`momentuHMM`: R package for analysis of telemetry data using generalized multivariate hidden Markov models of animal movement

Brett T. McClintock[1] and Théo Michelot[2]

---

[1]Marine Mammal Laboratory
Alaska Fisheries Science Center
NOAA National Marine Fisheries Service
Seattle, U.S.A.
Email: brett.mcclintock@noaa.gov

[2]School of Mathematics and Statistics
University of St Andrews
St Andrews, U.K.

September 26, 2019

Summary

1. Discrete-time hidden Markov models (HMMs) have become an immensely popular tool for inferring latent animal behaviors from telemetry data, largely because they are relatively fast and easy to implement when data streams are observed without error and at regular time intervals. While movement HMMs typically rely solely on location data, auxiliary biotelemetry and environmental data are powerful and readily-available resources for incorporating much more behavioral realism and inferring ecological relationships that would otherwise be difficult or impossible to infer from location data alone. However, there is a paucity of generalized user-friendly software available for implementing (multivariate) HMMs of animal movement. Furthermore, location measurement error, temporal irregularity, and other forms of missing data are often pervasive in telemetry studies (particularly in marine systems).

2. Here we provide a guide to using an open-source R package, `momentuHMM` version 1.5.0, that addresses many of the deficiencies in existing software. Features for multivariate HMMs in `momentuHMM` (pronounced "momentum") include: 1) tools for data pre-processing and visualization; 2) user-specified probability distributions for an unlimited number of data streams and latent behavior states, such as those based on location (e.g., step length, turning angle) and auxiliary biotelemetry data (e.g., from pressure, conductivity, heart rate, or motion sensors); 3) biased and correlated random walk movement models, including "activity centers" associated with attractive or repulsive forces; 4) user-specified design matrices and constraints for covariate modelling of initial distribution, state transition probability, and probability distribution parameters using linear model formulas familiar to most R users; 5) multiple imputation methods that account for observation error attributable to measurement error and temporally-irregular or missing data; 6) seamless integration of spatio-temporal covariate raster data; 7) cosinor and spline regression formulas for cyclical (e.g., daily, seasonal) and other complicated patterns; 8) discrete individual-level random effects on state transition probabilities; 9) hierarchical hidden Markov models for data streams and/or state switching at multiple time scales; 10) "recharge" models for an aggregated physiological process associated with state switching in heterogeneous environments; 11) model checking and selection; and 12) data simulation capabilities for study design, power analyses and assessing model performance, including simulation of location data subject to movement constraints (e.g. land for marine animals), temporal irregularity,

and/or measurement error.

3. After providing a brief introduction to (multivariate) HMMs for telemetry data, we demonstrate some of the capabilities of `momentuHMM` using real-world examples. This brief tutorial includes workflows for data formatting, model specification, model fitting, and diagnostics.

4. While many of the features of `momentuHMM` were motivated by animal movement data, the package can be used for analyzing any type of data that is amenable to (multivariate) HMMs. Practitioners interested in additional features for `momentuHMM` are encouraged to contact the authors.

Key-words animal biotelemetry, biologging, `crawl`, `moveHMM`, state-space model, state-switching

Contents

1 Introduction

Discrete-time hidden Markov models (HMMs) have become immensely popular for the analysis of animal telemetry data (e.g. Morales *et al.* 2004; Jonsen *et al.* 2005; Langrock *et al.* 2012; McClintock *et al.* 2012). In short, an HMM is a time series model composed of a (possibly multivariate) observation process $(\mathbf{Z}_1, \ldots, \mathbf{Z}_T)$, in which each data stream is generated by N state-dependent probability distributions, and where the unobservable (hidden) state sequence $(S_t \in \{1, \ldots, N\}, t = 1, \ldots, T)$ is assumed to be a Markov chain. The state sequence of the Markov chain is governed by (typically first-order) state transition probabilities, $\gamma_{ij}^{(t)} = \Pr(S_{t+1} = j \mid S_t = i)$ for $i, j = 1, \ldots, N$, and an initial distribution $\boldsymbol{\delta}^{(0)}$. The likelihood of an HMM can be succinctly expressed using the forward algorithm:

$$\mathcal{L} = \boldsymbol{\delta}^{(0)} \boldsymbol{\Gamma}^{(1)} \mathbf{P}(\mathbf{z}_1) \boldsymbol{\Gamma}^{(2)} \mathbf{P}(\mathbf{z}_2) \boldsymbol{\Gamma}^{(3)} \cdots \boldsymbol{\Gamma}^{(T-1)} \mathbf{P}(\mathbf{z}_{T-1}) \boldsymbol{\Gamma}^{(T)} \mathbf{P}(\mathbf{z}_T) \mathbf{1}^N, \tag{1}$$

where $\boldsymbol{\Gamma}^{(t)} = \left(\gamma_{ij}^{(t)}\right)$ is the $N \times N$ transition probability matrix, $\mathbf{P}(\mathbf{z}_t) = \text{diag}(p_1(\mathbf{z}_t), \ldots, p_N(\mathbf{z}_t))$, $p_s(\mathbf{z}_t)$ is the conditional probability density of $\mathbf{Z}_t$ given $S_t = s$, and $\mathbf{1}^N$ is a N-vector of ones (for a thorough introduction to HMMs see Zucchini *et al.* 2016).

One of the most common discrete-time animal movement HMMs for telemetry loca-

tion data is composed of two data streams, step length and turning angle (or bearing), which are calculated for each of the T time steps from the temporally-regular observations of an animal's position, (x_t, y_t), for $t = 1, \ldots, T+1$ (e.g. Morales *et al.* 2004; Langrock *et al.* 2012; McClintock *et al.* 2012). Step length (l_t) is typically calculated as the Euclidean distance between the locations (x_t, y_t) and (x_{t+1}, y_{t+1}), while turning angle (ϕ_t) is calculated as the change in bearing ($b_t = \text{atan2}(y_{t+1} - y_t, x_{t+1} - x_t)$) between the intervals $[t-1, t]$ and $[t, t+1]$ (e.g. $\phi_t = 0$ if $b_{t-1} = b_t$). For this HMM composed of 2 data streams, $\mathbf{z}_t = (l_t, \phi_t)$, and, conditional on the latent state S_t, independent probability distributions are typically assumed for each stream; that is, $p_s(\mathbf{z}_t) = p_s(l_t)p_s(\phi_t)$. Some common probability distributions for the step length data stream are the gamma or Weibull distributions, while the wrapped Cauchy or von Mises distributions are often employed for turning angle or bearing. For a fitted HMM, the Viterbi algorithm is used to compute the most likely sequence of underlying states (Zucchini *et al.* 2016). In movement HMMs, the states are often considered as proxies for animal behaviour.

While HMMs for animal movement based solely on location data are somewhat limited in the number and type of biologically-meaningful movement behavior states they are able to accurately identify, advances in biologging technology are now allowing the collection of valuable auxiliary biotelemetry data (e.g., dive activity, accelerometer, heart rate, stomach temperature), which, when combined with location data, allow for multivariate HMMs that can incorporate much more behavioral realism and facilitate inferences about complex ecological relationships that would otherwise be difficult or impossible to infer from location data alone (e.g. McClintock *et al.* 2013; DeRuiter *et al.* 2017; McClintock *et al.* 2017). Multivariate HMMs that utilize both location and auxiliary biotelemetry data can facilitate the identification of additional states that go beyond the $N = 2$ state approaches that are most frequently used by practitioners. For example, the most widely used 2-state HMMs for animal movement include "encamped" (or "foraging") and "exploratory" (or "transit") states characterized by area-restricted-search-type movements (shorter step lengths with little to no directional persistence) and migratory-type movements (longer step lengths with high directional persistence), respectively (Morales *et al.* 2004; Jonsen *et al.* 2005). However, very different behaviors can exhibit similar horizontal trajectories. For example, for herbivores such as North American elk (Morales *et al.* 2004) or central-place foragers such as harbour seals (McClintock *et al.* 2013), the horizontal trajectories of "resting" and "foraging"

movements can be very difficult to distinguish. Standard 2-state HMMs based solely on horizontal trajectory will tend to lump these behaviors together, and this could have unintended consequences if, for example, one intends to use the estimated state sequences to identify foraging habitat. In order to tweeze apart distinct behaviors with similar horizontal trajectories, additional states can be informed by auxiliary information (such as mandible accelerometer or dive data), incorporated as additional data stream(s) in a multivariate HMM.

When data streams are observed without error and at regular time intervals, a major advantage of HMMs is the relatively fast and efficient maximization of the likelihood using the forward algorithm (Eq. 1). However, location measurement error is rarely non-existent in animal-borne telemetry studies and depends on both the device and the system under study. For example, GPS errors are typically less than 50m, but Argos errors can exceed 10km (e.g. Costa *et al.* 2010). An extreme case of missing data can arise when location data are obtained with little or no temporal regularity, as in many marine mammal telemetry studies (e.g. Jonsen *et al.* 2005), such that few (if any) observations align with the regular time steps required by discrete-time HMMs. When explicitly accounting for uncertainty attributable to location measurement error, temporally-irregular observations, or other forms of missing data, one must typically fit (multivariate) HMMs using computationally-intensive (and often time-consuming) model fitting techniques such as Markov chain Monte Carlo (Jonsen *et al.* 2005; McClintock *et al.* 2012). However, complex analyses requiring novel statistical methods and custom model-fitting algorithms are not practical for many practitioners.

While statisticians have been applying HMMs to telemetry data for decades, R (R Core Team 2017) packages such as `bsam` (Jonsen *et al.* 2005), `moveHMM` (Michelot *et al.* 2016), and `swim` (Whoriskey *et al.* 2017) have recently helped make these models of animal movement behavior more accessible to the practitioners that are actually conducting telemetry studies. These advances represent important steps toward making HMMs of animal movement more accessible, but the models that can currently be implemented using existing software remain limited in many key respects. For example, existing HMM software for animal movement is limited to two data streams based solely on location data (e.g. step length and turning angle), and while `moveHMM` allows for a user-specified number of latent behavioral states (`bsam` and `swim` are limited to $N = 2$ states), it is typically difficult to identify >2 biologically-meaningful

behavior states from only 2 data streams (e.g. Morales *et al.* 2004; Beyer *et al.* 2013; McClintock *et al.* 2014). Both `moveHMM` and `swim` are designed for temporally-regular (or linearly-interpolated) location data with negligible measurement error, but the realities of animal-borne telemetry often yield temporally-irregular location data subject to error (particularly in aquatic environments). Other notable deficiencies of existing software include limited abilities to incorporate spatio-temporal environmental or individual covariates on parameters, biased (or directed) movements in response to attractive or repulsive forces (e.g. McClintock *et al.* 2012; Langrock *et al.* 2014), cyclical (e.g. daily, seasonal) and other more complicated behavioral patterns, or constraints on parameters.

To address these deficiencies in existing software, we developed a user-friendly R package, `momentuHMM` (Maximum likelihood analysis Of animal MovemENT behavior Using multivariate Hidden Markov Models), intended for practitioners wishing to implement more flexible and realistic (multivariate) HMM analyses of animal movement while accounting for common challenges associated with telemetry data (McClintock & Michelot 2018). Features for multivariate HMM analyses in `momentuHMM` include: 1) tools for data pre-processing and visualization; 2) user-specified probability distributions for an unlimited number of data streams and latent behavior states; 3) biased and correlated random walk movement models, including "activity centers" associated with attractive or repulsive forces (e.g. McClintock *et al.* 2012); 4) user-specified design matrices and constraints for covariate modelling of state transition probability and probability distribution parameters using linear model formulas familiar to most R users; 5) multiple imputation methods that account for observation error attributable to measurement error and temporally-irregular or missing data (Hooten *et al.* 2017; McClintock 2017); 6) seamless integration of spatio-temporal environmental covariate data (e.g., wind direction, forest cover, sea ice concentration) using the `raster` package (Hijmans 2016b); 7) cosinor (e.g. Cornelissen 2014) and spline regression formulas for cyclical and other complicated behavioral patterns; 8) discrete individual-level random effects on state transition probabilities (e.g. DeRuiter *et al.* 2017); 9) hierarchical hidden Markov models (e.g. Leos-Barajas *et al.* 2017; Adam *et al.* 2019) for data streams and/or state switching at multiple time scales; 10) "recharge" models for an aggregated physiological process associated with state switching in heterogeneous environments (Hooten *et al.* 2019); 11) model checking and selection; and 12) data simulation capabilities for

study design, power analyses and assessing model performance, including simulation of location data subject to movement constraints (e.g. land for marine animals), temporal irregularity, and/or measurement error.

In the following tutorial, we demonstrate some of the capabilities of `momentuHMM` using real-world examples, including an example of periodic cycles in African elephant movement, a 3-state ("resting", "foraging", "transit") northern fur seal example incorporating auxiliary dive activity data (McClintock *et al.* 2014), a loggerhead turtle example relating "foraging" and "transit" movements to ocean surface currents, a 5-state grey seal example incorporating biased movements toward haul-out and foraging locations (McClintock *et al.* 2012), a 4-state ("outbound", "searching", "foraging", "inbound") southern elephant seal example with biased movements toward and away from a colony (Michelot *et al.* 2017), a 3-state ("resting", "foraging", "transit") harbour seal example using population-level constraints on movement parameters (McClintock *et al.* 2013), a 6-state northern fulmar example incorporating biased movements relative to both static (i.e. colony) and dynamic (i.e. fishing vessels) activity centers (Pirotta *et al.* 2018), a 4-state long-finned pilot whale example including individual-level random effects on state transition probabilities (Isojunno *et al.* 2017), and hierarchical HMMs fitted to harbor porpoise, garter snake, Atlantic cod, and horn shark data (Leos-Barajas *et al.* 2017; Adam *et al.* 2019), and a recharge dynamics model for African buffalo movements in a heterogeneous environment (Hooten *et al.* 2019). Using simulated data, we also demonstrate how the group dynamic model of Langrock *et al.* (2014) can be implemented using `momentuHMM`. Finally, we demonstrate how to simulate movement subject to barriers or other constraints (e.g. land for marine animals) using potential functions (e.g. Brillinger *et al.* 2012). This brief tutorial includes workflows for data formatting, model specification, model fitting, and diagnostics. While many of the features of `momentuHMM` were motivated by animal movement data, the package can be used for analyzing any type of data that is amenable to (multivariate) HMMs. Additional information, including help files, data, examples, and package usage is available by downloading the `momentuHMM` package from CRAN (`https://cran.r-project.org`) or Github (`https://github.com/bmcclintock/momentuHMM`). This article describes `momentuHMM` version 1.5.0.

Table 1. *Workhorse functions for the R package momentuHMM.*

Function	Description
`crawlMerge`	Merge `crawlWrap` output with additional data streams or covariates
`crawlWrap`	Fit `crawl` models and predict temporally-regular locations
`fitHMM`	Fit a (multivariate) HMM to the data
`MIfitHMM`	Fit (multivariate) HMMs to multiple imputation data
`MIpool`	Pool `momentuHMM` model results across multiple imputations
`plot.crwData`	Plot `crawlWrap` output
`plot.miSum`	Plot summaries of multiple imputation `momentuHMM` models
`plot.momentuHMM`	Plot summaries of `momentuHMM` models
`plot.momentuHMMData`	Plot summaries of selected data streams and covariates
`plotPR`	Plot time series, qq-plots and sample ACFs of pseudo-residuals
`plotSat`	Plot locations on satellite image
`plotSpatialCov`	Plot locations on raster image
`plotStates`	Plot the (Viterbi-)decoded states and state probabilities
`plotStationary`	Plot stationary state probabilities
`prepData`	Pre-process data streams and covariates
`pseudoRes`	Calculate pseudo-residuals for `momentuHMM` models
`simData`	Simulate data from a (multivariate) HMM
`simHierData`	Simulate data from a (multivariate) hierarchical HMM
`stateProbs`	State probabilities for each time step
`viterbi`	Most likely state sequence (using the Viterbi algorithm)

2 momentuHMM overview

Before delving into some of the finer details, we will first provide an overview of the main features and functions of `momentuHMM` (pronounced "momentum"). While space is limited in this tutorial, further details on implementation can be found in the package's documentation and vignette. The workhorse functions of `momentuHMM` are listed in Table 1. Usage of several of these functions (e.g. `fitHMM`, `prepData`, `simData`) is deliberately very similar to equivalent functions in `moveHMM` (Michelot *et al.* 2016) , but the `momentuHMM` arguments for these functions have been generalized and expanded to accommodate a more flexible framework for data pre-processing, model specification, parameterization, and simulation. R users already familiar with `moveHMM` will therefore likely find it easy to immediately begin using `momentuHMM`.

One of the key features of `momentuHMM` is the ability to include an unlimited number of HMM data streams (e.g. step length, turning angle, dive activity, heart rate) arising

from a broad range of commonly used probability distributions (e.g. beta, categorical, gamma, normal, multivariate normal, Poisson, von Mises, Weibull), including (multivariate) normal random walks (section 2.6) that can be particularly useful for modeling positions directly (instead of step lengths and turning angles). Any of the parameters of the probability distributions used for the observed data can be modelled as a function of environmental and individual covariates using link functions (Tables 2 and 3). For any given "natural scale" (or "real scale") probability distribution parameter θ, all of the link functions (g) in `momentuHMM` are of the general form $g(\boldsymbol{\theta}) = \mathbf{X}_\theta \boldsymbol{\beta}_\theta$, where $\mathbf{X}_\theta$ is the $T \times K$ design matrix (composed of K covariates) and $\boldsymbol{\beta}_\theta$ is the correponding K-vector of "working scale" (or "beta scale") parameters for θ. For example, suppose step length is assumed to have a gamma distribution, $l_t \mid S_t = s \sim \text{gamma}(\mu_s, \sigma_s)$. In `momentuHMM`, the natural scale parameters for the gamma distribution are the (state-dependent) step length mean ($\mu_s > 0$) and standard deviation ($\sigma_s > 0$). Because both of these parameters must be positive, the log link function is a natural choice for modelling these parameters as a function of covariates, e.g., $\log(\boldsymbol{\mu}) = \mathbf{X}_\mu \boldsymbol{\beta}_\mu$ and $\log(\boldsymbol{\sigma}) = \mathbf{X}_\sigma \boldsymbol{\beta}_\sigma$.

The state transition probabilities ($\boldsymbol{\Gamma}^{(t)}$) and initial distribution ($\boldsymbol{\delta}^{(0)}$) can also be modelled as functions of covariates, using a multinomial logit link, as described e.g. by Michelot *et al.* (2016). Permissable R classes for covariates include `numeric`, `integer`, or `factor`. Factors can be particularly useful for specifying models with individual- or group-level (e.g. sex or age class) effects on state transition and probability distribution parameters. Spatio-temporal covariates can also be of classes `rasterLayer`, `rasterStack`, or `rasterBrick` (Hijmans 2016b), in which case `momentuHMM` automatically extracts the appropriate covariate values from the raster based on the time and location of each observation (see example in section 3.3).

2.1 Data preparation and visualization

For temporally-regular location data with negligible measurement error, the `prepData` function is used to create a `momentuHMMData` object that can be used for data visualization and further analysis. The arguments for `prepData` include:

- `data` A data frame with $T + 1$ rows including optionally a field 'ID' (identifiers for different individuals), coordinates from which step length ('step') and turning angle ('angle') data streams are to be calculated, any additional data streams,

Table 2. *Univariate data stream (z) probability distributions, natural parameters, and default link functions for covariate modelling. Probability distributions with positive support can be zero-inflated (with additional zero-mass parameters), while the beta distribution can be zero- and/or one-inflated (with additional one-mass parameters). If user-specified bounds are provided, then custom link functions are used instead of the defaults (see package documentation for further details). If both zero- and one-inflation are included, then a multinomial logit (mlogit) link is used because these probabilities must sum to less than one (in this case, any user-specified bounds for the zero- and one-inflation parameters are ignored). If circular-circular regression is specified for the mean of angular distributions ("vm" and "wrpcauchy"), then a link function based on Rivest et al. (2016) is used. The von Mises consensus distribution ("vmConsensus") is a von Mises circular-circular regression model where the concentration parameter depends on the level of agreement among short-term directional persistence and angular covariates. Users seeking additional univariate probability distributions are encouraged to contact the authors.*

Distribution	Support	Parameters	Link function[1]
Bernoulli ("`bern`")	$z_t \in \{0, 1\}$	`prob` $\in (0, 1)$	logit
Beta ("`beta`")	$z_t \in (0, 1)$	`shape1` > 0	log
		`shape2` > 0	log
		`zero-mass` $\in (0, 1)$	logit
		`one-mass` $\in (0, 1)$	logit
Categorical ("`cat`")	$z_t \in \{1, \ldots, k\}$	$\texttt{prob}_1, \ldots, \texttt{prob}_{k-1} \in (0, 1)$	mlogit
Exponential ("`exp`")	$z_t > 0$	`rate` > 0	log
		`zero-mass` $\in (0, 1)$	logit
Gamma ("`gamma`")	$z_t > 0$	`mean` > 0	log
		`sd` > 0	log
		`zero-mass` $\in (0, 1)$	logit
Log normal ("`lnorm`")	$z_t > 0$	`location` $\in \mathbb{R}$	identity
		`scale` > 0	log
		`zero-mass` $\in (0, 1)$	logit
Normal ("`norm`")	$z_t \in \mathbb{R}$	`mean` $\in \mathbb{R}$	identity
		`sd` > 0	log
Normal random walk ("`rw_norm`")	$z_t \in \mathbb{R}$	`mean` $\in \mathbb{R}$	identity
		`sd` > 0	log
Poisson ("`pois`")	$z_t \in \{0, 1, \ldots\}$	`lambda` > 0	log
Von Mises ("`vm`")	$z_t \in (-\pi, \pi]$	`mean` $\in (-\pi, \pi]$	tan(`mean`/2)
		`concentration` > 0	log
Von Mises ("`vmConsensus`")	$z_t \in (-\pi, \pi]$	`mean` $\in (-\pi, \pi]$	Rivest *et al.*
		`kappa` > 0	log
Weibull ("`weibull`")	$z_t > 0$	`shape` > 0	log
		`scale` > 0	log
		`zero-mass` $\in (0, 1)$	logit
Wrapped Cauchy ("`wrpcauchy`")	$z_t \in (-\pi, \pi]$	`mean` $\in (-\pi, \pi]$	tan(`mean`/2)
		`concentration` $\in (0, 1)$	logit

[1]Link functions (g) relate natural scale parameters ($\boldsymbol{\theta}$) to a $T \times K$ design matrix ($\mathbf{X}$) and $K-$vector of working scale parameters ($\boldsymbol{\beta} \in \mathbb{R}^K$) such that $g(\boldsymbol{\theta}) = \mathbf{X}\boldsymbol{\beta}$.

Table 3. *Multivariate data stream* ($\mathbf{z}$) *probability distributions, natural parameters, and default link functions for covariate modelling. If user-specified bounds are provided, then custom link functions are used instead of the defaults (see package documentation for further details). Users seeking additional multivariate probability distributions are encouraged to contact the authors.*

Distribution	Support	Parameters	Link function[2]
Bivariate normal ("`mvnorm2`")	$\mathbf{z}_t \in \mathbb{R}^2$	`mean.x` $\in \mathbb{R}$	identity
		`mean.y` $\in \mathbb{R}$	identity
		`sigma.x` > 0	log
		`sigma.xy` $\in \mathbb{R}$	identity
		`sigma.y` > 0	log
Bivariate normal random walk ("`rw_mvnorm2`")	$\mathbf{z}_t \in \mathbb{R}^2$	`mean.x` $\in \mathbb{R}$	identity
		`mean.y` $\in \mathbb{R}$	identity
		`sigma.x` > 0	log
		`sigma.xy` $\in \mathbb{R}$	identity
		`sigma.y` > 0	log
Trivariate normal ("`mvnorm3`")	$\mathbf{z}_t \in \mathbb{R}^3$	`mean.x` $\in \mathbb{R}$	identity
		`mean.y` $\in \mathbb{R}$	identity
		`mean.z` $\in \mathbb{R}$	identity
		`sigma.x` > 0	log
		`sigma.xy` $\in \mathbb{R}$	identity
		`sigma.xz` $\in \mathbb{R}$	identity
		`sigma.y` > 0	log
		`sigma.yz` $\in \mathbb{R}$	identity
		`sigma.z` > 0	log
Trivariate normal random walk ("`rw_mvnorm3`")	$\mathbf{z}_t \in \mathbb{R}^3$	`mean.x` $\in \mathbb{R}$	identity
		`mean.y` $\in \mathbb{R}$	identity
		`mean.z` $\in \mathbb{R}$	identity
		`sigma.x` > 0	log
		`sigma.xy` $\in \mathbb{R}$	identity
		`sigma.xz` $\in \mathbb{R}$	identity
		`sigma.y` > 0	log
		`sigma.yz` $\in \mathbb{R}$	identity
		`sigma.z` > 0	log

[1]Link functions (g) relate natural scale parameters ($\boldsymbol{\theta}$) to a $T \times K$ design matrix ($\mathbf{X}$) and $K-$vector of working scale parameters ($\boldsymbol{\beta} \in \mathbb{R}^K$) such that $g(\boldsymbol{\theta}) = \mathbf{X}\boldsymbol{\beta}$.

and any covariates identified in the `covNames` and `angleCovs` arguments. Alternatively, `data` can be a `crwData` object returned by `crawlWrap`.

- `type` Coordinate type; 'UTM' if easting-northing or 'LL' if longitude-latitude.
- `coordNames` Names of the two coordinate columns in `data`. If `coordNames=NULL` then step lengths, turning angles, and any location-based covariates (i.e., those specified by `spatialCovs`, `centers`, `centroids`, and `angleCovs`) are not calculated.
- `covNames` Character vector indicating the names of any covariates in `data`. Any variables in `data` (other than "ID") that are not identified in `covNames` or `angleCovs` are assumed to be data streams.
- `spatialCovs` List of Raster-class objects (Hijmans 2016b) containing spatiotemporally referenced covariates. Covariates specified by `spatialCovs` are extracted from the raster layer(s) based on the location data. Raster stacks may also be included, in which case the appropriate z values (e.g. time, date) must also be included in `data`.
- `centers` 2-column matrix providing the coordinates for any activity centers (e.g., potential centers of attraction or repulsion) from which distance and angle covariates will be calculated based on the location data and returned in the `momentuHMMData` object.
- `centroids` List where each element is a data frame containing the x-coordinates ('x'), y-coordinates ('y'), and times for a centroid (i.e., a dynamic activity center for which the coordinates can change over time) from which distance and angle covariates will be calculated based on the location data and returned in the `momentuHMMData` object.
- `angleCovs` Character vector indicating the names of any circular-circular regression angular covariates in `data` or `spatialCovs` that need conversion from standard direction (in radians relative to the x-axis) to turning angle (relative to previous movement direction).

Summary plots of the `momentuHMMData` object returned by `prepData` can be created for any data stream or covariate using the generic `plot` function.

If location data are temporally-irregular or subject to measurement error, then they are not suitable for `prepData`. In this case, `momentuHMM` can be used to perform a 2-stage multiple imputation approach (McClintock 2017). We discuss this pragmatic approach to incorporating uncertainty attributable to observation error and temporal irreglarity into multivariate HMM analyses in section 2.8.

2.2 HMM specification and fitting

Once a `momentuHMMData` object has been created using `prepData`, then the data are ready to be passed to the generalized multivariate HMM-fitting function `fitHMM`. There are many different options for specifying HMMs using `fitHMM`, so here we will only focus on several of the most important and useful features (further details of all `fitHMM` arguments are in the package documentation). The bare essentials of `fitHMM` include the arguments:

- `data` A `momentuHMMData` object
- `nbStates` Number of latent states (N)
- `dist` A named list indicating the probability distributions of the data streams.
- `estAngleMean` An optional named list indicating whether or not to estimate the angle mean for data streams with angular distributions (e.g. turning angle). If not estimated (the default), the angle mean is fixed to 0.
- `formula` Regression formula for the transition probability covariates
- `stationary` Logical indicating whether or not the initial distribution is considered equal to the stationary distribution (must be `FALSE` if `formula` includes covariates)
- `Par0` A named list containing vectors of starting values for the state-dependent probability distribution parameters of each data stream

These seven arguments are all that are needed in order to fit the HMMs currently supported in `moveHMM` (Michelot *et al.* 2016). For example, here is how the analysis of

15 "wild haggis" tracks described in Michelot *et al.* (2016) would be implemented using `momentuHMM`:

```
library(momentuHMM)
### Load raw data
rawHaggis<-read.csv("rawHaggises.csv")
### Process data
processedHaggis<-prepData(data=rawHaggis,covNames=c("slope","temp"))

### Fit HMM
# initial step distribution natural scale parameters
stepPar0 <- c(1,5,0.5,3) # (mu_1,mu_2,sd_1,sd_2)
# initial angle distribution natural scale parameters
anglePar0 <- c(0,0,1,8) # (mean_1,mean_2,concentration_1,concentration_2)
fitHaggis <- fitHMM(data = processedHaggis, nbStates = 2,
                    dist = list(step = "gamma", angle = "vm"),
                    Par0 = list(step = stepPar0, angle = anglePar0),
                    formula = ~ slope + I(slope^2),
                    estAngleMean = list(angle=TRUE))
```

Note that many of the arguments in `fitHMM` are lists, with each element of the list corresponding to a data stream. The list names provided in `dist`, `Par0`, and `estAngleMean` (e.g. 'step' and 'angle') must therefore have a corresponding column in `data` with the same name. Additional data streams can be added to the model by simply adding the additional elements to these list arguments (see examples in sections 3.2, 3.8, and 3.9).

As seen above, the `formula` argument can include many of the functions and operators commonly used to construct terms in R linear model formulas (e.g. `a*b`, `a:b`, `cos(a)`). The `formulaDelta` argument can be similarly used to specify covariate models for the initial distribution. The `formula` argument can also be used to specify transition probability matrix models that incorporate cyclical patterns (using the `cosinor` special function; see example in section 3.1), splines for explaining other more complicated patterns (e.g., `bs` and `ns` functions in the R base package `splines`), and factor variables (e.g., `formula=~ID` for individual-level effects). By default the `formula` argument applies to all state transition probabilities, but the special functions `state`, `toState`, and `betaCol` allow for state- and parameter-specific formulas to be specified (see examples in sections 3.4 and 3.8). While `betaCol` allows a formula to be specified for a specific

transition (e.g. state $3 \rightarrow 1$), `state` and `toState` allow a formula to be specified for all transitions from (e.g. $3 \rightarrow 1$, $3 \rightarrow 2$) and to (e.g. state $1 \rightarrow 3$, $2 \rightarrow 3$) specific states, respectively. The `betaCons` argument allows for equality constraints among any of the transition probability parameters (e.g. $\gamma_{12}^{(t)} = \gamma_{21}^{(t)}$; see example in section 3.8). Specific state transition probabilities can also be fixed to zero (or any other value) using the `fixPar` argument, which can be useful for incorporating more behavioral realism. For example, `fixPar` can be used to prohibit or enforce switching from one particular state to another (possibly as a function of spatio-temporal covariates).

Similar to the `formula` argument for state transition probability modelling, it is through the `DM` argument of `fitHMM` that models are specified for the state-dependent probability distribution parameters for each data stream. `DM` is a list argument containing an element for each data stream, but each element itself is also a list specifying the design matrix formulas for each parameter. For example, the following fits the exact same wild haggis model as above, but employs a user-specified (intercept-only) design matrix for the step length data stream:

```
stepDM <- list(mean = ~1, sd = ~1)

### Fit HMM using user-specified DM
fitHaggisDM <- fitHMM(data = processedHaggis, nbStates = 2,
                      dist = list(step = "gamma", angle = "vm"),
                      DM = list(step = stepDM),
                      Par0 = list(step = log(stepPar0), angle = anglePar0),
                      formula = ~ slope + I(slope^2),
                      estAngleMean = list(angle=TRUE))
```

Note that when `DM` is specified for a data stream, the initial parameter values (`Par0`) for that data stream now correspond to columns of the resulting design matrix and must be on the working scale instead of the natural scale. In this case, because the log link is used for the natural parameters of the gamma distribution, `Par0$step` was specified on the log scale. The functions `getPar`, `getPar0`, `checkPar0`, and `getParDM` are designed to assist users in the specification of design matrices and corresponding initial values on the working scale for any given model (see package documentation for further details). `DM` formulas are just as flexible as the `formula` argument and, in addition to common linear model formula functions and operators, can also include cyclical cosinor models (see section 3.1), splines, factor variables, and state-specific probability distribution

parameter formulas (see examples in sections 3.3 and 3.4). As with the state transition probabilities, working parameters for probability distributions can also be fixed to user-specified values using the `fixPar` argument.

Specification of design matrices using `DM` is not limited to formulas. Alternatively, "pseudo-design" matrices can be specified, using an R matrix with rows corresponding to the natural parameters and columns corresponding to the working parameters. The elements in the matrix may be numeric or character strings containing model formula terms (see examples in sections 3.4, 3.7, and 3.8). Using a pseudo-design matrix for step length, the following is yet another way to implement the exact same wild haggis model:

```
stepDMp <- matrix(c(1,0,0,0,
                    0,1,0,0,
                    0,0,1,0,
                    0,0,0,1),4,4,byrow=TRUE)
rownames(stepDMp) <- c("mean_1","mean_2","sd_1","sd_2")
colnames(stepDMp) <- c("mean_1:(Intercept)","mean_2:(Intercept)",
                       "sd_1:(Intercept)","sd_2:(Intercept)")

### Fit HMM using user-specified DM
fitHaggisDMp <- fitHMM(data = processedHaggis, nbStates = 2,
                       dist = list(step = "gamma", angle = "vm"),
                       DM = list(step = stepDMp),
                       Par0 = list(step = log(stepPar0), angle = anglePar0),
                       formula = ~ slope + I(slope^2),
                       estAngleMean = list(angle=TRUE))
```

(note that column and row names for pseudo-design matrices are not required but can be useful). Pseudo-design matrices allow for the sharing of common working parameters (such as intercept terms) among natural scale parameters, and this can be used to constrain natural scale parameters (e.g., $\mu_1 \leq \mu_2$) when used in tandem with the `workBounds` argument (see sections 3.2, 3.7, and 3.8). This is particularly useful for preventing state label switching when repeatedly fitting the same HMM using multiple imputation methods (see section 2.8).

2.3 Circular-circular regression model for the angle mean

Another noteworthy `fitHMM` argument, `circularAngleMean`, is a list argument that enables users to specify circular-circular regression models for the mean (μ) parameter of angular distributions, such as the wrapped Cauchy and von Mises, instead of circular-linear models based on the tangent link function (Table 2). When `circularAngleMean` is specified as `TRUE` for any given angular data stream (e.g. turning angle), then a special link function based on Rivest *et al.* (2016) is used:

$$\boldsymbol{\mu} = \text{atan2}(\sin(\mathbf{X}_\mu)\boldsymbol{\beta}_\mu, 1 + \cos(\mathbf{X}_\mu)\boldsymbol{\beta}_\mu), \tag{2}$$

where $\mathbf{X}_\mu$ is a $T \times K$ matrix composed of the turning angles between K angular covariates (e.g., wind direction, sea surface current direction) and the bearing of movement during the previous time step; that is, each element

$$x_{t,k} = \text{atan2}(\sin(r_{t,k} - b_{t-1}), \cos(r_{t,k} - b_{t-1})) \tag{3}$$

for angular covariate $r_{t,k}$ and $k = 1, \ldots, K$ (note that `prepData` and `MIfitHMM` calculate $\mathbf{X}_\mu$ based on the `angleCovs`, `centers`, or `centroids` arguments so users need not bother). Because this link function is designed for turning angles, a turning angle of 0 is provided as the reference angle (hence the "1+" preceeding the cosine term in Eq. 2). Thus as a trade-off between biased and correlated movements, the working parameters ($\boldsymbol{\beta}_\mu$) for the expected turning angle at time t weight the attractive (or repulsive) strengths of the angular covariates relative to directional persistence. When all $\boldsymbol{\beta}_\mu = 0$, the model reduces to a correlated random walk, but an increasingly biased random walk results as $\boldsymbol{\beta}_\mu$ gets larger (or smaller). Alternatively, `circularAngleMean` can be specified as a numeric scalar, where the value specifies the coefficient for the reference angle (i.e., directional persistence) term in Eq. 2. For example, setting `circularAngleMean` to 0 specifies a circular-circular regression model with no directional persistence term (thus specifying a biased random walk instead of a biased correlated random walk; see examples in sections 3.4, 3.5.2, and 3.6). Setting `circularAngleMean` to 1 is equivalent to setting it to `TRUE`, i.e., a circular-circular regression model with a coefficient of 1 for the directional persistence reference angle. Many interesting hypotheses about animal movmement can be addressed using circular-circular regression on movement direction, including the effects of wind, sea surface currents (see example in section 3.3), centers of

attraction or repulsion (see examples in sections 3.4, 3.5, and 3.8), group dynamic models (see example in section 3.6), and dynamic activity centers (see example in section 3.8.

The special function `angleFormula` can be included in `DM` formulas or pseudo-design matrices in order to model the circular-circular regression angle mean as a function of the relative strength (or importance) of angular covariates (Rivest *et al.* 2016):

$$\boldsymbol{\mu} = \text{atan2}((\mathbf{Z}_\mu \circ \sin(\mathbf{X}_\mu))\boldsymbol{\beta}_\mu, 1 + (\mathbf{Z}_\mu \circ \cos(\mathbf{X}_\mu))\boldsymbol{\beta}_\mu), \tag{4}$$

where $\mathbf{Z}_\mu$ is a $T \times K$ matrix of positive real covariates (e.g. wind speed, sea surface current speed) and $\circ$ is the Hadamard (i.e. element-wise) product. The special function `angleFormula` can also be used to specify group- or individual-level effects on the circular-circular regression angle mean coefficients ($\boldsymbol{\beta}_\mu$).

