visualisation of results. As an example, we use the data set of 15 wild haggises (*Haggis scoticus*; Figure 1) from Michelot, Langrock, and Patterson (2016). This vignette is only meant to be used as an example to learn about the functionalities of moveHMM, but please consult another source for more technical details (e.g., Michelot, Langrock, and Patterson (2016), Langrock et al. (2012), Zucchini, MacDonald, and Langrock (2016)).

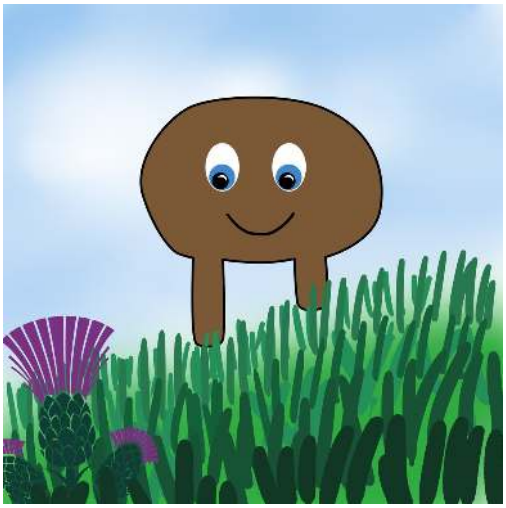


Figure 1: Wild haggis (*Haggis scoticus*)

1 Data preparation

The data set is included in moveHMM by default, which includes the first three tracks from the study of Michelot, Langrock, and Patterson (2016). This data set is already in the format expected by moveHMM, with columns:

- `ID`: an identifier for the track/individual;
- `x`: Easting or longitude coordinate;
- `y`: Northing or latitude coordinate.

These three variables are required, and should have these names. The other two columns, `slope` and `temp`, are covariates that will be included in the analysis later.

```
head(haggis_data)
```

```
  ID         x         y     slope      temp
1  1  0.000000  0.000000 25.957002 10.344959
2  1 -1.068761 -0.194650 18.606632  8.352531
3  1 -6.152549  2.051343 16.524004 13.529650
4  1 -6.703983  3.338480  9.154917 10.951095
5  1 -6.541667  3.553843  5.547686 11.243328
6  1 -7.160298  1.960377  8.129402 13.187280
```

The standard HMMs used in movement ecology, and implemented in this package, model step lengths and turning angles. The first step of the analysis is therefore to derive those variables from the observed locations. This can be done using the function `prepData`; here, we specify `type = "UTM"` to indicate that the locations are already projected (rather than longitude-latitude locations).

```
data <- prepData(haggis_data, type = "UTM")

head(data)
```

```
  ID      step      angle         x         y     slope      temp
1  1 1.0863417         NA  0.000000  0.000000 25.957002 10.344959
2  1 5.5578218 -0.5961622 -1.068761 -0.194650 18.606632  8.352531
3  1 1.4002860 -0.7500230 -6.152549  2.051343 16.524004 13.529650
4  1 0.2696813 -1.0506197 -6.703983  3.338480  9.154917 10.951095
5  1 1.7093394 -2.8660552 -6.541667  3.553843  5.547686 11.243328
6  1 1.1529149  2.3676683 -7.160298  1.960377  8.129402 13.187280
```

The output data frame has two new columns: `step` (step lengths), and `angle` (turning angle).

