State-switching step selection functions in hmmSSF

Natasha Klappstein & Théo Michelot

2023-06-09

Contents

We describe the analysis workflow for the R package hmmSSF, which implements state-switching step selection functions as described by Klappstein et al. [2023] (and originally proposed by Nicosia et al. [2017]). This package can be used to estimate behaviour-dependent habitat selection and movement parameters, bringing together hidden Markov models (Michelot et al. [2016]) and step selection functions (Signer et al. [2019]).

1 Word of caution

This model is fairly complex compared to standard step selection functions, and requires more care to avoid numerical problems. Indeed, the number of parameters is higher, and the problem of inference requires estimating the dynamics of an unobserved state process (used as proxy for the behavioural state of the animal).

In our experience, this model complexity often leads to numerical instability, i.e., failure of the fitting function to converge to the best parameter estimates. This is a difficult general problem, and here are some possible directions to explore to help with it:

- increase the number of control locations (see "Generating control step" below)
- use a simpler model formulation. The more complex the model, the more difficult it is to fit. It is usually good to start from the simplest model, i.e., a 2-state model with no covariates on the transition probabilities, and build up complexity if it seems possible.
- try different sets of starting values for the parameters (see "Starting parameter values" below)

2 Data preparation

We consider an example (simulated) track with 2500 locations from one animal, and one spatial covariate. The tracking data is shown on top of a heatmap of the covariate in Figure 1.

2.1 Format of tracking data

The package expects data to be provided as a data frame with the following columns:

- `ID`: identifier for track/animal
- `x`: Easting
- `y`: Northing

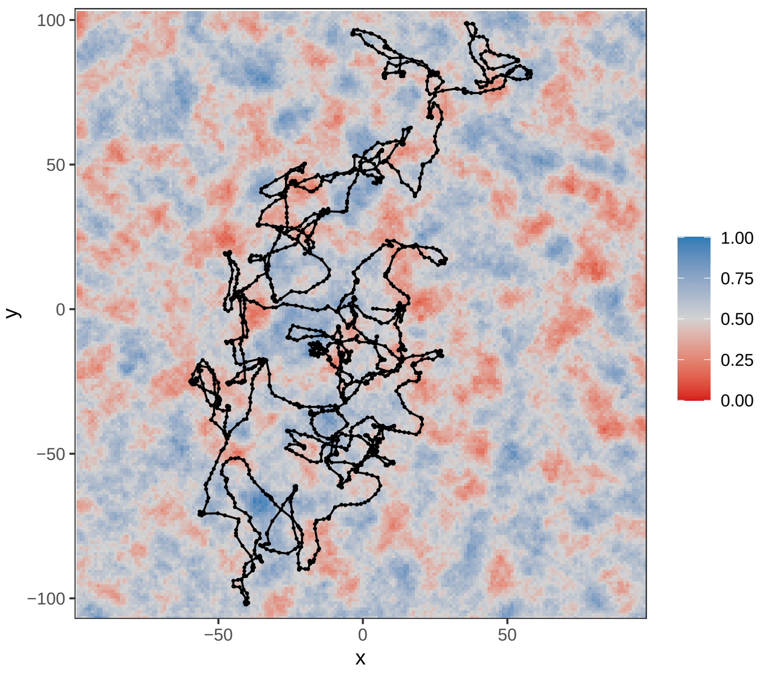


Figure 1: Example data.

- `time`: time of observation

The locations need to be projected from longitude-latitude to Easting-Northing prior to analysis, as the package uses Euclidean (planar) geometry to derive metrics such as step lengths and turning angles.

```
head(track)
```

```
  ID         x          y                time
1  1  3.517854  0.1967671 2020-01-01 00:00:00
2  1  4.781500  0.1262261 2020-01-01 01:00:00
3  1  8.635494 -0.2937678 2020-01-01 02:00:00
4  1 10.328140 -0.5135970 2020-01-01 03:00:00
5  1 11.852726 -0.1078853 2020-01-01 04:00:00
6  1 12.090271 -0.9006765 2020-01-01 05:00:00
```

2.2 Generating control steps

The most common method to fit a step selection function is to generate a set of "control" steps for each observed step, to weigh the suitability of the chosen move against alternatives. Forester et al. [2009] pointed out that it is advantageous to generate the control steps from distributions of movement variables that resemble the observed movement pattern. Klappstein et al. [2023] describe this as a form of importance sampling.