Also based on Rivest *et al.* (2016), the von Mises consensus distribution is a special von Mises circular-circular regression model where the concentration parameter (ρ) depends on the level of agreement among short-term directional persistence (i.e. moving forward) and the angular covariates:

$$\boldsymbol{\rho} = \kappa\sqrt{\left[(\mathbf{Z}_\mu \circ \sin(\mathbf{X}_\mu))\boldsymbol{\beta}_\mu\right]^2 + \left[1 + (\mathbf{Z}_\mu \circ \cos(\mathbf{X}_\mu))\boldsymbol{\beta}_\mu\right]^2}. \tag{5}$$

Note that the von Mises consensus distribution is parameterized in terms of μ and κ (see Table 2), but `momentuHMM` returns and plots real parameter estimates in terms of μ and ρ. When all $\boldsymbol{\beta}_\mu$ are non-negative, then the minimum and maximum values for ρ are $\kappa|1 - \min(\mathbf{Z}_\mu\boldsymbol{\beta}_\mu)|$ and $\kappa\left[1 + \max(\mathbf{Z}_\mu\boldsymbol{\beta}_\mu)\right]$, respectively. In the consensus model, κ can be interpreted as the concentration towards a turning angle of zero (i.e. moving forward) when the angular covariate components perfectly cancel out. See section 3.3 for example code using `angleFormula` and the von Mises consensus ("vmConsensus") distribution.

2.4 Individual-level random effects

HMM applications often assume the initial distribution and state transition probability matrix is the same for all individuals (i.e. "complete pooling" of the individuals' time series). But in reality, individuals often do not exhibit the same state-switching dynamics and there is individual-level variation. Individual heterogeneity can often be

well explained by covariates (e.g., sex, age class) and included in `formula`, but it is not always possible to identify (and/or measure) all of the important covariates that drive this variation. One option is to include separate state-switching dynamics for each individual (i.e. "no pooling") by specifying `formulaDelta = ~ID` and `formula = ~ID`, but this "fixed" effect approach can result in many additional parameters to estimate (it also doesn't explain very much about potential factors driving individual heterogeneity). Alternatively, generic individual heterogeneity in state-switching dynamics can be modeled as a "random" effect. While continuous-valued individual-level random effects can be computationally demanding, discrete-valued random effects are more computationally feasible and can be effective in "mopping up" individual heterogeneity in the initial distribution and state transition probabilities that is not explained by measurable covariates. Discrete-valued random effects have recently been used in HHMs of animal movement (e.g. McKellar *et al.* 2014; Towner *et al.* 2016; DeRuiter *et al.* 2017; Isojunno *et al.* 2017), and these "mixed" HMMs can be fitted with `fitHMM` (or `MIfitHMM`) through the `mixtures` and `formulaPi` arguments. The `mixtures` argument specifies the number of mixtures (K) in the model, where each mixture represents a possible initial distribution and transition probability matrix, and each individual time series is assumed to be driven by exactly one of these mixtures. For K mixtures, the mixture weight $(\pi_k; k = 1, \ldots, K)$ is the probability that the kth mixture underlies the state-switching dynamics for a given individual, and a model formula for $\boldsymbol{\pi}$ can be specified using the `formulaPi` argument. For example, Towner *et al.* (2016) found support for $K = 3$ mixtures and a sex covariate on $\boldsymbol{\pi}$ in their HMM for white shark movement, indicating that each of the three possible state-switching dynamics were exhibited differently for males and females; the random effects component of their model would be specified in `fitHMM` (or `MIfitHMM`) by simply setting `mixtures = 3` and `formulaPi = ~sex`. Note that because $\sum_{k=1}^{K} \pi_k = 1$, `momentuHMM` uses a multinomial logit link function for $\boldsymbol{\pi}$ when covariates are included in `formulaPi`. We demonstrate how to fit discrete-valued individual-level random effects on the initial distribution and state transition probabilities using the long-finned pilot whale example from Isojunno *et al.* (2017) in section 3.9.

2.5 Hierarchical hidden Markov models

Hierarchical hidden Markov models (HHMMs; see Leos-Barajas *et al.* 2017; Adam *et al.* 2019) can also be fitted in `momentuHMM`. HMMs with hierarchical structures allow for data streams and/or state transitions to occur at multiple regular time scales. For example, biotelemetry data are often collected at different time scales (e.g. 1-hr intervals for one data stream and 1-min intervals for another data stream) or state transitions can be governed by both larger- and finer-scale behavioral processes. HHMMs are integrated into the workhorse functions of `momentuHMM` and are specified via hierarchically-structured arguments for the data stream probability distributions (`hierDist`), behavioral states (`hierStates`), state transition probabilities (`hierFormula`, `hierBeta`), and initial distributions (`hierFormulaDelta`, `hierDelta`) using the `data.tree` package (Glur 2018). We demonstrate how the HHMM harbor porpoise and garter snake examples from Leos-Barajas *et al.* (2017) and the Atlantic cod and horn shark examples from Adam *et al.* (2019) can be fitted using `momentuHMM` in section 3.10.

2.6 Random walk probability distributions

`momentuHMM` includes several normal random walk probability distributions that can be specified in the `dist` argument (see Tables 2 and 3), including univariate (e.g. for modeling depths), bivariate (e.g. for modeling 2-D positions), and trivariate (e.g. for modeling 3-D positions) normal random walks. These can be particularly useful for modeling movement on positions directly instead of steps and turns. A random walk model assumes position at time t is a function of the position at time $t-1$; in its simplest form without any covariates, we have $x_t \sim N(x_{t-1}, \sigma^2)$ for the univariate case.

Multivariate normal distributions require some additonal book-keeping when preparing the data; the `altCoordNames` argument in `prepData` and `MIfitHMM` and the `mvnCoords` argument in `fitHMM` and `MIfitHMM` are designed to help properly format and identify multivariate coordinate data streams. For example, if a bivariate normal data stream name is "loc" (e.g. `dist=list(loc="mvnorm2")`), then the data must include columns "loc.x" and "loc.y" for the x- and y- coordinates, respectively. When using a multivariate normal random walk distribution, the previous position can be referenced in `DM` formulas or pseudo-design matrices. For example, for a bivariate normal random walk data stream named "mu" (e.g. `dist=list(mu="rw_mvnorm2")`), the previous position

can be refereced in `DM` as "mu.x_tm1" and "mu.y_tm1". This allows for persistence in velocity to be included via the special formula function `crw(x_tm1,lag)`, where argument `x_tm1` is the previous position (e.g. "mu.x_tm1" or "mu.y_tm1") and argument `lag` specifies the time lag for the persistence.

We demonstrate use of the bivariate normal random walk model for loggerhead turtle movements relative to ocean surface currents in section 3.3 and for African buffalo recharge dynamics in section 3.11. We also demonstrate how to simulate movement subject to barriers or other constraints (e.g. land for marine animals) using a bivariate normal random walk in section 3.12.

2.7 Recharge dynamics

Hooten *et al.* (2019) describe a novel way of modeling animal movement behavior based on an aggregated physiological process associated with decision making and movement in heterogeneous environments. In essence, their "recharge" model allows state switching to be a function of this process (i.e. the recharge function). For example, we can think of the recharge function as the gas tank of our car. When the gas tank is full, we are more-or-less free to drive wherever we want. However, when the tank gets low, we must eventually return to the same gas station (or find a new one) to refill our tank. In its simplest form, the recharge model associates "good" habitat with recharging (i.e. filling the tank) and less-suitable habitat with discharging (i.e. emptying the tank). The recharge function thus increases and decreases over time depending on the decision-making process of the individual, the resulting behavior, and the habitat conditions it encounters. By simply imbedding a recharge function into state transition probabilities, we can therefore begin to investigate models with an explicit, mechanistic connection to physiological dynamics! Hooten *et al.* (2019) formulated their recharge model in continuous time, but its discrete-time analogue can be implemented in `momentuHMM`. This is accomplished by including the `recharge(g0, theta)` special function in the transition probability matrix `formula`, where the arguments `g0` and `theta` are formulas for the initial recharge function at time $t = 0$ (g_0) and the recharge function coefficients ($\boldsymbol{\theta}$), respectively. For example, if one were to specify `formula = ~recharge(g0 = ~1, theta = ~cov1+cov2)` for a 2-state (e.g., state 1 = "charged" and state 2 = "discharged") model, the recharge function at time t (g_t) would

be:

$$g_t = g_0 + \sum_{j=1}^{t} \theta_0 + \text{cov1}_j\theta_1 + \text{cov2}_j\theta_2,$$

where cov1_j and cov2_j are the corresponding habitat covariate values for the individual's location at time j. We demonstrate how to fit a discrete-time version of the African buffalo example from Hooten *et al.* (2019) in section 2.7.

2.8 Multiple imputation

When location data are temporally-irregular or subject to measurement error, then they are not suitable for standard maximum-likelihood HMM analyses based on the forward algorithm (Eq. 1). In this case, `momentuHMM` can be used to perform the 2-stage multiple imputation approach of McClintock (2017). The basic concept is to first employ a single-state (i.e., $N = 1$) movement model that is relatively easy to fit but can accommodate location measurement error and temporally-irregular or missing observations (e.g. Johnson *et al.* 2008). The second stage involves repeatedly fitting the desired HMM to m temporally-regular realizations of the position process drawn from the model output of the first stage. Data streams or covariates that are dependent on location (e.g., step length, turning angle, habitat type, snow depth, sea surface temperature) will of course vary among the m realizations of the position process, and the pooled inferences across the HMM analyses therefore reflect location uncertainty.

There are three primary functions (`MIfitHMM`, `MIpool`, and `crawlWrap`) for performing multiple imputation HMM analyses in `momentuHMM`, and all rely on parallel processing to speed up computations. `crawlWrap` is a wrapper function for fitting the continuous-time correlated random walk (CTCRW) model of Johnson *et al.* (2008) to one or more tracks (subject to location measurement error and/or temporal irregularity) and then predicting temporally-regular tracks of the user's choosing (e.g. 15 min, hourly, daily) based on the CTCRW model output. `crawlWrap` returns a `crwData` object that can be used to draw m realization of the position process within the `MIfitHMM` function. `MIfitHMM` is essentially a wrapper function for `fitHMM` that repeatedly fits the same user-specified HMM to m imputed data sets and stores the output from each of the m model fits. If a `crwData` object is provided, then `MIfitHMM` will first draw m imputations based on the `crwData` output and then fit the specified HMM to each imputed data set. If users wish to use a movement model other than the CTCRW to

account for measurement error and temporal irregularity (e.g. Calabrese *et al.* 2016; Gurarie *et al.* 2017), or if other observation error processes (e.g. missing data) are to be accounted for in the imputation step, `MIfitHMM` can also be used for analysis of a list of m `momentuHMMData` objects that were imputed by the user. Based on the m model fits, the `MIpool` function calculates pooled estimates, standard errors, and confidence intervals for the working scale parameters, natural scale parameters (based on transformations of the pooled working parameters and mean or user-specified values for any covariates), state sequences, state probabilities, and activity budgets (i.e. the proportion of the T times step assigned to each state) using standard multiple imputation formulae (Rubin & Schenker 1986; McClintock 2017). `MIpool` can be called separately or within `MIfitHMM` (using the `poolEstimates` argument), and the function returns a `miSum` object containing the pooled output across all imputatons. See sections 3.2, 3.3, 3.4, and 3.7 for example HMM analyses that use multiple imputation to account for location measurement error and temporally irregularity.

2.9 Model visualization and diagnostics

The generic `plot` functions for `momentuHMM` models (`plot.momentuHMM` and `plot.miSum`) plot the data stream histograms along with their corresponding estimated probability distributions, the estimated natural parameters and state transition probabilities as a function of any covariates included in the model, and the tracks of all individuals (color-coded by the most likely state sequence). By default, the probability distributions are plotted based on the means of any covariate values, but user-specified covariate values for the plots can be provided using the `covs` argument. When the argument `plotCI=TRUE`, then confidence intervals for the natural parameters and state transition probabilities are also plotted. Confidence intervals are calculated from the working parameter estimates based on the delta method and finite-difference approximations of the first derivative for the transformation using the `numDeriv::grad` function (Gilbert & Varadhan 2016). For multiple imputation analyses (`plot.miSum`), all plots are based on the pooled parameter estimates and the means of any covariates (if not provided by the `covs` argument) across each imputation. Using the argument `errorEllipse`, `plot.miSum` will include estimated location error ellipses in the plots of individual tracks. The functions `plotSat`, `plotSpatialCov`, and `plotStates` (Table 1) provide further methods for visualizing model results.

Diagnostic tools include the calculation and plotting of pseudo-residuals (Zucchini *et al.* 2016) using the `pseudoRes` and `plotPR` functions, respectively. For discrete distributions (e.g. Bernoulli, Poisson), a continuity adjustment is used for calculating pseudo-residuals. Akaike's Information Criterion can be calculated for one or more models using the `AIC.momentuHMM` function.

2.10 Simulation

The functions `simData` (and `simHierData`) can be used to simulate multivariate HMM (or HHMM) data from scratch or based on the estimated parameters of existing `momentuHMM` or `miSum` models. The `simData` and `simHierData` arguments are very similar to those used for model specification in `fitHMM` (e.g., `dist`, `hierDist`, `DM`) and data preparation in `prepData` (e.g., `spatialCovs`, `centers`), but they include additonal arguments, `lambda` and `errorEllipse`, for simulating location data subject to temporal irregularity and measurement error, respectively. The `spatialCovs` argument allows for rasters of spatio-temporal covariate values to be utilized in simulation models, while the `centers` argument allows activity centers to be incorporated. Thus `simData` and `simHierData` can be used to simulate more ecologically-realistic tracks (potentially subject to observation error) that can be useful for study design, power analyses, and assessing model performance. Goodness-of-fit can also be investigated by drawing simulated data sets from a fitted model and comparing them to observed properties of the data (Morales *et al.* 2004). While `simData` and `simHierData` can be used for simulating tracks from fitted models, we note that it assumes the location data are Cartesian coordinates; the `simData` and `simHierData` functions are therefore not appropriate for simulating tracks from models that were fitted to unprojected (latitude and longitude) data.

3 Examples

We will now demonstrate some of the capabilities of `momentuHMM` using real telemetry data. These examples are intended for demonstration purposes only, and we do not claim these example analyses represent improvements relative to previous or alternative analyses for these data sets. While only some of the key workflow elements are included here, complete R code and further details for these analyses are available in the "vignettes" source directory.

3.1 African elephant

As our first example, we use an African elephant (*Loxodonta africana*) bull track described in Wall *et al.* (2014) and publicly available from the movebank.org data repository.

We can load the data from the URL (this requires an Internet connection):

```
URL <- paste0("https://www.datarepository.movebank.org/bitstream/handle/",
              "10255/move.373/Elliptical%20Time-Density%20Model%20%28Wall%",
              "20et%20al.%202014%29%20African%20Elephant%20Dataset%20%",
              "28Source-Save%20the%20Elephants%29.csv")
rawData <- read.csv(url(URL))
```

The data set contains two tracks; for this analysis, we only consider the first one. In addition to hourly locations, the tag also collected external temperature data. We subset the data frame to keep only the relevant rows and columns:

```
# select and rename relevant columns
rawData <- rawData[,c(11,3,4,5,6)]
colnames(rawData) <- c("ID","time","lon","lat","temp")

# only keep first track
rawData <- subset(rawData,ID==unique(ID)[1])
```

The data now has the following columns:

```
head(rawData)
```

```
##            ID                    time       lon      lat temp
## 1 Salif Keita 2008-03-22 17:00:00.000 -2.160167 15.65350   38
## 2 Salif Keita 2008-03-22 18:00:00.000 -2.160075 15.65452   35
## 3 Salif Keita 2008-03-22 19:00:00.000 -2.159902 15.65451   32
## 4 Salif Keita 2008-03-22 20:00:00.000 -2.159435 15.65489   30
## 5 Salif Keita 2008-03-22 21:00:00.000 -2.158113 15.65512   29
## 6 Salif Keita 2008-03-22 22:00:00.000 -2.157848 15.65461   28
```

Location measurement error is negligible for these terrestrial GPS data, although about 1% of the hourly observations collected between 22 March 2008 and 30 September 2010 are missing. Instead of simply ignoring these missing data, we can employ

`crawlWrap` to predict the missing locations based on the CTCRW model of Johnson *et al.* (2008) prior to conducting our HMM analysis.

To use `crawlWrap`, we convert times from factors to POSIX, and project the observed locations to UTM coordinates:

```
# convert times from factors to POSIX
rawData$time <- as.POSIXct(rawData$time,tz="GMT")

# project to UTM coordinates using package rgdal
library(rgdal)
llcoord <- SpatialPoints(rawData[,3:4],
                         proj4string=CRS("+proj=longlat +datum=WGS84"))
utmcoord <- spTransform(llcoord,CRS("+proj=utm +zone=30 ellps=WGS84"))

# add UTM locations to data frame
rawData$x <- attr(utmcoord,"coords")[,1]
rawData$y <- attr(utmcoord,"coords")[,2]
```

Then, we call `crawlWrap` to fit a CTCRW model and predict hourly locations:

```
# fit crawl model
crwOut <- crawlWrap(obsData=rawData, timeStep="hour",
                    theta=c(6.855, -0.007), fixPar=c(NA,NA))
```

Here the desired time step is specified by the `timeStep` argument, and `theta` and `fixPar` arguments are the same as for `crawl::crwMLE` (Johnson 2017). For the choice of initial parameters in `crawlWrap`, we refer the reader to the documentation of the package crawl, in particular `crawl::crwMLE` and `crawl::crwPredict`. We now have a complete set of temporally-regular location data.

Autocorrelation function (ACF) estimates suggest there are 24-hour cycles in the step length data, and this presents an opportunity to demonstrate the use of the `cosinor` function for incorporating cyclical behavior in model parameters using `momentuHMM`. We create a `momentuHMMData` object, and the 24-hour cosinor model covariate:

```
# create momentuHMMData object from crwData object
elephantData <- prepData(data=crwOut, covNames="temp")

# add cosinor covariate based on hour of day
elephantData$hour <- as.integer(strftime(elephantData$time, format = "%H", tz="GMT"))
```

As seen here, the function `prepData` can also be used for pre-processing the best predicted track data from `crawlWrap` output. The 24-hour cosinor covariate ("hour") is simply a set of integers $(0, 1, \ldots, 23)$ indicating the hour of day for each observation. The ACF plot of the step lengths, shown in Figure 1, was obtained with:

```
acf(elephantData$step[!is.na(elephantData$step)],lag.max=300)
```

Our aim is to fit a 2-state HMM to the elephant track that includes temperature effects on the turning angle concentration parameters and cycling temperature effects (with a 24-hour periodicity) on the step length and state transition probability parameters. Complex models such as this can require many parameters, and it can be challenging to choose good starting parameter values for the optimization. Here, we take an incremental approach, starting from a simpler model with no covariates. In `momentuHMM`, the function `getPar0` extracts initial parameters from a fitted (nested) HMM, given arguments for the more complex model.

For the covariate-free 2-state model, six initial parameters need to be chosen: for each state, the mean and standard deviation of the gamma distribution of step lengths, and the concentration of the wrapped Cauchy distribution of turning angles. Looking at the histograms of the step lengths and the turning angles (e.g. output by `plot(elephantData)`) is often useful to choose good starting parameter values.

```
# label states
stateNames <- c("encamped","exploratory")
# distributions for observation processes
dist = list(step = "gamma", angle = "wrpcauchy")

# initial parameters
Par0_m1 <- list(step=c(100,500,100,200),angle=c(0.3,0.7))

# fit model
m1 <- fitHMM(data = elephantData, nbStates = 2, dist = dist, Par0 = Par0_m1,
             estAngleMean = list(angle=FALSE), stateNames = stateNames)
```

To ensure convergence, we could also use the argument `retryFits` to specify the number of attempts to minimize the negative log-likelihood based on random perturbations of the parameter estimates at the current minimum.

We can build on complexity, by including the temperature and time of day as covariates in the state transition probabilities. We use the function `getPar0` to extract the new starting parameter values.

```
# formula for transition probabilities
formula <- ~ temp * cosinor(hour, period = 24)

# initial parameters (obtained from nested model m1)
Par0_m2 <- getPar0(model=m1, formula=formula)

# fit model
m2 <- fitHMM(data = elephantData, nbStates = 2, dist = dist, Par0 = Par0_m2$Par,
             beta0=Par0_m2$beta, stateNames = stateNames, formula=formula)
```

The special function `cosinor(hour, period = 24)` internally creates the cosinor model covariates, $\cos(2\pi \times hour/period)$ and $\sin(2\pi \times hour/period)$, and includes both terms (plus interactions with "temp") in the fitted model.

Finally, we can fit the more complex model, including the effect of temperature and time of day on the parameters of the state-dependent distributions of steps and angles.

```
# formulas for parameters of state-dependent observation distributions
DM <- list(step = list(mean = ~ temp * cosinor(hour, period = 24),
                       sd = ~ temp * cosinor(hour, period = 24)),
           angle = list(concentration = ~ temp))

# initial parameters (obtained from nested model m2)
Par0_m3 <- getPar0(model=m2, formula=formula, DM=DM)

# fit model
m3 <- fitHMM(data = elephantData, nbStates = 2, dist = dist, Par0 = Par0_m3$Par,
             beta0 = Par0_m3$beta, DM = DM, stateNames = stateNames,
             formula = formula)
```

The above model `m3` identifed a state of slow undirected movement ("encamped"), and a state of faster and more directed movement ("exploratory") (Figure 1). For a fitted model, the function `viterbi` computes the most likely state sequence:

```
# decode most likely state sequence
states <- viterbi(m3)
# derive percentage of time spent in each state
table(states)/nrow(elephantData)
```

Here, about 74% of the steps were attributed to the "encamped" state, and 26% were attributed to the "exploratory" state.

We can use `AIC(m1,m2,m3)` to compare the three fitted models in terms of AIC; here, `m3` is overwhelmingly supported by the AIC when compared to alternative models with fewer covariates.

The model can be visualized with the generic function `plot`, which was used for the plots shown in Figure 2, and the decoded track in Figure 1.

```
plot(m3, plotCI = TRUE, covs = data.frame(hour=12))
```

Interestingly, this model suggests step lengths and directional persistence for the "encamped" state decreased as temperature increased, step lengths for both states tended to decrease in the late evening and early morning, and transition probabilities from the "encamped" to "exploratory" state decreased as temperature increased (Figure 2).

Model fit can be assessed using the pseudo-residuals, with the functions `pseudoRes` and `plotPR`. The residual ACF plot shown in Figure 1 was produced by:

```
# compute pseudo-residuals for the steps and the angles
pr <- pseudoRes(m3)

# plot the ACF of step pseudo-residuals
acf(pr$stepRes[!is.na(pr$stepRes)],lag.max = 300)
```

Autocorrelation function plots of the pseudo-residuals indicate this model explained much of the periodicity in step length, although there does still appear to be some room for improvement.

3.2 Northern fur seal

In our second example, we use the northern fur seal (*Callorhinus ursinus*) example from McClintock *et al.* (2014) to demonstrate the use of additional data streams for distinguishing behaviors with similar horizontal trajectories in a multivariate HMM. The data

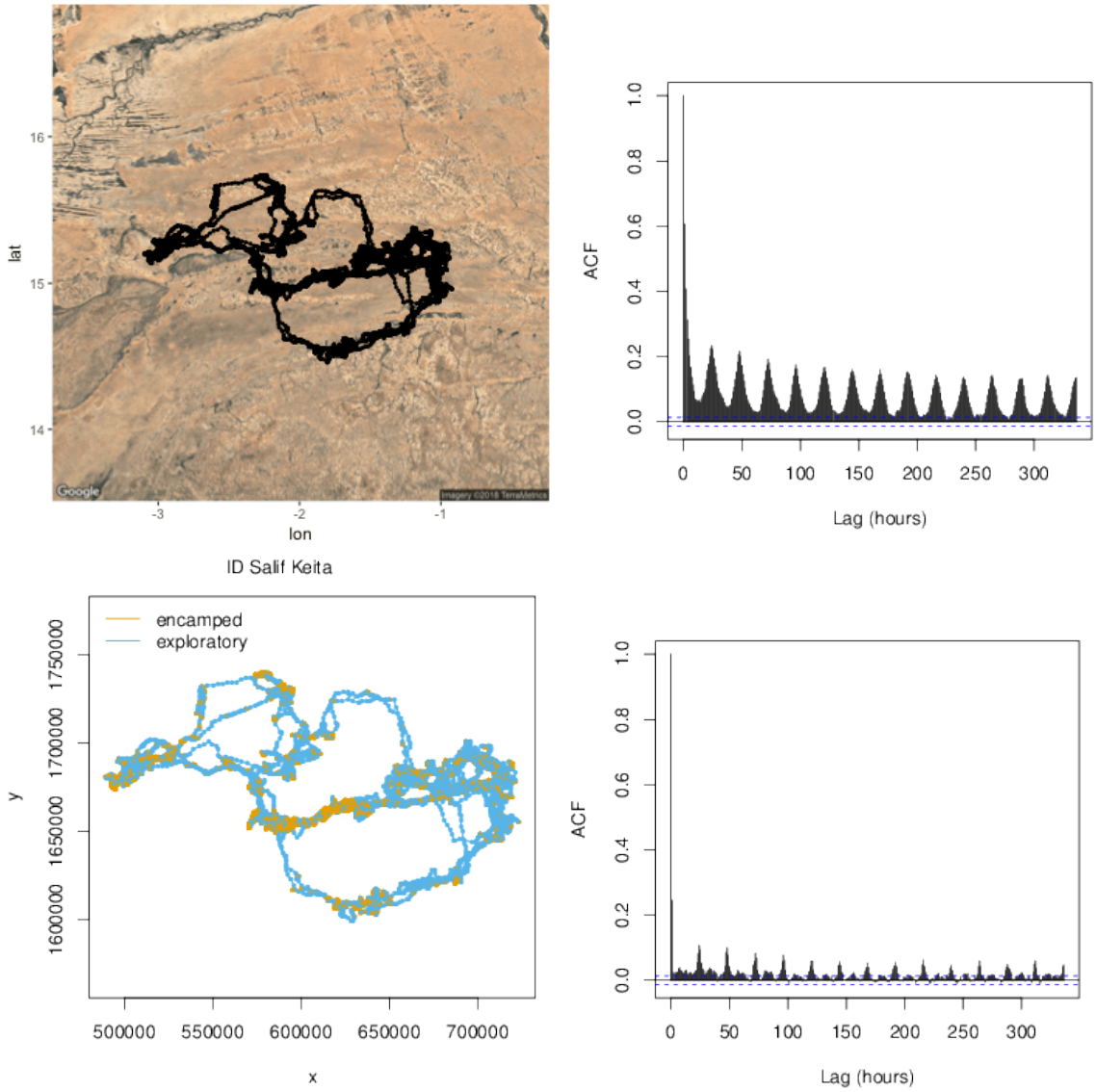


Figure 1. *Plot of the elephant track produced using the 'plotSat' function (top-left panel), autocorrelation function (ACF) plot of the corresponding step length data (top-right panel), plot of the Viterbi-decoded state sequence for the 2-state ("encamped" and "exploratory") model generated using the generic 'plot' function (bottom-left panel), and the step length pseudo-residual ACF plot for this model using the 'plotPR' function (bottom-right panel).*

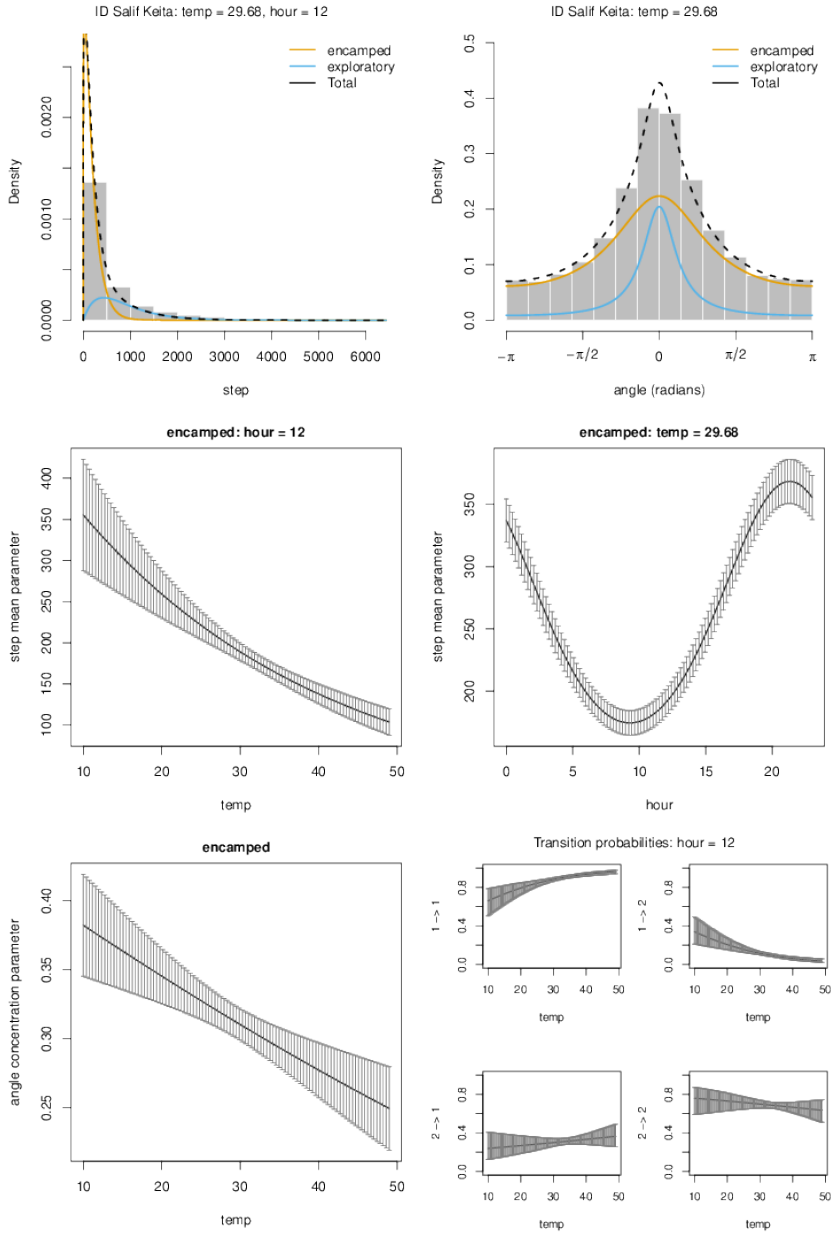


Figure 2. *Selected plots for the 2-state ("encamped" and "exploratory") African elephant example generated using the generic 'plot' function. Top panels present histograms of the step length (top-left) and turning angle (top-right) data along with the estimated state-dependent probability distributions based on the mean temperature (temp = 29.7 degrees celsius) at 12:00 GMT (hour = 12). Middle panels present estimates (and 95% confidence intervals) for the step length mean parameter of the "encamped" state as a function of temperature and hour of day. Bottom-left panel presents estimates for the turning angle concentration parameter of the "encamped" state as a function of temperature. Bottom-right panel presents estimated state transition probabilities (1 = "encamped", 2 = "exploratory") as a function of temperature at 12:00 GMT.*

consist of 241 temporally-irregular Fastloc GPS locations obtained during a foraging trip of a nursing female near the Pribilof Islands of Alaska, USA, from 10-17 October 2007. The tag included time-depth recording capabilities, and the dive activity data were summarized as the number of foraging dives over $T = 228$ temporally-regular 1 hr time steps. To fit the $N = 3$ state (1="resting", 2="foraging", 3="transit") of McClintock *et al.* (2014) using `momentuHMM`, we first used `crawlWrap` to predict temporally-regular locations at 1 hr time steps assuming a bivariate normal measurement error model and merged the results with the foraging dive data using the `crawlMerge` function. Then multiple imputation was used to account for location measurement error by repeatedly fitting the HMM to `nSims` realizations of the position process using `MIfitHMM`:

```
nbStates <- 3
stateNames <- c("resting", "foraging", "transit")
dist <- list(step = "gamma", angle = "wrpcauchy", dive = "pois")
Par0 <- getParDM(nbStates = nbStates, dist = dist,
                 Par = Par, DM = DM, workBounds = workBounds,
                 estAngleMean = list(angle = FALSE))
Fixpar <- list(dive = c(-100, NA, NA))
nfsFits <- MIfitHMM(crwOut, nSims = 100, nbStates = nbStates, dist = dist,
                    Par0 = Par0, DM = DM, workBounds = workBounds,
                    estAngleMean = list(angle = FALSE),
                    fixPar = fixPar, retryFits = 30,
                    stateNames=stateNames)
plot(nfsFits)
```

Here we specified a gamma distribution for step length ('step'), wrapped Cauchy distribution for turning angle ('angle'), and Poisson distribution for the number of foraging dives ('dive'). The function `getParDM` was used to organize the starting values for the data stream working parameters (`Par0`) in the correct format based on `DM`, `workBounds`, and estimates of the natural parameters (`Par`) from McClintock *et al.* (2014). The `DM` and `workBounds` arguments were specified to avoid label switching among the `nSims` imputed data model fits and enforce similar state-dependent probability distribution constraints as McClintock *et al.* (2014); for example, constraining the Poisson rate parameters such that the "foraging" state tends to have higher numbers of foraging dives than the "transit" state ($\lambda_2 > \lambda_3$; see Eq. 11 in section 3.7 for more details on parameter constraints using `DM` in conjunction with the `userBounds` and `workBounds` arguments). To prohibit foraging dives for the "resting" state, we used the `fixPar` argument to

effectively fix the Poisson rate parameter to zero on the natural scale (i.e. $\lambda_1 \approx 0$). To help deal with the problem of convergence to local maxima, the `retryFits` argument allows users to specify the number of times to attempt to re-fit each model using random perturbations of the parameter estimates as the starting values for optimization.

The results are very similar to those of the discrete-time model of McClintock *et al.* (2014), with periods of foraging often followed by resting (Figure 3). The "activity budgets" (i.e. the proportion of time steps allocated to each state) calculated by `MIpool` based on the estimated state sequences for each imputation were 0.31 (95% CI: 0.25−0.37) for "resting", 0.28 (95% CI: 0.23−0.35) for "foraging", and 0.41 (95% CI: 0.33−0.5) for "transit".

3.3 Loggerhead turtle

For our third example, we demonstrate how to model movement direction and step length as a function of angular covariates using hitherto unpublished loggerhead turtle (*Caretta caretta*) data for a captive-raised juvenile released in 2012 on the coast of North Carolina, USA. The data consist of 165 temporally-irregular Argos locations subject to measurement error and rasters of daily ocean surface currents collected between 20 November and 19 December 2012. Assuming a gamma distribution for step length (l_t) and a wrapped Cauchy distribution for turning angle (ϕ_t), we model the mean step length parameter (μ_t^l) as a function of ocean surface current speed (w_t) and direction (r_t) relative to the bearing of movement (b_t):

$$\mu_t^l = \exp(\beta_0^l + \beta_1^l w_t \cos(b_t - r_t)), \tag{6}$$

and the turning angle mean parameter (μ_t^ϕ) as a trade-off between short-term directional persistence and bias in the direction of ocean surface currents using the circular-circular regression link function:

$$\mu_t^\phi = \text{atan2}(\sin(d_t)\beta^\phi, 1 + \cos(d_t)\beta^\phi), \tag{7}$$

where $d_t = \text{atan2}(\sin(r_t - b_{t-1}), \cos(r_t - b_{t-1}))$.

We wish to fit a 2-state HMM to the turtle data, with a "foraging" state unaffected by currents and a "transit" state potentially influenced by ocean surface currents as in Eqs. 6 and 7. We used `crawlWrap` to predict $T = 350$ temporally-regular locations at 2

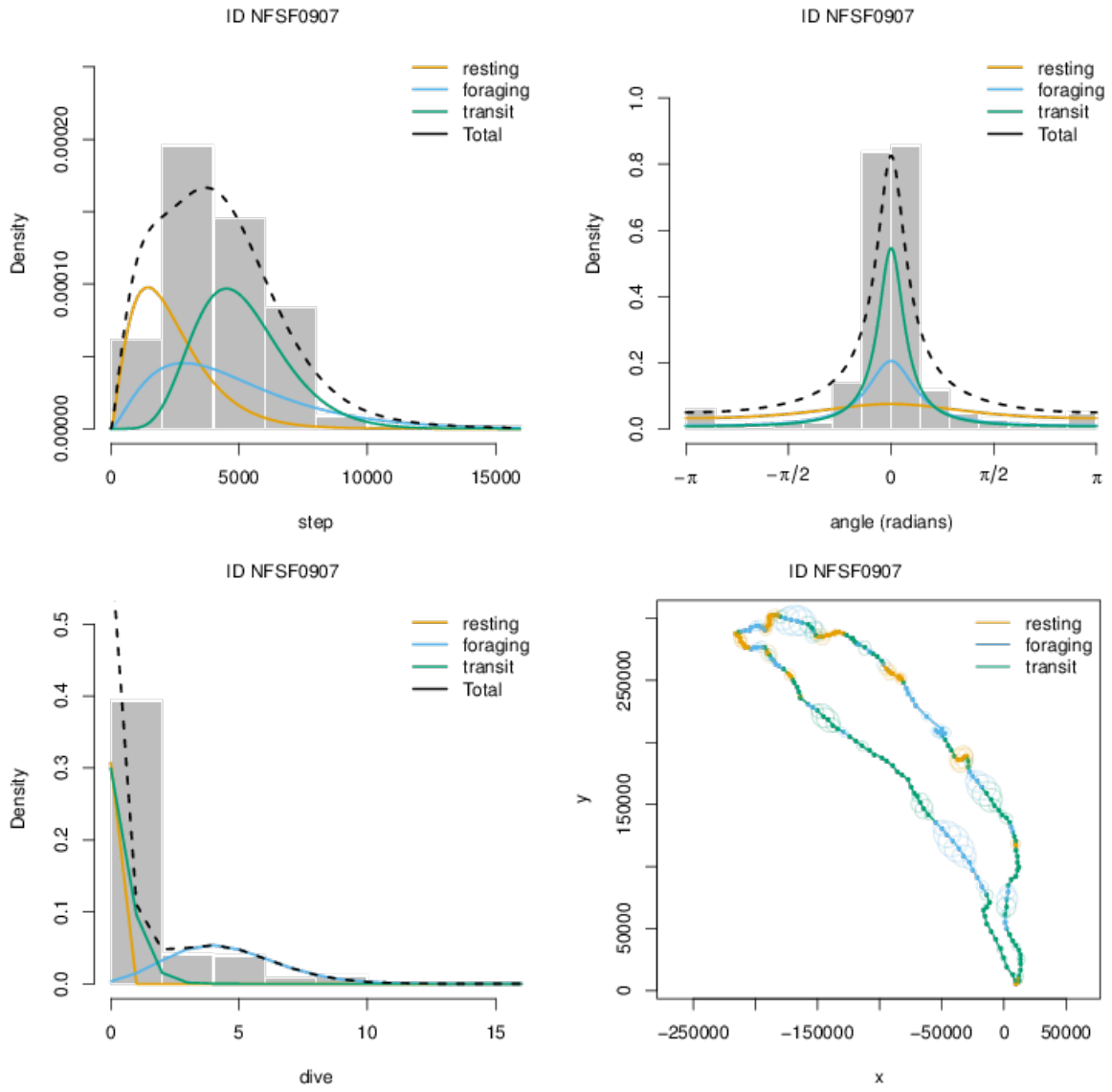


Figure 3. *Plots of the northern fur seal example results generated using the generic 'plot' function. The estimated probability distributions for step length (top-left panel), turning angle (top-right panel), and number of foraging dives (bottom-left panel) for the 3-state ("resting", "foraging", and "transit") model are plotted along with histograms of these data streams. The temporally-regular predicted locations (and 95% ellipsoidal confidence bands) and estimated states are plotted in the bottom-right panel. All estimates are pooled across multiple imputations of the position process and thus reflect uncertainty attributable to location measurement error and temporally-irregular observations.*

hr time steps assuming a bivariate normal measurement error model that accounts for the Argos location quality class (i.e. 3,2,1,0,A,B) of each observation. We then again used multiple imputation to account for location uncertainty by repeatedly fitting the HMM to `nSims` realizations of the position process using `MIfitHMM`. We first draw `nSims` realizations of the position process and extract the corresponding spatial covariates from the raster bricks for ocean surface current speed ("speedBrick") and direction ("dirBrick") using `MIfitHMM` with `fit=FALSE`:

```
miTurtleData <- MIfitHMM(crwOut, nSims = 100, fit=FALSE,
                 spatialCovs = list(w = speedBrick, d = dirBrick, r = dirBrick),
                 angleCovs = "d")
```

When the `fit` argument is `FALSE`, `MIfitHMM` returns a list of length `nSims` composed of `momentuHMMData` objects (`miData`). For convenience and ease of interpretation, we manually added an additional covariate ($angle\_osc = \cos(b_t - r_t)$) to each of the imputed data sets and fitted the 2-state HMM using Eqs. 6 and 7 for state 2 ("transit"):

```
nbStates<-2
dist <- list(step = "gamma", angle = "wrpcauchy")
DM <- list(step = list(mean = ~state2(w:angle_osc), sd = ~1),
           angle = list(mean = ~state2(d), concentration= ~1))
turtleFits <- MIfitHMM(miTurtleData$miData, nbStates = nbStates, dist = dist,
                       Par0 = Par0, DM = DM,
                       estAngleMean = list(angle = TRUE),
                       circularAngleMean = list(angle = TRUE))
plot(turtleFits, plotCI = TRUE, covs = data.frame(angle_osc = cos(0)))
```

Note that the `state2` special function in `DM` indicates the covariate formulas are specific to state 2 ("transit") and the `circularAngleMean` argument indicates that circular-circular regression link function is to be used on the mean turning angle parameter as in Eq. 7.