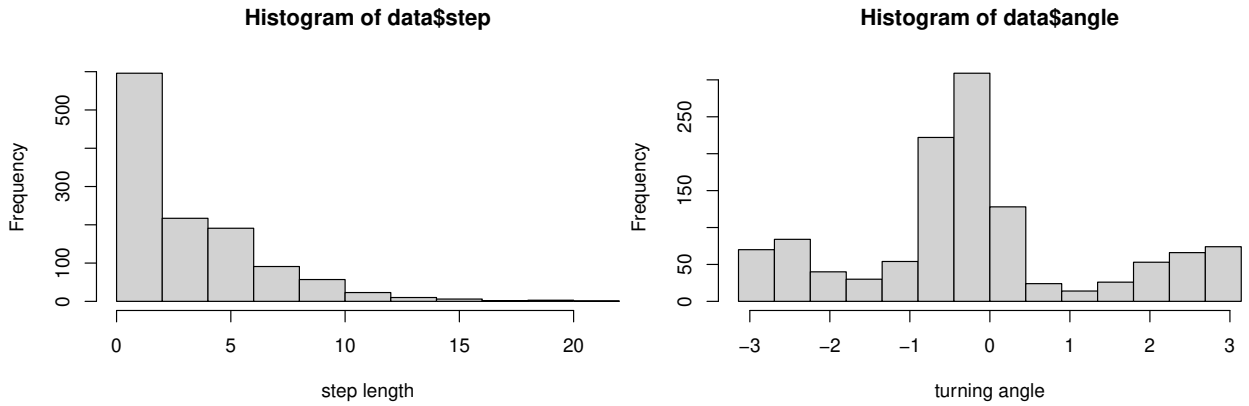
2 Model fitting

In moveHMM, the models are fitted using the function `fitHMM`. It implements maximum likelihood estimation, using the numerical optimiser `nlm`. The likelihood function captures how plausible the data are, given a set of parameters, and `nlm` explores the space of parameter values to find the one that maximise the likelihood. For this reason, it is necessary to choose initial parameter values, from where `nlm` can start its exploration, and these are specified using the `stepPar0` and `anglePar0` arguments in `fitHMM`. The choice of starting values can affect the performance of the optimiser, and in some cases its ability to find the maximum likelihood estimates. Some guidance on how to choose those values is provided in another vignette, called "A short guide to choosing initial parameter values for the estimation in moveHMM". In summary, one approach is to look at the empirical distributions of step lengths and turning angles, to make an educated guess about what parameter values are plausible.

In the following, we consider a 2-state HMM where, in each state, a gamma distribution models step length, and a von Mises distribution models turning angle. In moveHMM, the gamma distribution is specified in terms of a mean and standard deviation, and we pick some initial values based on the histogram of observed step lengths. The von Mises distribution has a mean and a concentration parameter, and reasonable starting values can be chosen based on a histogram of observed turning angles. Please refer to the dedicated vignette for more guidance about selecting initial parameters.

```
hist(data$step, xlab = "step length")
hist(data$angle, breaks = seq(-pi, pi, length = 15), xlab = "turning angle")

stepPar0 <- c(1, 5, 1, 5)
anglePar0 <- c(pi, 0, 0.3, 5)
```



We can fit the model using `fitHMM`, with arguments `nbStates` (number of states), `stepPar0` (initial step length parameters), and `anglePar0` (initial turning angle parameters). By default, the function uses the gamma and von Mises distributions, so we don't need to specify them here, but the arguments `stepDist` and `angleDist` could be included to use different distributions.

```
mod1 <- fitHMM(data = data, nbStates = 2,
               stepPar0 = stepPar0, anglePar0 = anglePar0)

mod1
```

```
Value of the maximum log-likelihood: -3462.487

Step length parameters:
-----------------------
       state 1  state 2
mean 0.9977756 5.009977
sd   0.4951516 3.044708

Turning angle parameters:
-------------------------
                 state 1    state 2
mean          -3.109852 -0.3082705
concentration  1.035834  8.7682400

Regression coeffs for the transition probabilities:
---------------------------------------------------
             1 -> 2    2 -> 1
intercept -1.736619 -2.002033
```

```
Transition probability matrix:
------------------------------
          [,1]      [,2]
[1,] 0.8502571 0.1497429
[2,] 0.1189896 0.8810104

Initial distribution:
---------------------
[1] 0.06369954 0.93630046
```

The object returned by `fitHMM` is a list that contains, among other things, estimates of all model parameters. Those are automatically shown in a convenient layout when the model object is printed, but they can also be accessed as `mod1$mle`.

Note that the parameter estimates are pooled across individuals; i.e., a common model is fitted to all tracks in the data set. This is based on the assumption that all animals follow similar movement patterns, and parameters can then be viewed as an average across individuals. In cases where this assumption doesn't hold, one option is to include individual-specific covariates on the transition probabilities, to capture some of the inter-individual heterogeneity. (See section on covariates below.)