In hmmSSF, the control steps are generated using the function `get_controls()`, with multiple possible distributions (estimated from the observed data). The sampling distributions are specified by the argument `distr`, with the following options:

- if `distr = "uniform"`, then the control steps are generated by sampling points uniformly on a disc around each observed location, with radius set to the maximum observed step length.
- if `distr` is set to `"gamma"` or `"exp"`, then this is used as a distribution of distances (estimated from the distribution of observed step lengths) to generate control steps, and the turning angles are uniform.
- if `distr` is specified as a vector of two character strings, then the first is used as distribution of distances (`"gamma"` or `"exp"`), and the second as a distribution of turning angles (`"vm"` or `"wrpcauchy"`) to generate control steps.

The number of control steps also needs to be specified, with larger numbers leading to better approximations but higher computational cost. In general, the choice of the distribution and number of control locations is difficult, as it can affect the results, but there is no general guidance. We encourage users to test different distributions and numbers of control locations, to assess the sensitivity of their inferences.

For this example, we decide to generate 50 control locations per observed step, using a gamma distribution of distances. The output has the same columns as the original data frame, and a few new ones: `stratum` identifies the stratum to which a location belongs, and `obs` is a binary variable indicating whether a location is observed or not.

```r
data <- get_controls(obs = track, n_controls = 50, distr = "gamma")

head(data)
```

```
  ID stratum obs        x          y       step       angle                time
1  1       3   1 8.635494 -0.2937678 3.87681125 -0.05278238 2020-01-01 02:00:00
2  1       3   0 5.675062 -0.6194955 1.16385268 -0.63969012 2020-01-01 02:00:00
3  1       3   0 5.143816  0.8498971 0.80930400  1.16238281 2020-01-01 02:00:00
4  1       3   0 1.847962  2.1092420 3.54090345  2.60293491 2020-01-01 02:00:00
5  1       3   0 4.835666  0.1277922 0.05418846  0.08467172 2020-01-01 02:00:00
6  1       3   0 3.789019  0.1708322 0.99348242 -3.13074097 2020-01-01 02:00:00
```

In the importance sampling approach, the control steps need to be weighted during model fitting based on how likely they were under the distribution used to generate them. This technical detail is hidden in hmmSSF, and the weights are automatically computed and stored as an attribute of the data frame (for later use in `fitHMMSSF()`). For this reason, the control steps need to have been generated by `get_controls()`, rather than some other method (e.g., the amt package).

2.3 Extracting spatial covariates

The spatial covariate needs to be evaluated at all observed and control locations. The raster is stored as a `SpatRaster`, and we use the package terra for this purpose.

```
data$cov1 <- extract(cov1, data[,c("x", "y")])$layer
```

3 Model specification and model fitting

3.1 Model formulation

The main modelling decisions are the choice of the number of states in the HMM, and the choice of the SSF formula within each state.

There is no general method to select the best number of states, and we recommend leaning towards fewer states (2 or 3) in most applications to avoid numerical problems. (See also Pohle et al. [2017] for a discussion of this issue.) Here, we use two states, which we hope can separate slow and fast movement phases.

The formula should include movement variables as well as covariates (Avgar et al. [2016]). For example, including step length (and possibly its log) captures the preference for some range of movement speeds, and including the cosine of turning angle captures directional persistence. See Avgar et al. [2016] and Klappstein et al. [2023] for more details. We use a formula with step length, log step length, cosine of turning angle, and the spatial covariate `cov1`.

```
# number of HMM states
n_states <- 2

# SSF formula
ssf_formula <- ~ step + log(step) + cos(angle) + cov1
```