For the "transit" state, pooled parameter estimates indicated step lengths increased with ocean surface current speed and as the bearing of movement aligned with ocean surface current direction ($\beta_1^l = 0.4, 95\%$ CI: $0.09 - 0.7$; Figure 4). The estimated wrapped Cauchy distribution for turning angle had mean angles (μ_t^ϕ) biased towards the direction of ocean surface currents for each time step ($\beta^\phi = 0.26, 95\%$ CI: $0.04 - 0.48$), with

concentration parameter $\rho_2^\phi = 0.86$ (95% CI: 0.82−0.9) indicating turning angles were concentrated at μ_t^ϕ. Thus movement during the "transit" state appears to strongly follow ocean surface currents (mean $angle\_osc = 0.88, SD = 0.22$), while movement during the "foraging" state exhibited shorter step lengths ($\mu_1^l = 3001\text{m}, 95\%$ CI: $2439 - 3563$) perpendicular to ocean surface currents (mean $angle\_osc = 0.07, SD = 0.27$), with some directional persistence ($\rho_1^\phi = 0.49, 95\%$ CI: $0.37 - 0.61$). The turtle spent 0.56 (95% CI: 0.46−0.66) of the 2 hr time steps in the "foraging" state and 0.44 (95% CI: 0.34−0.54) of time steps in the "transit" state as it travelled northeast along a predominant current until it (presumably) found an attractive foraging patch (Figure 4).

It may often make more sense to weight angular covariates (such as ocean surface current direction) by their relative strength or importance. For example, weak ocean surface currents may be less likely to influence movement direction than strong ocean surface currents. This could easily be included in our turtle model using the `angleFormula(cov, strength, by)` special function in `DM`, where `cov` is an angle covariate (e.g. wind direction), `strength` is an optional positive real covariate (e.g. wind speed), and `by` is an optional factor variable for individual- or group-level effects (e.g. ID, sex):

```
DM$angle = list(mean = ~state2(angleFormula(d, strength = w)),
                concentration= ~1)
```

which would yield the following model for the "transit" state mean angle parameter:

$$\mu_t^\phi = \text{atan2}(w_t \sin(d_t)\beta^\phi, 1 + w_t \cos(d_t)\beta^\phi). \tag{8}$$

Still another option would be to use the von Mises consensus model, where the concentration parameter would now depend on the level of agreement between short-term directional persistence (i.e. going forward) and ocean surface currents:

```
dist$angle = "vmConsensus"
DM$angle = list(mean = ~state2(angleFormula(d, strength = w)),
                kappa = ~1)
```

which would yield the following model for the "transit" state concentration parameter:

$$\rho_t^\phi = \kappa\sqrt{[w_t \sin(d_t)\beta^\phi]^2 + [1 + w_t \cos(d_t)\beta^\phi]^2}. \tag{9}$$

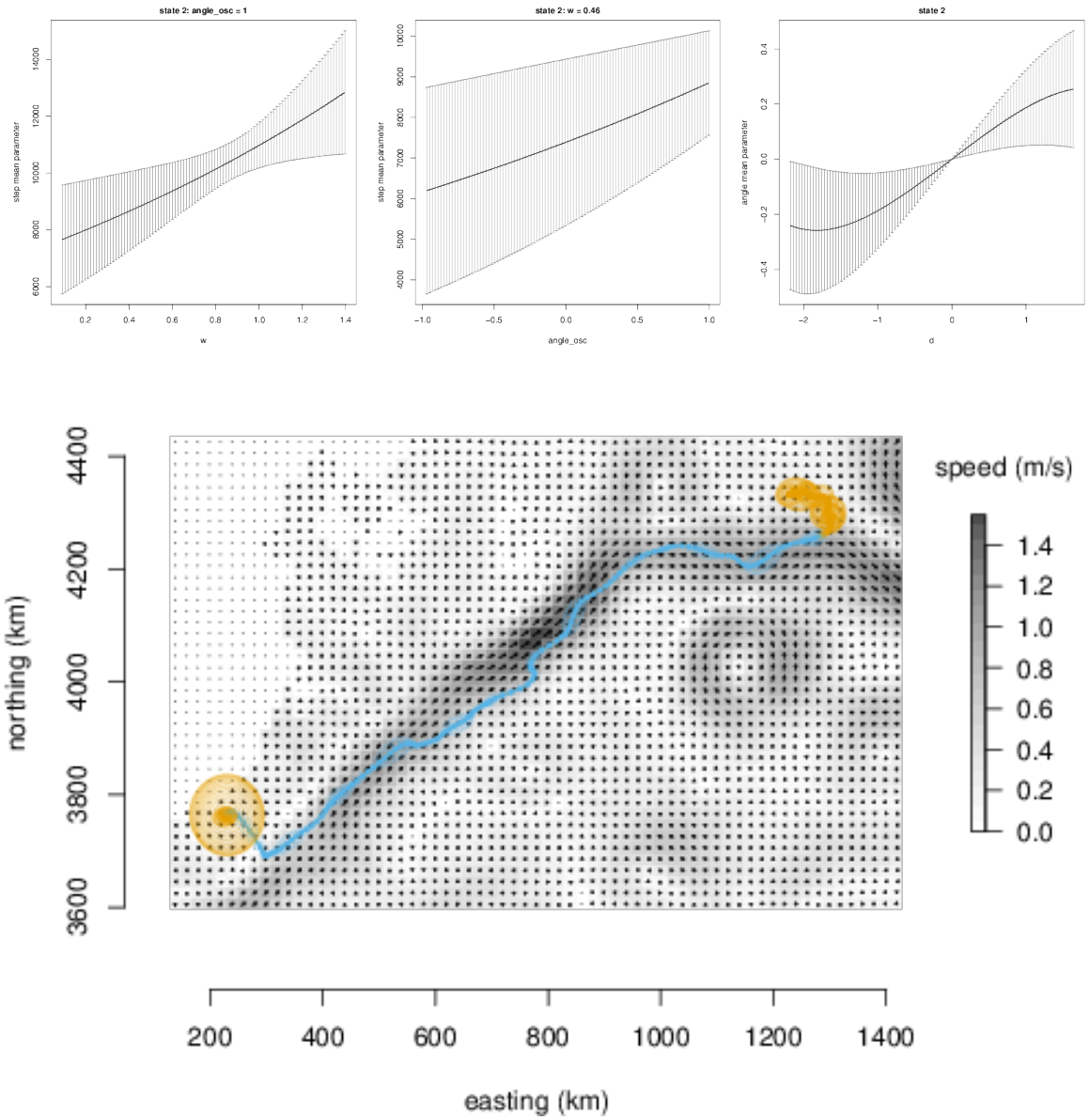


Figure 4. *Selected results from the loggerhead turtle example. Top panels include estimates and 95% confidence intervals for the mean step length parameter as a function of ocean surface current speed (w) when ocean surface current direction (r_t) is the same as the bearing (b_t) of movement (i.e. $angle\_osc = \cos(b_t - r_t) = 1$; top-left panel), mean step length parameter as a function of $angle\_osc$ at the mean ocean surface current speed ($w = 0.46$ m/s; top-middle panel), and mean turning angle parameter as a function of $d_t = atan2(\sin(r_t - b_{t-1}), \cos(r_t - b_{t-1}))$ (top-right panel). Bottom panel plots the pooled track, 95% error ellipse confidence bands, and state (orange = "foraging", blue = "transit") estimates based on multiple imputations of the position process relative to ocean surface current speed (m/s) and direction on 2 December 2012.*

If there were multiple turtles in this dataset, then individual-level effects could be included on μ^{ϕ} by simply specifying `angleFormula(d, strength = w, by = ID)` or `angleFormula(d, by = ID)` (with no strength effects).

One disadvantage of modeling steps and turns as above is that the fitted model cannot be properly simulated using `simData`. This is because `simData` is unable to calculate new realizations of the constructed covariate ($angle\_osc = \cos(b_t - r_t)$). However, we can implement a very similar model on the positions directly using a bivariate normal random walk. While arguably more intuitive, modeling the positions directly also has the added benefit that the fitted model can be properly simulated using `simData`. Similar to the continuous-time potential function approach of Brillinger *et al.* (2012) and Hooten *et al.* (2019), we can model the positions $\boldsymbol{\mu} = (\mu_x, \mu_y)$ as a bivariate normal random walk where the position at time t is a function of the position at time $t-1$ and the ocean surface current velocity vectors $V(\boldsymbol{\mu}_{t-1}) = (u_{t-1}, v_{t-1})$, where u is easting and v is northing:

$$\boldsymbol{\mu}_t \mid S_t = s \sim \mathcal{N}\left(\boldsymbol{\mu}_{t-1} + (\boldsymbol{\mu}_{t-1} - \boldsymbol{\mu}_{t-2})\beta_1 + V(\boldsymbol{\mu}_{t-1})\beta_2 I(s=2), \sigma_s^2 \mathbf{I}\right), \tag{10}$$

where $I()$ is the indicator function and $\mathbf{I}$ is a 2×2 identity matrix. This is analogous to our model for steps and turns, where $(\boldsymbol{\mu}_{t-1} - \boldsymbol{\mu}_{t-2})$ accounts for persistence in velocity. Thus when in state 1 (i.e., $S_t = 1$) the movement model is a correlated random walk, but when in state 2 (i.e., $S_t = 2$) the movement model includes a potential function surface based on ocean surface currents.

Recall that multivariate normal distributions require some additonal book-keeping when preparing the data; the `altCoordNames` argument in `prepData` and `MIfitHMM` and the `mvnCoords` argument in `fitHMM` and `MIfitHMM` are designed to help properly format and identify multivariate coordinate data streams. For example, if a bivariate normal data stream name is "loc" (e.g. `dist=list(loc="mvnorm2")`), then the data must include columns "loc.x" and "loc.y" for the x- and y- coordinates, respectively. When using a multivariate normal random walk distribution, the previous position can be referenced in `DM` formulas or pseudo-design matrices. For example, for a bivariate normal random walk data stream named "mu" (e.g. `dist=list(mu="rw_mvnorm2")`), the previous position can be refereced in `DM` as "mu.x_tm1" and "mu.y_tm1". This allows for persistence in velocity to be included as in Eq. 10 via the special formula function `crw(x_tm1,lag)`, where argument `x_tm1` is the previous position (e.g. "mu.x_tm1" or

"mu.y_tm1") and argument `lag` specifies the time lag for the persistence (`lag=1` in this example, but higher order lags could also be included). A complete demonstration of how to implement this bivariate normal random walk model can be found in the "turtleExample_rw_mvnorm2.R" script in the `momentuHMM` "vignettes" source directory (or at https://github.com/bmcclintock/momentuHMM).

3.4 Grey seal

For our next example, we perform a similar analysis of a grey seal (*Halichoerus grypus*) track that was originally conducted by McClintock *et al.* (2012) using Bayesian methods and (computationally-intensive) Markov chain Monte Carlo. The data consist of 1045 temporally-irregular Fastloc GPS locations collected in the North Sea between 9 April and 11 August 2008. Because the seal repeatedly visited the same haul-out and foraging locations, it provides a nice example for demonstrating how to implement biased movements relative to activity centers using `momentuHMM`. McClintock *et al.* (2012) fitted a 5-state model to these data that included three center of attraction states, with movement biased towards two haul-out sites ("Abertay" and "Farne Islands") and a foraging area ("Dogger Bank"), and two "exploratory" states ("low speed", "high speed") that were unassociated with an activity center. After using `crawlWrap` to predict $T = 1515$ temporally-regular locations at 2 hr time steps including a bivariate normal measurement error model, we can perform a very similar analysis to McClintock *et al.* (2012) in `momentuHMM` by using the `centers` argument and state-specific functions for the probability distribution parameters. A cluster analysis on the observed locations using the R package `dtwclust` (Sarda-Espinosa 2017) identified three centroids with coordinates that were nearly identical to the three activity centers ("Abertay", "Farne Islands", and "Dogger Bank") identified by McClintock *et al.* (2012). We use these coordinates to derive covariates relative to the activity centers when drawing `nSims` realizations of the position process:

```
crwSim <- MIfitHMM(crwOut, nSims = 100, fit=FALSE,
                   center = centers)
```

Specifying the `centers` argument results in the calculation of two covariates for each activity center: the distance (with '.dist' suffix) and angle (with '.angle' suffix)

from each location at time t. These covariates can then be used to model parameters as a function of the distance and angle to activity centers for each time step:

```
dist <- list(step = "weibull", angle = "wrpcauchy")
distFormula <- ~state1(I(Abertay.dist>2500)) + state2(I(Farne.dist>2500))
                  + state3(I(Dogger.dist>15000))
angleFormula <- ~state1(Abertay.angle) + state2(Farne.angle)
                  + state3(Dogger.angle)
stepDM <- list(shape = distFormula, scale = distFormula)
angleDM <- list(mean = angleFormula, concentration = distFormula)
DM <- list(step = stepDM, angle = angleDM)
```

Similar to McClintock *et al.* (2012), we assume a Weibull distribution for step length where both the shape and scale parameter depend on the distance from the location at time t to each activity center. For the activity centers on land ("Abertay" and "Farne"), we allow the (state-dependent) step length parameters to change when the seal is beyond 2500m of the haulout. For the "Dogger" activity center, we allow the parameters to change when the seal is beyond 15000m of this (presumably) foraging area. We thus allow the movement behavior to change within these activity center states upon entering or leaving the vicinity of these sites. We assume a wrapped Cauchy distribution for turning angle with (state-dependent) mean angle derived from the direction to each activity center at time t, and the concentration parameter is modeled similarly to the step length parameters. For the two "exploratory" states, we assumed they are simple random walks unaffected by proximity to activity centers. To complete our model specification, we use the `knownStates` argument to assign the seal to the corresponding activity center state whenever it was within the 2500m (haul-out area) or 15000m (foraging area) thresholds for each imputed data set:

```
greySealFits <- MIfitHMM(miDat, nSims = 400,
                         nbStates = 5, dist = dist,
                         Par0 = Par0, beta0 = beta0, fixPar = fixPar,
                         formula = distFormula,
                         estAngleMean = list(angle=TRUE),
                         circularAngleMean = list(angle=0),
                         DM = DM, knownStates = knownStates)
plot(greySealFits, plotCI = TRUE)
```

As with the step length and turning angle concentration parameters, the state transition probabilities are also allowed to change as a function of distance to activity centers (as specified by the `formula` argument). The starting values (`Par0` and `beta0`) for each imputation were extracted from a single HMM fitted to the best predicted locations from `crawlWrap`, and `circularAngleMean=list(angle=0)` was used to remove short-term directional persistence (and thus formulate the model as a mixture of biased and simple random walks).

Estimated activity budgets for the 5 states of this multiple imputation HMM were 0.28 (0.27 – 0.3) for the "Abertay" haul-out state, 0.12 (0.11 – 0.13) for the "Farne Islands" haul-out state, 0.37 (0.35 – 0.38) for the "Dogger Bank" foraging state, 0.11 (0.05 – 0.2) for a low-speed "exploratory" state, and 0.12 (0.07 – 0.21) for a high-speed "exploratory" state. All three activity center states exhibited shorter step lengths and less biased movements when within the vicinity of these targets (Figure 5). Results from this analysis were thus very similar to those of McClintock *et al.* (2012), but this implementation required far less computation time and no custom model-fitting algorithms.

The `simData` function can be used to simulate tracks from a fitted model:

```
greySealSim<-simData(model = greySealFits, centers = centers,
                     initialPosition = centers[1,],
                     obsPerAnimal = 1515)
```

A simulated track is presented along with the fitted track in Figure 6. While potentially useful for study design, power analysis, and prediction, the `simData` function can also be helpful in assessing goodness of fit by repeatedly drawing simulated data sets from a fitted model and comparing them to observed properties of the data (e.g. Morales *et al.* 2004).

3.5 Southern elephant seals

Here, we analyse the southern elephant seal (*Mirounga leonina*) data from Michelot *et al.* (2017) using `momentuHMM`. The data set consists of 15 tracks, each encompassing (at most) one foraging trip, starting from Kerguelen Island. We want to fit the model described by Michelot *et al.* (2017), with the four following states:

1. outbound trip from the colony to the ice;

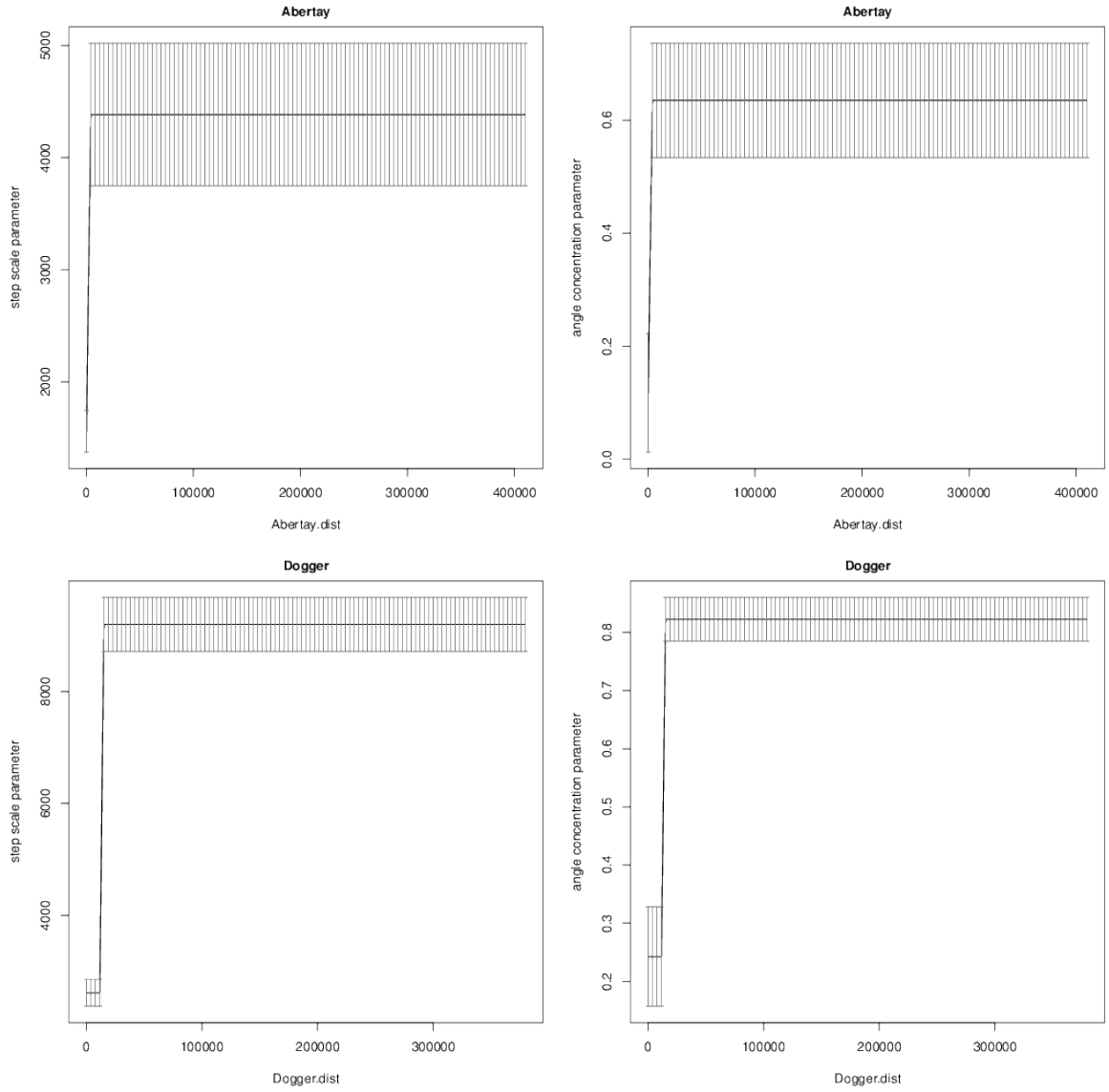


Figure 5. *Selected results from the grey seal example. Panels include estimates and 95% confidence intervals for the "Abertay" haul-out state step length scale parameter as a function of distance in meters ('Abertay.dist'; top-left panel), "Abertay" haul-out state turning angle concentration parameter as a function of distance (top-right panel), "Dogger Bank" foraging state step length scale parameter as a function of distance ('Dogger.dist'; bottom-left panel), and the "Dogger Bank" foraging state turning angle concentration parameter as a function of distance (bottom-right panel).*

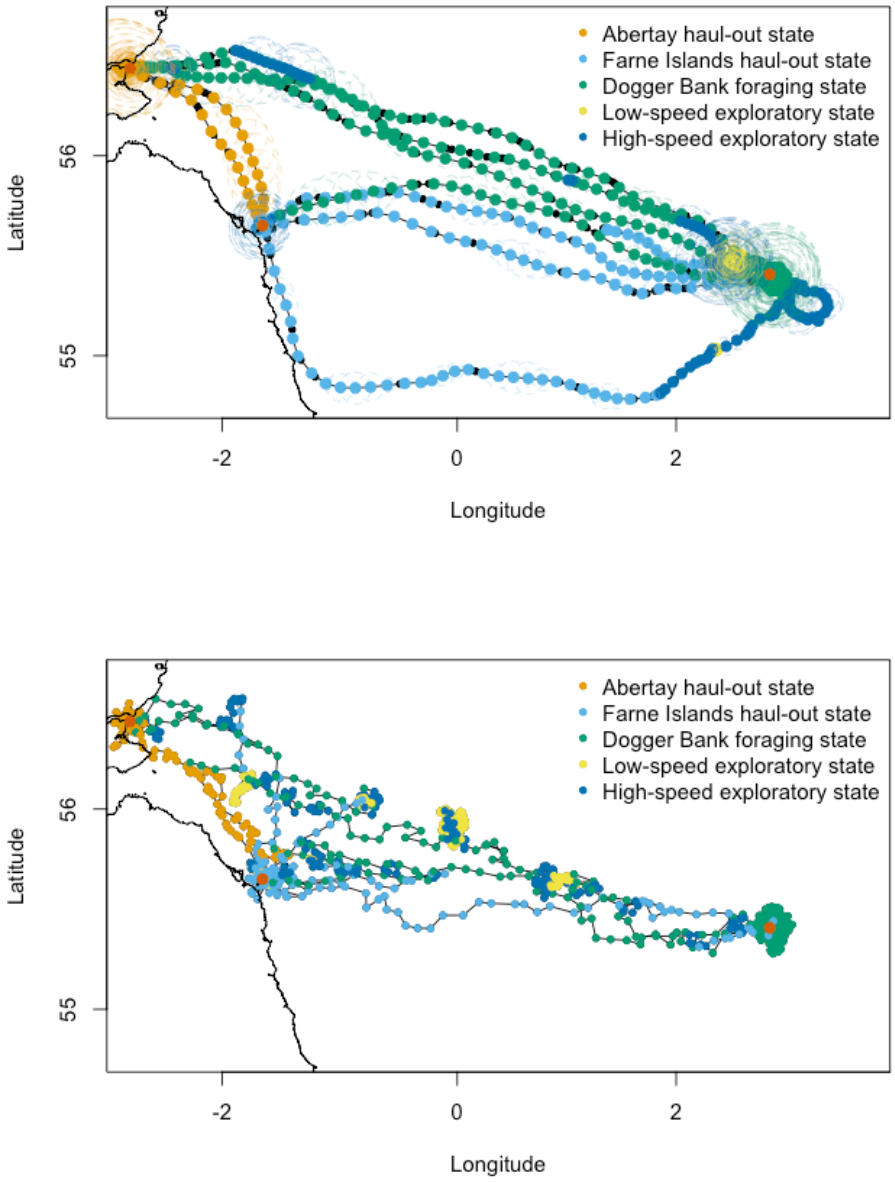


Figure 6. *Fitted and simulated tracks from the grey seal example. This seal tended to move in a clockwise fashion between two haul-out locations ("Abertay" and "Farne Islands") and a foraging area ("Dogger Bank") in the North Sea. Top panel plots the pooled track, 95% error ellipse confidence bands, and state estimates based on the 5-state HMM fitted to multiple imputations of the position process. Red points indicate the locations of the three activity centers. Black points indicate the (temporally-irregular) observed locations. Bottom panel presents the locations and states for a track simulated from the fitted model using the 'simData' function.*

2. searching;

3. foraging;

4. inbound trip from the ice to the colony.

The data set has three columns: "ID" (track ID), "x" (longitude), and "y" (latitude):

```
head(tracks)
```

```
##   ID        x         y
## 1  1 70.60946 -49.60737
## 2  1 70.82908 -50.08287
## 3  1 70.90029 -50.32835
## 4  1 70.85766 -50.54746
## 5  1 70.63792 -50.90529
## 6  1 70.48480 -50.99666
```

From the locations, we use `prepData` to derive the step lengths and turning angles, as well as the distance and bearing (relative to previous movement direction as in Eq. 3) to the Kerguelen Island colony (with coordinates 70° longitude and -49° latitude):

```
center <- matrix(c(70,-49),nrow=1,dimnames=list("colony"))
data <- prepData(data=tracks, type="LL", centers=center)
```

Note that distances are in kilometers and angles are based on initial bearings (using geosphere::bearing; Hijmans 2016a) when calculated from longitude and latitude coordinates.

3.5.1 Model 1: no covariates

We start by fitting a covariate-free 4-state correlated random walk model, which we will use to extract starting parameter values for more complex models. We use the argument `fixPar` to fix some transition probabilities to zero, following Michelot *et al.* (2017). We set to `NA` the columns of unconstrained transition probabilities, and we fix the intercept of the other columns to a large negative number (here -100) to set the corresponding transition probabilities to be virtually zero (i.e. impossible transition). As in Michelot *et al.* (2017), we set transition probabilities from outbound to forage, outbound to inbound, search to outbound, forage to outbound, forage to inbound,

inbound to outbound, inbound to search, and inbound to forage to be effectively zero.

```
stateNames <- c("outbound","search","forage","inbound")

# initial parameters
stepPar0 <- c(25,5,1,25,10,5,3,10)
anglePar0 <- c(15,5,2,15)

# constrain transition probabilities
fixbeta <- matrix(c(NA,-100,-100,-100,NA,NA,-100,NA,-100,-100,-100,-100),
                  nrow=1)

m1 <- fitHMM(data=data, nbStates=4, dist=list(step="gamma",angle="vm"),
             Par0=list(step=stepPar0, angle=anglePar0),
             fixPar=list(beta=fixbeta), stateNames = stateNames)
```

3.5.2 Model 2

This model mimics the formulation of Michelot *et al.* (2017). We model the effect of the distance to colony on the transition probability from outbound to search, and of the time since departure on the transition probability from search to inbound.

```
# time spent since left colony
time <- NULL
for(id in unique(data$ID)) {
    nbSubObs <- length(which(data$ID==id))

    # approximately in months for interval = 9.6h
    time <- c(time, (1:nbSubObs)/75)
}

data$time <- time

# compute time since departure and include in formula below
formula <- ~ colony.dist + time
```

As before, we constrain the transition probability matrix to prevent some of the transitions (e.g. from forage to inbound, etc.). We define a 3×12 matrix for the beta parameters, in which each column corresponds to a transition ($1 \to 2, 1 \to 3, 1 \to 4, 2 \to$

1, . . .), and each row corresponds to a covariate (intercept, distance to center, time since departure). We set to `NA` the columns of unconstrained transition probabilities, and we again fix the intercept of the other columns to a large negative number (here -100) to set the corresponding transition probabilities to be virtually zero (i.e. impossible transition).

```
fixbeta <- matrix(c(NA,-100,-100,-100,NA,NA,-100,NA,-100,-100,-100,-100,
                    NA,   0,   0,   0, 0, 0,   0, 0,   0,   0,   0,   0,
                     0,   0,   0,   0, 0,NA,   0, 0,   0,   0,   0,   0),
                  nrow=3,byrow=TRUE)
```

Biased random walks are used to model the movement in states 1 and 4, with repulsion away from the colony in the outbound trip, and attraction towards the colony in the inbound trip. For that purpose, we include 'colony.angle' as a covariate on the angle mean of the von Mises distributions for turning angles in states 1 and 4.

```
angleFormula <- ~ state1(colony.angle) + state4(colony.angle)
```

To specify the direction of the bias (away from or towards the colony), we fix the parameters linking the mean turning angle to the direction of the colony. Because we will remove the correlated random walk component of Eq. 2 for states 1 and 4 (by setting `circularAngleMean=list(angle=0)`; see section 2.3), we fix the coefficient to -1 for state 1 (so that the mean direction is away from the colony), and we fix the coefficient to $+1$ for state 4 (so that the mean direction is towards the colony). Note that with only a single angular covariate, the magnitude of these fixed coefficients is not important; only the sign is important (i.e. positive for attraction, negative for repulsion). The four other parameters correspond to the angle concentrations and should be estimated (NAs in `fixPar`).

```
fixPar <- list(angle=c(-1,1,NA,NA,NA,NA),beta=fixbeta)
```

Because no covariates are specified for the mean angle of state 2 (searching) and state 3 (foraging), these states are reduced to correlated random walks with a mean turning angle of zero (i.e. $\text{atan2}(0, 0) = 0$; see Eq. 2).

We can now fit the second model with starting parameter values extracted from the simpler model using `getPar0`. In `fitHMM`, we use the arguments `estAngleMean` and

`circularAngleMean` to indicate that the angle mean is to be estimated using circular-circular regression (with short-term directional persistence removed for states 1 and 4).

```
Par0 <- getPar0(model=m1, nbStates=4,
                DM=list(angle=list(mean=angleFormula, concentration=~1)),
                estAngleMean=list(angle=TRUE),
                circularAngleMean=list(angle=0), formula=formula)

m2 <- fitHMM(data=data, nbStates=4, dist=list(step="gamma",angle="vm"),
             Par0=list(step=Par0$Par$step, angle=Par0$Par$angle),
             beta0=Par0$beta, fixPar=fixPar, formula=formula,
             DM=list(angle=list(mean=angleFormula, concentration=~1)),
             estAngleMean=list(angle=TRUE), circularAngleMean=list(angle=0),
             stateNames = stateNames)
```

Instead of relying entirely on `fixPar` for parameter constraints, an equivalent model for the transition probabilities could be specified using the special function `betaCol` in `formula`:

```
formula <- ~ betaCol1(colony.dist) + betaCol6(time)

fixbeta <- matrix(c(NA,-100,-100,-100,NA,NA,-100,NA,-100,-100,-100,-100,
                    rep(NA,12),
                    rep(NA,12)),
                  nrow=3,byrow=TRUE)

fixPar <- list(angle=c(-1,1,NA,NA,NA,NA),beta=fixbeta)
```

Here `betaCol1(colony.dist)` specifies an effect of distance to colony only on the transition from state 1 to state 2 (which corresponds to the first column of the beta matrix) and `betaCol6(time)` specifies an effect of time since departure only on the transition from state 2 to state 4 (which corresponds to the sixth column of the beta matrix). When the special function `betaCol` is used, then `fitHMM` automatically fixes the appropriate elements in the second ('colony.dist') and third ('time') rows of the beta matrix to zero (without the user needing to do so manually using `fixPar`). However, note that the first row (corresponding to the intercept terms) must still be manually fixed to achieve the desired constraints on the transition probability matrix.

3.5.3 Model 3

In addition to the covariates included in model 2, we add the effect of distance to colony on the step length and turning angle concentration parameters for states 1 and 4. We specify the following formulas:

```
distFormula <- ~ state1(colony.dist) + state4(colony.dist)
stepDM <- list(mean=distFormula, sd=distFormula)
angleDM <- list(mean=angleFormula, concentration=distFormula)
```

The initial parameters are extracted from model 2, again using the function `getPar0`. Instead of fixing the mean direction of movement like in model 2, we estimate it here as a trade-off between short-term directional persistence and bias toward (or away) from the colony (i.e. a biased correlated random walk as in Eq. 2).

```
# remove fixed angle parameters
fixPar <- list(beta=fixbeta)

# get starting parameters from m2
Par0 <- getPar0(model=m2, nbStates=4,
                DM = list(step=stepDM, angle=angleDM),
                estAngleMean=list(angle=TRUE),
                circularAngleMean=list(angle=TRUE),
                formula=formula)

# the bias is estimated rather than fixed
Par0$Par$angle[c("mean_1:(colony.angle)","mean_4:(colony.angle)")] <- 0

m3 <- fitHMM(data=data, nbStates=4, dist=list(step="gamma",angle="vm"),
             Par0=list(step=Par0$Par$step, angle=Par0$Par$angle),
             beta0=Par0$beta, fixPar=fixPar, formula=formula,
             DM = list(step=stepDM, angle=angleDM),
             estAngleMean=list(angle=TRUE),
             circularAngleMean=list(angle=TRUE),
             stateNames = stateNames)
```

The three fitted model can be compared with `AIC(m1,m2,m3)`, which overwhelmingly supports model 3. The most likely state sequence is obtained with `viterbi(m3)`. Figure 7 shows a map of the state-decoded track. The estimated circular-circular regression coefficients for the angle means of state 1 (outbound) and state 4 (inbound)

were -0.66 (95% CI: $-0.82 - -0.5$) and 0.4 (95% CI: $0.32 - 0.49$), respectively, thus indicating biased correlated random walks with repulsion away from the colony during outbound movements and attraction towards the colony during inbound movements. The estimated regression coefficients for the step length mean and turning angle concentration parameters for states 1 and 4 suggest that step lengths decreased and turning angles became more concentrated at the mean angle as distance to colony increased (Figure 8)

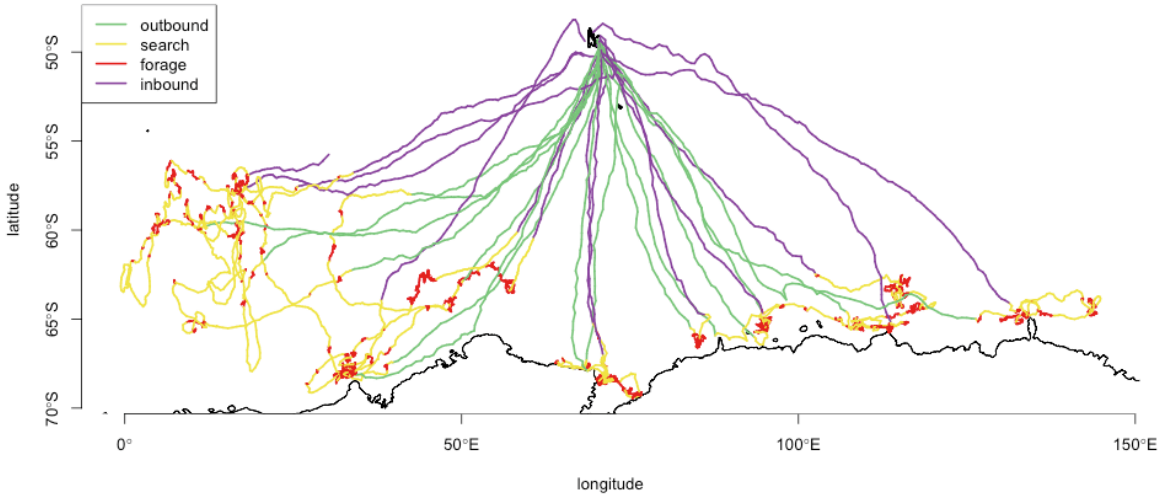


Figure 7. *The 15 elephant seal tracks, colored by the most likely state sequence.*

3.6 Group dynamic animal movement

Here we demonstrate how `momentuHMM` can be used to simulate and fit the group dynamic animal movement model of Langrock *et al.* (2014). In group dynamic models, groups (e.g., herds, packs, schools) are allowed to influence the movement of social individuals. One way to accomplish this is to model individual movements as being attracted to a group "centroid". Depending on the system, the centroid could simply be the location of the group center (e.g., the mathematical centroid of the group) or group leader (e.g., alpha wolf) at times $t = 1, \ldots, T$. In this sense, the centroid can be considered a dynamic activity center that changes position over time, and these models are not necessarily limited to groups. For example, the centroid could instead

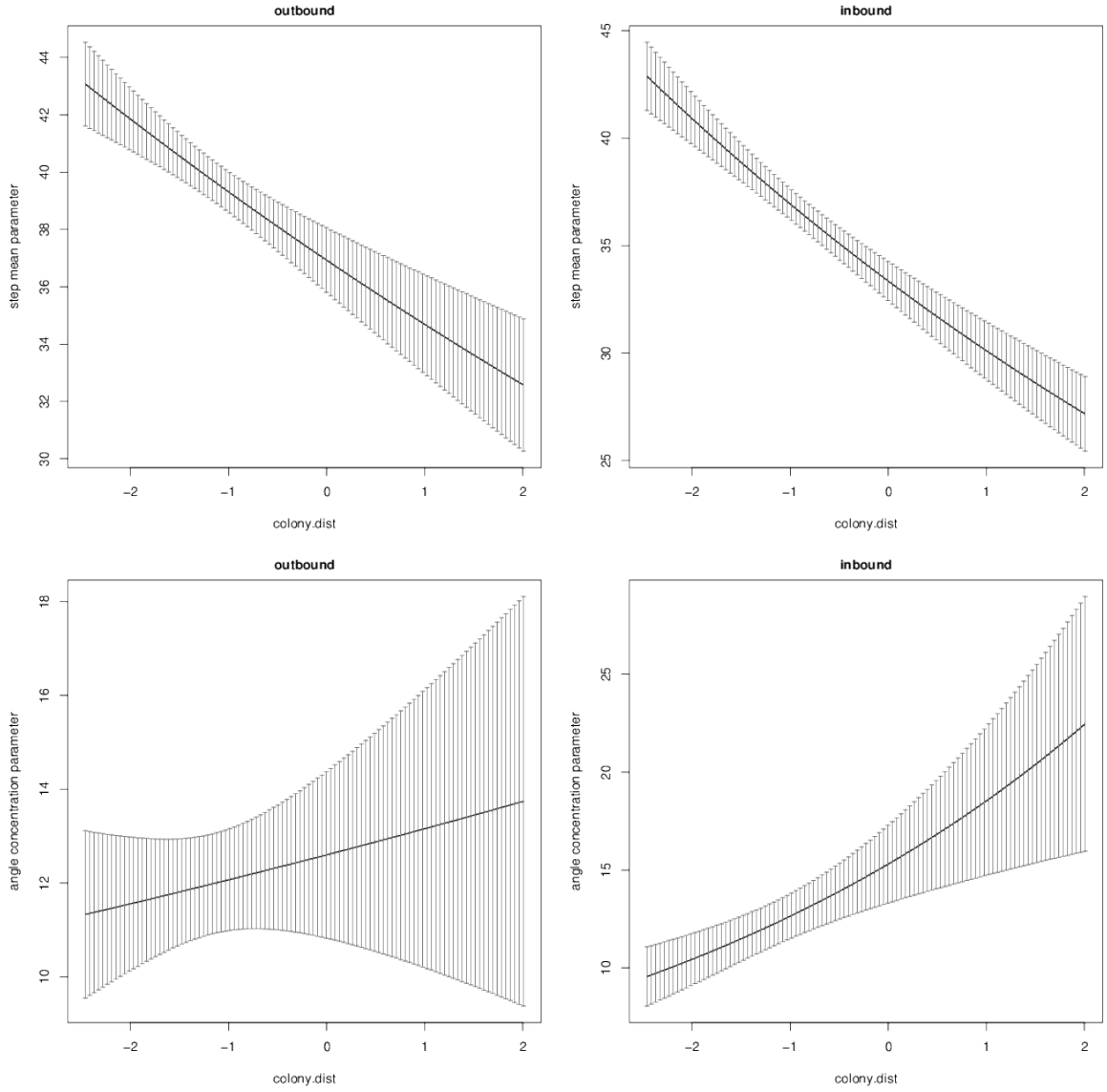


Figure 8. *Selected results from the elephant seal example. Panels include estimates and 95% confidence intervals for the "outbound" mean step length parameter (top-left panel), "inbound" mean step length parameter (top-right panel), "outbound" turning angle concentration parameter (bottom-left panel), and "inbound" turning angle concentration parameter (bottom-right panel) as a function of distance to colony ('colony.dist'). Distance to colony has been standardized based on a mean of 2539 km (SD = 1021.3).*

refer to predators, competitors, or human activity (in which case the centroid might be repulsive rather than attractive!).