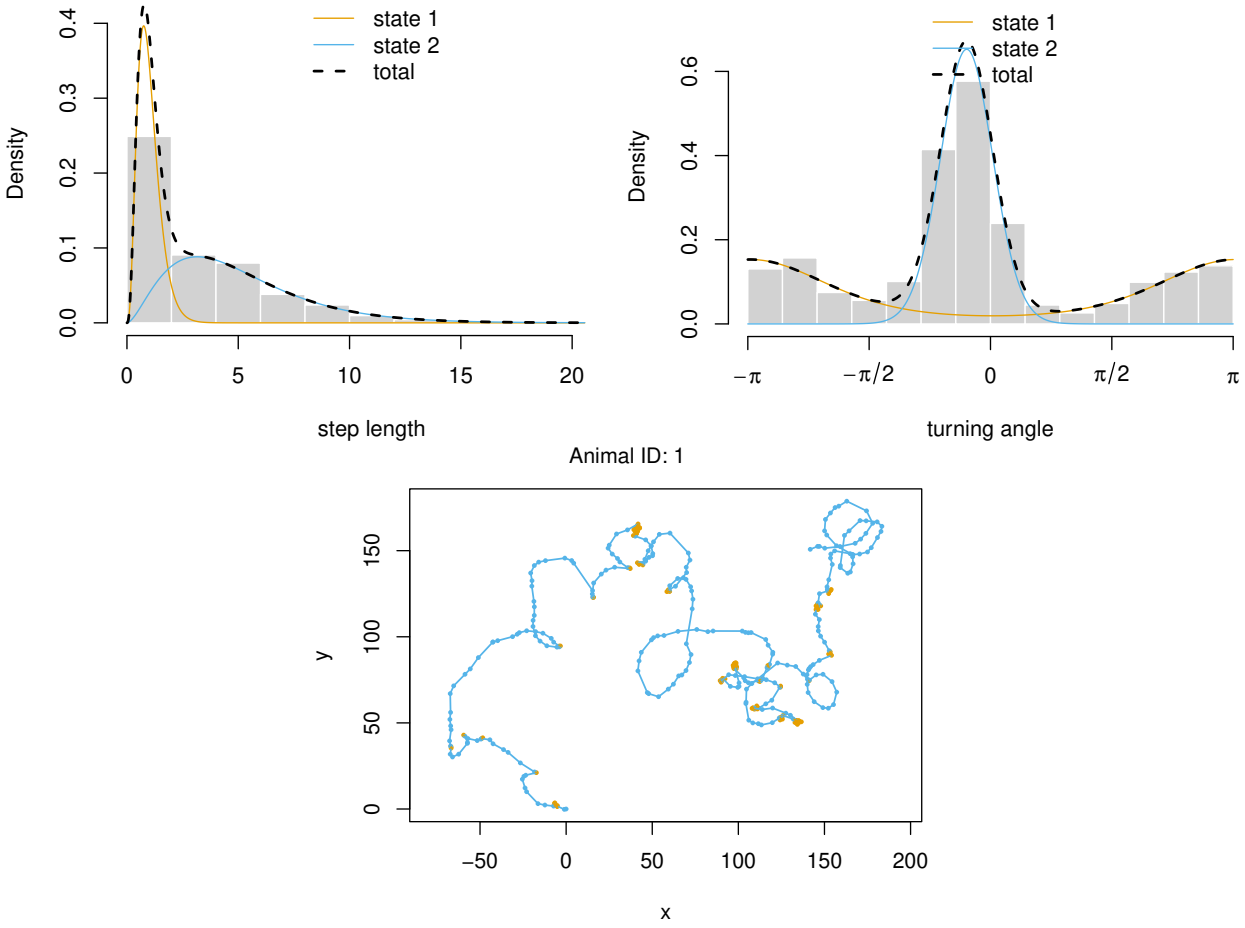
3 Model visualisation

To understand and interpret a fitted model, it is often helpful to visualise it. The `plot` function can be called on the model object to generate a few plots, including:

- histograms of the step lengths, overlaid with estimated distributions in each state. This is helpful to check whether the fitted distributions match the data well, and to interpret each state (e.g., does state 1 capture long or short step lengths? how variable are step lengths in state 2?).
- histograms of the turning angles, overlaid with estimated distributions in each state.
- plots of the movement tracks, coloured by estimated state. This provides useful information about where an when animals followed each movement type, what movement patterns were captured in each state, etc.

```
plot(mod1, ask = FALSE, animals = 1)
```

```
Decoding states sequence... DONE
```