3.2 Starting parameter values

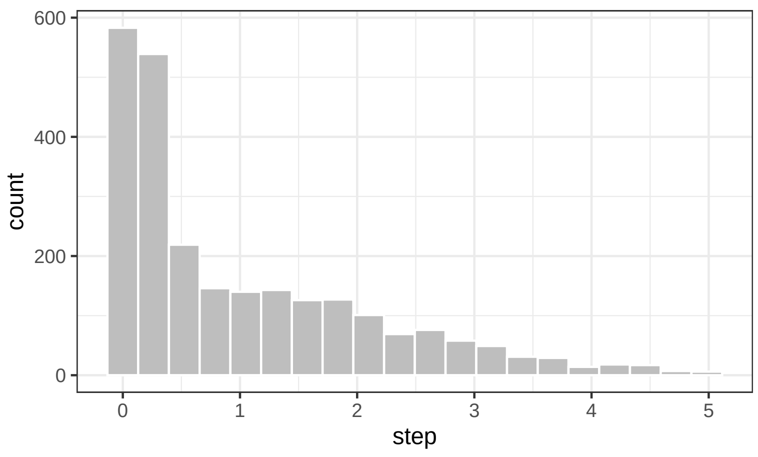
In hmmSSF, the model is fitted through numerical likelihood optimisation, and starting values need to be chosen for the model parameters. The choice of these values does not affect the model specification, but poorly chosen values can lead to numerical problems (e.g., failure of optimiser to converge). The general idea is that, if starting values are close to the "true" parameters, it will be easier for the optimiser to converge. In practice, the true parameters are not known, and so our best bet is to choose plausible values given the data.

Here, we use the fact that the selection parameters for `step` and `log(step)` are linked to the mean μ_L and standard deviation σ_L of a gamma distribution through the relationships:

$$\beta_L = -\frac{\mu_L}{\sigma_L^2}, \quad \beta_{\log(L)} = \frac{\mu_L^2}{\sigma_L^2} - 2.$$

We can choose some hypothesised mean step length for each state by looking at the empirical distribution, and transform them using these formulas to get starting values for the selection coefficients. To visualise step lengths, we subset the data to observed steps (i.e., removing control steps).

```r
ggplot(subset(data, obs == 1), aes(x = step)) +
  geom_histogram(col = "white", fill = "grey", bins = 20)
```



We might want state 1 to capture slow movement (e.g., steps between 0 and 1km), and state 2 to capture faster movement (e.g., steps between 1 and 3km), so we choose a mean of 0.2km for state 1 and 2km for state 2. We use the same value as the mean for the standard deviation in each state. Plugging these values into the above equations yields (-5, -0.5) for β_L, and (-1, -1) for $\beta_{\log(L)}$.

Similarly, the selection parameter for `cos(angle)` is the concentration of a von Mises distribution (Klappstein et al. [2023]). The larger the concentration, the more peaked the distribution is around zero, i.e., the more directed the movement. We choose a starting value of 0.2 in state 1 and 5 in state 2, based on the expectation that slow movement is less directional than fast movement.

The parameters should be passed as a matrix, with one row for each covariate in the formula (here, 4), and one column for each state (here, 2).

```
# initial values for SSF parameters
ssf_par0 <- matrix(c(-5, -0.5,
                     -1, -1,
                     0.2, 5,
                     3, -1),
                   ncol = 2, byrow = TRUE)
```

3.3 Model fitting

Finally, we can pass the model formulation, the data, and the starting parameter values to the function `fitHMMSSF()` for model fitting.

```
# fit model
mod <- hmmSSF(ssf_formula = ssf_formula,
              n_states = n_states,
              data = data,
              ssf_par0 = ssf_par0)
```

The fitted model object can then be printed to find the estimated parameter values.

```
mod
```

```
Negative log-likelihood: -1337.694
Convergence code: 0

SSF model:
~step + log(step) + cos(angle) + cov1
                 mle    low    upp
step.S1       -3.662 -4.005 -3.319
log(step).S1  -1.001 -1.068 -0.933
cos(angle).S1  0.202  0.125  0.279
cov1.S1        0.131 -1.816  2.079
step.S2       -1.916 -2.110 -1.723
log(step).S2   1.896  1.511  2.280
cos(angle).S2  4.785  4.361  5.210
```