Dynamic activity centers can be simulated in `simData` using the `centroids` argument. Following simulation scenario A of Langrock *et al.* (2014), we first simulate a group centroid as a single-state (i.e. $N = 1$) biased correlated random walk relative to the origin:

```
dist <- list(step="gamma", angle="vm")
nbObs <- 250

Parc <- list(step = c(15,10),
             angle = c(0.15,log(1)))

DMc <- list(angle=list(mean = ~center1.angle,
                       concentration=~1))

centroidData <- simData(nbStates=1, dist=dist, Par=Parc, DM=DMc,
                        circularAngleMean = list(angle = TRUE),
                        centers = matrix(0,1,2),
                        obsPerAnimal = nbObs)
```

Now we can use the simulated centroid track (Fig. 9) as a dynamic activity center and simulate the movement of a group of 20 individuals as a 2-state mixture of a biased random walk (relative to the centroid) and a correlated random walk (independent of the centroid):

```
nbAnimals <- 20
nbStates <- 2
stateNames <- c("group","solitary")

Par <- list(step = c(30,50,15,25),
            angle = c(1,log(2.5),log(5)))

beta <- matrix(c(-2.944439,-1.734601),1,nbStates)

DM <- list(angle=list(mean = ~state1(centroid.angle),
                      concentration = ~1))

# calculate stationary distribution
gamma <- diag(nbStates)
```

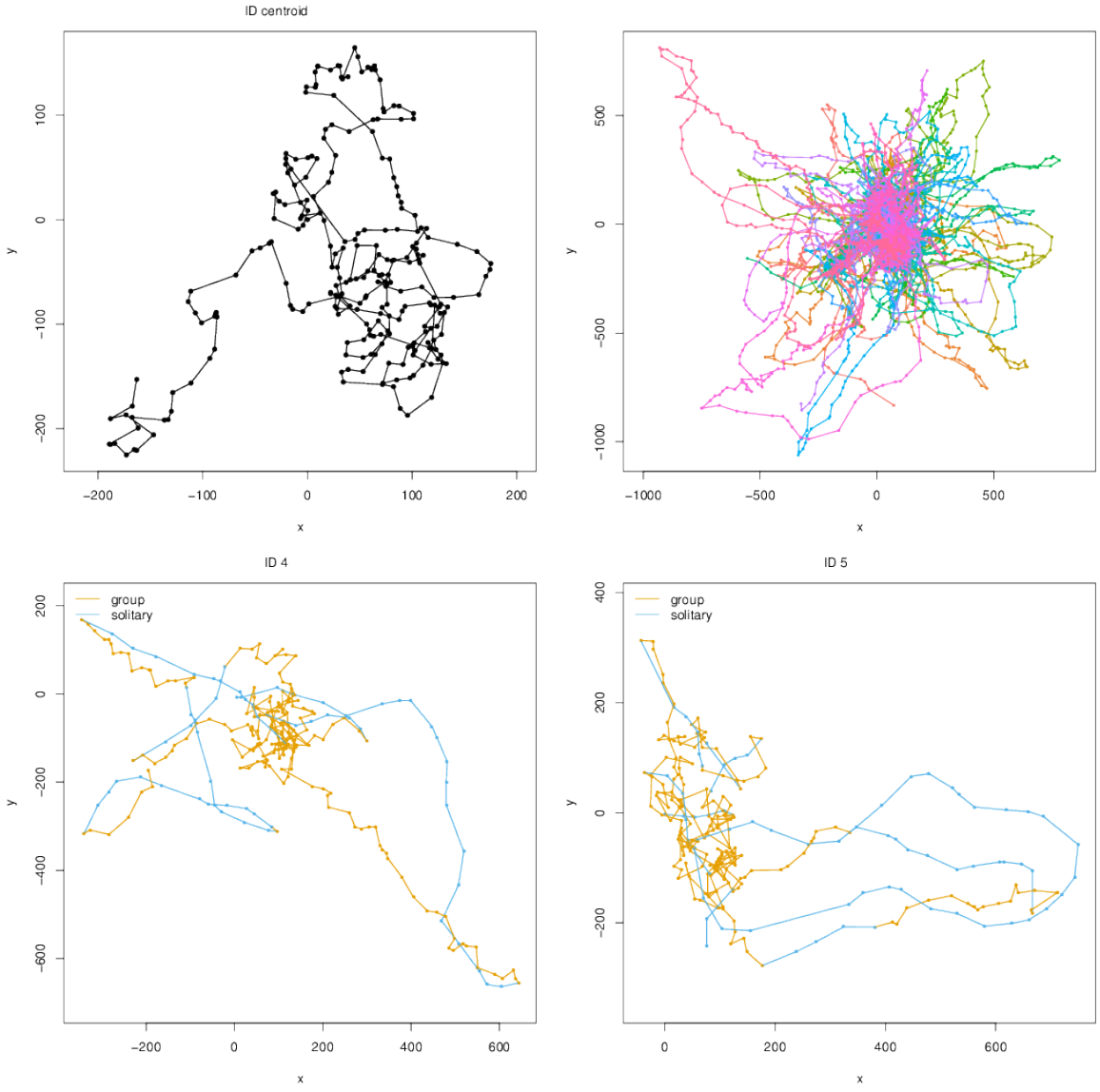


Figure 9. *Selected results from the group dynamic animal movement example. Panels include the simulated centroid path (top-left panel), the simulated paths of 20 individuals where state 1 ("group") includes biased movements towards the centroid and state 2 ("solitary") is a correlated random walk independent of the group centroid (top-right panel), and two fitted tracks that are colored by the most likely state sequence (bottom panels).*

```
gamma[!gamma] <- exp(beta)
gamma <- t(gamma)
gamma <- gamma/apply(gamma,1,sum)
delta <- solve(diag(nbStates) - t(gamma) + 1, rep(1, nbStates))

# draw random initial locations for each individual
initialPositions <- vector("list")
for (i in 1:nbAnimals) {
  initialPositions[[i]] <- runif(2, -10, 10)
}

# create centroid data frame
cD <- data.frame(x = centroidData$x, y = centroidData$y)

groupData <- simData(nbAnimals=nbAnimals, nbStates=nbStates, dist=dist,
                     Par = Par, beta = beta, delta = delta, DM = DM,
                     circularAngleMean = list(angle = 0),
                     centroids = list(centroid = cD),
                     obsPerAnimal = nbObs,
                     initialPosition = initialPositions,
                     states = TRUE, stateNames = stateNames)
```

Here state 1 ("group") has biased movements toward the centroid and state 2 ("solitary") is simply a correlated random walk independent of the group centroid (Fig. 9). Note that despite this being a 2-state HMM, the working scale parameters for turning angle (`Par$angle`) only includes 3 parameters (1 for the angle mean and 2 for the concentration parameters). This is because under the circular-circular regression model, no working parameter is specified[3] for the reference turning angle of zero (i.e., the component for short-term directional persistence; see Eq. 2) and no angular covariates were specified in the model for state 2 ("solitary"). Thus the first parameter corresponds to the working scale parameter of the `centroid.angle` covariate for state 1 ("group"), while the second and third parameters are the working scale parameters for the concentration parameters for states 1 and 2, respectively. In this case, we remove the correlated random walk component for state 1 by setting `circularAngleMean = list(angle = 0)`, and the angle mean parameter for state 1 was set at a positive value (+1) to enforce a biased random walk with attraction towards

[3]More accurately, the working parameter for the reference angle is automatically fixed to $\beta_0 = 1$ (or whatever scalar is provided by the circularAngleMean argument).

the group centriod. As there is only a single angular covariate here, the magnitude of this value is not important; only the sign matters (i.e. positive for attraction, negative for repulsion).

Finally, we can fit the group dynamic model using `fitHMM`:

```
Par0 <- list(step = c(30,50,15,25),
             angle = c(1,log(2.5),log(5)))

fixPar <- list(angle=c(1,NA,NA))

groupFit <- fitHMM(groupData, nbStates=nbStates, dist=dist, Par=Par0,
                   DM = DM, stationary = TRUE,
                   estAngleMean = list(angle = TRUE),
                   circularAngleMean = list(angle = 0), fixPar = fixPar,
                   stateNames = stateNames)
```

3.7 Harbour seals

Here we demonstrate how more complicated parameter constraints can be implemented using the `userBounds` and `workBounds` arguments in `fitHMM` and `MIfitHMM`. This example is based on the harbour seal analysis of McClintock *et al.* (2013). Using individual-level random effects on probability distribution parameters, McClintock *et al.* (2013) performed a Bayesian analysis of population-level activity budgets for 3-states ("resting", "foraging", and "transit"). While `momentuHMM` cannot be used to replicate this analysis exactly, we can perform a similar analysis in the absence of individual-level random effects. Here we will focus on several specific parameter constraints, but the full example code can be found in the "vignettes" source directory.

The harbour seal data consist of 17 individuals (10 male, 7 female) and, as in the northern fur example in section 3.2, both location and dive activity data. The location data were obtained at temporally-irregular intervals, while the dive activity data were obtained at regular 2-hour time steps. We therefore first used `crawlWrap` to fit and predict locations for all 17 tracks at 2-hour time steps and then used `crawlMerge` to merge the predicted locations with the dive activity data. We then fitted several different models assuming: 1) no individual- or sex-level effects on all parameters (i.e., the "null" model); 2) sex-level effects on all parameters; and 3) individual-level effects on all parameters. Based on `fitHMM` fits for the best predicted tracks, AIC overwhelmingly

supported the model including individual-level effects on all parameters, but for simplicity we will use the model including no individual- or sex-level effects to demonstrate how the constraints of McClintock *et al.* (2013) can be implemented in `momentuHMM`. In addition to the lack of individual-level random effects in our example, we also depart from McClintock *et al.* (2013) in our use of zero-inflation parameters to account for steps of length zero (i.e., $l_t = 0$) and time steps with no dive activity (i.e., $\omega_t = 0$). Unlike McClintock *et al.* (2013), note that our model 3 also includes individual-level fixed effects on the state transition probabilities.

McClintock *et al.* (2013) fit their 3-state model using more complicated constraints on the probability distribution parameters than any of our previous vignette examples. In addition to relational constraints among the "resting", "foraging", and "transit" states similar to those used in the northern fur seal example (section 3.2), these constraints included upper bounds for the shape and scale parameters of the Weibull distribution for step length, a minimum value for the "transit" concentration parameter of the wrapped Cauchy distribution for turning angle, and bounds on the shape parameters of the beta distribution for dive activity specifically chosen to prevent any "bathtub" shaped distributions for the proportion of each time step spent diving below 1.5m.

Before we demonstrate how to implement these constraints, we should first provide more detail on exactly how the `DM`, `userBounds`, and `workBounds` arguments work together in `fitHMM` (and `MIfitHMM`). While the `DM` argument should now be familiar, we have thus far spent little time discussing the latter two arguments. The `userBounds` argument specifies the lower and upper bounds for the natural scale parameters as a 2-column matrix. By default all working scale parameters ($\boldsymbol{\beta}_\theta$) are bounded on the real line, but the `workBounds` argument can be used to specify the lower and upper bounds for the working scale parameters as a 2-column matrix. Specifically, `momentuHMM` calculates natural scale parameters with finite bounds as

$$\boldsymbol{\theta} \;=\; (\mathbf{U}_\theta - \mathbf{L}_\theta)\, g^{-1}\left(\mathbf{X}_\theta \boldsymbol{\beta}^*_\theta\right) + \mathbf{L}_\theta,$$

where $\mathbf{L}_\theta$ is the lower bound on the natural scale and $\mathbf{U}_\theta$ is the upper bound on the natural scale. Note that $\boldsymbol{\beta}^*_\theta = \boldsymbol{\beta}_\theta$ under the default values for `workBounds`. For natural

scale parameters with finite lower bounds and infinite upper bounds, we have

$$\boldsymbol{\theta} = g^{-1}\left(\mathbf{X}_\theta \boldsymbol{\beta}^*_\theta\right) + \mathbf{L}_\theta.$$

When `workBounds` is specified, then additional link functions are used on the working scale parameters. For example, for working scale parameters with finite bounds, we have

$$\boldsymbol{\beta}^*_\theta \;\; = \;\; \left(\mathbf{U}_{\beta_\theta} - \mathbf{L}_{\beta_\theta}\right) \text{logit}^{-1}\left(\boldsymbol{\beta}_\theta\right) + \mathbf{L}_{\beta_\theta},$$

where $\mathbf{L}_{\beta_\theta}$ is the lower bound on the working scale and $\mathbf{U}_{\beta_\theta}$ is the upper bound on the working scale. When constraining working parameters with finite lower bounds (e.g. zero) and infinite upper bounds, we have

$$\boldsymbol{\beta}^*_\theta = \exp\left(\boldsymbol{\beta}_\theta\right) + \mathbf{L}_{\beta_\theta}.$$

When constraining working parameters with infinite lower bounds and finite upper bounds (e.g. zero), we have

$$\boldsymbol{\beta}^*_\theta = -\left(\exp\left(-\boldsymbol{\beta}_\theta\right) - \mathbf{U}_{\beta_\theta}\right).$$

Although optimization within `fitHMM` and `MIfitHMM` is always performed on $\boldsymbol{\beta}_\theta$, note that $\boldsymbol{\beta}^*_\theta$ (and a delta method approximation for the variance of this transformation) is returned by `CIbeta`, `MIpool`, and `print` function calls.

For the Weibull distribution parameters for step length, McClintock *et al.* (2013) constrained the shape parameter $a_s < 5$ (to prevent too "peaked" distributions) and the scale parameter less than the maximum distance a harbour seal could travel in 2 hours at 2 m/s (i.e., $b_s < 14400$m) for $s \in \{1 = \text{"resting"}, 2 = \text{"foraging"}, 3 = \text{"transit"}\}$. They also constrained $b_3 \geq b_2 \geq b_1$. This is easily accomplished in `momentuHMM` using the `DM`, `userBounds`, and `workBounds` arguments:

```
nbStates <- 3
stateNames <- c("resting", "foraging", "transit")
dist <- list(step = "weibull", angle = "wrpcauchy", dive = "beta")
stepDM<-matrix(c(1,0,0,0,0,0,0,0,0,
                 0,1,0,0,0,0,0,0,0,
                 0,0,1,0,0,0,0,0,0,
                 0,0,0,1,0,0,0,0,0,
```

```
                0,0,0,1,1,0,0,0,0,
                0,0,0,1,1,1,0,0,0,
                0,0,0,0,0,0,1,0,0,
                0,0,0,0,0,0,0,1,0,
                0,0,0,0,0,0,0,0,1),nrow=3*nbStates,byrow=TRUE,
        dimnames=list(c(paste0("shape_",1:nbStates),
                        paste0("scale_",1:nbStates),
                        paste0("zeromass_",1:nbStates)),
                      c(paste0("shape_",1:nbStates,":(Intercept)"),
                        "scale:(Intercept)","scale_2","scale_3",
                        paste0("zeromass_",1:nbStates,":(Intercept)"))))
stepworkBounds<-matrix(c(rep(-Inf,4),0,0,rep(-Inf,3),
                   rep(Inf,ncol(stepDM))),ncol(stepDM),2,
                   dimnames=list(colnames(stepDM),c("lower","upper")))
stepBounds<-matrix(c(0,5,
                     0,5,
                     0,5,
                     0,14400,
                     0,14400,
                     0,14400,
                     0,1,
                     0,1,
                     0,1),nrow=3*nbStates,byrow=TRUE,
                    dimnames=list(rownames(stepDM),c("lower","upper")))
```

When included in `workBounds` and `userBounds` for the step length data stream, 'stepworkBounds' and 'stepBounds' above constrain the parameters $\beta^*_{l,5} > 0$ and $\beta^*_{l,6} > 0$ such that $14400 \geq b_3 \geq b_2 \geq b_1 \geq 0$:

$$\begin{aligned} b_1 &= (14400 - 0)\,\text{logit}^{-1}\left(\beta_{l,4}\right) + 0 \\ b_2 &= (14400 - 0)\,\text{logit}^{-1}\left(\beta_{l,4} + \exp(\beta_{l,5}) + 0\right) + 0 \\ b_3 &= (14400 - 0)\,\text{logit}^{-1}\left(\beta_{l,4} + \exp(\beta_{l,5}) + 0 + \exp(\beta_{l,6}) + 0\right) + 0. \end{aligned}$$

In order to force the "transit" state to have strong directional persistence, McClintock *et al.* (2013) constrained the concentration parameter $\rho_3 > 0.75$. In the absence of relational constraints, `userBounds` could be used to constrain $\rho_3 > 0.75$. However, because we also wish to constrain $\rho_2 \leq \rho_3$, we must make use of the `workBounds` argument. We can constrain $\rho_3 \geq 0.75$, $\rho_2 \leq \rho_3$, and $\rho_s \leq 0.95$ ($s \in \{1, 2, 3\}$) using the following combination of `DM`, `userBounds`, and `workBounds` arguments:

```
angleDM <- matrix(c(1,0,0,
                    0,1,1,
                    0,1,0),nrow=nbStates,byrow=TRUE,
                  dimnames=list(paste0("concentration_",1:nbStates),
                                c("concentration_1:(Intercept)",
                                  "concentration_23:(Intercept)",
                                  "concentration_2")))
angleBounds <- matrix(c(0,0.95,
                        0,0.95,
                        0,0.95),nrow=nbStates,byrow=TRUE,
                      dimnames=list(rownames(angleDM),c("lower","upper")))
transitcons <- stats::qlogis((0.75 - angleBounds[3,1])
                             /(angleBounds[3,2] - angleBounds[3,1]))
angleworkBounds <- matrix(c(-Inf,transitcons,-Inf,
                            rep(Inf,2),0),ncol(angleDM),2,
                          dimnames=list(colnames(angleDM),c("lower","upper")))
```

When 'angleworkBounds' and 'angleBounds' are respectively included in `workBounds` and `userBounds` for the turning angle data stream, this yields

$$\begin{aligned}
\rho_1 &= (0.95-0)\,\text{logit}^{-1}(\beta_{\phi,1}) + 0 \\
\rho_2 &= (0.95-0)\,\text{logit}^{-1}(\exp(\beta_{\phi,2}) + 1.32 - (\exp(-\beta_{\phi,3}) - 0)) + 0 \\
\rho_3 &= (0.95-0)\,\text{logit}^{-1}(\exp(\beta_{\phi,2}) + 1.32) + 0.
\end{aligned}$$

Here we constrained $\rho_s \leq 0.95$ to avoid numerical convergence issues that can arise with sparse data sets as $\rho_s \to 1$. Also note that when using `workBounds` to enforce a specific constraint on the natural scale, `userBounds` should not also include the corresponding natural parameter constraint(s). For example, because we are constraining $\rho_3 \geq 0.75$ with the `workBounds` argument, we did not also include a 0.75 lower bound for ρ_3 in 'angleBounds' above.

For the beta distribution of the dive activity data, McClintock *et al.* (2013) constrained the shape1 (υ_s) and shape2 (δ_s) parameters as follows:

$$\begin{aligned}
1 \leq \upsilon_1 \leq \delta_1 \leq 10 \\
1 \leq \delta_2 \leq \upsilon_2 \leq 10
\end{aligned}$$

where $\upsilon_2 = \upsilon_3$ and $\delta_2 = \delta_3$. These constraints can be imposed using the following combination of `DM`, `userBounds`, and `workBounds` arguments:

```
omegaDM <- matrix(c(1,0,0,0,0,0,
                    0,0,1,1,0,0,
                    0,0,1,1,0,0,
                    1,1,0,0,0,0,
                    0,0,1,0,0,0,
                    0,0,1,0,0,0,
                    0,0,0,0,1,0,
                    0,0,0,0,0,1,
                    0,0,0,0,0,1),nrow=nbStates*3,byrow=TRUE,
                  dimnames=list(c(paste0("shape1_",1:nbStates),
                                  paste0("shape2_",1:nbStates),
                                  paste0("zeromass_",1:nbStates)),
                                c("shape_1:(Intercept)","shape2_1",
                                  "shape_2:(Intercept)","shape1_2",
                                  "zeromass_1:(Intercept)",
                                  "zeromass_23:(Intercept)")))
omegaworkBounds <- matrix(c(-Inf,0,-Inf,0,-Inf,-Inf,
                            rep(Inf,ncol(omegaDM))),ncol(omegaDM),2,
                          dimnames=list(colnames(omegaDM),c("lower","upper")))
omegaBounds <- matrix(c(1,10,
                        1,10,
                        1,10,
                        1,10,
                        1,10,
                        1,10,
                        0,1,
                        0,1,
                        0,1),nrow=nbStates*3,byrow=TRUE,
                      dimnames=list(rownames(omegaDM),c("lower","upper")))
```

Lastly, we wish to impose some biologically-meaningful constraints on the zero-inflation parameters using the `fixPar` argument. Because we would never expect a harbour seal in the "transit" state to exhibit a step length of zero, it makes sense to constrain the zero-mass step length parameter for the "transit" state to (effectively) zero. Similarly, we would not expect a harbour seal in the "foraging" or "transit" states to exhibit no dive activity, and it therefore also makes sense to constrain the zero-mass dive activity parameters for these states to zero. We can accomplish this using the following `fixPar` argument:

```
fixPar <- list(step=c(rep(NA,nbStates*2),NA,NA,stats::qlogis(1.e-100)),
               omega=c(rep(NA,4),NA,stats::qlogis(1.e-100)))
```

Putting it all together, we can fit our constrained model assuming no individual- or sex-level effects using `MIfitHMM`:

```
DM <- list(step = stepDM, angle = angleDM, omega = omegaDM)
userBounds <- list(step = stepBounds,
                   angle = angleBounds,
                   omega = omegaBounds)
workBounds <- list(step = stepworkBounds,
                   angle = angleworkBounds,
                   omega = omegaworkBounds)
hsFits <- MIfitHMM(crwOut, nSims = 30,
                   nbStates = nbStates, dist = dist, Par0 = Par0,
                   DM = DM, workBounds = workBounds,
                   userBounds = userBounds, workBounds = workBounds,
                   fixPar = fixPar, stateNames = stateNames)
```

As was mentioned earlier, we found overwhelming AIC support for the individual-level effects model relative to the sex-level effects models and the null model above. While our best supported `momentuHMM` model included individual-level fixed effects, the estimated tracks (Fig. 10) and inferences about population-level activity budgets were similar to the individual-level random effects model of McClintock *et al.* (2013). Estimated activity budgets for the males were 0.36 (95% CI: $0.35 - 0.37$) for "resting", 0.53 (95% CI: $0.52 - 0.55$) for "foraging", and 0.11 (95% CI: $0.1 - 0.12$) for "transit". Activity budgets for females were 0.3 (95% CI: $0.29 - 0.3$) for "resting", 0.61 (95% CI: $0.6 - 0.62$) for "foraging", and 0.09 (95% CI: $0.09 - 0.1$) for "transit". We found considerable individual variation in the state transition probabilities (Fig. 11), and when comparing the estimated activity budgets of our analysis with those of McClintock *et al.* (2013), we suspect the more noticeable differences between the time spent in the "resting" and "foraging" states for males is attributable to our having included individual-level effects on the state transition probabilities.

3.8 Northern fulmars

Using Bayesian analysis methods, Pirotta *et al.* (2018) fit a 6-state biased random walk model to northern fulmar (*Fulmarus glacialis*) tracks in northern Scotland, UK. These

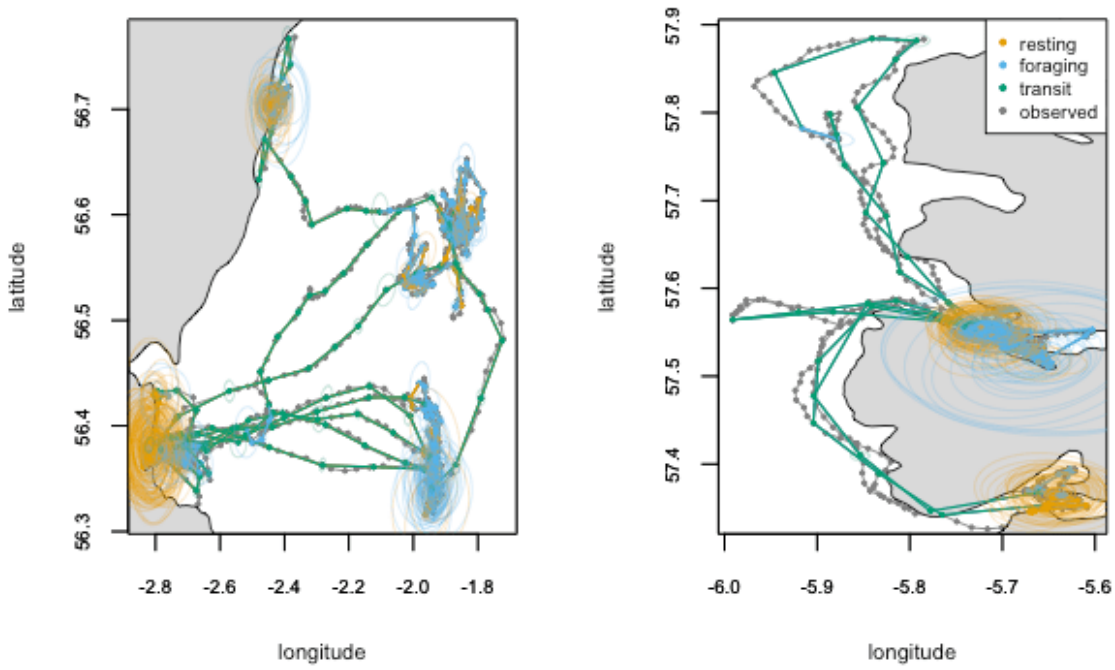


Figure 10. *Two harbour seal tracks, colored by the most likely state sequence.*

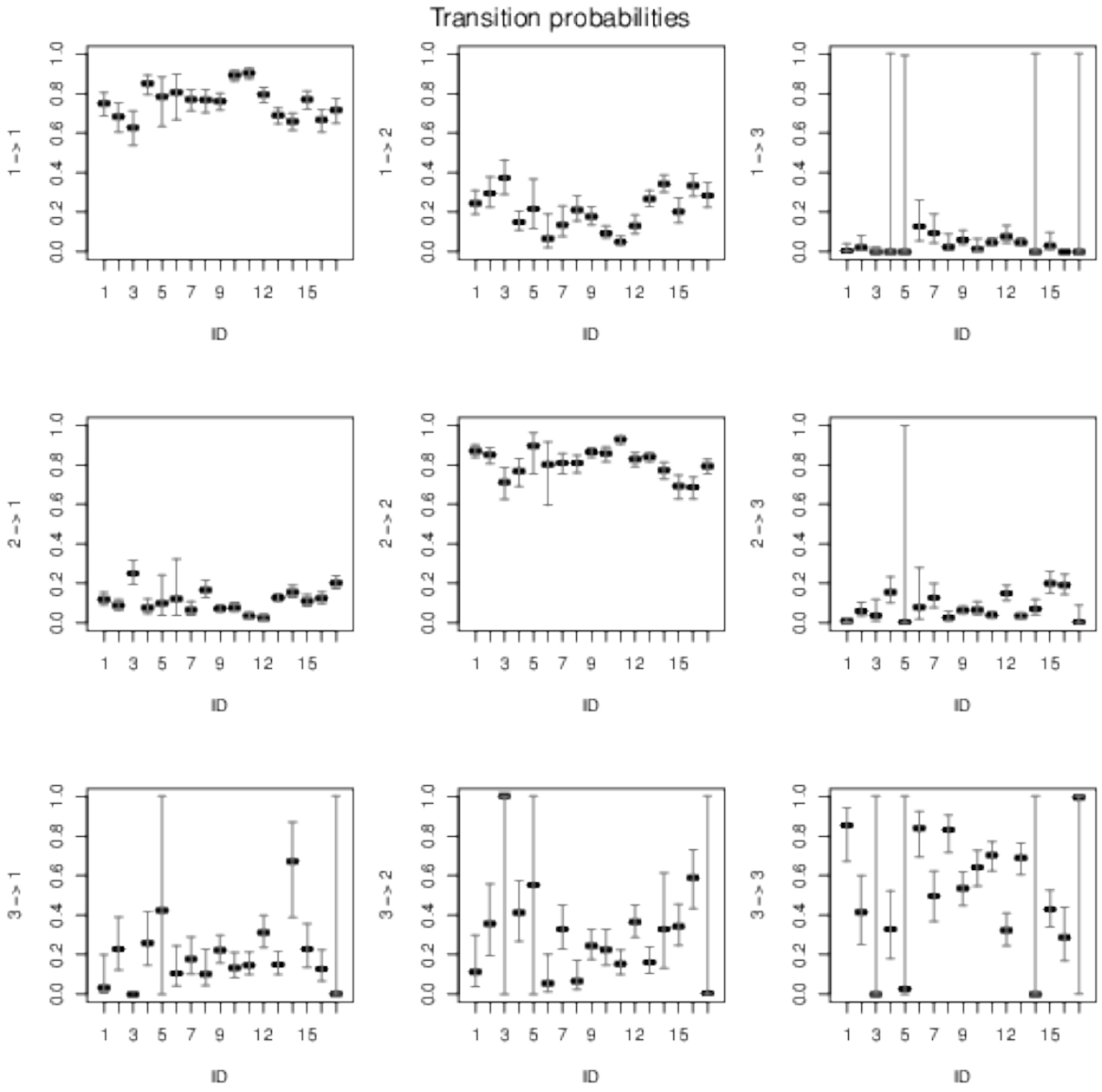


Figure 11. *Estimated individual-level state transition probabilities for the harbour seal example.*

states included biased movements relative to a colony in Eynhallow (59.12° N, 3.1° W) and fishing vessels that frequently work in the area. Pirotta *et al.* (2018) framed their model as having two latent state process. Under the first state process, the direction of movement could be biased away from the colony ("sea"), towards the nearest fishing vessel ("boat"), or towards the colony ("colony"). Under the second state process, the movement mode could either be fast and directionally-persistent ("transit") or area-restricted search ("ARS"). Thus the six states are "sea ARS", "sea transit", "boat ARS", "boat transit", "colony ARS" and "colony transit". Pirotta *et al.* (2018) also allowed the distance to the nearest fishing vessel and time since leaving the colony to affect state transitions to the "boat" and "colony" states. Here we demonstrate how a very similar (but not identical) model can be implemented in `momentuHMM`.

The data are provided in a Dryad repository, but these will require some additional formatting and preparation. We first load the raw data, create the required "ID" column based on individual trips, convert time stamps to class `POSIXct`, and project the northern fulmar ("Longitude", "Latitude") and nearest fishing vessel ("Boat_Longitude", "Boat_Latitude") locations using the `sp` package:

```
library(sp)

# load data provided by Pirotta et al
fulmarURL <- "https://datadryad.org/stash/downloads/file_stream/45899"
raw_data <- read.csv(url(fulmarURL),
                     stringsAsFactors = FALSE)

raw_data$ID <- raw_data$tripID
raw_data$Date <- as.POSIXct(raw_data$Date,tz="UTC",
                            format="%d/%m/%Y %H:%M")

# project data
oldProj <- CRS("+proj=longlat +datum=WGS84")
newProj <- CRS("+init=epsg:27700")
coordinates(raw_data) <- c("Longitude","Latitude")
proj4string(raw_data) <- oldProj
raw_data <- as.data.frame(spTransform(raw_data, newProj))

coordinates(raw_data) <- c("Boat_Longitude","Boat_Latitude")
proj4string(raw_data) <- oldProj
raw_data <- as.data.frame(spTransform(raw_data, newProj))
```

For movements away from the colony ("sea ARS" and "sea transit"), Pirotta *et al.* (2018) included bias in the direction of the farthest location from the colony for a given trip. We can use the `centers` argument of `prepData` to identify these locations ("max_dist") and then calculate the expected angle for the "sea" states ("sea.angle") using `momentuHMM:::distAngle`:

```
# use prepData to calculate colony distance covariate ('sea.angle')
colony <- data.frame(x = -3.1, y = 59.12)
coordinates(colony) <- c("x", "y")
proj4string(colony) <- oldProj
colony <- as.matrix(as.data.frame(spTransform(colony, newProj)))
rownames(colony) <- "colony"
colony_dist <- prepData(raw_data, coordNames = c("Longitude","Latitude"),
                        centers = colony)

# calculate "sea" mean angle covariate
sea.angle <- NULL
for(id in unique(colony_dist$ID)) {
  idat <- subset(colony_dist,ID==id)
  nbSubObs <- length(which(colony_dist$ID==id))
  max_dist <- as.numeric(idat[which.max(idat$colony.dist),c("x","y")])
  max_angle <- momentuHMM:::distAngle(colony,colony,max_dist)[2]
  sea.angle <- c(sea.angle, rep(max_angle,nbSubObs))
}
raw_data$sea.angle <- sea.angle
```

Next we calculate the time since leaving colony covariate ("time"):

```
# calculate time since left colony covariate ('time')
time <- aInd <- NULL
for(id in unique(raw_data$ID)) {
  idInd <- which(raw_data$ID==id)
  aInd <- c(aInd,idInd[1])
  nbSubObs <- length(idInd)
  time <- c(time, (1:nbSubObs)/nbSubObs)
}
raw_data$time <- time
```

To complete our data preparation, we convert the nearest fishing vessel data to the `centriods` argument format and use `prepData` to calculate step lengths, turn angles, and our "sea", "boat", and "colony" covariates:

```
# get boat data into centroids argument format
boat_data <- list(boat=data.frame(Date = raw_data$Date,
                                  x = raw_data$Boat_Longitude,
                                  y = raw_data$Boat_Latitude))

# format and merge all data and covariates for analysis
fulmar_data <- prepData(raw_data, coordNames = c("Longitude","Latitude"),
                        centers = colony,
                        centroids = boat_data,
                        covNames = "time",
                        angleCovs = "sea.angle")

# momentuHMM doesn't like data streams and covariates to have same name,
# so create identical data column with different name
fulmar_data$d <- fulmar_data$boat.dist

# standarize boat.dist covariate
fulmar_data$boat.dist <- scale(fulmar_data$boat.dist)
```

Note that we use the `centers` argument for the colony (because its location is static) and the `centroids` argument for the nearest fishing vessel (because its location is dynamic).