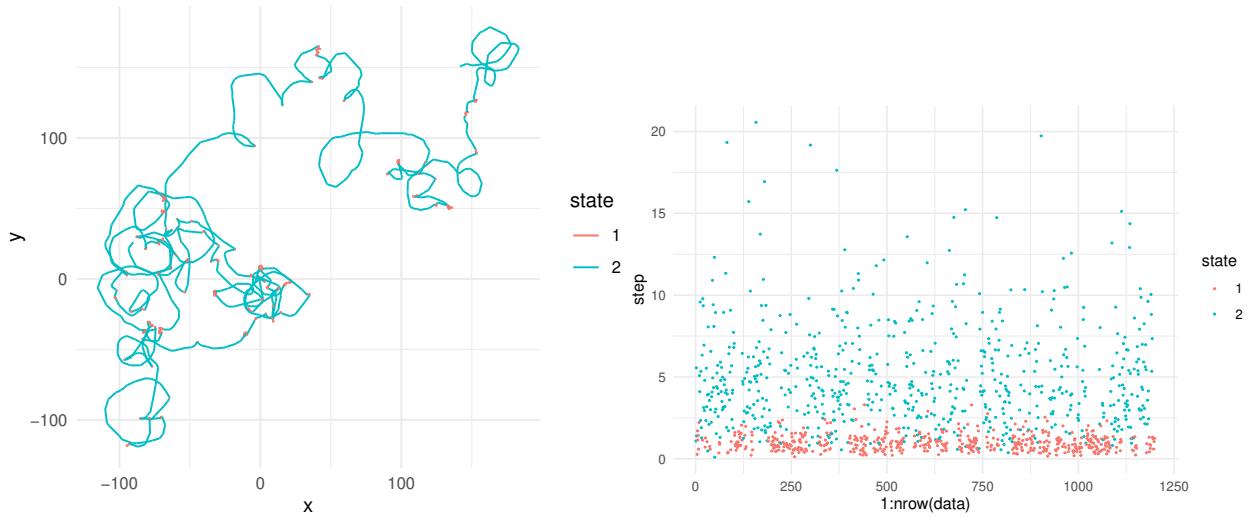
The first state captures short step lengths (mostly between 0 and 2 km) and undirected movement (flat distribution of turning angles), whereas the second state captures long step lengths (between 0 and 15 km) and directed movement (peak of turning angles at zero). In practice, these two states might therefore be used as proxies for two different behavioural states.

4 State process inference

As shows in the plots above, it is possible to estimate a state for each observation, which is often of great applied interest. The most common way to do this is to use the Viterbi algorithm. The function `viterbi` returns a sequence of states (i.e., vector of numbers between 1 and `nbStates`), and these can for example be used to generate plots of the tracks, or of the step lengths, coloured by the most likely state sequence.

```
# Add most likely state sequence to data
data$state <- factor(viterbi(mod1))
```

```r
# Plot tracks coloured by state
ggplot(data, aes(x, y, col = state, group = ID)) +
    geom_path() +
    coord_equal()
# Plot step lengths coloured by state
ggplot(data, aes(x = 1:nrow(data), y = step, col = state, group = ID)) +
    geom_point(size = 0.3)
```



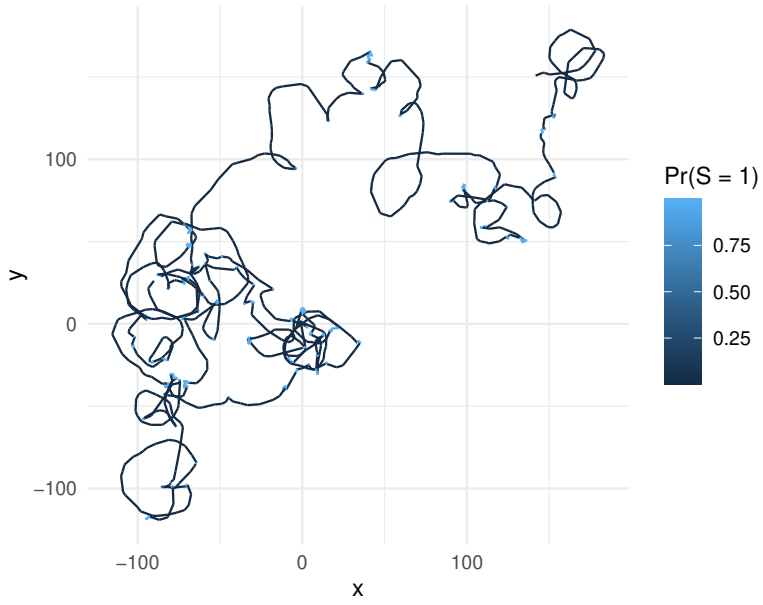
An alternative to the Viterbi algorithm is to compute state probabilities, i.e., the probability of being in each state for each observation in the data. State probabilities provide more detailed information about the uncertainty on the state allocation, and they can be computed with the function `stateProbs`. The output is a matrix with one column for each state and one row for each observation.

```r
sp <- stateProbs(m = mod1)
head(sp)
```

```
             [,1]         [,2]
[1,] 9.588177e-02 9.041182e-01
[2,] 1.215788e-06 9.999988e-01
[3,] 3.145690e-01 6.854310e-01
[4,] 9.586125e-01 4.138746e-02
[5,] 1.000000e+00 1.260704e-08
[6,] 1.000000e+00 9.504989e-10
```

```r
# Add prob of state 1 to data set
data$sp1 <- sp[,1]
```

```
ggplot(data, aes(x, y, col = sp1, group = ID)) +
    geom_path() +
    coord_equal() +
    labs(col = "Pr(S = 1)")
```