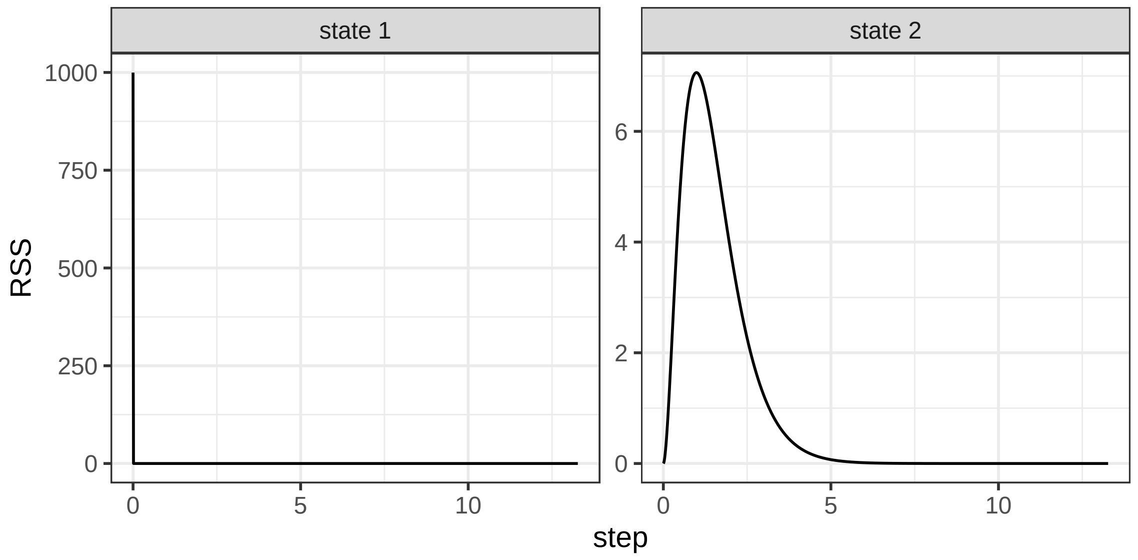
```
cov1.S2       -0.793 -2.091  0.505

TPM model:
~1
      S1    S2
S1 0.901 0.099
S2 0.134 0.866
```

4 Interpretation of results

The function `plot_ssf()` can be used to plot the step selection function against any covariate included in the model. **Important:** the y axis shows relative likelihoods of selection, and y values should only be interpreted by comparing two covariate values. More precisely, for some covariate x (chosen as `var` argument) with estimated selection parameter β, the plot shows $k \times \exp(\beta x)$ against a grid of x values. The multiplicative constant k is arbitrary, which is why the scale of the y axis should only be interpreted in a relative way.

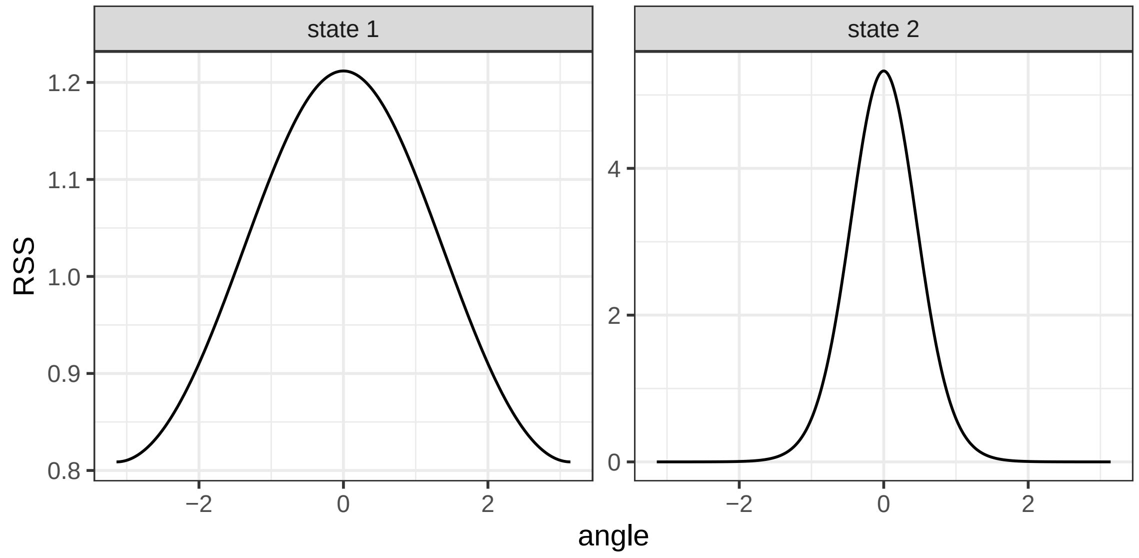
We first plot selection for step length L, $\exp(\beta_1 L + \beta_2 \log(L))$, in each state. The two distributions of step length are very different: state 1 captures shorter steps, and state 2 longer steps.

```
plot_ssf(mod = mod, var = "step")
```