Now that we've formatted the data, we're ready to specify the 6-state HMM. Using 10 min time steps, Pirotta *et al.* (2018) included three data streams in their model: step length ("step"), turn angle ("angle"), and distance to nearest boat ("d"). These were respectively modelled using Weibull, wrapped Cauchy, and log-normal distributions:

```
nbStates <- 6

stateNames <- c("seaARS", "seaTr",
                "boatARS", "boatTr",
                "colonyARS", "colonyTr")

dist <- list(step = "weibull",
             angle = "wrpcauchy",
             d = "lnorm")
```

Similar to the harbour seal example (section 3.7), Pirotta *et al.* (2018) used relational parameter constraints that can be specified in `momentuHMM` using pseudo-design matrices:

```
# specify data stream probability distribution parameter constraints
stepDM <- matrix(c(1,0,0,0,
                   0,1,0,0,
                   1,0,0,0,
                   0,1,0,0,
                   1,0,0,0,
                   0,1,0,0,
                   0,0,1,0,
                   0,0,1,1,
                   0,0,1,0,
                   0,0,1,1,
                   0,0,1,0,
                   0,0,1,1),2*nbStates,4,byrow=TRUE,
                 dimnames=list(c(paste0("shape_",1:nbStates),
                                 paste0("scale_",1:nbStates)),
                               c("shape:ARS","shape:Tr",
                                 "scale:(Intercept)","scale:Tr")))

# constrain scale parameters such that Tr > ARS
stepworkBounds <- matrix(c(-Inf,Inf,
                           -Inf,Inf,
                           -Inf,Inf,
                           0,Inf),ncol(stepDM),2,byrow=TRUE,
                         dimnames=list(colnames(stepDM),c("lower","upper")))

# include trip-level effects on angle mean concentration parameter
nbTrips <- length(unique(fulmar_data$ID))
angleDM <- matrix(c("sea.angle",0,0,0,0,rep(0,2*nbTrips),
                    "sea.angle",0,0,0,0,rep(0,2*nbTrips),
                    0,"boat.angle",0,0,0,rep(0,2*nbTrips),
                    0,"boat.angle",0,0,0,rep(0,2*nbTrips),
                    0,0,"colony.angle",0,0,rep(0,2*nbTrips),
                    0,0,"colony.angle",0,0,rep(0,2*nbTrips),
                    0,0,0,1,0,paste0("ID",1:nbTrips),rep(0,nbTrips),
                    0,0,0,1,1,paste0("ID",1:nbTrips),paste0("ID",1:nbTrips),
                    0,0,0,1,0,rep(0,2*nbTrips),
                    0,0,0,1,1,rep(0,2*nbTrips),
                    0,0,0,1,0,rep(0,2*nbTrips),
                    0,0,0,1,1,rep(0,2*nbTrips)),2*nbStates,3+2+2*nbTrips,byrow=TRUE,
                  dimnames=list(c(paste0("mean_",1:nbStates),
                                  paste0("concentration_",1:nbStates)),
                                c("mean:sea","mean:boat","mean:colony",
```

```
                                    "concentration:(Intercept)","concentration:Tr",
                                    paste0("concentration:ID",1:nbTrips,":(Intercept)"),
                                    paste0("concentration:ID",1:nbTrips,":Tr"))))

# constrain concentration parameters such that Tr > ARS
angleworkBounds <- matrix(c(-Inf,Inf,
                            -Inf,Inf,
                            -Inf,Inf,
                            -Inf,Inf,
                            0,Inf,
                            rep(c(-Inf,Inf),nbTrips),
                            rep(c(0,Inf),nbTrips)),ncol(angleDM),2,byrow=TRUE,
                          dimnames=list(colnames(angleDM),c("lower","upper")))

dDM <- matrix(c(1,1,0,0,
                1,1,0,0,
                1,0,0,0,
                1,0,0,0,
                1,1,0,0,
                1,1,0,0,
                0,0,1,1,
                0,0,1,1,
                0,0,1,0,
                0,0,1,0,
                0,0,1,1,
                0,0,1,1),2*nbStates,4,byrow=TRUE,
              dimnames=list(c(paste0("location_",1:nbStates),
                              paste0("scale_",1:nbStates)),
                            c("location:(Intercept)","location:noboat",
                              "scale:(Intercept)","scale:noboat")))

# constrain location and scale parameters such that sea and colony > boat
dworkBounds <- matrix(c(-Inf,Inf,
                        0,Inf,
                        -Inf,Inf,
                        0,Inf),ncol(dDM),2,byrow=TRUE,
                      dimnames=list(colnames(dDM),c("lower","upper")))

DM <- list(step = stepDM, angle = angleDM, d = dDM)

workBounds <- list(step = stepworkBounds,
                   angle = angleworkBounds,
```

```
                    d = dworkBounds)
```

To complete our model specification, we use the `toState` special function to model transitions to the "boat" and "colony" states as a function of distance to nearest fishing vessel ("boat.dist") and time since leaving colony ("time"), respectively. Following Pirotta *et al.* (2018), we will use the `knownStates` argument to fix the initial state to "sea transit". We will also use the `fixPar` argument to fix the initial state probabilities (because we are assuming these are known) and, as in section 3.6, constrain the model to a biased random walk by fixing the mean angle working scale parameters to a large positive value:

```
# state transition formula similar to Pirotta et al
formula <- ~ toState3(boat.dist) + toState4(boat.dist) +
             toState5(time) + toState6(time)

# specify knownStates
# Priotta et al assumed all animals start in state 2 ('seaTr')
knownStates <- rep(NA,nrow(fulmar_data))
knownStates[aInd] <- 2

# fix delta_2 = 1 because assuming initial state is known for each track
fixPar <- list(delta=c(100,rep(0,nbStates-2)))
fixPar$delta <- exp(c(0,fixPar$delta))/sum(exp(c(0,fixPar$delta)))
# Constrain model to BRW (instead of BCRW)
fixPar$angle <- c(rep(1.e+7, 3), rep(NA, 2+2*nbTrips))
```

Lastly, we used the `betaCons` argument to impose similar constraints as Pirotta *et al.* (2018) for the transition probability parameters. We accomplish this by using `betaCons` to set the transition probability working parameter intercept terms equal among the three "ARS" states and the three "transit" states. We also use `betaCons` to constrain the effects of 'boat.dist' and 'time' to be identical for each of the two movement modes ("ARS" and "transit") within each of the three biased movement states ("sea", "boat", and "colony"). `betaCons` must be a matrix of the same dimension as `beta0` and be composed of integers, where each beta working parameter is sequentially indexed in a column-wise fashion. Equality constraints can then be incorporated by having parameters share the same index. In this example we have:

```
betaCons

##             1 -> 2 1 -> 3 1 -> 4 1 -> 5 1 -> 6 2 -> 1 2 -> 3 2 -> 4 2 -> 5
## (Intercept)      1      4      1      4      1     16     16     22     16
## boat.dist        2      5      5     11     14     17      5      5     26
## time             3      6      9     12     12     18     21     24     12
##             2 -> 6 3 -> 1 3 -> 2 3 -> 4 3 -> 5 3 -> 6 4 -> 1 4 -> 2 4 -> 3
## (Intercept)     22      4      1      1      4      1     16     22     16
## boat.dist       29     32     35     38     41     44     47     50     38
## time            12     33     36     39     42     42     48     51     54
##             4 -> 5 4 -> 6 5 -> 1 5 -> 2 5 -> 3 5 -> 4 5 -> 6 6 -> 1 6 -> 2
## (Intercept)     16     22      4      1      4      1      1     16     22
## boat.dist       56     59     62     65     68     68     74     77     80
## time            42     42     63     66     69     72     75     78     81
##             6 -> 3 6 -> 4 6 -> 5
## (Intercept)     16     22     16
## boat.dist       68     68     89
## time            84     87     75
```

Again, `betaCons` constrains any of the transition probability matrix working parameters with the same index to be equal to one another. For example, all of the intercept terms indexed by a '1' ($1 \to 2$, $1 \to 4$, $1 \to 6$, $3 \to 2$, $3 \to 4$, $3 \to 6$, $5 \to 2$, $5 \to 4$, and $5 \to 6$) are equal, and these terms correspond to the transitions from the "ARS" movement mode (states 1, 3, and 5) to the "transit" movement mode (states 2, 4, and 6). Similarly, all of the intercept terms indexed by a '16' are equal and correspond to the transitions from the "transit" movement mode to the "ARS" movement mode. For further details on the `betaCons` argument, see the `fitHMM` help file and the northern fulmar example code in the "vignettes" source directory.

Now we are ready to fit our 6-state HMM:

```
m2 <- fitHMM(fulmar_data, nbStates, dist,
             Par0 = Par0$Par, beta0 = Par0$beta0,
             formula = formula,
             estAngleMean = list(angle = TRUE),
             circularAngleMean = list(angle = TRUE),
             DM = DM, workBounds = workBounds, betaCons = betaCons,
             fixPar = fixPar, knownStates = knownStates,
             stateNames = stateNames)
```

With decent starting values, this model required about 2 min to fit on a standard desktop computer (macOS El Capitan, 2.8 GHz Intel Core i7, 16 GB RAM). For com-

parison, Pirotta *et al.* (2018) required about 18 hr to fit their Bayesian model using MCMC (2.9 GHz Intel Core i7, 16 GB RAM).

We can compare the estimated activity budgets with those of Pirotta *et al.* (2018) using the `timeInStates` function:

```
timeInStates(m2)
```

```
##      seaARS     seaTr   boatARS     boatTr colonyARS  colonyTr
## 1 0.2582469 0.1687088 0.1705938 0.06503299 0.1677663 0.1696513
```

Pirotta *et al.* (2018) estimated 0.28 ("seaARS"), 0.20 ("seaTr"), 0.18 ("boatARS"), 0.06 ("boatTr"), 0.12 ("colonyARS"), and 0.16 ("colonyTr"). While these are very similar, there a handful of state assignments that differ between the analyses. These differences could be attributable to several factors, including: 1) the use of informative priors in the Bayesian analysis of Pirotta *et al.* (2018); 2) our use of fixed trip-level effects on the "sea" state turn angle concentration parameters; and 3) Pirotta *et al.* (2018) assumed the state transition probability covariates ("boat.dist" and "time") only affected the movement direction states ("sea", "boat", "colony"), but in our `momentuHMM` implementation the covariates can affect state transitions for both the movement direction ("sea", "boat", "colony") and the movement mode ("ARS", "transit").

We can also examine activity budgets by individual bird (which are indexed in the raw data "birdID" column), where it is clear that the first 3 individuals tended to spend a larger proportion of their foraging trips in the "boat" states:

```
timeInStates(m2, by = "birdID")
```

```
##   birdID     seaARS       seaTr    boatARS     boatTr  colonyARS
## 1      1 0.35564854 0.117154812 0.22594142 0.11297071 0.09623431
## 2      2 0.14102564 0.134615385 0.28205128 0.10256410 0.25641026
## 3      3 0.08333333 0.269607843 0.35294118 0.06862745 0.04411765
## 4      4 0.49019608 0.196078431 0.00000000 0.04575163 0.05882353
## 5      5 0.61475410 0.008196721 0.00000000 0.00000000 0.30327869
## 6      6 0.00000000 0.235294118 0.05882353 0.02673797 0.32085561
##     colonyTr
## 1 0.09205021
```

```
## 2 0.08333333
## 3 0.18137255
## 4 0.20915033
## 5 0.07377049
## 6 0.35828877
```

Finally, we can create a plot similar to Pirotta *et al.* (2018) using the `plotSat` function (Figure 12):

```
plotSat(m2, zoom = 7, shape = c(17,1,17,1,17,1), size = 2,
        col = rep(c("#E69F00", "#56B4E9", "#009E73"), each = 2),
        stateNames = c("sea ARS", "sea Transit",
                       "boat ARS", "boat Transit",
                       "colony ARS", "colony Transit"),
        projargs = newProj, ask = FALSE)
```

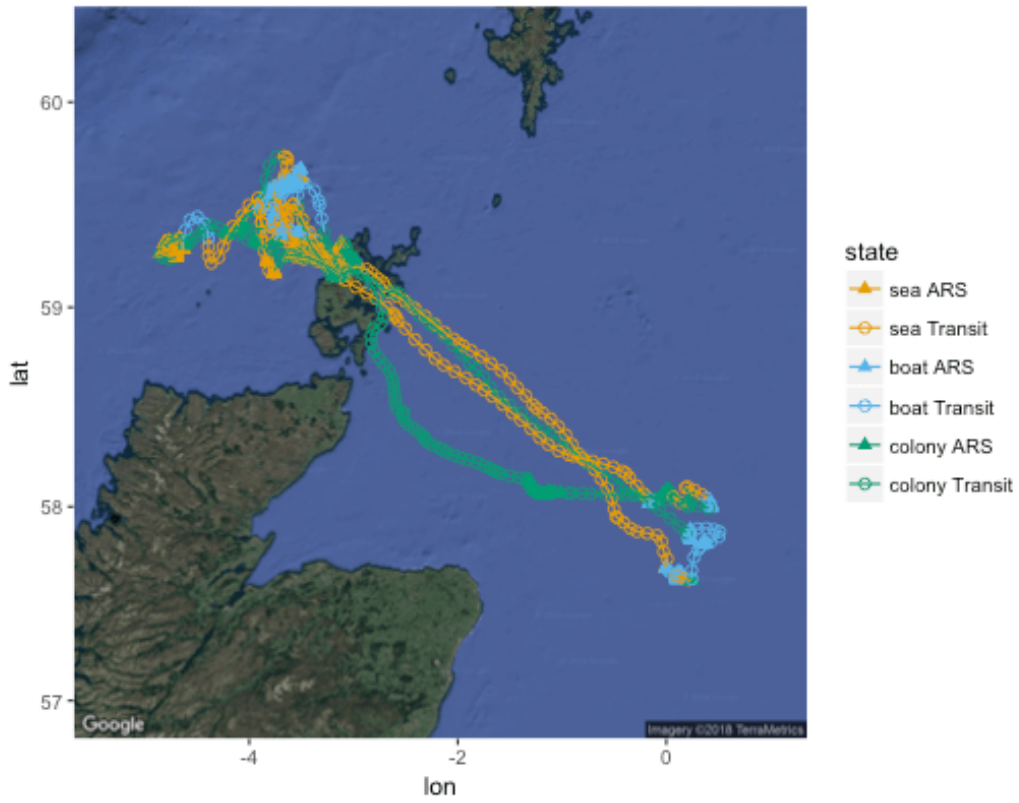


Figure 12. *Seven northern fulmar tracks, colored by the most likely state sequence.*

3.9 Pilot whales

Here we demonstrate how to include individual heterogeneity in state-switching dynamics via discrete-valued random effects by fitting the 4-state ("exploratory", "foraging", "crowded", "directed") mixed HHM for long-finned pilot whales described in Isojunno *et al.* (2017). The pilot whale data consist of 11 data streams collected from 15 individuals, and, as usual, we must first prepare the data for `fitHMM` using `prepData`. Let's summarize our prepared data using the `summary` function:

```
summary(pilotData, dataNames=names(pilotData)[-1])
```

```
## HMM data for 15 individuals:
##
## gm08_150c -- 156 observations
## gm08_154d -- 142 observations
## gm08_159a -- 192 observations
## gm09_137b -- 152 observations
## gm09_138a -- 223 observations
## gm09_156b -- 254 observations
## gm10_000a -- 119 observations
## gm10_143a -- 146 observations
## gm10_152b -- 50 observations
## gm10_157b -- 130 observations
## gm10_158d -- 148 observations
## gm13_137a -- 144 observations
## gm13_149a -- 88 observations
## gm13_169a -- 190 observations
## gm14_180a -- 180 observations
##
##
## Data summaries:
##
##     dive.dur        dive.depth        GR.speed2      dive.pitchvar2
##  Min.   : 0.14   Min.   :  0.86   Min.   :0.025   Min.   :0.010
##  1st Qu.: 0.64   1st Qu.:  5.36   1st Qu.:0.838   1st Qu.:0.042
##  Median : 0.81   Median :  7.08   Median :1.272   Median :0.070
##  Mean   : 1.40   Mean   : 26.36   Mean   :1.367   Mean   :0.100
##  3rd Qu.: 1.41   3rd Qu.: 11.73   3rd Qu.:1.732   3rd Qu.:0.121
##  Max.   :13.75   Max.   :617.37   Max.   :8.824   Max.   :0.673
##                                   NA's   :892     NA's   :142
##  breath.headchange     GR.size        GR.tight     dive.CS.pres
```

```
##  Min.   :-3.12587   Min.   : 0    Min.   :0.00   Min.   :0.0
##  1st Qu.:-0.19089   1st Qu.: 6    1st Qu.:1.00   1st Qu.:0.0
##  Median : 0.00066   Median : 9    Median :1.00   Median :1.0
##  Mean   :-0.00407   Mean   :11    Mean   :0.82   Mean   :0.6
##  3rd Qu.: 0.18903   3rd Qu.:13    3rd Qu.:1.00   3rd Qu.:1.0
##  Max.   : 3.05424   Max.   :79    Max.   :1.00   Max.   :1.0
##  NA's   :261        NA's   :665   NA's   :1128   NA's   :905
##   dive.SS.pres     presurf        postsurf
##  Min.   :0.00   Min.   :0.00   Min.   :0.00
##  1st Qu.:0.00   1st Qu.:0.00   1st Qu.:0.00
##  Median :1.00   Median :1.00   Median :1.00
##  Mean   :0.56   Mean   :0.74   Mean   :0.74
##  3rd Qu.:1.00   3rd Qu.:1.00   3rd Qu.:1.00
##  Max.   :1.00   Max.   :1.00   Max.   :1.00
##  NA's   :905    NA's   :15     NA's   :15
```

After specifying the data stream probability distributions and starting values for the parameters (based on those reported by Isojunno *et al.* 2017), let's first fit the null model with no discrete-valued individual-level random effects on the state-switching dynamics:

```
## 11 data streams
dist <- list(dive.dur = "weibull",
             dive.depth = "gamma",
             GR.speed2 = "gamma",
             dive.pitchvar2 = "beta",
             breath.headchange = "vm",
             GR.size = "pois",
             GR.tight = "bern",
             dive.CS.pres = "bern",
             dive.SS.pres = "bern",
             presurf = "bern",
             postsurf = "bern")

## initial values
Par0 <- list(dive.dur = c(1.9, 2.72, 1.64, 4.21,
                          1.3, 8.14, 1.53, 0.79),
             dive.depth = c(10.23, 315.91, 10.74, 5.51,
                            4.93, 233.12,  6.32, 1.91),
             GR.speed2 = c(1.15, 1.32, 1.36, 1.57,
                           0.66, 0.51, 0.77, 0.76),
```

```
             dive.pitchvar2 = c( 2.21, 2.88,  1.82,  3.18,
                                 17.94, 6.04, 16.06, 55.36),
             breath.headchange = c(3.07, 5.65, 2.64, 18.02),
             GR.size = c(6, 7.39, 20.39, 9.52),
             GR.tight = c(0.89, 0.66, 0.76, 0.81),
             dive.CS.pres = c(0.76, 0.99, 0.41, 0.47),
             dive.SS.pres = c(0.72, 0.98, 0.39, 0.41),
             presurf = c(0.76, 0.99, 0.71, 0.71),
             postsurf = c(0.81, 0.96, 0.72, 0.66))

beta0 <- matrix(c(-2.38, -3.86, -1.22,
                   0.21,  -1.6, -0.47,
                  -3.56, -4.15, -2.29,
                  -1.17, -3.05, -2.54),nrow=1)

stateNames <- c("exploratory","foraging","crowded","directed")
```

Isojunno *et al.* (2017) found that model selection criteria favored models that assume the initial distribution is the stationary distribution, so we'll set `stationary=TRUE`:

```
## fit model with single mixture on TPM
fitmix1 <- fitHMM(pilotData, nbStates=4, dist=dist,
                  Par0=Par0, beta0=beta0,
                  stationary=TRUE,
                  stateNames=stateNames,
                  nlmPar=list(hessian=FALSE))
fitmix1

## Value of the maximum log-likelihood: -18510.4
##
##
## dive.dur parameters:
## --------------------
##       exploratory foraging  crowded  directed
## shape    1.895587 2.719570 1.640040 4.2099390
## scale    1.301088 8.142114 1.529318 0.7911992
##
## dive.depth parameters:
## ----------------------
##      exploratory foraging   crowded directed
## mean   10.221159 315.7408 10.752753 5.507151
## sd      4.920867 233.2394  6.325075 1.908221
```

```
##
## GR.speed2 parameters:
## --------------------
##      exploratory  foraging crowded  directed
## mean   1.1469499 1.3170625 1.36225 1.5706933
## sd     0.6602282 0.5121064 0.77157 0.7628831
##
## dive.pitchvar2 parameters:
## -------------------------
##        exploratory foraging   crowded  directed
## shape1    2.205733 2.876755  1.823291  3.186465
## shape2   17.948930 6.036487 16.067474 55.491769
##
## breath.headchange parameters:
## -----------------------------
##               exploratory foraging  crowded directed
## mean             0.000000 0.000000 0.000000  0.00000
## concentration    3.068759 5.653926 2.644184 18.04806
##
## GR.size parameters:
## -------------------
##        exploratory foraging  crowded directed
## lambda    5.996763 7.392536 20.38838 9.522725
##
## GR.tight parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.8885557 0.6629627 0.7609689 0.8072405
##
## dive.CS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7630851 0.9892991 0.4077435 0.4703789
##
## dive.SS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7167555 0.9789121 0.3955766 0.4115657
##
## presurf parameters:
## -------------------
##      exploratory  foraging crowded  directed
```

```
## prob     0.76011 0.9924097 0.70787 0.7068425
##
## postsurf parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.8081057 0.9619604 0.7164094 0.6568226
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##              1 -> 2    1 -> 3    1 -> 4    2 -> 1    2 -> 3     2 -> 4
## (Intercept) -2.3725 -3.871793 -1.230078 0.2005482 -1.623498 -0.4895434
##                3 -> 1    3 -> 2    3 -> 4    4 -> 1    4 -> 2    4 -> 3
## (Intercept) -3.543939 -4.120499 -2.272679 -1.163733 -3.035013 -2.539986
##
## Transition probability matrix:
## ------------------------------
##             exploratory   foraging    crowded   directed
## exploratory  0.71106663 0.06630503 0.01480512 0.20782321
## foraging     0.40303342 0.32979502 0.06503813 0.20213342
## crowded      0.02516983 0.01414110 0.87095004 0.08973904
## directed     0.21699916 0.03340193 0.05479726 0.69480165
##
## Initial distribution:
## ---------------------
## exploratory    foraging     crowded    directed
##  0.36640726  0.05854032  0.22161529  0.35343713
```

(note that we've set the `nlmPar` option `hessian=FALSE` simply to speed up the compile time for this vignette).

Now we'll fit a model with $K = 2$ mixtures of discrete-valued individual-level random effects by setting `mixtures=2`, but first we'll set some starting values based on the null model (using `getPar0`) and check our model specification (using `checkPar0`):

```
Par0_mix2 <- getPar0(fitmix1, mixtures=2)

## Error in !m$conditions$fit:  invalid argument type

Par0_mix2$beta$beta[1,] <- c(-2.26, -3.93, -0.58,
                              0.03, -2.25, -0.26,
                             -3.38, -4.79, -2.82,
                             -1.06,  -3.3, -3.43)
```

```
Par0_mix2$beta$beta[2,] <- c(-2.51, -3.32, -2.63,
                             0.03, -1.26, -0.12,
                            -96.8, -3.62, -1.75,
                            -1.76, -2.14, -1.38)
Par0_mix2$beta$pi <- c(0.73, 0.27)
```

Note that because `mixtures` is > 1, the starting values for `beta0` must now be specified as a list with elements named `beta` (containing the starting values for the t.p.m. parameters) and/or `pi` (containing the starting values for the mixture probability parameters).

```
# check model specification
checkPar0(pilotData, nbStates=4, dist=dist,
          Par0=Par0_mix2$Par, beta0=Par0_mix2$beta,
          stationary=TRUE,
          mixtures=2,
          stateNames=stateNames)

##
## dive.dur parameters:
## --------------------
##       exploratory foraging  crowded  directed
## shape    1.895587 2.719570 1.640040 4.2099390
## scale    1.301088 8.142114 1.529318 0.7911992
##
## dive.depth parameters:
## ----------------------
##      exploratory foraging   crowded directed
## mean   10.221159 315.7408 10.752753 5.507151
## sd      4.920867 233.2394  6.325075 1.908221
##
## GR.speed2 parameters:
## ---------------------
##      exploratory  foraging crowded  directed
## mean   1.1469499 1.3170625 1.36225 1.5706933
## sd     0.6602282 0.5121064 0.77157 0.7628831
##
## dive.pitchvar2 parameters:
## -------------------------
##        exploratory foraging   crowded  directed
## shape1    2.205733 2.876755  1.823291  3.186465
```

```
## shape2   17.948930 6.036487 16.067474 55.491769
##
## breath.headchange parameters:
## ----------------------------
##               exploratory foraging  crowded directed
## concentration    3.068759 5.653926 2.644184 18.04806
##
## GR.size parameters:
## -------------------
##        exploratory foraging  crowded directed
## lambda    5.996763 7.392536 20.38838 9.522725
##
## GR.tight parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.8885557 0.6629627 0.7609689 0.8072405
##
## dive.CS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7630851 0.9892991 0.4077435 0.4703789
##
## dive.SS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7167555 0.9789121 0.3955766 0.4115657
##
## presurf parameters:
## -------------------
##      exploratory  foraging crowded  directed
## prob     0.76011 0.9924097 0.70787 0.7068425
##
## postsurf parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.8081057 0.9619604 0.7164094 0.6568226
##
## Mixture probabilities (pi):
## ---------------------------
## mix1 mix2
## 0.73 0.27
##
```

```
## Regression coeffs for the transition probabilities (beta):
## ----------------------------------------------------------
##                   1 -> 2 1 -> 3 1 -> 4 2 -> 1 2 -> 3 2 -> 4 3 -> 1 3 -> 2
## (Intercept)_mix1  -2.26  -3.93  -0.58   0.03  -2.25  -0.26  -3.38  -4.79
## (Intercept)_mix2  -2.51  -3.32  -2.63   0.03  -1.26  -0.12 -96.80  -3.62
##                   3 -> 4 4 -> 1 4 -> 2 4 -> 3
## (Intercept)_mix1  -2.82  -1.06  -3.30  -3.43
## (Intercept)_mix2  -1.75  -1.76  -2.14  -1.38
```

We can see above that by setting `mixtures=2` we now have $K = 2$ sets of state transition probability matrix parameters, each suffixed by a mixture label (`_mix1` or `_mix2`). We also now have $K = 2$ mixture probability parameters ($\boldsymbol{\pi}$), where the first (π_1) corresponds to parameters suffixed with `_mix1` and the second (π_2) corresponds to parameters suffixed with `_mix2`. Now let's fit the $K = 2$ mixture model from Isojunno *et al.* (2017):

```
fitmix2 <- fitHMM(pilotData, nbStates=4, dist=dist,
                  Par0=Par0_mix2$Par, beta0=Par0_mix2$beta,
                  stationary=TRUE,
                  mixtures=2,
                  stateNames=stateNames, nlmPar=list(hessian=FALSE))

## ======================================================================
## Fitting HMM with 4 states and 11 data streams
## ----------------------------------------------------------------------
##  dive.dur ~ weibull(shape=~1, scale=~1)
##  dive.depth ~ gamma(mean=~1, sd=~1)
##  GR.speed2 ~ gamma(mean=~1, sd=~1)
##  dive.pitchvar2 ~ beta(shape1=~1, shape2=~1)
##  breath.headchange ~ vm(concentration=~1)
##  GR.size ~ pois(lambda=~1)
##  GR.tight ~ bern(prob=~1)
##  dive.CS.pres ~ bern(prob=~1)
##  dive.SS.pres ~ bern(prob=~1)
##  presurf ~ bern(prob=~1)
##  postsurf ~ bern(prob=~1)
```

```
##
##  Transition probability matrix formula:  ~1
##
##  Initial distribution formula:  ~1
##
##  Number of mixtures:  2
##  Mixture probability formula:  ~1
## ============================================================================
## DONE
```

```
fitmix2
```

```
## Value of the maximum log-likelihood: -18481.29
##
##
## dive.dur parameters:
## --------------------
##       exploratory foraging  crowded  directed
## shape    1.936296 2.765043 1.648316 4.0151432
## scale    1.372185 8.194463 1.525709 0.7855839
##
## dive.depth parameters:
## ----------------------
##      exploratory foraging   crowded directed
## mean    10.65323 319.2099 10.708474 5.650962
## sd       5.19198 230.5566  6.319645 1.960661
##
## GR.speed2 parameters:
## ---------------------
##      exploratory  foraging  crowded  directed
## mean   1.1379085 1.3188728 1.379888 1.5330244
## sd     0.6637213 0.5154339 0.774373 0.7589017
##
## dive.pitchvar2 parameters:
## --------------------------
##        exploratory foraging   crowded  directed
## shape1    2.156273 2.954203  1.788241  2.791914
## shape2   17.462342 6.134090 15.932112 44.398296
##
## breath.headchange parameters:
## -----------------------------
```

```
##               exploratory foraging  crowded directed
## mean              0.00000 0.000000 0.000000   0.0000
## concentration     2.86254 5.650729 2.636543  17.5883
##
## GR.size parameters:
## -------------------
##        exploratory foraging crowded directed
## lambda    5.893962 7.428444 20.5167 9.348098
##
## GR.tight parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.9054784 0.6575736 0.7645632 0.7937692
##
## dive.CS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7570289 0.9892898 0.3978535 0.5077637
##
## dive.SS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded directed
## prob   0.7246258 0.9795692 0.3849078 0.440141
##
## presurf parameters:
## -------------------
##      exploratory  foraging   crowded  directed
## prob   0.7519146 0.9923487 0.7113906 0.7172221
##
## postsurf parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.8176395 0.9621881 0.7139607 0.6647271
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                      1 -> 2    1 -> 3     1 -> 4     2 -> 1    2 -> 3
## (Intercept)_mix1 -2.255164 -3.921555 -0.5878246 0.03912368 -2.230029
## (Intercept)_mix2 -2.474720 -3.374581 -2.6435725 0.03630417 -1.202661
##                       2 -> 4     3 -> 1    3 -> 2    3 -> 4    4 -> 1
## (Intercept)_mix1 -0.2580981  -3.390892 -4.797065 -2.838810 -1.053870
## (Intercept)_mix2 -0.0930870 -96.800000 -3.526245 -1.723148 -1.730594
```

```
##                      4 -> 2    4 -> 3
## (Intercept)_mix1 -3.310025 -3.418480
## (Intercept)_mix2 -2.081930 -1.406935
##
## Mixture probabilities:
## ----------------------
##      mix1      mix2
## 0.7334777 0.2665223
##
## Transition probability matrix:
## ------------------------------
##                    exploratory    foraging    crowded   directed
## exploratory_mix1 5.951668e-01 0.062407036 0.01179041 0.33063572
## foraging_mix1    3.561367e-01 0.342472347 0.03682443 0.26456654
## crowded_mix1     3.060502e-02 0.007500662 0.90873748 0.05315684
## directed_mix1    2.458530e-01 0.025753712 0.02310673 0.70528657
## exploratory_mix2 8.406711e-01 0.070773201 0.02877825 0.05977742
## foraging_mix2    3.192174e-01 0.307836357 0.09247212 0.28047408
## crowded_mix2     7.555383e-43 0.024351947 0.82787036 0.14777769
## directed_mix2    1.145485e-01 0.080613179 0.15832611 0.64651221
##
## Initial distribution:
## ---------------------
##      exploratory   foraging   crowded  directed
## mix1   0.3283189 0.05061176 0.1756211 0.4454483
## mix2   0.3359432 0.07584579 0.3323006 0.2559104
```

Based on our fitted model, we can calculate the probability of each individual being in a particular mixture using the `mixtureProbs` function and the t.p.m. for each mixture using the `getTrProbs` function:

```
## calculate mixture probabilities for each individual
round(mixtureProbs(fitmix2),4)

##                mix1   mix2
## ID:gm08_150c 1.0000 0.0000
## ID:gm08_154d 1.0000 0.0000
## ID:gm08_159a 1.0000 0.0000
## ID:gm09_137b 1.0000 0.0000
## ID:gm09_138a 0.0004 0.9996
## ID:gm09_156b 0.9969 0.0031
## ID:gm10_000a 0.0000 1.0000
```

```
## ID:gm10_143a 1.0000 0.0000
## ID:gm10_152b 0.9999 0.0001
## ID:gm10_157b 0.0000 1.0000
## ID:gm10_158d 1.0000 0.0000
## ID:gm13_137a 1.0000 0.0000
## ID:gm13_149a 1.0000 0.0000
## ID:gm13_169a 0.0038 0.9962
## ID:gm14_180a 1.0000 0.0000

## calculate state transition probabilities for each mixture
trProbs2 <- getTrProbs(fitmix2, covIndex=1)

# mixture 1
round(trProbs2[[1]][,,1],2)

##             exploratory foraging crowded directed
## exploratory        0.60     0.06    0.01     0.33
## foraging           0.36     0.34    0.04     0.26
## crowded            0.03     0.01    0.91     0.05
## directed           0.25     0.03    0.02     0.71

# mixture 2
round(trProbs2[[2]][,,1],2)

##             exploratory foraging crowded directed
## exploratory        0.84     0.07    0.03     0.06
## foraging           0.32     0.31    0.09     0.28
## crowded            0.00     0.02    0.83     0.15
## directed           0.11     0.08    0.16     0.65
```