5 Covariates

HMMs have been extended to allow for covariate effects on the transition probabilities. This approach is often interesting to answer ecological questions of the form "does [some covariate] have an effect on the animal's behaviour?".

We model the haggises' transition probabilities as functions of two covariates: temperature, and slope. We expect a non-linear effect of slope, so we include a quadratic term for that covariate. The model formula can be specified using the usual R syntax, and passed through the argument `formula` in `fitHMM`. The other arguments are unchanged.

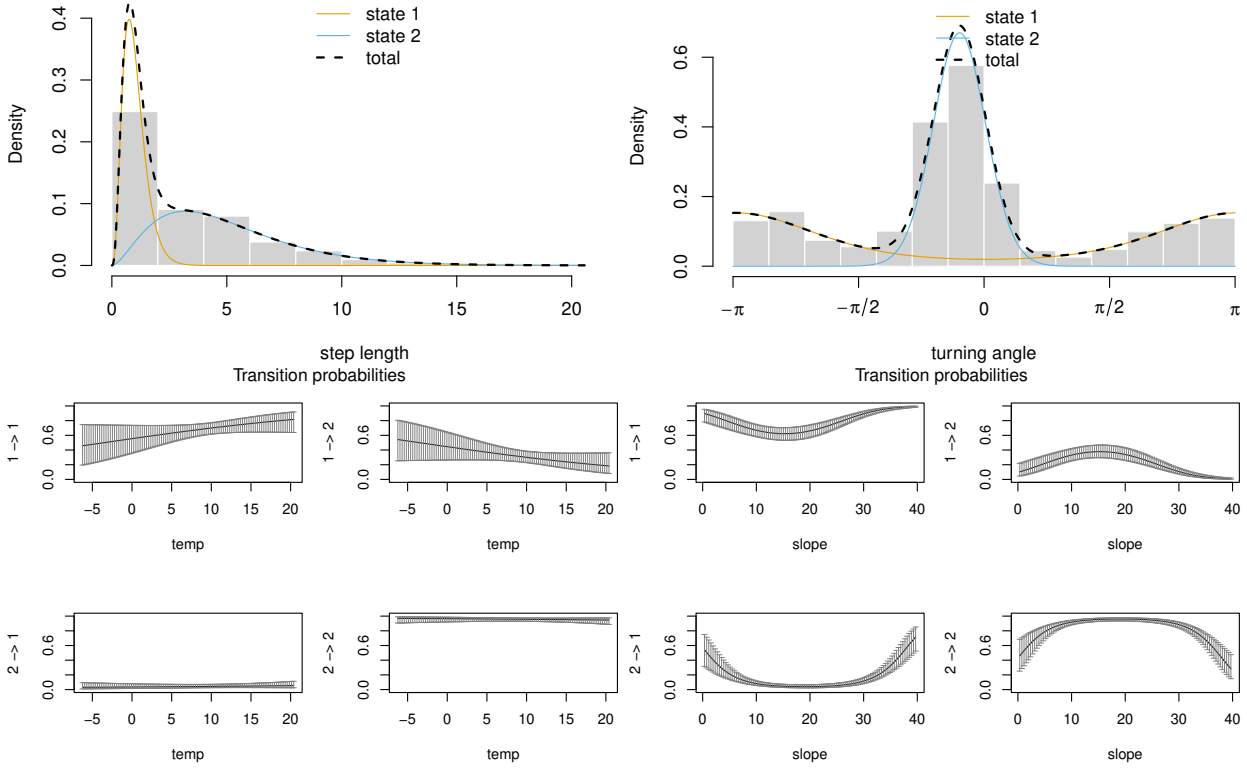
```
mod2 <- fitHMM(data = data, nbStates = 2,
               stepPar0 = stepPar0, anglePar0 = anglePar0,
               formula = ~ temp + slope + I(slope^2))
```