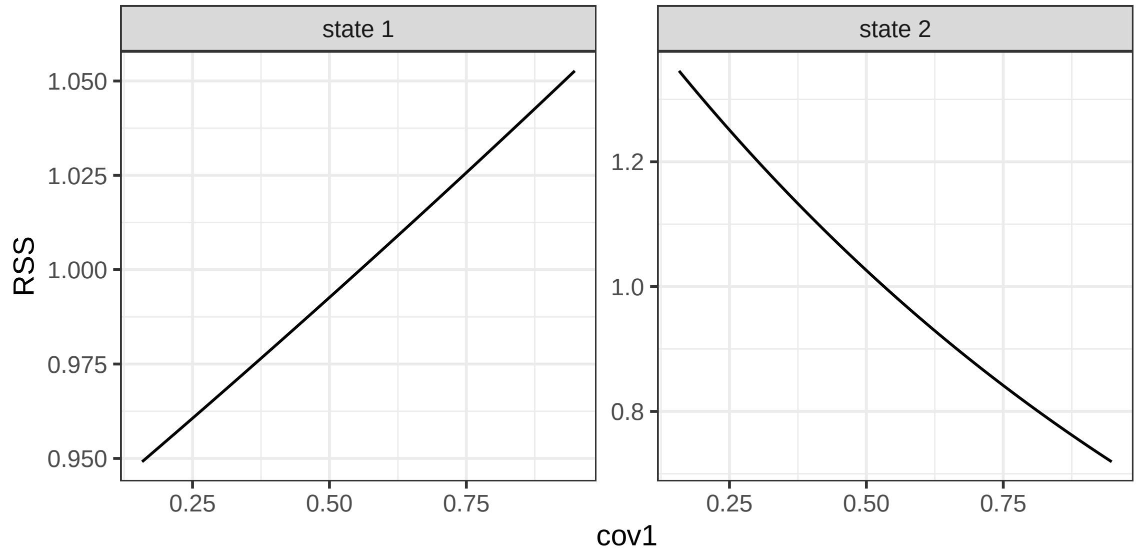
Similarly, we can plot selection for turning angle α, i.e, $\exp(\beta_3 \cos(\alpha))$. Movement in both state tends to be directed (i.e., with a peak at zero), and the directionality is stronger in state 2. This is consistent with the expectation that fast movement also tends to be more directed.

```
plot_ssf(mod = mod, var = "angle")
```



Finally, we plot the selection for the spatial covariate `cov1`, i.e., $\exp(\beta_4 x)$. The results suggest that the animal selected for the covariate in state 1 (positive relationship), but tended to avoid it in state 2 (negative relationship).

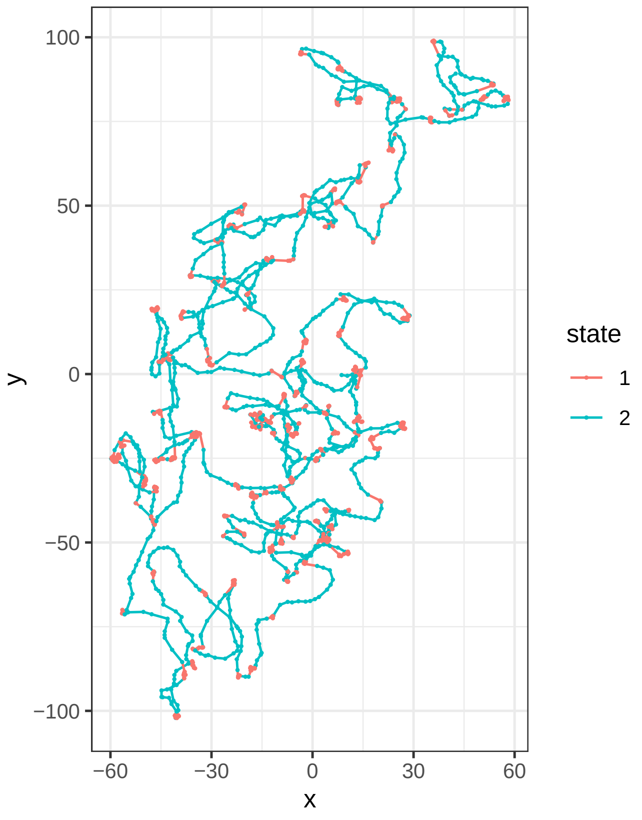
```
plot_ssf(mod = mod, var = "cov1")
```