Now let's fit a model with $K = 3$ mixtures by setting some starting values with the help of `getPar0`:

```
Par0_mix3 <- getPar0(fitmix2, mixtures=3)

## Error in !m$conditions$fit:  invalid argument type

Par0_mix3$beta$beta[1,] <- c(-2.15, -4.31, -1.09,
                              0.28, -1.88,  -0.3,
                             -3.5, -4.71, -3.11,
                            -0.68, -2.49,  -2.6)
```

```
## Error in Par0_mix3$beta$beta[1, ] <- c(-2.15, -4.31, -1.09, 0.28, -1.88,
:  object 'Par0_mix3' not found
```

```
Par0_mix3$beta$beta[2,] <- c(  -2.5,  -2.47,  0.63,
                             -17.22, -13.18,  0.59,
                              -3.92, -13.96, -2.27,
                              -1.25,  -3.57, -3.75)
```

```
## Error in Par0_mix3$beta$beta[2, ] <- c(-2.5, -2.47, 0.63, -17.22, -13.18,
:  object 'Par0_mix3' not found
```

```
Par0_mix3$beta$beta[3,] <- c(-2.71, -3.48, -3.01,
                             -0.35, -1.12,  -0.1,
                             -96.8, -2.98, -1.53,
                             -2.29, -2.07, -1.55)
```

```
## Error in Par0_mix3$beta$beta[3, ] <- c(-2.71, -3.48, -3.01, -0.35, -1.12,
:  object 'Par0_mix3' not found
```

```
Par0_mix3$beta$pi <- c(0.4, 0.4, 0.2)
```

```
## Error in Par0_mix3$beta$pi <- c(0.4, 0.4, 0.2):  object 'Par0_mix3' not
found
```

and then calling `fitHMM` with `mixtures=3`:

```
fitmix3 <- fitHMM(pilotData, nbStates=4, dist=dist,
                  Par0=Par0_mix3$Par, beta0=Par0_mix3$beta,
                  stationary=TRUE,
                  mixtures=3,
                  stateNames=stateNames,
                  nlmPar=list(hessian=FALSE))
```

```
## ============================================================================
## Fitting HMM with 4 states and 11 data streams
## ----------------------------------------------------------------------------
##  dive.dur ~ weibull(shape=~1, scale=~1)
##  dive.depth ~ gamma(mean=~1, sd=~1)
##  GR.speed2 ~ gamma(mean=~1, sd=~1)
##  dive.pitchvar2 ~ beta(shape1=~1, shape2=~1)
```

```
##  breath.headchange ~ vm(concentration=~1)
##  GR.size ~ pois(lambda=~1)
##  GR.tight ~ bern(prob=~1)
##  dive.CS.pres ~ bern(prob=~1)
##  dive.SS.pres ~ bern(prob=~1)
##  presurf ~ bern(prob=~1)
##  postsurf ~ bern(prob=~1)
##
##  Transition probability matrix formula:  ~1
##
##  Initial distribution formula:  ~1
##
##  Number of mixtures:  3
##  Mixture probability formula:  ~1
## ============================================================================
## DONE
```

```
fitmix3
```

```
## Value of the maximum log-likelihood: -18445.14
##
##
## dive.dur parameters:
## --------------------
##       exploratory foraging  crowded  directed
## shape    1.975978 2.813776 1.648341 3.8522580
## scale    1.422555 8.246565 1.520763 0.7864722
##
## dive.depth parameters:
## ----------------------
##      exploratory foraging   crowded directed
## mean   10.977415 322.5798 10.638125 5.769243
## sd      5.427491 228.1005  6.314569 2.013745
##
## GR.speed2 parameters:
## ---------------------
##      exploratory  foraging   crowded  directed
## mean   1.1385507 1.3201513 1.3861439 1.5065078
```

```
## sd     0.6663534 0.5203517 0.7776481 0.7581291
##
## dive.pitchvar2 parameters:
## -------------------------
##        exploratory foraging   crowded  directed
## shape1     2.15122 3.028038  1.764795  2.524053
## shape2    17.62926 6.225084 15.844942 37.637633
##
## breath.headchange parameters:
## -----------------------------
##               exploratory foraging  crowded directed
## mean             0.000000 0.000000 0.000000  0.00000
## concentration    2.768482 5.696489 2.629183 17.09466
##
## GR.size parameters:
## -------------------
##        exploratory foraging  crowded directed
## lambda    5.672187 7.460572 20.48158 9.345636
##
## GR.tight parameters:
## --------------------
##      exploratory  foraging   crowded  directed
## prob   0.9098293 0.6533947 0.7686185 0.7903651
##
## dive.CS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7618571 0.9892938 0.3923022 0.5248959
##
## dive.SS.pres parameters:
## ------------------------
##      exploratory  foraging   crowded  directed
## prob   0.7364702 0.9804547 0.3799649 0.4544787
##
## presurf parameters:
## -------------------
##      exploratory  foraging   crowded  directed
## prob    0.748499 0.9922603 0.7107198 0.7227432
##
## postsurf parameters:
## --------------------
##      exploratory  foraging   crowded  directed
```

```
## prob   0.8213494 0.9624941 0.7156723 0.6705817
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                     1 -> 2    1 -> 3     1 -> 4      2 -> 1     2 -> 3
## (Intercept)_mix1 -2.152085 -4.306731 -1.0867497   0.2791947  -1.880898
## (Intercept)_mix2 -2.498738 -2.469649  0.6277231 -17.2200002 -13.180019
## (Intercept)_mix3 -2.714484 -3.475361 -3.0076135  -0.3548059  -1.124543
##                       2 -> 4     3 -> 1     3 -> 2    3 -> 4     4 -> 1
## (Intercept)_mix1 -0.29671906  -3.501065  -4.714406 -3.111360 -0.6813091
## (Intercept)_mix2  0.58694161  -3.926036 -13.960064 -2.270529 -1.2524123
## (Intercept)_mix3 -0.09724842 -96.800000  -2.975692 -1.531841 -2.2900315
##                     4 -> 2    4 -> 3
## (Intercept)_mix1 -2.488782 -2.599644
## (Intercept)_mix2 -3.567966 -3.753564
## (Intercept)_mix3 -2.073131 -1.549459
##
## Mixture probabilities:
## ----------------------
##      mix1      mix2      mix3
## 0.4018629 0.3980198 0.2001172
##
## Transition probability matrix:
## ------------------------------
##                    exploratory     foraging      crowded   directed
## exploratory_mix1 6.816492e-01 7.923599e-02 9.186948e-03 0.22992783
## foraging_mix1    4.108636e-01 3.107742e-01 4.737849e-02 0.23098376
## crowded_mix1     2.783618e-02 8.272980e-03 9.227895e-01 0.04110136
## directed_mix1    3.041933e-01 4.990848e-02 4.467117e-02 0.60122702
## exploratory_mix2 3.289318e-01 2.703446e-02 2.783241e-02 0.61620128
## foraging_mix2    1.187207e-08 3.573367e-01 6.746335e-07 0.64266266
## crowded_mix2     1.756192e-02 7.706355e-07 8.904877e-01 0.09194960
## directed_mix2    2.136992e-01 2.109457e-02 1.752130e-02 0.74768494
## exploratory_mix3 8.721443e-01 5.777009e-02 2.699344e-02 0.04309217
## foraging_mix3    2.390740e-01 3.408966e-01 1.107235e-01 0.30930588
## crowded_mix3     7.202219e-43 4.025737e-02 7.891729e-01 0.17056974
## directed_mix3    7.035018e-02 8.739041e-02 1.475339e-01 0.69472552
##
## Initial distribution:
## ---------------------
##      exploratory   foraging   crowded  directed
## mix1   0.3870062 0.06849400 0.2553623 0.2891375
```

```
## mix2   0.2012576 0.02880544 0.1502894 0.6196476
## mix3   0.3234760 0.08566424 0.2940851 0.2967747

## calculate mixture probabilities for each individual
round(mixtureProbs(fitmix3),4)

##                mix1   mix2  mix3
## ID:gm08_150c 0.0000 1.0000 0e+00
## ID:gm08_154d 0.0000 1.0000 0e+00
## ID:gm08_159a 0.0000 1.0000 0e+00
## ID:gm09_137b 1.0000 0.0000 0e+00
## ID:gm09_138a 0.0000 0.0000 1e+00
## ID:gm09_156b 1.0000 0.0000 0e+00
## ID:gm10_000a 0.0000 0.0000 1e+00
## ID:gm10_143a 1.0000 0.0000 0e+00
## ID:gm10_152b 0.0276 0.9724 0e+00
## ID:gm10_157b 0.0000 0.0000 1e+00
## ID:gm10_158d 0.0006 0.9994 0e+00
## ID:gm13_137a 1.0000 0.0000 0e+00
## ID:gm13_149a 1.0000 0.0000 0e+00
## ID:gm13_169a 0.9991 0.0000 9e-04
## ID:gm14_180a 0.0000 1.0000 0e+00

## calculate state transition probabilities for each mixture
trProbs3 <- getTrProbs(fitmix3, covIndex=1)

# mixture 1
round(trProbs3[[1]][,,1],2)

##             exploratory foraging crowded directed
## exploratory        0.68     0.08    0.01     0.23
## foraging           0.41     0.31    0.05     0.23
## crowded            0.03     0.01    0.92     0.04
## directed           0.30     0.05    0.04     0.60

# mixture 2
round(trProbs3[[2]][,,1],2)

##             exploratory foraging crowded directed
## exploratory        0.33     0.03    0.03     0.62
## foraging           0.00     0.36    0.00     0.64
## crowded            0.02     0.00    0.89     0.09
## directed           0.21     0.02    0.02     0.75
```

```
# mixture 3
round(trProbs3[[3]][,,1],2)

##             exploratory foraging crowded directed
## exploratory        0.87     0.06    0.03     0.04
## foraging           0.24     0.34    0.11     0.31
## crowded            0.00     0.04    0.79     0.17
## directed           0.07     0.09    0.15     0.69
```

And let's do the same with $K = 4$:

```
fitmix4 <- fitHMM(pilotData, nbStates=4, dist=dist,
                  Par0=Par0_mix4$Par, beta0=Par0_mix4$beta,
                  stationary=TRUE,
                  mixtures=4,
                  stateNames=stateNames,
                  nlmPar=list(hessian=FALSE))
```

For comparison to the null and random effects models, let's also fit a model including individual-level fixed effects:

```
Par0_fix <- getPar0(fitmix1, formula= ~ID, formulaDelta= ~ID)
Par0_fix$delta[1,] <- log(fitmix1$mle$delta[1,-1]/fitmix1$mle$delta[1,1])

fitfix <- fitHMM(pilotData, nbStates=4, dist=dist,
                  formula = ~ID, formulaDelta = ~ID,
                  Par0=Par0_fix$Par, beta0=Par0_fix$beta,
                  delta0=Par0_fix$delta,
                  stateNames=stateNames,
                  nlmPar=list(hessian=FALSE))

## ============================================================================
## Fitting HMM with 4 states and 11 data streams
## ----------------------------------------------------------------------------
##  dive.dur ~ weibull(shape=~1, scale=~1)
##  dive.depth ~ gamma(mean=~1, sd=~1)
##  GR.speed2 ~ gamma(mean=~1, sd=~1)
##  dive.pitchvar2 ~ beta(shape1=~1, shape2=~1)
##  breath.headchange ~ vm(concentration=~1)
```

```
##  GR.size ~ pois(lambda=~1)
##  GR.tight ~ bern(prob=~1)
##  dive.CS.pres ~ bern(prob=~1)
##  dive.SS.pres ~ bern(prob=~1)
##  presurf ~ bern(prob=~1)
##  postsurf ~ bern(prob=~1)
##
##  Transition probability matrix formula:  ~ID
##
##  Initial distribution formula:  ~ID
## ============================================================================
## DONE
```

Based on AIC, we find overwhelming support for the discrete-valued individual-level random effects model with $K = 3$ mixtures:

```
AIC(fitmix1,fitmix2,fitmix3,fitmix4,fitfix)

## Error in !m$conditions$fit:  invalid argument type

AICweights(fitmix1,fitmix2,fitmix3,fitmix4,fitfix)

## Error in !m$conditions$fit:  invalid argument type
```

3.10 Hierarchical HMMs

As we already noted in section 2.5, HMMs with hierarchical structures allow for data streams and/or state transitions to occur at multiple regular time scales. Leos-Barajas *et al.* (2017) provide two examples where state transitions are allowed to occur at both "coarse" and "fine" time scales, while Adam *et al.* (2019) provide two examples where both data streams and state transitions occur at both "coarse" and "fine" time scales. Here we demonstrate how all four of these examples can be fitted in `momentuHMM`. The key to fitting (and simulating) hierarchical hidden Markov models (HHMMs) in `momentuHMM` is specifying certain `fitHMM` (and `simHierData`) arguments hierarchically using the `data.tree` package (Glur 2018). For HHMMs, instead of simply specifying

the number of states (`nbStates`), distributions (`dist`), and a single t.p.m. (`formula`) or initial distribution (`formulaDelta`) formula, the `hierStates` argument specifies the hierarchical nature of the states, the `hierDist` argument specifies the hierarchical nature of the data streams, and the `hierFormula` or `hierFormulaDelta` arguments specify a t.p.m. or initial distribution formula for each level of the hierarchy. All are specified as `Node` objects from the `data.tree` package. In the examples below, we focus on how to implement these HHMMs in `momentuHMM` and refer readers to Leos-Barajas *et al.* (2017) and Adam *et al.* (2019) for specific details about the data sets and the particular models being fitted.

3.10.1 Harbor porpoise

In order to replicate the harbor porpoise HHMM example from Leos-Barajas *et al.* (2017) in `momentuHMM`, we must first use `prepData` to prepare our hierarchical data. This requires an additional field named `level` to be included in `data` that identifies the level of the hierarchy for each observation. These levels must be ordered from the coarsest to finest time scales, and must also indicate the initial observations at each level of the hierarchy (except for the coarsest level). For example, if there are $M = 3$ time scales in the hierarchy (e.g. "coarse", "medium", and "fine" scales), then the `level` field must include $2M - 1 = 5$ ordered factors: "`1`" (corresponding to coarse-scale observations), "`2i`" (initial medium-scale observations), "`2`" (medium-scale observations), "`3i`" (initial fine-scale observations), and "`3`" (fine-scale observations). Regardless of the number of levels in the hierarchy, note that for each individual the `level` field for the first observation must always be "`1`", the second obervation must always be "`2i`", the third observation must always be "`2`", and the last observation must always be from level M. Also note that every "`1`" observation must be followed by "`2i`", every "`2i`" must be followed by one or more "`2`", every "`3i`" must be preceded by "`2`" and followed by one or more "`3`", and, after the first observation, every "`1`" must be preceded by an observation from level M.

In the harbor porpoise example from Leos-Barajas *et al.* (2017), there are only $M = 2$ levels in the hierarchy so the $2M - 1 = 3$ ordered `level` factors are "`1`" (coarse level), "`2i`" (initial fine level), and "`2`" (fine level). After downloading the data, we can manually add the `level` field to our data frame as follows:

```r
# load harbor porpoise data from Leos-Barajas et al
load(url(paste0("https://static-content.springer.com/esm/",
                "art%3A10.1007%2Fs13253-017-0282-9/MediaObjects/",
                "13253_2017_282_MOESM2_ESM.rdata")))

# convert date_time to POSIX
data <- lapply(data,function(x)
                {x$date_time <- as.POSIXct(x$date_time,tz="UTC"); x})

porpoiseData <- NULL
for(i in 1:length(data)){
  coarseInd <- data.frame(date_time=as.POSIXct(format(data[[i]]$date_time[1],
                                                      format="%Y-%m-%d %H:%M"),
                                               tz="UTC"),
                          level=c("1","2i"),
                          dive_duration=NA,
                          maximum_depth=NA,
                          dive_wiggliness=NA)
  tmp <- rbind(coarseInd,data.frame(data[[i]],level="2"))
  porpoiseData <- rbind(porpoiseData,tmp)
}

head(porpoiseData)
```

```
##             date_time level dive_duration maximum_depth dive_wiggliness
## 1 2015-11-02 14:43:00     1            NA            NA              NA
## 2 2015-11-02 14:43:00    2i            NA            NA              NA
## 3 2015-11-02 14:43:11     2            28          8.19            5.50
## 4 2015-11-02 14:44:08     2            18          8.19            3.00
## 5 2015-11-02 14:44:35     2            14          6.19            5.25
## 6 2015-11-02 14:44:58     2            17          7.44            2.50
```

By including the `level` field, we will be able to specify when the coarse-scale state switching can occur (i.e., a coarse scale t.p.m. will be used when `level=1`), the start of each fine-scale interval (i.e. a fine-scale initial distribution will be used when `level=2i`), and when the fine-scale state switching can occur (i.e. a fine scale t.p.m. will be used when `level=2`).

Now that we have labeled each observation in our data with a `level` factor, we can prepare our HHMM data using `prepData`:

```r
# prepare hierarchical data
porpoiseData <- prepData(data = porpoiseData,
                         coordNames = NULL,
                         hierLevels = c("1", "2i", "2"))

# summarize prepared data
summary(porpoiseData, dataNames = names(porpoiseData)[-1])
```

```
## Hierarchical HMM data for 1 individual:
##
## Animal1 -- 8135 observations
##
##
## Data summaries:
##
##    date_time                    level     dive_duration maximum_depth
##  Min.   :2015-11-02 14:43:00   1 : 275   Min.   :  1   Min.   :  2.2
##  1st Qu.:2015-11-06 02:01:24   2i: 275   1st Qu.: 15   1st Qu.:  7.2
##  Median :2015-11-08 14:14:28   2 :7585   Median : 43   Median : 16.2
##  Mean   :2015-11-08 11:59:48             Mean   : 58   Mean   : 22.2
##  3rd Qu.:2015-11-11 04:43:07             3rd Qu.: 90   3rd Qu.: 30.2
##  Max.   :2015-11-14 00:58:50             Max.   :248   Max.   :169.4
##                                          NA's   :550   NA's   :550
##  dive_wiggliness
##  Min.   :  0.0
##  1st Qu.:  3.8
##  Median : 15.0
##  Mean   : 23.4
##  3rd Qu.: 36.0
##  Max.   :253.5
##  NA's   :562
```

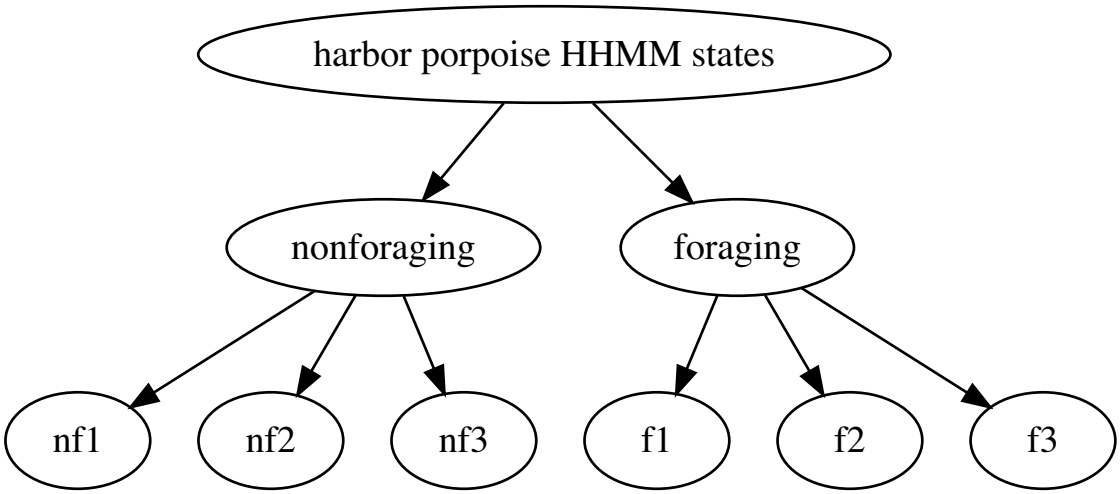
For HHMMs, we must use the `hierLevels` argument to identify the ordered factor levels that `prepData` can expect to find in the `level` field. When `hierLevels` is specified, `prepData` assumes the data are intended for a HHMM analysis and assigns the classes `hierarchical` and `momentuHierHMMData` to the returned object (and corresponding methods for these classes will hereafter be used when calling functions such as `fitHMM`). Because `prepData` assumes the `level` field is a factor with levels ordered according to `hierLevels`, an error is returned if the order of `hierLevels` is not consistent with the `level` field (or vice versa):

```
prepData(data = porpoiseData,
                 coordNames = NULL,
                 hierLevels = c("1", "2", "2i"))

## Error in prepData.hierarchical(data = porpoiseData, coordNames = NULL,
:  hierLevels must be ordered factors of the form:
##      '1', '2i', '2', ..., 'Mi', 'M' where M is the number of levels in
the hierarchy
```

Note that because there are no location data in this example, we have set `coordNames` to `NULL`. Also note that in this example there are no data streams observed at the coarse time scale (`level=1`), but these could be easily included (if available; see sections 3.10.3 and 3.10.4). By definition there are no data streams observed at the initial fine scale (`level=2i`), but covariates (if available) could be included here for modeling the fine-scale initial distributions or state transition probabilities.

Now that our hierarchical data are prepared, we are ready to specify the HHMM. We will start with the hierarchical nature of the states, which is specified using the `hierStates` argument in `fitHMM`:

```
library(data.tree)

### define hierarchical HMM
### states 1-3 = coarse state 1 (nonforaging)
### states 4-6 = coarse state 2 (foraging)
hierStates <- data.tree::Node$new("harbor porpoise HHMM states")
hierStates$AddChild(name="nonforaging")
hierStates$nonforaging$AddChild(name="nf1", state=1)
hierStates$nonforaging$AddChild(name="nf2", state=2)
hierStates$nonforaging$AddChild(name="nf3", state=3)
hierStates$AddChild(name="foraging")
hierStates$foraging$AddChild(name="f1", state=4)
hierStates$foraging$AddChild(name="f2", state=5)
hierStates$foraging$AddChild(name="f3", state=6)

plot(hierStates)
```



Here we can see that the coarse-scale "nonforaging" and "foraging" states are both composed of three fine-scale states. Note that the `Node` attribute `state` is required in `hierStates` and determines the index for each state in our HHMM. Alternatively, we could specify the exact same `Node` as:

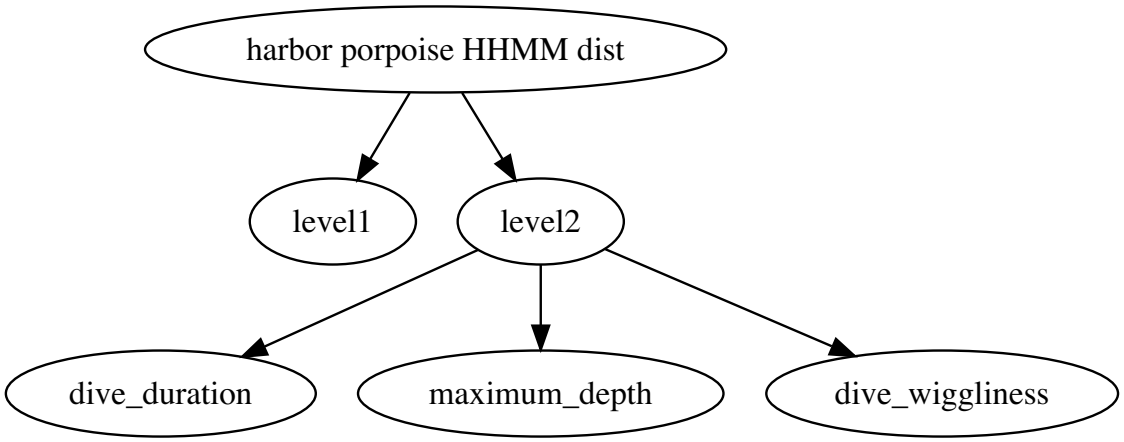
```
hierStates <- data.tree::as.Node(list(name="harbor porpoise HHMM states",
                                      nonforaging=list(nf1=list(state=1),
                                                       nf2=list(state=2),
                                                       nf3=list(state=3)),
                                      foraging=list(f1=list(state=4),
                                                    f2=list(state=5),
                                                    f3=list(state=6))))
```

The name for any of the "children" added to a node are user-specified and are akin to the `stateNames` argument in `fitHMM` for a standard HMM. While these names are arbitrary, the name and state attributes must be unique.

Next we will specify the hierarchical nature of the data streams for the `hierDist` argument in `fitHMM`:

```
# data stream distributions
# level 1 = coarse scale (no data streams)
# level 2 = fine scale (dive_duration, maximum_depth, dive_wiggliness)
```

```
hierDist <- data.tree::Node$new("harbor porpoise HHMM dist")
hierDist$AddChild(name="level1")
hierDist$AddChild(name="level2")
hierDist$level2$AddChild(name="dive_duration", dist="gamma")
hierDist$level2$AddChild(name="maximum_depth", dist="gamma")
hierDist$level2$AddChild(name="dive_wiggliness", dist="gamma")

plot(hierDist)
```



The `Node` attribute `dist` is required in `hierDist` and specifies the probability distribution for each data stream at each level of the hierarchy. In this case, `level1` (corresponding to coarse-scale observations with `level=1`) has no data streams, and each of the data streams for `level2` (corresponding to fine-scale observations with `level=2`) is assigned a gamma distribution. The first level of the `hierDist` Node must include a child for each level of the hierarchy, and the name for each child must be of the form `paste0("level",i)` for `i`$\in 1, 2, \ldots, M$, where M is the number of levels in the hierarchy; in this case, $M = 2$ and the names for each of the children at the first level of the `hierDist` Node must be `level1` and `level2`. The children of these levels must be "leaves" (i.e. they have no children), where the `name` attribute indicates the data stream and the `dist` attribute indicates the probability distribution.

Leos-Barajas *et al.* (2017) did not include any covariates on the t.p.m. or initial

distribution for either level of the hierarchy, but, for demonstration purposes, here is how we would use the `hierFormula` and `hierFormulaDelta` arguments to specify the t.p.m. and initial distribution formula for each level of the hierarchy in `fitHMM`:

```
# define hierarchical t.p.m. formula(s)
hierFormula <- data.tree::Node$new("harbor porpoise HHMM formula")
hierFormula$AddChild(name="level1", formula=~1)
hierFormula$AddChild(name="level2", formula=~1)
```

```
# define hierarchical initial distribution formula(s)
hierFormulaDelta <- data.tree::Node$new("harbor porpoise HHMM formulaDelta")
hierFormulaDelta$AddChild(name="level1", formulaDelta=~1)
hierFormulaDelta$AddChild(name="level2", formulaDelta=~1)
```

The `Node` attribute `formula` (or `formulaDelta`) is required in `hierFormula` (or `hierFormulaDelta`) and specifies the t.p.m. (or initial distribution) formula for each level of the hierarchy. Each child in `hierFormula` or `hierFormulaDelta` must be a leaf with a name of the form `paste0("level",i)` for `i`$\in 1, 2, \ldots, M$, where M is the number of levels in the hierarchy.

Leos-Barajas *et al.* (2017) assume the data stream probability distributions do not depend on the coarse-scale state, so we can constrain the state-dependent parameters for states 1 ("nf1") and 4 ("f1"), states 2 ("nf2") and 5 ("f2"), and states 3 ("nf3") and 6 ("f3") to be equal using the `DM` argument:

```
# defining starting values
dd.mu0 = rep(c(5,30,100),hierStates$count)
dd.sigma0 = rep(c(5,15,40),hierStates$count)
md.mu0 = rep(c(5,15,40),hierStates$count)
md.sigma0 = rep(c(2,5,20),hierStates$count)
dw.mu0 = rep(c(2,10,40),hierStates$count)
dw.sigma0 = rep(c(2,10,20),hierStates$count)
dw.pi0 = rep(c(0.2,0.01,0.01),hierStates$count)

Par0 <- list(dive_duration=c(dd.mu0,dd.sigma0),
             maximum_depth=c(md.mu0,md.sigma0),
             dive_wiggliness=c(dw.mu0,dw.sigma0,dw.pi0))

nbStates <- length(hierStates$Get("state",filterFun=data.tree::isLeaf))
```

```
# constrain fine-scale data stream distributions to be same
dw_DM <- matrix(cbind(kronecker(c(1,1,0,0,0,0),diag(3)),
                      kronecker(c(0,0,1,1,0,0),diag(3)),
                      kronecker(c(0,0,0,0,1,1),diag(3))),
                nrow=nbStates*3,
                ncol=9,
                  dimnames=list(c(paste0("mean_",1:nbStates),
                                  paste0("sd_",1:nbStates),
                                  paste0("zeromass_",1:nbStates)),
                                paste0(rep(c("mean","sd","zeromass"),each=3),
                                       c("_14:(Intercept)",
                                         "_25:(Intercept)",
                                         "_36:(Intercept)"))))

DM <- list(dive_duration=dw_DM[1:(2*nbStates),1:6],
           maximum_depth=dw_DM[1:(2*nbStates),1:6],
           dive_wiggliness=dw_DM)

# get initial parameter values for data stream probability distributions
Par <- getParDM(porpoiseData,hierStates=hierStates,hierDist=hierDist,
                Par=Par0,DM=DM)
```

The (optional) last step for fitting this HHMM is specifying starting values for the t.p.m. and initial distribution parameters for each level of the hierarchy. In this case, we set them to "nudge" coarse state 1 (i.e., fine-scale states $1-3$) to "nonforaging" and coarse state 2 (i.e., fine-scale states $4-6$) to "foraging":

```
# initial values ('beta') for t.p.m. at each level of hierarchy
hierBeta <- data.tree::Node$new("harbor porpoise beta")
hierBeta$AddChild(name="level1",beta=matrix(c(-1, -1),1))
hierBeta$AddChild(name="level2")
hierBeta$level2$AddChild(name="nonforaging",beta=matrix(c(0,-1,1,0,1,1),1))
hierBeta$level2$AddChild(name="foraging",beta=matrix(c(-1,1,1,2,0,3),1))

# initial values ('delta') for initial distribution at each level of hierarchy
hierDelta <- data.tree::Node$new("harbor porpoise delta")
hierDelta$AddChild(name="level1",delta=matrix(0,1))
hierDelta$AddChild(name="level2")
hierDelta$level2$AddChild(name="nonforaging",delta=matrix(c(30, 30),1))
hierDelta$level2$AddChild(name="foraging",delta=matrix(c(-6, 2),1))
```

The `Node` attribute `beta` (or `delta`) is required in `hierBeta` (or `hierDelta`) and specifies the starting values for the t.p.m. (or initial distribution) at each level of the hierarchy. For each level of the hierarchy, these values are specified just as in `beta0` (or `delta0`) for a standard HMM fitted with `fitHMM`. However, for HHMMs the starting values in `hierDelta` must always be provided as a matrix on the working scale (even if no covariates are included in `hierFormulaDelta`). Any additional arguments pertaining to t.p.m. or initial distribution parameters (such as `workBounds$beta`, `betaCons`, `fixPar$beta`, `workBounds$delta`, `deltaCons`, and `fixPar$delta`) must also be specified as `data.tree` Nodes (with `Node` attributes `workBounds`, `betaCons`, `fixPar`, `workBounds`, `deltaCons`, and `fixPar`, respectively).

Before fitting the HHMM, let's first check that everything is in order using the `checkPar0` function:

```
# check hierarchical model specification and parameters
checkPar0(porpoiseData,hierStates=hierStates,hierDist=hierDist,Par0=Par,
          hierFormula=hierFormula,hierFormulaDelta=hierFormulaDelta,
          DM=DM,hierBeta=hierBeta,hierDelta=hierDelta)

##
## Regression coeffs for dive_duration parameters:
## ----------------------------------------------
##      mean_14:(Intercept) mean_25:(Intercept) mean_36:(Intercept)
## [1,]            1.609438            3.401197             4.60517
##      sd_14:(Intercept) sd_25:(Intercept) sd_36:(Intercept)
## [1,]          1.609438           2.70805          3.688879
##
## Regression coeffs for maximum_depth parameters:
## ----------------------------------------------
##      mean_14:(Intercept) mean_25:(Intercept) mean_36:(Intercept)
## [1,]            1.609438             2.70805            3.688879
##      sd_14:(Intercept) sd_25:(Intercept) sd_36:(Intercept)
## [1,]         0.6931472          1.609438          2.995732
##
## Regression coeffs for dive_wiggliness parameters:
## ------------------------------------------------
##      mean_14:(Intercept) mean_25:(Intercept) mean_36:(Intercept)
## [1,]           0.6931472            2.302585            3.688879
##      sd_14:(Intercept) sd_25:(Intercept) sd_36:(Intercept)
## [1,]         0.6931472          2.302585          2.995732
##      zeromass_14:(Intercept) zeromass_25:(Intercept)
```

```
## [1,]               -1.386294                -4.59512
##      zeromass_36:(Intercept)
## [1,]                -4.59512
##
## ------------------------------------------------------------
## Regression coeffs for the transition probabilities (beta):
## ------------------------------------------------------------
## ------------------------  level1  ----------------------
##                        1 -> 4 4 -> 1
## I((level == "1") * 1)     -1     -1
##
## ------------------------  level2  ----------------------
##                        1 -> 2 1 -> 3 2 -> 2 2 -> 3 3 -> 2 3 -> 3
## I((level == "2") * 1)      0     -1      1      0      1      1
##
##                        4 -> 5 4 -> 6 5 -> 5 5 -> 6 6 -> 5 6 -> 6
## I((level == "2") * 1)     -1      1      1      2      0      3
##
## ------------------------------------------------------------
##
## ------------------------------------------------------------
## Regression coeffs for the initial distribution (delta):
## ------------------------------------------------------------
## ------------------------  level1  ----------------------
##             state 4
## (Intercept)       0
##
## ------------------------  level2  ----------------------
##                         state 2 state 3
## I((level == "2i") * 1)      30      30
##
##                         state 5 state 6
## I((level == "2i") * 1)      -6       2
##
## ------------------------------------------------------------
```

Note that for HHMMs, the reference states for the t.p.m. are determined by the lowest index for each state at the coarsest level in the hierarchy; in this case, the reference states are state 1 for "nonforaging" and state 4 for "foraging". This differs from standard HMMs fitted with `fitHMM` (where the reference states can be user-specified with the `betaRef` argument).

Since everything looks good, we're now ready to fit our HHMM:

```
# fit hierarchical HMM
hhmm <- fitHMM(data=porpoiseData,hierStates=hierStates,hierDist=hierDist,
               hierFormula=hierFormula,hierFormulaDelta=hierFormulaDelta,
               Par0=Par,hierBeta=hierBeta,hierDelta=hierDelta,
               DM=DM,nlmPar=list(hessian=FALSE))

## ============================================================================
## Fitting hierarchical HMM with 6 states and 3 data streams
## ----------------------------------------------------------------------------
##  dive_duration ~ gamma(mean:  custom, sd:  custom)
##  maximum_depth ~ gamma(mean:  custom, sd:  custom)
##  dive_wiggliness ~ gamma(mean:  custom, sd:  custom, zeromass:  custom)
##
##  Transition probability matrix formula:  ~0 + I((level == "1") * 1) +
I((level == "2i") * 1) + I((level == "2") * 1)
##
##  Initial distribution formula:  ~1
## ============================================================================
## DONE
```

Before examining the output for our fitted HHMM, it's worth noting here that when `data` is a `momentuHierHMMData` object, `fitHMM` "shoehorns" the HHMM into a standard HMM by constraining the t.p.m. and initial distribution according to the hierarchical structure defined by `hierStates`. This is why the printed t.p.m. formula above may look a little strange at first. The initial distribution formula printed above pertains only to the initial distribution at the coarsest level of the hierarchy, and the initial distributions for all other levels are imbedded in the t.p.m. Let's look a little deeper:

```
hhmm$conditions$fixPar$beta

##                         1 -> 2 1 -> 3 1 -> 4 1 -> 5 1 -> 6 2 -> 2 2 -> 3
## I((level == "1") * 1)  -1e+10 -1e+10     NA -1e+10 -1e+10 -1e+10 -1e+10
## I((level == "2i") * 1)     NA     NA -1e+10 -1e+10 -1e+10 -1e+10 -1e+10
## I((level == "2") * 1)      NA     NA -1e+10 -1e+10 -1e+10     NA     NA
##                         2 -> 4 2 -> 5 2 -> 6 3 -> 2 3 -> 3 3 -> 4 3 -> 5
```

```
## I((level == "1") * 1)      NA -1e+10 -1e+10 -1e+10 -1e+10     NA -1e+10
## I((level == "2i") * 1) -1e+10 -1e+10 -1e+10 -1e+10 -1e+10 -1e+10 -1e+10
## I((level == "2") * 1)  -1e+10 -1e+10 -1e+10     NA     NA -1e+10 -1e+10
##                        3 -> 6 4 -> 1 4 -> 2 4 -> 3 4 -> 5 4 -> 6 5 -> 1
## I((level == "1") * 1)  -1e+10     NA -1e+10 -1e+10 -1e+10 -1e+10     NA
## I((level == "2i") * 1) -1e+10 -1e+10 -1e+10 -1e+10     NA     NA -1e+10
## I((level == "2") * 1)  -1e+10 -1e+10 -1e+10 -1e+10     NA     NA -1e+10
##                        5 -> 2 5 -> 3 5 -> 5 5 -> 6 6 -> 1 6 -> 2 6 -> 3
## I((level == "1") * 1)  -1e+10 -1e+10 -1e+10 -1e+10     NA -1e+10 -1e+10
## I((level == "2i") * 1) -1e+10 -1e+10 -1e+10 -1e+10 -1e+10 -1e+10 -1e+10
## I((level == "2") * 1)  -1e+10 -1e+10     NA     NA -1e+10 -1e+10 -1e+10
##                        6 -> 5 6 -> 6
## I((level == "1") * 1)  -1e+10 -1e+10
## I((level == "2i") * 1) -1e+10 -1e+10
## I((level == "2") * 1)      NA     NA

hhmm$conditions$betaCons

##                        1 -> 2 1 -> 3 1 -> 4 1 -> 5 1 -> 6 2 -> 2 2 -> 3
## I((level == "1") * 1)       1      1      7      1      1      1      1
## I((level == "2i") * 1)      2      5      1      1      1      1      1
## I((level == "2") * 1)       3      6      1      1      1     18     21
##                        2 -> 4 2 -> 5 2 -> 6 3 -> 2 3 -> 3 3 -> 4 3 -> 5
## I((level == "1") * 1)       7      1      1      1      1      7      1
## I((level == "2i") * 1)      1      1      1      1      1      1      1
## I((level == "2") * 1)       1      1      1     33     36      1      1
##                        3 -> 6 4 -> 1 4 -> 2 4 -> 3 4 -> 5 4 -> 6 5 -> 1
## I((level == "1") * 1)       1     46      1      1      1      1     46
## I((level == "2i") * 1)      1      1      1      1     56     59      1
## I((level == "2") * 1)       1      1      1      1     57     60      1
##                        5 -> 2 5 -> 3 5 -> 5 5 -> 6 6 -> 1 6 -> 2 6 -> 3
## I((level == "1") * 1)       1      1      1      1     46      1      1
## I((level == "2i") * 1)      1      1      1      1      1      1      1
## I((level == "2") * 1)       1      1     72     75      1      1      1
##                        6 -> 5 6 -> 6
## I((level == "1") * 1)       1      1
## I((level == "2i") * 1)      1      1
## I((level == "2") * 1)      87     90

hhmm$conditions$fixPar$delta

##             state 2 state 3 state 4 state 5 state 6
## (Intercept)  -1e+10  -1e+10      NA  -1e+10  -1e+10
```

```
hhmm$conditions$deltaCons

##             state 2 state 3 state 4 state 5 state 6
## (Intercept)       1       1       3       1       1
```

We can see that even though we did not explicitly specify `fixPar$beta`, `betaCons`, `fixPar$delta`, or `deltaCons`, the working parameters for the t.p.m. and initial distribution for each level of the hierarchy are constrained accordingly by making certain state transition and initial distribution probabilities equal and/or effectively zero. For example, these constraints do not allow fine-scale state switches between "nonforaging" (states $1-3$) and "foraging" (states $4-6$) when `level=2`. Similar to how the t.p.m. reference states are defined, higher-level (i.e. "parent") states are indexed based on the lowest state index of their "children". For example, "nonforaging" is indexed by state 1 and "foraging" is indexed by state 4, and only transitions to states 1 or 4 are permitted when `level=1`. Likewise, because `delta` corresponds to the initial distribution at the coarsest-scale, the initial distribution probabilities are effectively zero for all states except states 1 and 4.

Let's now examine our fitted HHMM:

```
hhmm

## Value of the maximum log-likelihood: -88242.08
##
##
## dive_duration parameters:
## -------------------------
##           nf1      nf2       nf3       f1       f2        f3
## mean 5.637624 32.18047 106.82948 5.637624 32.18047 106.82948
## sd   4.372832 15.13961  38.41245 4.372832 15.13961  38.41245
##
## maximum_depth parameters:
## -------------------------
##           nf1       nf2      nf3       f1        f2       f3
## mean 3.740051 13.193085 39.61714 3.740051 13.193085 39.61714
## sd   1.426271  5.288809 18.11188 1.426271  5.288809 18.11188
##
## dive_wiggliness parameters:
## ---------------------------
##                nf1          nf2          nf3        f1           f2
```

```
## mean     1.4566105 11.297304400 4.582790e+01 1.4566105 11.297304400
## sd       1.2051899  7.683421367 2.437341e+01 1.2051899  7.683421367
## zeromass 0.3089819  0.007866278 2.885413e-04 0.3089819  0.007866278
##                    f3
## mean     4.582790e+01
## sd       2.437341e+01
## zeromass 2.885413e-04
##
##
## ------------------------------------------------------------------
## Regression coeffs for the transition probabilities:
## ------------------------------------------------------------------
## -------------------------  level1  -------------------------
##                          1 -> 4    4 -> 1
## I((level == "1") * 1) -1.29882 -1.285164
##
## -------------------------  level2  -------------------------
##                           1 -> 2    1 -> 3  2 -> 2    2 -> 3    3 -> 2
## I((level == "2") * 1) 0.07465917 -1.014811 0.90907 -0.424281 0.6198106
##                           3 -> 3
## I((level == "2") * 1) 0.7823011
##
##                           4 -> 5    4 -> 6    5 -> 5   5 -> 6    6 -> 5
## I((level == "2") * 1) -0.5797742 0.7135748 0.7070142 1.621093 0.4255634
##                          6 -> 6
## I((level == "2") * 1) 2.701054
##
## ------------------------------------------------------------------
##
## ------------------------------------------------------------------
## Transition probability matrix (based on mean covariate values):
## ------------------------------------------------------------------
## -------------------------  level1  -------------------------
##             nonforaging  foraging
## nonforaging   0.7856364 0.2143636
## foraging      0.2166725 0.7833275
##
## -------------------------  level2  -------------------------
##           nf1       nf2       nf3
## nf1 0.4098381 0.4416074 0.1485545
## nf2 0.2417647 0.6000632 0.1581721
## nf3 0.1982132 0.3683942 0.4333926
```

```
##
##            f1         f2        f3
## f1 0.27767750 0.15550628 0.5668162
## f2 0.12366226 0.25077807 0.6255597
## f3 0.05738593 0.08782643 0.8547876
##
## ------------------------------------------------------------
##
## ------------------------------------------------
## Regression coeffs for the initial distribution:
## ------------------------------------------------
## --------------------  level1  --------------------
##               state 4
## (Intercept) -6.718877
##
## --------------------  level2  --------------------
##                         state 2  state 3
## I((level == "2i") * 1) 30.34116 29.65686
##
##                          state 5 state 6
## I((level == "2i") * 1) -6.010936  1.7169
##
## ------------------------------------------------
##
## ------------------------------------------------
## Initial distribution:
## ------------------------------------------------
## --------------------  level1  --------------------
##            nonforaging    foraging
## ID:Animal1   0.9987936 0.001206437
##
## --------------------  level2  --------------------
##                     nf1       nf2       nf3
## nonforaging 4.42208e-14 0.6646966 0.3353034
##
##                 f1           f2        f3
## foraging 0.1522141 0.0003731974 0.8474127
##
## ------------------------------------------------
```

These estimates are nearly identical to those reported by Leos-Barajas *et al.* (2017). The very slight differences are attributable to Leos-Barajas *et al.* (2017) assuming that the

initial distribution for each level of the hierarchy is equal to the stationary distribution. However, because it constrains the t.p.m. based on `level`, this stationarity assumption is not possible when fitting HHMMs in `momentuHMM`.