Plotting the model now also returns plots of the transition probabilities over a grid of values of each covariate (with 95% pointwise confidence intervals if `plotCI = TRUE`). In this example, it looks like there is no clear effect of temperature, but a clear effect of slope. The probability of remaining in state 2 (or of switching from state 1 to state 2) is highest when the slope is between 10 and 30 degrees, suggesting that wild haggises move faster in that range of slopes.

(See Michelot, Langrock, and Patterson (2016) for more details about the interpretation, but it has to do with their morphology.)

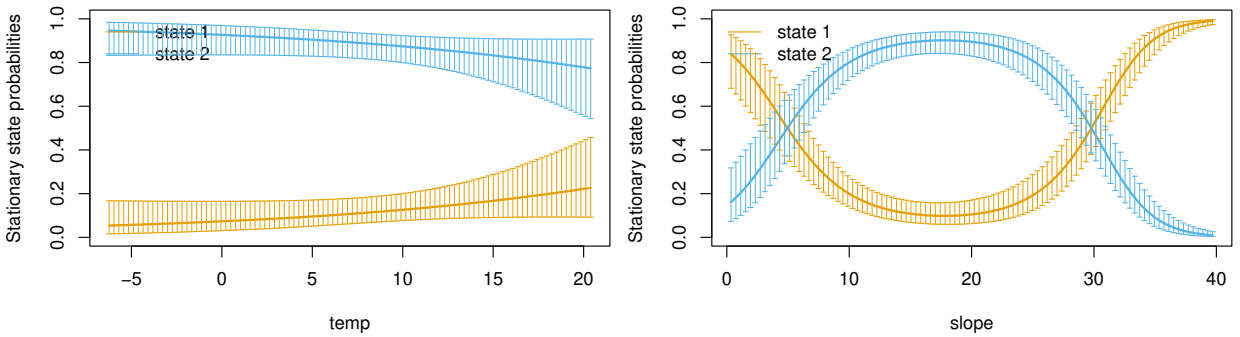
```
plot(mod2, ask = FALSE, plotTracks = FALSE, plotCI = TRUE)
```

```
Decoding states sequence... DONE
```



Another useful output to interpret covariate effects is the stationary state probabilities. These capture the probability of being in each state over the long term, and they can be derived for a grid of covariate values to assess how animals' activity budget depends on a covariate. The probabilities can be computed with `stationary`, and plotted with `plotStationary`. The stationary state probabilities make it even clearer that haggises tend to spend time in state 2 when the slope is between 10 and 30, but are likely to be in state 1 for small or large slope values.

```
plotStationary(mod2, plotCI = TRUE)
```



Models can be compared using the `AIC` function, for example to decide whether or not to keep a covariate. Here, the model with covariate is strongly preferred by AIC.

```
AIC(mod1, mod2)
```

```
  Model      AIC
1  mod2 6814.784
2  mod1 6946.975
```