Another useful output is the most likely state sequence, which can be computed with the function `viterbi_decoding()`. We can use it to visualise the movement track coloured by states, which confirms that states 1 and 2 correspond to slow and fast movement, respectively.

```
# get most likely state sequence
states <- viterbi_decoding(mod = mod)
```

```
# data frame for plotting (remove first two obs - not used in model)
df <- data.frame(x = track$x[-(1:2)],
                 y = track$y[-(1:2)],
                 state = factor(states))

ggplot(df, aes(x, y, col = state, group = NA)) +
  geom_path() +
  geom_point(size = 0.2) +
  coord_equal()
```



5 Covariates on transition probabilities

The effects of covariates on the transition probabilities of the hidden process can be included with the argument `tpm_formula`. Klappstein et al. [2023] describes this model, and discusses when a variable should be included in this part of the model (rather than in the SSF directly, say). Briefly, this option is useful to answer questions such as "Does this covariate affect the proportion of time spent in the different behaviours?".

We add a column `tod` to the data frame, which is a numeric variable for the time of day.

To include its effect on the transition probabilities, we define a formula with cyclical effects (with period = 24), based on trigonometric functions (Towner et al. [2016]).

```
# add time of day covariate to data frame
data$tod <- lubridate::hour(data$time) + lubridate::minute(data$time) / 60

# cyclical effect of time of day
tpm_formula <- ~ cos(2 * pi * tod / 24) + sin(2 * pi * tod / 24)

# fit model with covariates
mod2 <- hmmSSF(ssf_formula = ssf_formula,
               tpm_formula = tpm_formula,
               n_states = 2,
               data = data,
               ssf_par0 = ssf_par0)
```

Printing the model now outputs the estimated covariate effects for the transition probabilities. We

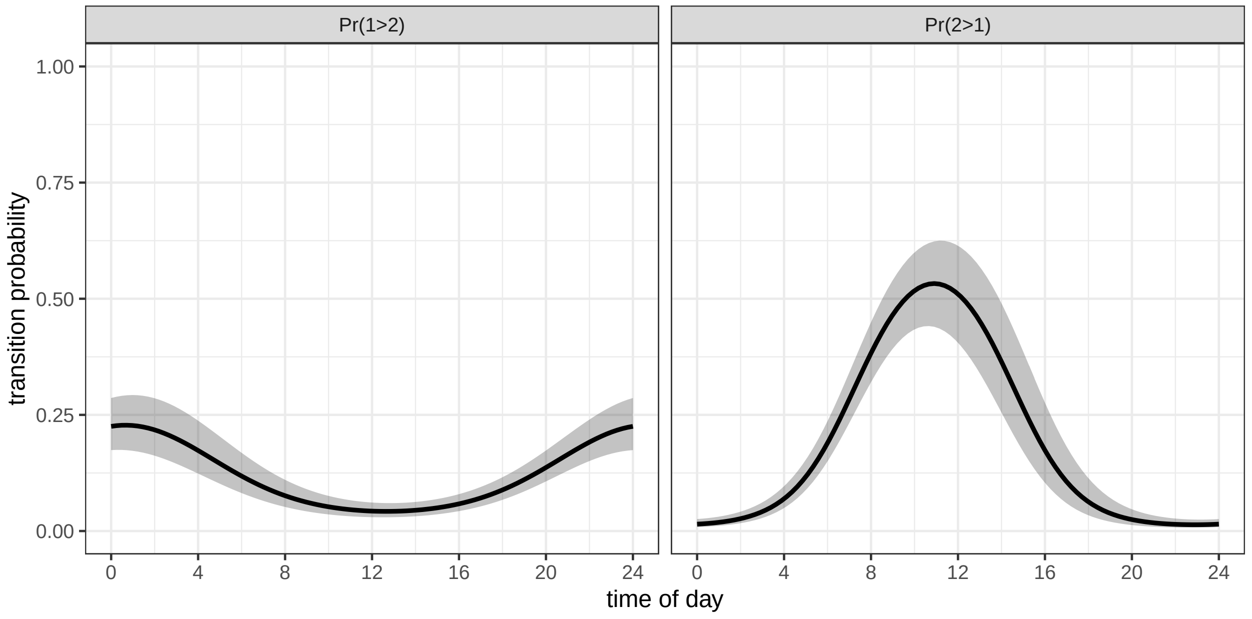
```
# print parameter estimates
mod2
```

```
Negative log-likelihood: -1454.32
Convergence code: 0

SSF model:
~step + log(step) + cos(angle) + cov1
                  mle    low    upp
step.S1        -3.574 -3.902 -3.246
log(step).S1   -0.991 -1.058 -0.923
cos(angle).S1   0.185  0.108  0.262
cov1.S1         3.269  1.357  5.181
step.S2        -1.716 -1.886 -1.546
log(step).S2    1.399  1.076  1.722
cos(angle).S2   4.273  3.906  4.640
cov1.S2         0.485 -0.773  1.742

TPM model:
~cos(2 * pi * tod/24) + sin(2 * pi * tod/24)
```

```
                            mle    low    upp
(Intercept).1>2          -2.170 -2.380 -1.961
cos(2 * pi * tod/24).1>2  0.936  0.655  1.216
sin(2 * pi * tod/24).1>2  0.162 -0.130  0.454
(Intercept).2>1          -2.077 -2.397 -1.758
cos(2 * pi * tod/24).2>1 -2.118 -2.501 -1.735
sin(2 * pi * tod/24).2>1  0.625  0.240  1.009
```