As in a standard HMM, we can decode the most likely state sequence using the `viterbi` function:

```
states <- viterbi(hhmm)

## Error in !m$conditions$fit:  invalid argument type

length(states)

## Error in eval(expr, envir, enclos):  object 'states' not found

head(states)

## Error in head(states):  object 'states' not found
```

but we can also obtain the most likely state sequences at each level of the hierarchy by setting the argument `hierarchical=TRUE`

```
hStates <- viterbi(hhmm, hierarchical=TRUE)

## Error in !m$conditions$fit:  invalid argument type

lapply(hStates,length)

## Error in lapply(hStates, length):  object 'hStates' not found

head(hStates$level1)

## Error in head(hStates$level1):  object 'hStates' not found

head(hStates$level2)

## Error in head(hStates$level2):  object 'hStates' not found
```

We can plot the estimated state probabilities for each level of the hierarchy (Figures 13 and 14) using the `plotStates` function:

```
plotStates(hhmm)
```

We can also calculate the stationary probabilities of each state for each level of the hierarchy:

```
# stationary distributions
stats <- stationary(hhmm)

# coarse scale
stats[[1]]$level1[1,]

## nonforaging    foraging
##   0.5026783   0.4973217

# fine scale
lapply(stats[[1]]$level2,function(x) x[1,])

## $nonforaging
##       nf1       nf2       nf3
## 0.2793776 0.5060948 0.2145275
##
## $foraging
##         f1         f2         f3
## 0.08308895 0.11164215 0.80526890
```

Finally, we can simulate from our fitted HHMM using the `simHierData` function. This requires that we specify the number of observations for each level of the hierarchy as a `data.tree Node` (with attribute `obs`) using the `obsPerLevel` argument:

```
obsPerLevel <- data.tree::Node$new("simHierData")

# number of level 1 observations
obsPerLevel$AddChild("level1",obs=100)

# number of level 2 observations that follow each level 1 observation
obsPerLevel$AddChild("level2",obs=25)

simHHMM <- simHierData(model=hhmm,
                       obsPerLevel = obsPerLevel, states = TRUE)

## ======================================================================
```

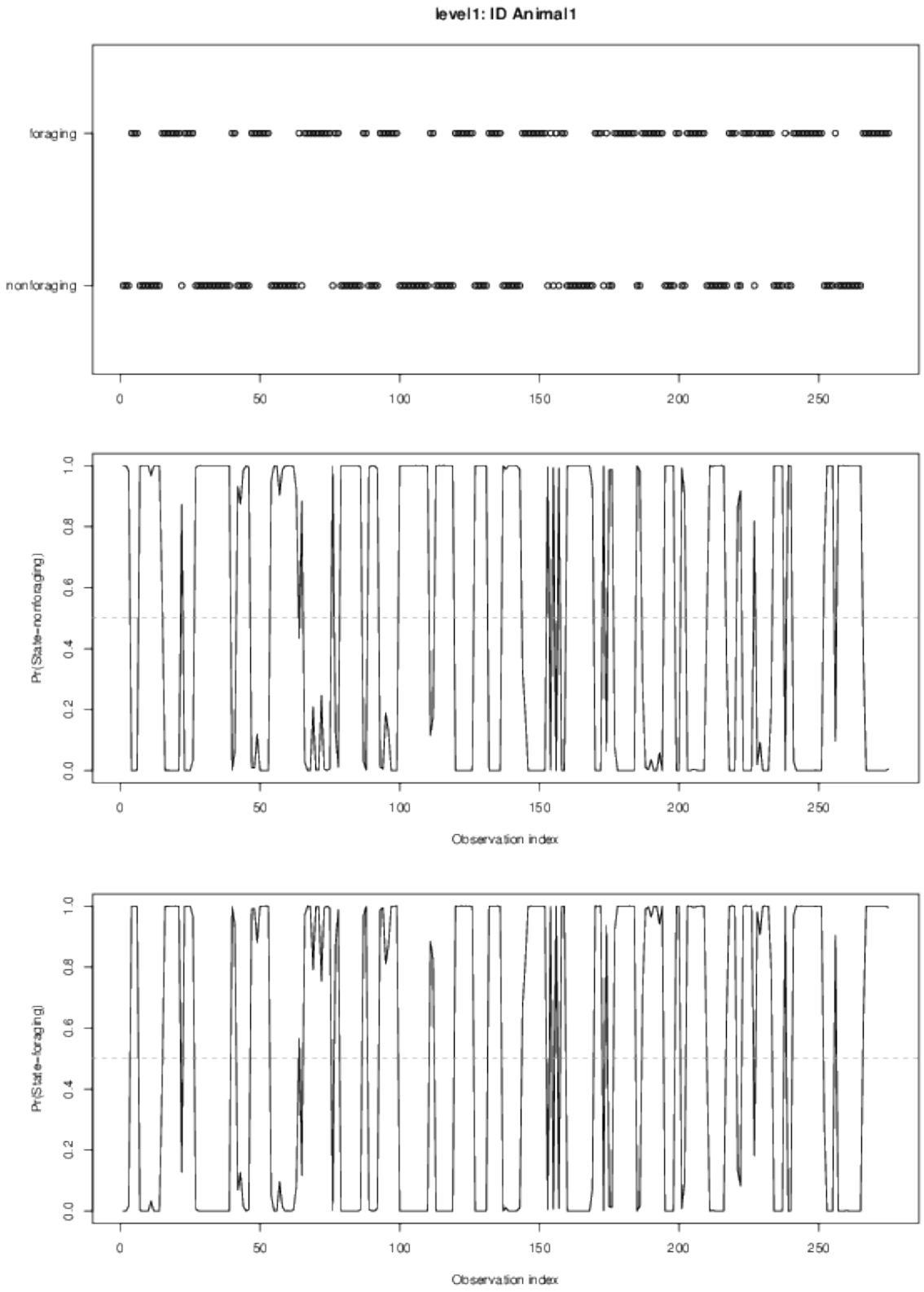


Figure 13. *Coarse-scale state probabilities for the harbor porpoise example.*

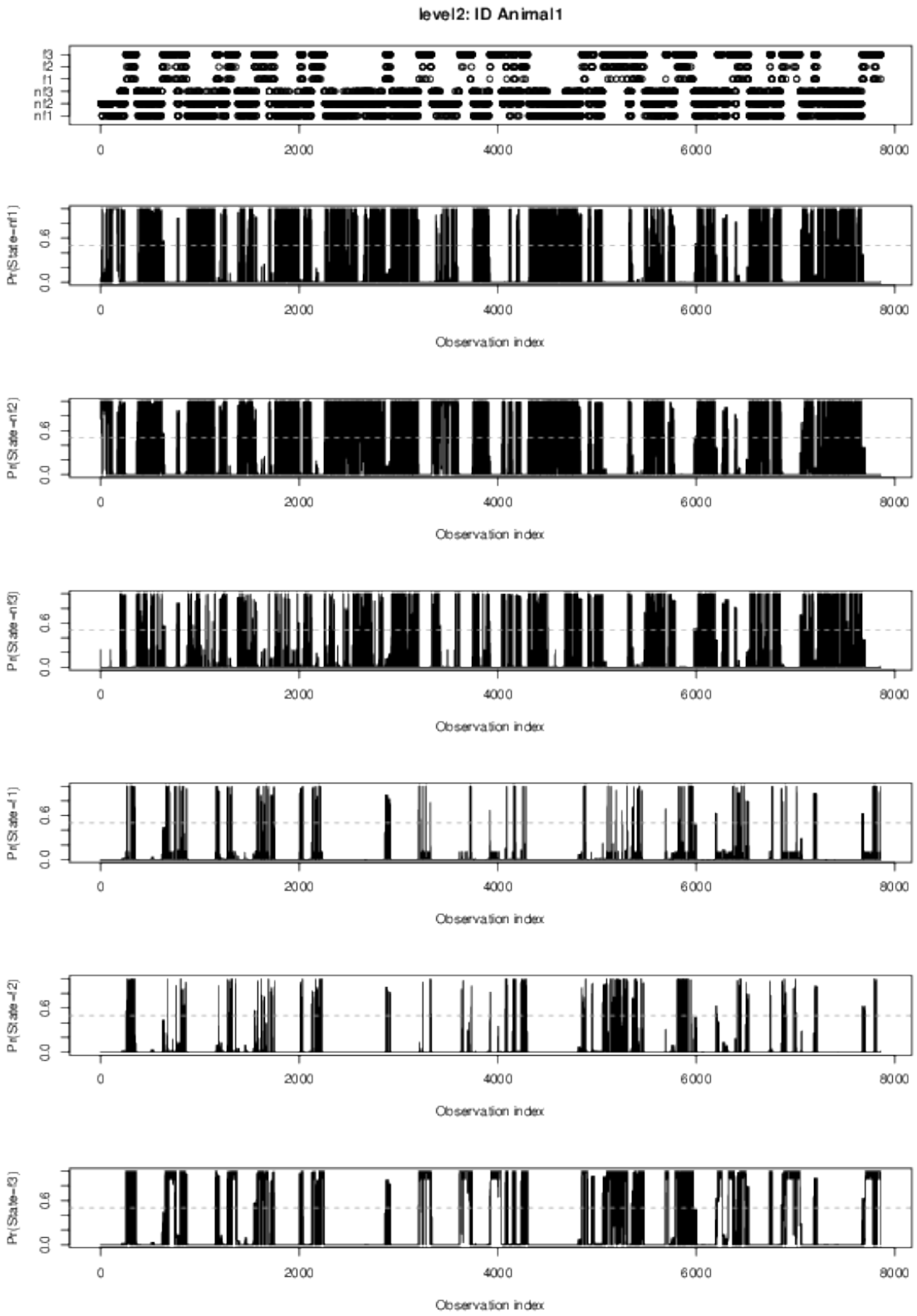


Figure 14. *Fine-scale state probabilities for the harbor porpoise example.*

```
## Simulating hierarchical HMM with 6 states and 3 data streams
## ------------------------------------------------------------------------
##  dive_duration ~ gamma(mean:  custom, sd:  custom)
##  maximum_depth ~ gamma(mean:  custom, sd:  custom)
##  dive_wiggliness ~ gamma(mean:  custom, sd:  custom, zeromass:  custom)
##
##  Transition probability matrix formula:  ~0 + I((level == "1") * 1) +
I((level == "2i") * 1) + I((level == "2") * 1)
##
##  Initial distribution formula:  ~1
## ========================================================================
## DONE
```

```
head(simHHMM)
```

```
##   ID level dive_duration maximum_depth dive_wiggliness states
## 1  1     1            NA            NA              NA      1
## 2  1    2i            NA            NA              NA      2
## 3  1     2      50.75401     19.444787      11.0461151      2
## 4  1     2      16.89706     16.166635      32.4377175      2
## 5  1     2      11.40743      4.024188       0.6959951      1
## 6  1     2      85.22059     35.200063      60.0786026      3
```

3.10.2 Garter snakes

Next we'll quickly demonstrate how to perform the HHMM analysis for the garter snake movement data in Leos-Barajas *et al.* (2017) using `momentuHMM`. This is also a 2-level HHMM, but now we include three coarse-scale states each composed of three fine-scale states (for a total of $N = 9$ states). As before, we must first add the `level` field to our data to indicate the level of the hierarchy for each observation and then create a `momentuHierHMMData` object with `prepData`:

```
# load garter snake data from Leos-Barajas et al
load(url(paste0("https://static-content.springer.com/esm/",
                "art%3A10.1007%2Fs13253-017-0282-9/MediaObjects/",
                "13253_2017_282_MOESM1_ESM.rdata")))
```

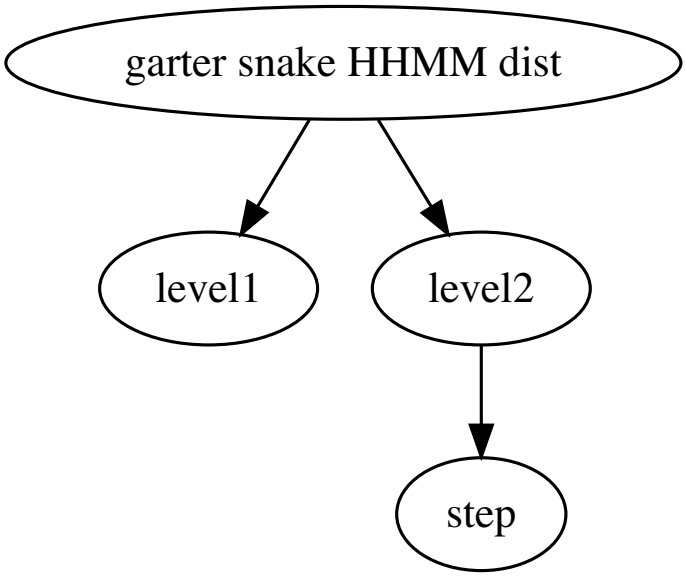
```r
W <- dim(dataAr)[3] # number of individuals
M <- dim(dataAr)[2] # number of time series per individual

### add 2 extra rows for each time step where coarse scale behavior switches occur
# level=1  indicates when coarse-scale behavior switching can occur
# level=2i indicates start of each fine-scale interval
# level=2  indicates when fine-scale behavior switching can occur
snakeData <- NULL
for(w in 1:W){
  coarseInd <- data.frame(ID=w,level=c("1","2i"),step=NA)
  for(m in 1:M){
    tmp <- rbind(coarseInd,data.frame(ID=w,level="2",step=sqrt(dataAr[,m,w])))
    snakeData <- rbind(snakeData,tmp)
  }
}

# prepare hierarchical data
snakeData <- prepData(snakeData,coordNames=NULL,hierLevels=c("1","2i","2"))

# summarize prepared data
summary(snakeData)
```

```
## Hierarchical HMM data for 19 individuals:
##
## 1 -- 606 observations
## 2 -- 606 observations
## 3 -- 606 observations
## 4 -- 606 observations
## 5 -- 606 observations
## 6 -- 606 observations
## 7 -- 606 observations
## 8 -- 606 observations
## 9 -- 606 observations
## 10 -- 606 observations
## 11 -- 606 observations
## 12 -- 606 observations
## 13 -- 606 observations
## 14 -- 606 observations
## 15 -- 606 observations
## 16 -- 606 observations
## 17 -- 606 observations
## 18 -- 606 observations
## 19 -- 606 observations
```

```
##
##
## Data summaries:
##
##       step       level
##  Min.   :0.011   1 :  114
##  1st Qu.:0.334   2i:  114
##  Median :0.700   2 :11286
##  Mean   :0.790
##  3rd Qu.:1.149
##  Max.   :3.130
##  NA's   :1496
```

The sole data stream for this example is step length at the fine-scale level, and no coordinates are provided (hence `coordNames=NULL`). As in section 3.10.1, this example has no data streams observed at the coarse-scale level, but these could be easily included (if available; see sections 3.10.3 and 3.10.4). Let's now specify the probability distribution for the fine-scale step length data stream via the `hierDist` data tree Node:

```
### data stream distributions:
### level 1 = coarse level (no data streams)
### level 2 = fine level (step="gamma")
hierDist <- data.tree::Node$new("garter snake HHMM dist")
hierDist$AddChild(name="level1")
hierDist$AddChild(name="level2")
hierDist$level2$AddChild(name="step", dist="gamma")

plot(hierDist)
```



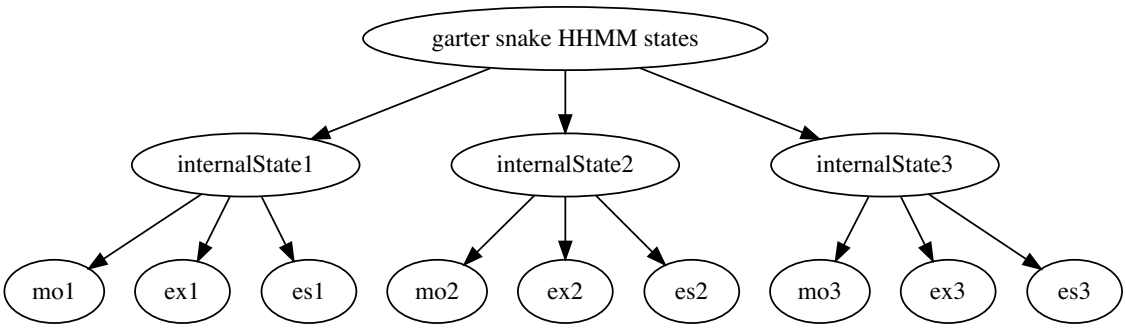
Next we define our HHMM structure via the `hierStates` data tree Node:

```
### define hierarchical HMM: states 1-3 = coarse state 1
###                          states 4-6 = coarse state 2
###                          states 7-9 = coarse state 3
hierStates <- data.tree::Node$new("garter snake HHMM states")
hierStates$AddChild(name="internalState1")
hierStates$internalState1$AddChild(name="mo1", state=1) # motionless
hierStates$internalState1$AddChild(name="ex1", state=2) # slow exploratory
hierStates$internalState1$AddChild(name="es1", state=3) # rapid escape
hierStates$AddChild(name="internalState2")
hierStates$internalState2$AddChild(name="mo2", state=4) # motionless
hierStates$internalState2$AddChild(name="ex2", state=5) # slow exploratory
hierStates$internalState2$AddChild(name="es2", state=6) # rapid escape
hierStates$AddChild(name="internalState3")
hierStates$internalState3$AddChild(name="mo3", state=7) # motionless
hierStates$internalState3$AddChild(name="ex3", state=8) # slow exploratory
hierStates$internalState3$AddChild(name="es3", state=9) # rapid escape
```

Or, equivalently:

```
hierStates <- data.tree::as.Node(list(name="garter snake HHMM states",
                                      internalState1=list(mo1=list(state=1),
                                                          ex1=list(state=2),
```

```
                                                    es1=list(state=3)),
                                        internalState2=list(mo2=list(state=4),
                                                            ex2=list(state=5),
                                                            es2=list(state=6)),
                                        internalState3=list(mo3=list(state=7),
                                                            ex3=list(state=8),
                                                            es3=list(state=9))))
plot(hierStates)
```



As in the harbor porpoise example (section 3.10.1), we follow Leos-Barajas *et al.* (2017) and assume the fine-scale data stream probability distribution does not depend on coarse-scale state:

```
# defining start values for step data stream
mu0 <- c(0.121,0.678,1.375)
sd0 <- c(0.06,0.321,0.4875)
Par0 <- list(step=c(rep(mu0,hierStates$count),rep(sd0,hierStates$count)))

nbStates <- length(hierStates$Get("state",filterFun=data.tree::isLeaf))

# constrain data stream distributions to be same for coarse-scale states
DM <- list(step=matrix(cbind(kronecker(c(1,1,1,0,0,0),diag(3)),
                             kronecker(c(0,0,0,1,1,1),diag(3))),
                       nrow=nbStates*2,
```

```
                        ncol=6,
                        dimnames=list(c(paste0("mean_",1:nbStates),
                                        paste0("sd_",1:nbStates)),
                                      paste0(rep(c("mean","sd"),each=3),
                                             c("_147:(Intercept)",
                                               "_258:(Intercept)",
                                               "_369:(Intercept)")))))

# initial parameter values for data stream probability distributions
Par <- getParDM(snakeData,
                hierStates=hierStates,hierDist=hierDist,
                Par=Par0,DM=DM)
```

We do not include any covariates in `hierFormula` or `hierFormulaDelta`, so all that's left before fitting the model is the (optional) step of specifying starting values for the t.p.m. and initial distribution at each level of the hierarchy (based on values reported by Leos-Barajas *et al.* 2017):

```
hierBeta <- data.tree::Node$new("garter snake beta")
hierBeta$AddChild(name="level1")
hierBeta$AddChild(name="level2")
hierBeta$level2$AddChild(name="internalState1")
hierBeta$level2$AddChild(name="internalState2")
hierBeta$level2$AddChild(name="internalState3")

hierDelta <- data.tree::Clone(hierBeta)
hierDelta$name <- "garter snake delta"

# reference states for level1
level1states <- hierStates$Get(function(x) data.tree::Aggregate(x,"state",min),
                               filterFun=function(x) x$level==2)

hierBeta$level1$beta <- matrix(c(1.24, 0.44, 1.1, -0.87, -1.40, -1.11),
                               nrow=1,
                               ncol=hierStates$count*(hierStates$count-1),
                               byrow=TRUE,
                               dimnames=list("(Intercept)",
                                        c(sapply(level1states,function(x)
                                          paste(
                                           rep(x,each=hierStates$count-1),
                                           "->",
```

```
                                        level1states[-which(level1states==x)])))))

hierDelta$level1$delta <- matrix(rep(c(-2.5,-3.5),hierStates$count-1),
                                 nrow=1,
                                 ncol=(hierStates$count-1),
                                 byrow=TRUE,
                                 dimnames=list("(Intercept)",
                                               paste("state",level1states[-1])))

beta0_level2 <- delta0_level2 <- list()
beta0_level2$internalState1 <- c(-2.99, -5.06, 3.93, 1.25, 34.84, 35.97)
beta0_level2$internalState2 <- c(-1.72, -2.78, 3.54, 2.83, 34.72, 36.21)
beta0_level2$internalState3 <- c(-5.10, -17.69, 5.76, -11.34, 33.39, 37.37)

delta0_level2$internalState1 <- c(-1.39, 0.16)
delta0_level2$internalState2 <- c(-1.27, 2.33)
delta0_level2$internalState3 <- c(0.34, -0.26)

for(jj in 1:hierStates$count){
  j <-  names(hierStates$children)[jj]

  # reference states for internalState j
  ref <- hierStates[[j]]$Get(function(x)
                             data.tree::Aggregate(x,"state",min),
                            filterFun=function(x) x$level==2)

  # states for internalState j
  states <- hierStates[[j]]$Get("state",filterFun = isLeaf)

  dimNames <- list("(Intercept)",
                   paste0(rep(states,each=hierStates[[j]]$count-1),
                          " -> ",
                          states[-which(states==ref)]))

  hierBeta$level2[[j]]$beta <- matrix(beta0_level2[[j]],
                                      nrow=1,
                                      ncol=hierStates[[j]]$count*
                                           (hierStates[[j]]$count-1),
                                      byrow=TRUE,
                                      dimnames=dimNames)

  hierDelta$level2[[j]]$delta <- matrix(delta0_level2[[j]],
```

```
                                        nrow=1,
                                        ncol=(hierStates[[j]]$count-1),
                                        byrow=TRUE,
                                        dimnames=list("(Intercept)",
                                                      names(states)[-1]))
}
```

Let's check our model specification:

```
checkPar0(snakeData,hierStates=hierStates,hierDist=hierDist,
          Par0=Par,DM=DM,
          hierBeta=hierBeta,hierDelta=hierDelta)

##
## Regression coeffs for step parameters:
## -------------------------------------
##      mean_147:(Intercept) mean_258:(Intercept) mean_369:(Intercept)
## [1,]            -2.111965            -0.388608             0.3184537
##      sd_147:(Intercept) sd_258:(Intercept) sd_369:(Intercept)
## [1,]          -2.813411          -1.136314          -0.718465
##
##
## ------------------------------------------------------------
## Regression coeffs for the transition probabilities (beta):
## ------------------------------------------------------------
## -------------------------  level1  ------------------------
##                       1 -> 4 1 -> 7 4 -> 1 4 -> 7 7 -> 1 7 -> 4
## I((level == "1") * 1)   1.24   0.44    1.1  -0.87   -1.4  -1.11
##
## -------------------------  level2  ------------------------
##                       1 -> 2 1 -> 3 2 -> 2 2 -> 3 3 -> 2 3 -> 3
## I((level == "2") * 1)  -2.99  -5.06   3.93   1.25  34.84  35.97
##
##                       4 -> 5 4 -> 6 5 -> 5 5 -> 6 6 -> 5 6 -> 6
## I((level == "2") * 1)  -1.72  -2.78   3.54   2.83  34.72  36.21
##
##                       7 -> 8 7 -> 9 8 -> 8 8 -> 9 9 -> 8 9 -> 9
## I((level == "2") * 1)   -5.1 -17.69   5.76 -11.34  33.39  37.37
##
## ------------------------------------------------------------
##
## ------------------------------------------------------------
```

```
## Regression coeffs for the initial distribution (delta):
## ------------------------------------------------------------
## ------------------------  level1  ------------------------
##             state 4 state 7
## (Intercept)    -2.5    -3.5
##
## ------------------------  level2  ------------------------
##                          state 2 state 3
## I((level == "2i") * 1)   -1.39    0.16
##
##                          state 5 state 6
## I((level == "2i") * 1)   -1.27    2.33
##
##                          state 8 state 9
## I((level == "2i") * 1)    0.34   -0.26
##
## ------------------------------------------------------------
```

Again note that higher-level (i.e. "parent") states are indexed based on the lowest state index of their "children". For example, "internalState1" is indexed by state 1, "internalState2" is indexed by state 4, and "internalState3" is indexed by state 7. We can also examine the starting values for the t.p.m. on the real scale ("`gamma`") at each level of the hierarchy using the `getTrProbs` function:

```
iTrProbs <- getTrProbs(snakeData,hierStates=hierStates,
                       hierBeta=hierBeta,hierDist=hierDist)

# t.p.m. at first time step for level1
iTrProbs$level1$gamma[,,1]

##                internalState1 internalState2 internalState3
## internalState1      0.1664359      0.5751380     0.25842616
## internalState2      0.6791965      0.2260849     0.09471861
## internalState3      0.1564547      0.2090903     0.63445500

# t.p.m. at first time step for level2
lapply(iTrProbs$level2,function(x) x$gamma[,,1])

## $internalState1
##              mo1        ex1         es1
## mo1 9.464024e-01 0.04759215 0.006005453
```

```
## ex1 1.805141e-02 0.91894296 0.063005628
## es1 1.806578e-16 0.24416110 0.755838899
##
## $internalState2
##              mo2       ex2        es2
## mo2 8.057338e-01 0.1442797 0.04998652
## ex2 1.907946e-02 0.6576103 0.32331027
## es2 1.534365e-16 0.1839217 0.81607827
##
## $internalState3
##              mo3         ex3          es3
## mo3 9.939402e-01 0.006059801 2.063911e-08
## ex3 3.141213e-03 0.996858749 3.734204e-08
## es3 5.785955e-17 0.018342891 9.816571e-01
```

Now let's fit our garter snake HHMM:

```
# fit hierarchical HMM
hhmm <- fitHMM(snakeData,hierStates=hierStates,hierDist=hierDist,
               Par0=Par,DM=DM,hierBeta=hierBeta,hierDelta=hierDelta,
               nlmPar=list(hessian=FALSE))

## ======================================================================
## Fitting hierarchical HMM with 9 states and 1 data stream
## ----------------------------------------------------------------------
##  step ~ gamma(mean:  custom, sd:  custom)
##
##  Transition probability matrix formula:  ~0 + I((level == "1") * 1) +
I((level == "2i") * 1) + I((level == "2") * 1)
##
##  Initial distribution formula:  ~1
## ======================================================================
## DONE
```

```
hhmm

## Value of the maximum log-likelihood: -2060.361
##
```

```
##
## step parameters:
## ----------------
##             mo1       ex1       es1        mo2       ex2       es2
## mean 0.12148244 0.6778589 1.3749453 0.12148244 0.6778589 1.3749453
## sd   0.05906919 0.3217373 0.4876126 0.05906919 0.3217373 0.4876126
##             mo3       ex3       es3
## mean 0.12148244 0.6778589 1.3749453
## sd   0.05906919 0.3217373 0.4876126
##
##
## ------------------------------------------------------------
## Regression coeffs for the transition probabilities:
## ------------------------------------------------------------
## -------------------------  level1  -------------------------
##                          1 -> 4    1 -> 7   4 -> 1    4 -> 7    7 -> 1
## I((level == "1") * 1) 1.245638 0.4309528 1.102461 -0.870203 -1.397521
##                           7 -> 4
## I((level == "1") * 1) -1.114278
##
## -------------------------  level2  -------------------------
##                           1 -> 2    1 -> 3   2 -> 2   2 -> 3   3 -> 2
## I((level == "2") * 1) -2.993152 -5.064987 3.927787 1.246237 34.83412
##                           3 -> 3
## I((level == "2") * 1) 35.96589
##
##                           4 -> 5    4 -> 6   5 -> 5   5 -> 6  6 -> 5
## I((level == "2") * 1) -1.721131 -2.775771 3.540737 2.834405 34.7087
##                           6 -> 6
## I((level == "2") * 1) 36.19201
##
##                           7 -> 8    7 -> 9   8 -> 8    8 -> 9   9 -> 8
## I((level == "2") * 1) -5.098125 -17.69051 5.761101 -11.34101 33.38974
##                           9 -> 9
## I((level == "2") * 1) 37.36932
##
## ------------------------------------------------------------
##
## ------------------------------------------------------------
## Transition probability matrix (based on mean covariate values):
## ------------------------------------------------------------
## -------------------------  level1  -------------------------
```

```
##                internalState1 internalState2 internalState3
## internalState1      0.1662821      0.5778557     0.25586212
## internalState2      0.6797456      0.2257114     0.09454295
## internalState3      0.1569221      0.2083028     0.63477506
##
## -------------------------  level2  --------------------------
##              mo1        ex1         es1
## mo1 9.465724e-01 0.04745091 0.005976649
## ex1 1.809245e-02 0.91899585 0.062911698
## es1 1.814804e-16 0.24383417 0.756165834
##
##              mo2       ex2        es2
## mo2 8.056946e-01 0.1441095 0.05019591
## ex2 1.904305e-02 0.6568391 0.32411785
## es2 1.560287e-16 0.1849270 0.81507302
##
##              mo3         ex3          es3
## mo3 9.939289e-01 0.006071107 2.062827e-08
## ex3 3.137768e-03 0.996862195 3.726331e-08
## es3 5.789852e-17 0.018350461 9.816495e-01
##
## ------------------------------------------------------------
##
## ------------------------------------------------
## Regression coeffs for the initial distribution:
## ------------------------------------------------
## --------------------  level1  --------------------
##               state 4   state 7
## (Intercept) -2.536649 -3.509269
##
## --------------------  level2  --------------------
##                           state 2   state 3
## I((level == "2i") * 1) -1.387284 0.1582614
##
##                           state 5  state 6
## I((level == "2i") * 1) -1.274323 2.327475
##
##                           state 8    state 9
## I((level == "2i") * 1) 0.3387744 -0.2607024
##
## ------------------------------------------------
##
```

```
## ------------------------------------------------
## Initial distribution:
## ------------------------------------------------
## --------------------  level1  --------------------
##       internalState1 internalState2 internalState3
## ID:1       0.9016727     0.07135036     0.02697695
## ID:2       0.9016727     0.07135036     0.02697695
## ID:3       0.9016727     0.07135036     0.02697695
## ID:4       0.9016727     0.07135036     0.02697695
## ID:5       0.9016727     0.07135036     0.02697695
## ID:6       0.9016727     0.07135036     0.02697695
## ID:7       0.9016727     0.07135036     0.02697695
## ID:8       0.9016727     0.07135036     0.02697695
## ID:9       0.9016727     0.07135036     0.02697695
## ID:10      0.9016727     0.07135036     0.02697695
## ID:11      0.9016727     0.07135036     0.02697695
## ID:12      0.9016727     0.07135036     0.02697695
## ID:13      0.9016727     0.07135036     0.02697695
## ID:14      0.9016727     0.07135036     0.02697695
## ID:15      0.9016727     0.07135036     0.02697695
## ID:16      0.9016727     0.07135036     0.02697695
## ID:17      0.9016727     0.07135036     0.02697695
## ID:18      0.9016727     0.07135036     0.02697695
## ID:19      0.9016727     0.07135036     0.02697695
##
## --------------------  level2  --------------------
##                       mo1       ex1       es1
## internalState1 0.4130141 0.1031514 0.4838346
##
##                       mo2        ex2      es2
## internalState2 0.0867179 0.02424808 0.889034
##
##                      mo3       ex3      es3
## internalState3 0.315086 0.4421371 0.242777
##
## ------------------------------------------------
```

These estimates are virtually identical to Leos-Barajas *et al.* (2017); the only (very slight) difference is in the estimates for the coarse-scale initial distribution ($\boldsymbol{\delta}^{(0)}$) because, unlike in Leos-Barajas *et al.* (2017), the forward algorithm in `momentuHMM` (Eq. 1) includes a state transition between time steps $t = 0$ and $t = 1$.