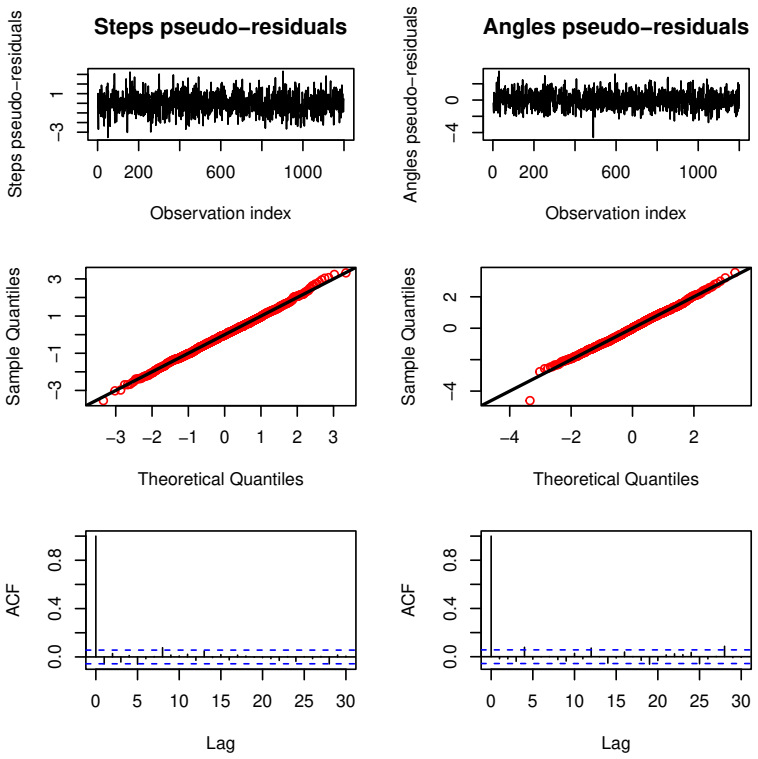
6 Model checking

A chosen HMM formulation rests on many assumptions, including the number of states, the types of distributions used for step length and turning angle, and the dependence structure of the data-generating process. Visual inspection of the model outputs (e.g., fitted distributions) is a first step to check whether it seems to capture features in the data well. Pseudo-residuals are another tool to assess goodness-of-fit in HMMs. Similarly to residuals in linear models, they should be independent and follow a standard normal distribution if all model assumptions were satisfied. Deviations from these patterns suggest lack of fit.

Pseudo-residuals can be computed with `pseudoRes`, which returns a vector for step lengths and one for turning angles. The function `plotPR` creates three plots of the pseudo-residuals for each data variable: a time series plot, a quantile-quantile (QQ) plot against the standard normal distribution, and an autocorrelation function (ACF) plot.

```
plotPR(mod2)
```

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



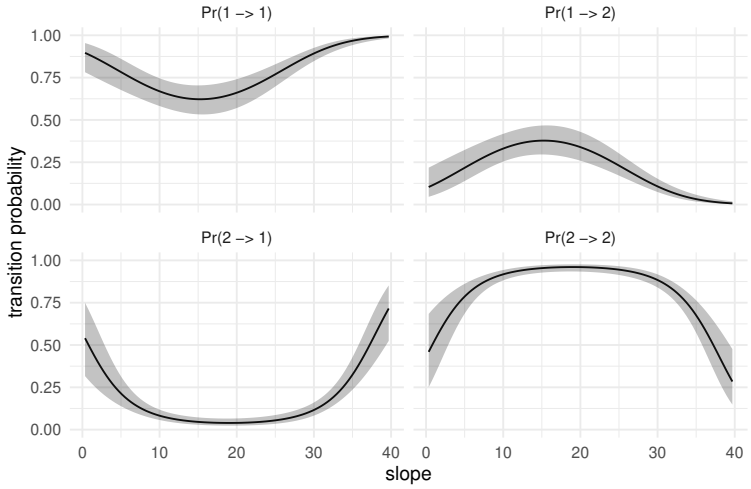
In the QQ plot, deviations from the diagonal line suggest that the estimated distributions do not capture the empirical distribution of the corresponding variable. Here, the points are aligned on the diagonal, indicating that the fit is good. The ACF plots can be used to detect residual autocorrelation (bars that stretch beyond the confidence band around zero). In this case, the ACF is virtually zero, suggesting that the dependence assumptions of the model are satisfied.

7 Custom plots

The function `getPlotData` creates data frames in a format that is convenient for creating custom plots of the model, including state-dependent distributions (`type = "dist"`), transition probabilities (`type = "tpm"`), and stationary state probabilities (`type = "stat"`). The option `format` specifies whether the data frame should be wide (typically for base R plots) or long (typically for ggplot). With this, we can recreate the plot of transition probabilities shown above, in ggplot.