We can predict the transition probabilities over a grid of times of day, to plot the effect. The function `predict_tpm()` returns a list with elements `mle` (point estimates), `upper` (upper bound of 95% confidence interval), and `lower` (lower bound of interval). We use these predictions to create a plot of the transition probabilities as functions of the time of day, with 95% (pointwise) confidence bands.

```
# predict transition probabilities
new_data <- data.frame(tod = seq(0, 24, length = 100))
tpm <- predict_tpm(mod = mod2, new_data = new_data)

# create data frame for plotting
df <- data.frame(tod = new_data$tod,
                 mle = c(tpm$mle[1,2,], tpm$mle[2,1,]),
                 low = c(tpm$lower[1,2,], tpm$lower[2,1,]),
                 upp = c(tpm$upper[1,2,], tpm$upper[2,1,]),
                 name = rep(c("Pr(1>2)", "Pr(2>1)"), each = nrow(new_data)))

# plot transition probabilities against time of day
ggplot(df, aes(tod, mle)) +
  geom_ribbon(aes(ymin = low, ymax = upp), alpha = 0.3) +
  geom_line(linewidth = 1) +
  facet_wrap("name") +
  labs(x = "time of day", y = "transition probability") +
  scale_x_continuous(breaks = (0:6)*4) +
  scale_y_continuous(limits = c(0, 1))
```



The results indicate that the animal was most likely to switch from fast to slow movement in the middle of the day, and from slow to fast during the night.

References

Tal Avgar, Jonathan R Potts, Mark A Lewis, and Mark S Boyce. Integrated step selection analysis: bridging the gap between resource selection and animal movement. *Methods in Ecology and Evolution*, 7(5):619–630, 2016.

James D Forester, Hae Kyung Im, and Paul J Rathouz. Accounting for animal movement in estimation of resource selection functions: sampling and data analysis. *Ecology*, 90(12): 3554–3565, 2009.

Natasha Jean Klappstein, Len Thomas, and Theo Michelot. Flexible hidden Markov models for behaviour-dependent habitat selection. *Movement Ecology*, 11(1):1–13, 2023.

Théo Michelot, Roland Langrock, and Toby A Patterson. moveHMM: an R package for the statistical modelling of animal movement data using hidden Markov models. *Methods in Ecology and Evolution*, 7(11):1308–1315, 2016.

Aurélien Nicosia, Thierry Duchesne, Louis-Paul Rivest, and Daniel Fortin. A multi-state conditional logistic regression model for the analysis of animal movement. *Annals of Applied Statistics*, 2017.

Jennifer Pohle, Roland Langrock, Floris M Van Beest, and Niels Martin Schmidt. Selecting the number of states in hidden Markov models: pragmatic solutions illustrated using animal movement. *Journal of Agricultural, Biological and Environmental Statistics*, 22:270–293, 2017.

Johannes Signer, John Fieberg, and Tal Avgar. Animal movement tools (amt): R package for managing tracking data and conducting habitat selection analyses. *Ecology and evolution*, 9(2):880–890, 2019.

Alison V Towner, Vianey Leos-Barajas, Roland Langrock, Robert S Schick, Malcolm J Smale, Tami Kaschke, Oliver JD Jewell, and Yannis P Papastamatiou. Sex-specific and individual preferences for hunting strategies in white sharks. *Functional Ecology*, 30(8):1397–1407, 2016.