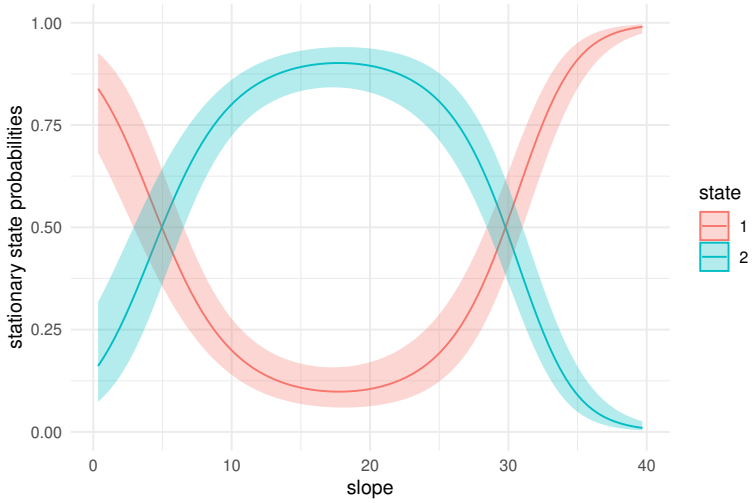
```
# Plot of transition probs as function of slope
plotData1 <- getPlotData(m = mod2, type = "tpm", format = "long")
ggplot(plotData1$slope, aes(slope, mle)) +
    facet_wrap("prob") +
    geom_line() +
```

```
    geom_ribbon(aes(ymin = lci, ymax = uci), alpha = 0.3) +
    labs(y = "transition probability")
```



Similarly, we can create a plot of stationary state probabilities as functions of slope.

```
# Plot of stationary state probs as function of slope
plotData2 <- getPlotData(m = mod2, type = "stat", format = "long")
ggplot(plotData2$slope, aes(slope,  mle, col = factor(state))) +
    geom_line() +
    geom_ribbon(aes(ymin = lci, ymax = uci, col = NULL,
                    fill = factor(state)), alpha = 0.3) +
    labs(fill = "state", col = "state", y = "stationary state probabilities")
```



8 Other features

8.1 `predict` functions

The functions `predictTPM` and `predictStationary` compute the transition probabilities (or stationary state probabilities) for a data frame of covariate values passed as input, possibly with confidence intervals. For example, let's say we want to predict the transition probability matrix for temp = 0 and slope = 17.

```
new_data <- data.frame(temp = 0, slope = 17)
new_data
```

```
  temp slope
1    0    17
```

```
tpm <- predictTPM(m = mod2, newData = new_data, returnCI = TRUE)
tpm
```

```
$mle
, , 1

            toState1  toState2
fromState1 0.4650446 0.5349554
fromState2 0.0328305 0.9671695


$lci
, , 1

             toState1  toState2
fromState1 0.27370706 0.3327452
fromState2 0.01418785 0.9258726


$uci
, , 1

             toState1  toState2
fromState1 0.66725478 0.7262929
fromState2 0.07412736 0.9858121
```

The element `mle` ("maximum likelihood estimate") is the estimate, and `lci` and `uci` are the bounds of the confidence interval.

8.2 `knownStates`

The argument `knownStates` of `fitHMM` can be used to fix states that are known a priori. This can be useful in cases where some observations have been classified into behaviours a priori, and the problem of interest is to classify other points into those states. The models are then semi-supervised, because we are giving some information to the model about what the states should be (rather than letting them be completely data-driven).

In practice, `knownStates` should be a vector of same length as the data, where each element is either NA (if the state is not known at that time step), or a number between 1 and `nbStates` (when the state is known).

8.3 `fit = FALSE`

It is sometimes useful to create a model without fitting it. For example, we may fit a model on one data set, and then wish to use that model to estimate the state sequence of another data set. This is possible using the option `fit = FALSE` in `fitHMM`. Specifically, we can follow these steps:

1. Fit a model to the first data set in the conventional way (as described previously).
2. Create a model for the second data set using `fitHMM` with `fit = FALSE`, passing the estimated parameters of the previous model as starting values.
3. Use `viterbi` (or `stateProbs`) on the second model.

This procedure might be helpful for data exploration, or in cases where the full data set is very large and it is only possible to fit a model to a subset.

References

Langrock, Roland, Ruth King, Jason Matthiopoulos, Len Thomas, Daniel Fortin, and Juan M Morales. 2012. "Flexible and Practical Modeling of Animal Telemetry Data: Hidden Markov Models and Extensions." *Ecology* 93 (11): 2336–42.

Michelot, Théo, Roland Langrock, and Toby A Patterson. 2016. "moveHMM: An R Package for the Statistical Modelling of Animal Movement Data Using Hidden Markov Models." *Methods in Ecology and Evolution* 7 (11): 1308–15.

Zucchini, Walter, Iain L MacDonald, and Roland Langrock. 2016. *Hidden Markov Models for Time Series: An Introduction Using R, Second Edition*. Chapman; Hall/CRC.