As usual, we can check pseudo-residuals using `plotPR` (Figure 15):

```
plotPR(hhmm)
```

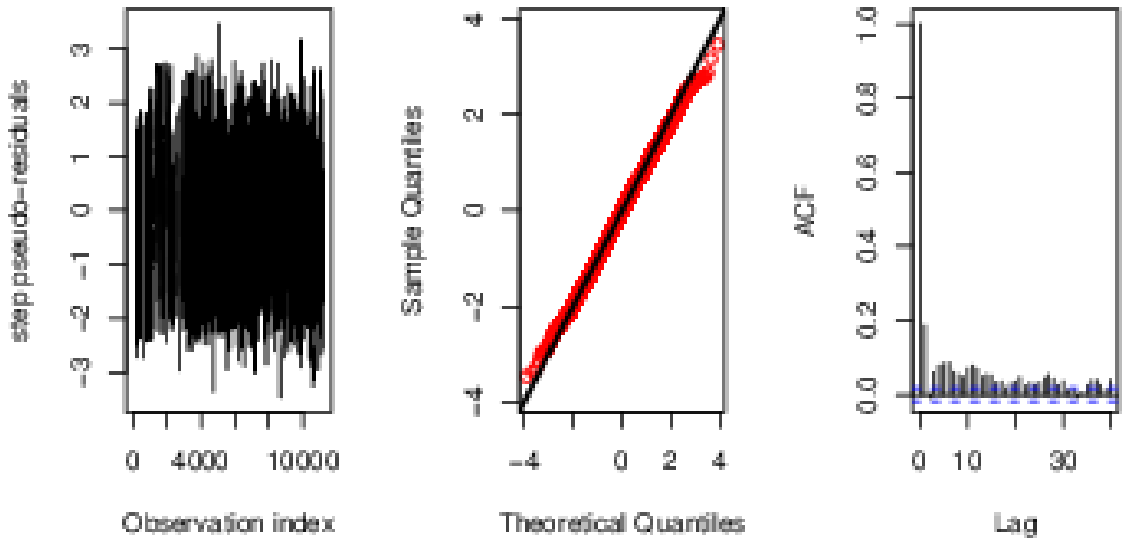


Figure 15. *Pseudo-residual plot for the HHMM garter snake example.*

and simulate from our fitted HHMM using `simHierData`:

```
obsPerLevel <- data.tree::Node$new("simHierData")

# number of level 1 observations
obsPerLevel$AddChild("level1",obs=M)

# number of level 2 observations that follow each level 1 observation
obsPerLevel$AddChild("level2",obs=dim(dataAr)[1])

simHHMM <- simHierData(nbAnimals=W,
                       model=hhmm,
                       obsPerLevel = obsPerLevel, states = TRUE)

## ======================================================================
## Simulating hierarchical HMM with 9 states and 1 data stream
## ----------------------------------------------------------------------
##  step ~ gamma(mean:  custom, sd:  custom)
```

```
##
##  Transition probability matrix formula:  ~0 + I((level == "1") * 1) +
I((level == "2i") * 1) + I((level == "2") * 1)
##
##  Initial distribution formula:  ~1
## ============================================================================
## DONE
```

```
head(simHHMM)
```

```
##   ID level     step        x  y states
## 1  1     1       NA       NA NA      4
## 2  1    2i       NA       NA NA      6
## 3  1     2 1.992639 0.000000  0      6
## 4  1     2 1.182153 1.992639  0      5
## 5  1     2 1.045767 3.174792  0      5
## 6  1     2 1.285761 4.220559  0      6
```

3.10.3 Atlantic cod

Now we will demonstrate how HHMMs with data streams observed at multiple time scales can be fitted using `momentuHMM`. In their Atlantic cod example, Adam *et al.* (2019) fit a 9-state HHMM to coarse-scale horizontal (i.e., step length and turn angle) and fine-scale vertical movement data. The coarse-scale states were "resting/foraging" (hereafter "resForage"), "mobile/foraging" (hereafter "mobForage"), and "travelling/migrating" (hereafter "transit"), each of which was composed of three fine-scale states. To begin our analysis, we must first load and prepare the data (available for download from Adam *et al.* 2019):

```
# load the data from Adam et al
load("Atlantic_cod_data_set.RData")
```

```
# coarse-scale data
data <- data.frame(level="1",
                   step=steps,
                   angle=angles,
                   vertical=NA,
```

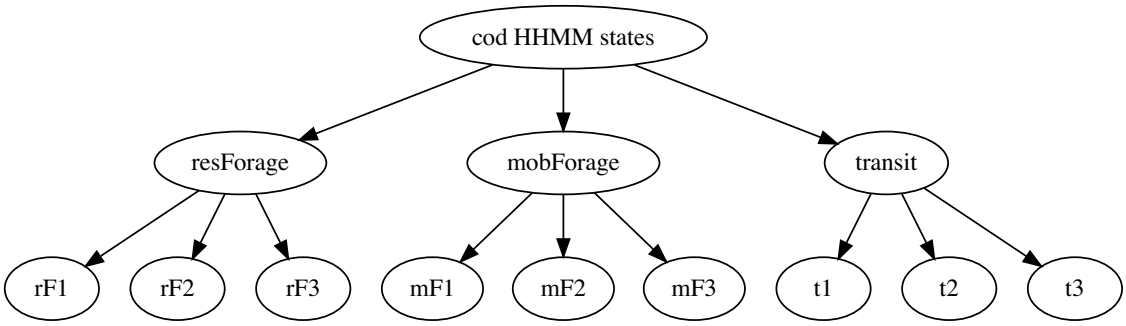
```
                  time=0)

### add extra rows for fine-scale data
# level=1  indicates when coarse-scale behavior switching can occur
# level=2i indicates start of each fine-scale interval
# level=2  indicates when fine-scale behavior switching can occur
codData <- NULL
timeSeq <- seq(from=0,to=23+5/6,length=144) # time of day covariate
for(i in 1:nrow(data)){
  fineInd <- data.frame(level="2",
                        step=NA,
                        angle=NA,
                        vertical=verticals[[i]],
                        time=timeSeq)
  tmp <- rbind(data[i,,drop=FALSE],
               data.frame(level="2i",
                          step=NA,
                          angle=NA,
                          vertical=NA,
                          time=0),
               fineInd)
  codData <- rbind(codData,tmp)
}

# prepare hierarchical data
codData <- prepData(codData, coordNames=NULL,
                    covNames="time",
                    hierLevels=c("1","2i","2"))
head(codData)

##        ID level     step angle vertical      time
## 1 Animal1     1 5.236266    NA       NA 0.0000000
## 2 Animal1    2i       NA    NA       NA 0.0000000
## 3 Animal1     2       NA    NA   0.1285 0.0000000
## 4 Animal1     2       NA    NA   0.0499 0.1666667
## 5 Animal1     2       NA    NA   0.1417 0.3333333
## 6 Animal1     2       NA    NA   0.1713 0.5000000

# data summary
summary(codData,dataNames=names(codData)[-1])

## Hierarchical HMM data for 1 individual:
##
```

```
## Animal1 -- 42486 observations
##
##
## Data summaries:
##
##  level           step           angle          vertical
##  1 :  291   Min.   : 0.21   Min.   :-3.12   Min.   : 0.000
##  2i:  291   1st Qu.: 2.86   1st Qu.:-0.36   1st Qu.: 0.083
##  2 :41904   Median : 5.49   Median : 0.01   Median : 0.182
##             Mean   : 8.24   Mean   :-0.03   Mean   : 0.418
##             3rd Qu.:11.89   3rd Qu.: 0.34   3rd Qu.: 0.440
##             Max.   :31.49   Max.   : 3.12   Max.   :18.223
##             NA's   :42196   NA's   :42197   NA's   :582
##       time
##  Min.   : 0.0
##  1st Qu.: 5.7
##  Median :11.8
##  Mean   :11.8
##  3rd Qu.:17.8
##  Max.   :23.8
##
```

From sections 3.10.1 and 3.10.2, we should now be familiar with how to specify HHMMs with state transitions at multiple time scales. We will therefore focus on how to accommodate data streams that are observed at multiple time scales here, but complete details and code for fitting this model can be found in the "codExample.R" script in the `momentuHMM` "vignettes" source directory (or at https://github.com/bmcclintock/momentuHMM). First we specify the hierarchical nature of the states as a `data.tree` Node:

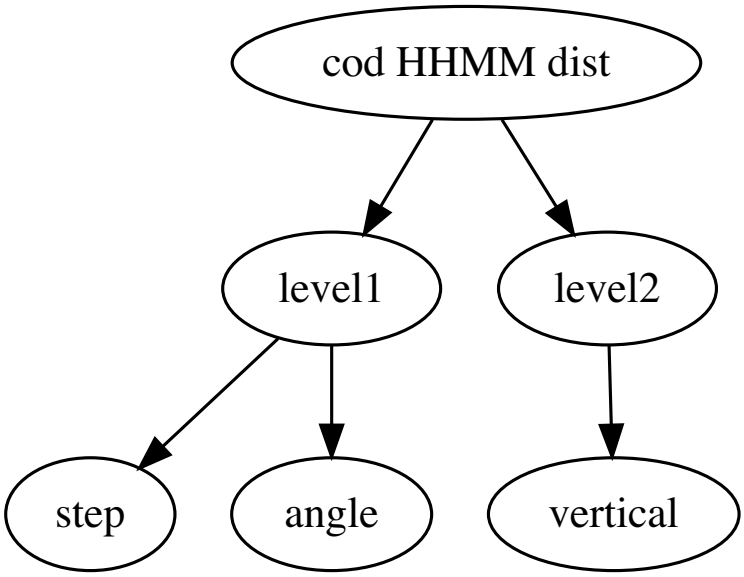
```
### define hierarchical HMM
# states 1-3 = coarse state 1 (resident/foraging)
# states 4-6 = coarse state 2 (mobile/foraging)
# states 7-9 = coarse state 3 (travelling/migrating)
hierStates <- data.tree::Node$new("cod HHMM states")
hierStates$AddChild("resForage")   # resident/foraging
hierStates$resForage$AddChild("rF1", state=1)
hierStates$resForage$AddChild("rF2", state=2)
hierStates$resForage$AddChild("rF3", state=3)
hierStates$AddChild("mobForage")  # mobile/foraging
```

```
hierStates$mobForage$AddChild("mF1", state=4)
hierStates$mobForage$AddChild("mF2", state=5)
hierStates$mobForage$AddChild("mF3", state=6)
hierStates$AddChild("transit")    # travelling/migrating
hierStates$transit$AddChild("t1", state=7)
hierStates$transit$AddChild("t2", state=8)
hierStates$transit$AddChild("t3", state=9)

plot(hierStates)
```



Next we specify the hierarchical nature of the data streams as a `data.tree` Node:

```
# data stream distributions
# level 1 = coarse level (step="gamma", angle="vm")
# level 2 = fine level (vertical="gamma")
hierDist <- data.tree::Node$new("cod HHMM dist")
hierDist$AddChild("level1")
hierDist$level1$AddChild("step", dist="gamma")
hierDist$level1$AddChild("angle", dist="vm")
hierDist$AddChild("level2")
hierDist$level2$AddChild("vertical", dist="gamma")

plot(hierDist)
```



We then constrain the fine-scale states within each coarse-scale state to have the same parameters for the "step" and "angle" distributions using the `DM` argument:

```
nbStates <- length(hierStates$Get("state",filterFun=data.tree::isLeaf))

# constrain coarse-scale parameters for fine-scale states
DM <- list(step=matrix(kronecker(diag(6),c(1,1,1)),
                       nrow=2*nbStates,
                       ncol=6,
                       dimnames=list(paste0(rep(c("mean_","sd_"),each=nbStates)
                                            ,1:nbStates),
                                     c(paste0(rep(c("mean_","sd_"),each=3),
                                              1:length(hierStates$children),
                                              ":(Intercept)")))))
DM$angle <- DM$step
dimnames(DM$angle) <- list(paste0(rep(c("mean_","concentration_"),each=nbStates)
                                  ,1:nbStates),
                           c(paste0(rep(c("mean_","concentration_"),each=3),
                                    1:length(hierStates$children),
                                    ":(Intercept)")))
```

and obtain starting values on the working scale using `getParDM`:

```
### defining start values based on those reported by Adam et al
hm.mu0 <- c(5.482, 6.786, 14.914)
hm.sigma0 <- c(4.27, 4.714, 11.242)

ha.mu0 <- c(0.011, -0.299, 0.044)
ha.kappa0 <- c(1.571, 1.426, 2.15)

vm.mu0 <- vm.sigma0 <- vm.pi0 <- list()
vm.mu0[[1]] <- c(0.116, 0.303, 0.691)
vm.mu0[[2]] <- c(0.109, 0.056, 0.351)
vm.mu0[[3]] <- c(0.125, 0.514, 1.987)

vm.sigma0[[1]] <- c(0.096, 0.261, 0.636)
vm.sigma0[[2]] <- c(0.043, 0.047, 0.342)
vm.sigma0[[3]] <- c(0.109, 0.462, 1.878)

vm.pi0[[1]] <- c(0.014, 0.003, 1.050e-06)
vm.pi0[[2]] <- c(1.791e-08, 0.035, 0.002)
vm.pi0[[3]] <- c(0.012, 2.933e-04, 3.462e-09)

Par0 <- list(step=c(rep(hm.mu0,each=3),rep(hm.sigma0,each=3)),
             angle=c(rep(ha.mu0,each=3),rep(ha.kappa0,each=3)),
             vertical=c(unlist(vm.mu0),unlist(vm.sigma0),unlist(vm.pi0)))

# starting values for data stream parameters on the working scale
Par <- getParDM(codData,
                hierStates=hierStates,
                hierDist=hierDist,
                Par=Par0,
                DM=DM,
                estAngleMean = list(angle=TRUE))
```

Adam *et al.* (2019) included a periodic time-of-day covariate on the fine-scale state transition probabilities, and we specify this via the `hierFormula` argument:

```
# define hierarchical t.p.m. formula(s)
hierFormula <- data.tree::Node$new("cod HHMM formula")
hierFormula$AddChild("level1", formula=~1)
hierFormula$AddChild("level2", formula=~cosinor(time, period=24))
```

All that remains is (optionally) specifying starting values for the initial distribution

(`hierDelta`) and t.p.m. (`hierBeta`) parameters, which we'll base on those reported by Adam *et al.* (2019) to speed up the optimization:

```
hierBeta <- data.tree::Node$new("cod beta")
hierBeta$AddChild("level1",
          beta=matrix(c(-18.585, -2.86, -2.551, -1.641, -2.169, -2.415),
                      nrow=1,
                      ncol=length(hierStates$children)
                       *(length(hierStates$children)-1)))
hierBeta$AddChild("level2")
hierBeta$level2$AddChild("resForage",
          beta=matrix(c(-2.562, -3.403,  2.765, -1.607,  2.273,  4.842,
                         -0.665,  -0.26, -0.681, -0.149, -2.728, -2.798,
                         -0.027,   0.26,  0.191,  0.667,  0.123, -0.262),
                      nrow=3,
                      ncol=length(hierStates$resForage$children)
                       *(length(hierStates$resForage$children)-1),
                      byrow=TRUE))
hierBeta$level2$AddChild("mobForage",
          beta=matrix(c(-2.156, -3.662,   3.01,  0.597, -0.313,  2.897,
                          0.067,  -1.22, -0.799, -0.797,   0.15,  0.379,
                         -0.112, -0.195, -0.269, -0.215,  1.539,  0.728),
                      nrow=3,
                      ncol=length(hierStates$mobForage$children)
                       *(length(hierStates$mobForage$children)-1),
                      byrow=TRUE))
hierBeta$level2$AddChild("transit",
          beta=matrix(c(-2.53, -4.279,  2.507, -0.228, 10.803, 12.873,
                         -0.04,  1.221, -0.301,  0.284, -0.106, -0.077,
                         0.629, -0.226, -0.253, -0.303,  0.011,  0.036),
                      nrow=3,
                      ncol=length(hierStates$transit$children)
                       *(length(hierStates$transit$children)-1),
                      byrow=TRUE))

hierDelta <- data.tree::Node$new("cod delta")
hierDelta$AddChild("level1",delta=matrix(c(15.776, 4.78),1))
hierDelta$AddChild("level2")
hierDelta$level2$AddChild("resForage",delta=matrix(c(-0.643, -2.416),1))
hierDelta$level2$AddChild("mobForage",delta=matrix(c(1.181, 0.46),1))
hierDelta$level2$AddChild("transit",delta=matrix(c(-0.357, -0.624),1))
```

```r
# check hierarchical model specification and parameters
checkPar0(codData,
          hierStates = hierStates,
          hierDist = hierDist,
          hierFormula = hierFormula,
          Par0 = Par, hierBeta = hierBeta, hierDelta = hierDelta,
          DM = DM,
          estAngleMean = list(angle=TRUE))
```

```
##
## Regression coeffs for step parameters:
## --------------------------------------
##      mean_1:(Intercept) mean_2:(Intercept) mean_3:(Intercept)
## [1,]            1.70147           1.914862             2.7023
##      sd_1:(Intercept) sd_2:(Intercept) sd_3:(Intercept)
## [1,]         1.451614         1.550537         2.419657
##
## Regression coeffs for angle parameters:
## ---------------------------------------
##      mean_1:(Intercept) mean_2:(Intercept) mean_3:(Intercept)
## [1,]        0.005500055         -0.1506238         0.02200355
##      concentration_1:(Intercept) concentration_2:(Intercept)
## [1,]                   0.4517124                   0.3548733
##      concentration_3:(Intercept)
## [1,]                   0.7654678
##
## vertical parameters:
## --------------------
##            rF1   rF2      rF3       mF1   mF2   mF3    t1        t2
## mean     0.116 0.303 6.91e-01 1.090e-01 0.056 0.351 0.125 0.5140000
## sd       0.096 0.261 6.36e-01 4.300e-02 0.047 0.342 0.109 0.4620000
## zeromass 0.014 0.003 1.05e-06 1.791e-08 0.035 0.002 0.012 0.0002933
##                 t3
## mean     1.987e+00
## sd       1.878e+00
## zeromass 3.462e-09
##
## ------------------------------------------------------------
## Regression coeffs for the transition probabilities (beta):
## ------------------------------------------------------------
## -------------------------  level1  ------------------------
##                          1 -> 4 1 -> 7 4 -> 1 4 -> 7 7 -> 1 7 -> 4
## I((level == "1") * 1) -18.585  -2.86 -2.551 -1.641 -2.169 -2.415
```

```
##
## ------------------------  level2  ----------------------
##                                                    1 -> 2 1 -> 3 2 -> 2
## I((level == "2") * 1)                              -2.562 -3.403  2.765
## I((level == "2") * cosinorCos(time, period = 24)) -0.665 -0.260 -0.681
## I((level == "2") * cosinorSin(time, period = 24)) -0.027  0.260  0.191
##                                                    2 -> 3 3 -> 2 3 -> 3
## I((level == "2") * 1)                              -1.607  2.273  4.842
## I((level == "2") * cosinorCos(time, period = 24)) -0.149 -2.728 -2.798
## I((level == "2") * cosinorSin(time, period = 24))  0.667  0.123 -0.262
##
##                                                    4 -> 5 4 -> 6 5 -> 5
## I((level == "2") * 1)                              -2.156 -3.662  3.010
## I((level == "2") * cosinorCos(time, period = 24))  0.067 -1.220 -0.799
## I((level == "2") * cosinorSin(time, period = 24)) -0.112 -0.195 -0.269
##                                                    5 -> 6 6 -> 5 6 -> 6
## I((level == "2") * 1)                               0.597 -0.313  2.897
## I((level == "2") * cosinorCos(time, period = 24)) -0.797  0.150  0.379
## I((level == "2") * cosinorSin(time, period = 24)) -0.215  1.539  0.728
##
##                                                    7 -> 8 7 -> 9 8 -> 8
## I((level == "2") * 1)                              -2.530 -4.279  2.507
## I((level == "2") * cosinorCos(time, period = 24)) -0.040  1.221 -0.301
## I((level == "2") * cosinorSin(time, period = 24))  0.629 -0.226 -0.253
##                                                    8 -> 9 9 -> 8 9 -> 9
## I((level == "2") * 1)                              -0.228 10.803 12.873
## I((level == "2") * cosinorCos(time, period = 24))  0.284 -0.106 -0.077
## I((level == "2") * cosinorSin(time, period = 24)) -0.303  0.011  0.036
##
## ------------------------------------------------------------
##
## ------------------------------------------------------------
## Regression coeffs for the initial distribution (delta):
## ------------------------------------------------------------
## ------------------------  level1  ------------------------
##             state 4 state 7
## (Intercept)  15.776    4.78
##
## ------------------------  level2  ------------------------
##                         state 2 state 3
## I((level == "2i") * 1)  -0.643  -2.416
##
```

```
##                          state 5 state 6
## I((level == "2i") * 1)     1.181    0.46
##
##                          state 8 state 9
## I((level == "2i") * 1)    -0.357  -0.624
##
## ------------------------------------------------------------
```

and we are now ready to fit the HHMM:

```
hhmm <- fitHMM(codData,
               hierStates = hierStates,
               hierDist = hierDist,
               hierFormula = hierFormula,
               Par0 = Par, hierBeta = hierBeta, hierDelta = hierDelta,
               DM = DM,
               estAngleMean = list(angle=TRUE))

plot(hhmm, plotCI=TRUE, ask=FALSE)

# plot stationary distributions and CIs
plotStationary(hhmm, plotCI=TRUE)
```

The resulting estimates are virtually identical to Adam *et al.* (2019); the slight differences are attributable to: 1) the forward algorithm in `momentuHMM` (Eq. 1) includes a state transition between time steps $t = 0$ and $t = 1$; and 2) `momentuHMM` uses the lowest fine-scale state index for each coarse-scale state as the mlogit-link reference state for the initial distribution and t.p.m.Nevertheless, we can see that the estimated coarse-scale data stream probability distribution (Figure 16) and fine-scale stationary probabilities as a function of time of day (Figure 17) are very similar.

3.10.4 Horn shark

For our final HHMM example, we'll quickly demonstrate how the horn shark example from Adam *et al.* (2019) can be fitted in `momentuHMM`. The data streams in this example consist of coarse-scale categorical step lengths (`stepCat`) based on estimated geopositions at 2 second intervals over 194 distinct segments and, within each 2 second interval, fine-scale accelerometer data that were summarized as 50 sequential values of overall dynamic body acceleration (`odba`). The analysis included 8 categories for step

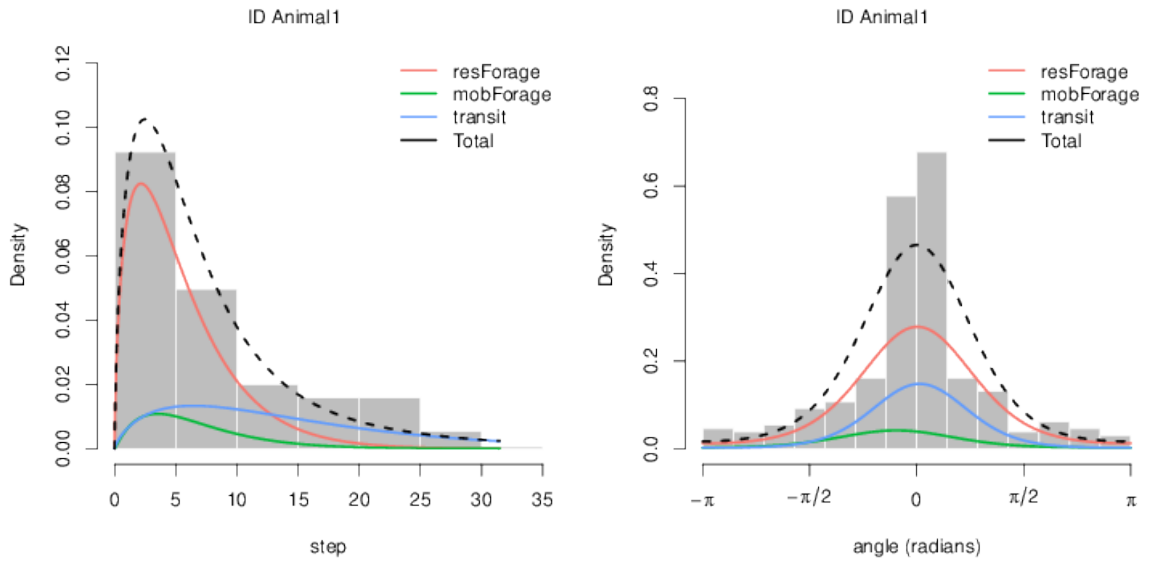


Figure 16. *Estimated state-dependent distributions of coarse-scale step lengths (left panel) and turning angles (right panel) of an Atlantic cod.*

length to construct a so-called histogram distribution of step lengths, where category 1 indicates zero step lengths and categories $2-8$ were defined by increasing cutoffs at 0.00075, 0.00125, 0.00175, 0.00225, 0.00275, 0.00325, and 0.00375 m, respectively. The 9-state HHMM included three coarse-scale states ("activity", "resting", and "transit"), each composed of three fine-scale states.

First we load and prepare the data (available for download from Adam *et al.* 2019):

```
# load the data from Adam et al
load("horn_shark_data_set.RData")
```

```
# coarse-scale data
data <- data.frame(ID=unlist(mapply(function(x) rep(paste0("seg",x),
                                                    nrow(steps[[x]])-1),
                                    1:length(steps))),
                   level="1",
                   steps=unlist(lapply(steps,function(x) x$steps[-nrow(x)])),
                   stepCat=unlist(lapply(steps,function(x) x$cats[-nrow(x)])),
                   odbas=NA)
data$stepCat[which(is.na(data$steps))] <- NA

### add extra rows for fine-scale data
```

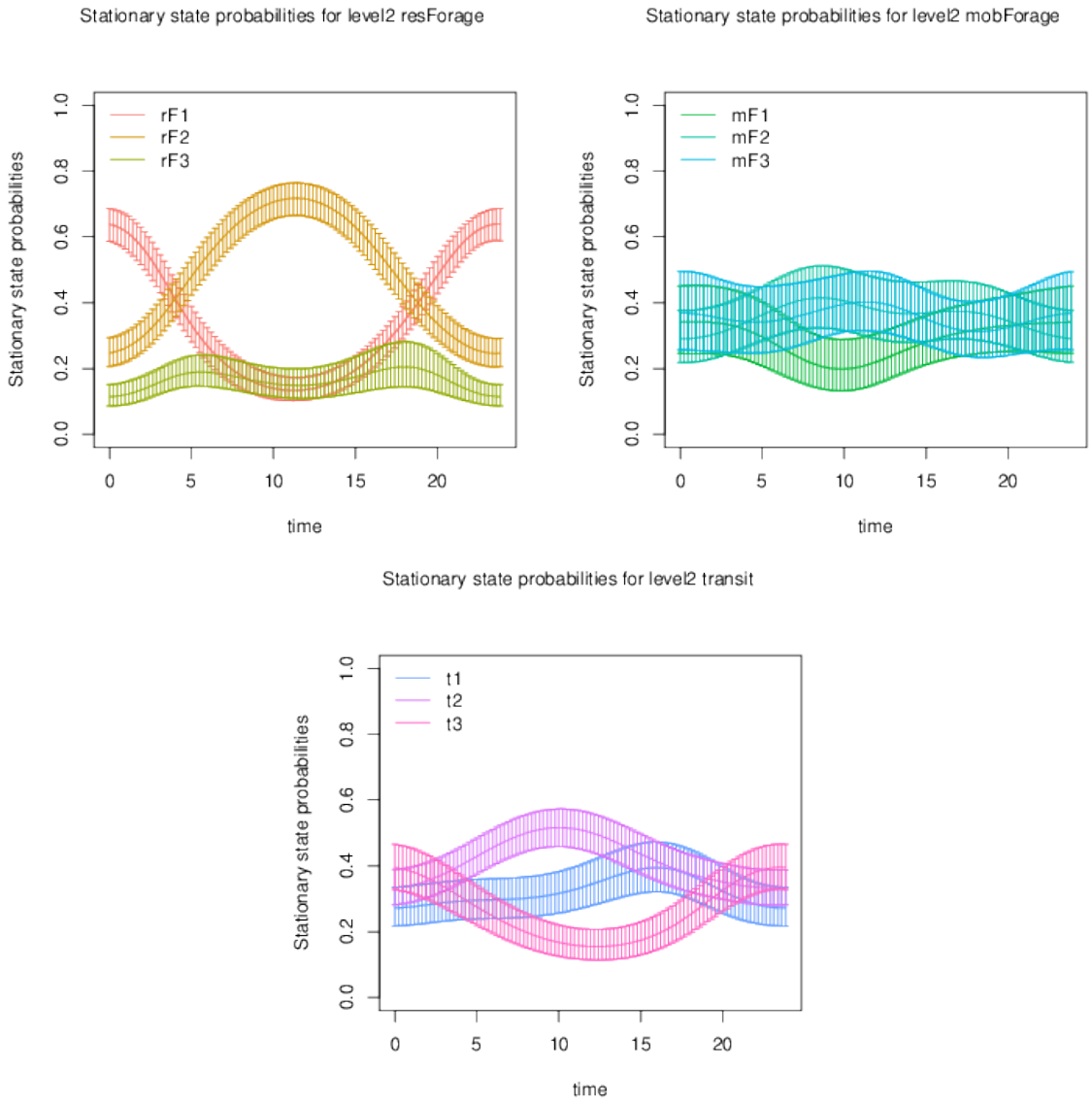


Figure 17. *Stationary distributions of the fine-scale state processes for an Atlantic cod as a function of time of day for the coarse-scale states corresponding to "resting/foraging" (top-left panel), "mobile/foraging" (top-right panel), and "travelling/migrating" (bottom panel).*

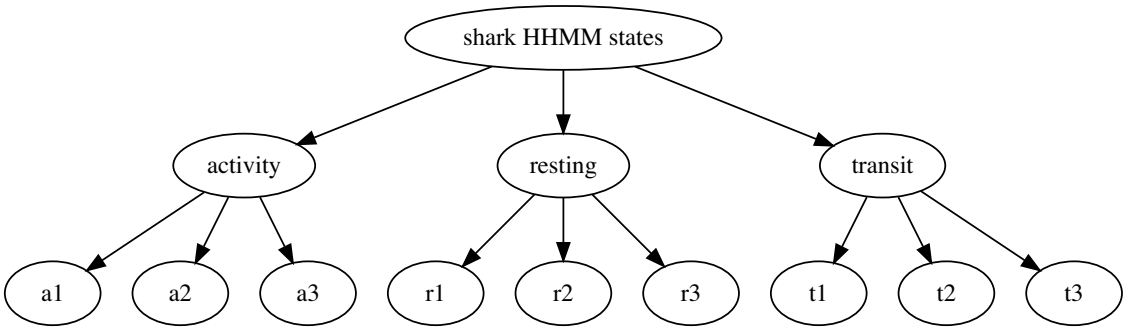
```
# level=1  indicates when coarse-scale behavior switching can occur
# level=2i indicates start of each fine-scale interval
# level=2  indicates when fine-scale behavior switching can occur
odbas <- unlist(odbas)
sharkData <- NULL
for(i in 1:nrow(data)){
  fineInd <- data.frame(ID=data$ID[i],
                        level="2",
                        steps=NA,
                        stepCat=NA,
                        odbas=odbas[(i-1)*50+1:50])
  tmp <- rbind(data[i,,drop=FALSE],
               data.frame(ID=data$ID[i],
                          level="2i",
                          steps=NA,
                          stepCat=NA,
                          odbas=NA),
               fineInd)
  sharkData <- rbind(sharkData,tmp)
}

# prepare hierarchical data
sharkData <- prepData(sharkData,coordNames=NULL,
                      hierLevels=c("1","2i","2"))
head(sharkData)
```

```
##     ID level       steps stepCat      odbas
## 1 seg1     1 0.001447295       4         NA
## 2 seg1    2i          NA      NA         NA
## 3 seg1     2          NA      NA 0.07733020
## 4 seg1     2          NA      NA 0.07424251
## 5 seg1     2          NA      NA 0.07866384
## 6 seg1     2          NA      NA 0.07899367
```

Note that because the 194 segments were observed irregularly in bouts of time over the coarse of one night, Adam *et al.* (2019) essentially treated each segment as a different track whereby the HHMM "resets" at the beginning of each segment; this can be accomplished in `momentuHMM` by simply assigning each segment its own `ID` as was done above.

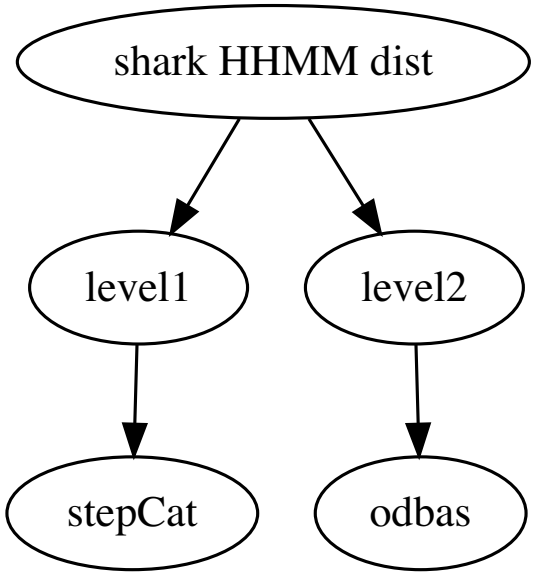
Next we define the hierarchical nature of the states and data streams:

```
### define hierarchical HMM
# states 1-3 = coarse state 1 (high activity)
# states 4-6 = coarse state 2 (resting)
# states 7-9 = coarse state 3 (travelling)
hierStates <- data.tree::Node$new("shark HHMM states")
hierStates$AddChild("activity") # zero distance travelled, high activity
hierStates$activity$AddChild("a1", state=1)
hierStates$activity$AddChild("a2", state=2)
hierStates$activity$AddChild("a3", state=3)
hierStates$AddChild("resting")  # zero distance travelled, low activity
hierStates$resting$AddChild("r1", state=4)
hierStates$resting$AddChild("r2", state=5)
hierStates$resting$AddChild("r3", state=6)
hierStates$AddChild("transit")  # travelling
hierStates$transit$AddChild("t1", state=7)
hierStates$transit$AddChild("t2", state=8)
hierStates$transit$AddChild("t3", state=9)

plot(hierStates)
```




```
nbStates <- length(hierStates$Get("state",filterFun=data.tree::isLeaf))
nCat <- 8 # number of stepCat categories
```

```r
# data stream distributions
# level 1 = coarse level (stepCat="cat8")
# level 2 = fine level (odbas="gamma")
hierDist <- data.tree::Node$new("shark HHMM dist")
hierDist$AddChild("level1")
hierDist$level1$AddChild("stepCat", dist=paste0("cat",nCat))
hierDist$AddChild("level2")
hierDist$level2$AddChild("odbas", dist="gamma")

plot(hierDist)
```



This is the first example in the vignette that uses a categorical data stream probability distribution, so it is perhaps worth describing this in a little more detail. When specifying categorical distributions, the number of categories must be indicated. In this case, there are 8 `stepCat` categories, so we specify this as `cat8`. Generally, categorical distributions are specified as `paste0("cat",nCat)`, where `nCat` is an integer greater than 2 (note that a categorical distribution with only 2 categories is simply a Bernoulli distribution). The categorical distribution parameters are `nCat` probabilities that sum to 1, so the mlogit link is used and only `nCat` -1 working parameters are estimated in order to obtain the `nCat` categorical probabilities on the real scale.

Both the "activity" and "resting" coarse-scale states were assumed to have zero distance travelled (i.e., `stepCat` $= 1$), while the "transit" state was assumed to have

> 0 distance travelled (i.e., `stepCat` $\in 2, \ldots, 8$). We therefore need to constrain the categorical step length probabilities for each fine-scale state within each coarse-scale state accordingly:

```
### starting values based on Adam et al
mu0 <- c(0.191, 0.323, 0.721, 0.084, 0.15, 0.228, 0.094, 0.191, 0.39)
sd0 <- c(0.047, 0.051, 0.248, 0.021, 0.025, 0.033, 0.026, 0.039, 0.159)
probs0 <- c(1e+10, -1e+10, 0.958, 5.064, 3.261, 4.021, 0.473, 2.568)

# constrain coarse-scale parameters for fine-scale states
DM <- list(stepCat=matrix(cbind(c(rep(c(1,1,1),2),
                                  rep(0,(nCat-1)*nbStates-6)),
                                kronecker(diag(nCat-1),
                                          c(0,0,0,0,0,0,1,1,1))),
                          nrow=(nCat-1)*nbStates,
                          ncol=8,
                          dimnames=list(paste0(rep(paste0("prob",
                                                          1:(nCat-1),
                                                          "_"),
                                                   each=nbStates),
                                               1:nbStates),
                                        c(paste0(c("prob1_12",
                                                   paste0("prob",
                                                          1:(nCat-1),
                                                          "_3")),
                                                 ":(Intercept)")))))
head(DM$stepCat,nbStates)

##         prob1_12:(Intercept) prob1_3:(Intercept) prob2_3:(Intercept)
## prob1_1                    1                   0                   0
## prob1_2                    1                   0                   0
## prob1_3                    1                   0                   0
## prob1_4                    1                   0                   0
## prob1_5                    1                   0                   0
## prob1_6                    1                   0                   0
## prob1_7                    0                   1                   0
## prob1_8                    0                   1                   0
## prob1_9                    0                   1                   0
##         prob3_3:(Intercept) prob4_3:(Intercept) prob5_3:(Intercept)
## prob1_1                   0                   0                   0
## prob1_2                   0                   0                   0
## prob1_3                   0                   0                   0
```

```
## prob1_4                    0                    0                    0
## prob1_5                    0                    0                    0
## prob1_6                    0                    0                    0
## prob1_7                    0                    0                    0
## prob1_8                    0                    0                    0
## prob1_9                    0                    0                    0
##         prob6_3:(Intercept) prob7_3:(Intercept)
## prob1_1                    0                    0
## prob1_2                    0                    0
## prob1_3                    0                    0
## prob1_4                    0                    0
## prob1_5                    0                    0
## prob1_6                    0                    0
## prob1_7                    0                    0
## prob1_8                    0                    0
## prob1_9                    0                    0

Par0 <- list(stepCat = probs0,
             odbas = c(mu0,sd0))

fixPar <- list(stepCat=c(1.e+10,-1.e+10,rep(NA,6)))
```

By fixing the working parameter corresponding to the probability of observing step length category 1 (`prob1`) for coarse-scale state 1 (fine-scale states $1-3$) and coarse-scale state 2 (fine-scale states $4-6$) to a very large positive number, we have effectively fixed `prob1` $= 1$ for these states. Likewise, by fixing the working parameter corresponding to `prob1` for coarse-scale state 3 (fine-scale states $7-9$) to a very large negative number, we have effectively fixed `prob1` $= 0$ for this state.

All that remains is (optionally) specifying starting values for the initial distribution (`hierDelta`) and t.p.m. (`hierBeta`) parameters, and let's check our model specification before fitting using `checkPar0`:

```
### t.p.m. starting values based on Adam et al
hierBeta <- data.tree::Node$new("shark beta")
hierBeta$AddChild("level1",
   beta=matrix(c(-0.651, -0.169, -1.884, -0.369, -1.596, -0.737),
               ncol=length(hierStates$children)
                    *(length(hierStates$children)-1)))
hierBeta$AddChild("level2")
hierBeta$level2$AddChild("activity",
```

```
   beta=matrix(c(-2.902, -14.032, 3.059, -1.37, 8.098, 11.66),
               ncol=length(hierStates$activity$children)
                    *(length(hierStates$activity$children)-1)))
hierBeta$level2$AddChild("resting",
   beta=matrix(c(-3.264, -14.279, 3.107, 0.252, 13.861, 16.468),
               ncol=length(hierStates$resting$children)
                    *(length(hierStates$resting$children)-1)))
hierBeta$level2$AddChild("transit",
   beta=matrix(c(-3.21, -21.32, 3.463, -0.598, 14.636, 17.811),
               ncol=length(hierStates$transit$children)
                    *(length(hierStates$transit$children)-1)))

### initial distribution starting values based on Adam et al
hierDelta <- data.tree::Node$new("shark delta")
hierDelta$AddChild("level1",
   delta=matrix(c(0.582, 2.894),
                ncol=length(hierStates$children)-1))
hierDelta$AddChild("level2")
hierDelta$level2$AddChild("activity",
   delta=matrix(c(-0.001, -1.1),
                ncol=length(hierStates$activity$children)-1))
hierDelta$level2$AddChild("resting",
   delta=matrix(c(-0.103, -0.105),
                ncol=length(hierStates$resting$children)-1))
hierDelta$level2$AddChild("transit",
   delta=matrix(c(0.24, -0.777),
                ncol=length(hierStates$transit$children)-1))

# check hierarchical model specification and parameters
checkPar0(sharkData,
          hierStates=hierStates,
          hierDist=hierDist,
          Par0=Par0,
          DM=DM,
          hierBeta=hierBeta,
          hierDelta=hierDelta,
          fixPar=fixPar)

##
## Regression coeffs for stepCat parameters:
## ----------------------------------------
##      prob1_12:(Intercept) prob1_3:(Intercept) prob2_3:(Intercept)
## [1,]                 1e+10               -1e+10                0.958
```

```
##      prob3_3:(Intercept) prob4_3:(Intercept) prob5_3:(Intercept)
## [1,]               5.064               3.261               4.021
##      prob6_3:(Intercept) prob7_3:(Intercept)
## [1,]               0.473               2.568
##
## odbas parameters:
## -----------------
##         a1    a2    a3    r1    r2    r3    t1    t2    t3
## mean 0.191 0.323 0.721 0.084 0.150 0.228 0.094 0.191 0.390
## sd   0.047 0.051 0.248 0.021 0.025 0.033 0.026 0.039 0.159
##
##
## ------------------------------------------------------------
## Regression coeffs for the transition probabilities (beta):
## ------------------------------------------------------------
## ------------------------  level1  ------------------------
##                         1 -> 4 1 -> 7 4 -> 1 4 -> 7 7 -> 1 7 -> 4
## I((level == "1") * 1) -0.651 -0.169 -1.884 -0.369 -1.596 -0.737
##
## ------------------------  level2  ------------------------
##                         1 -> 2  1 -> 3 2 -> 2 2 -> 3 3 -> 2 3 -> 3
## I((level == "2") * 1) -2.902 -14.032  3.059  -1.37  8.098  11.66
##
##                         4 -> 5  4 -> 6 5 -> 5 5 -> 6 6 -> 5 6 -> 6
## I((level == "2") * 1) -3.264 -14.279  3.107  0.252 13.861 16.468
##
##                         7 -> 8 7 -> 9 8 -> 8 8 -> 9 9 -> 8 9 -> 9
## I((level == "2") * 1)  -3.21 -21.32  3.463 -0.598 14.636 17.811
##
## ------------------------------------------------------------
##
## ------------------------------------------------------------
## Regression coeffs for the initial distribution (delta):
## ------------------------------------------------------------
## ------------------------  level1  ------------------------
##             state 4 state 7
## (Intercept)   0.582   2.894
##
## ------------------------  level2  ------------------------
##                          state 2 state 3
## I((level == "2i") * 1)  -0.001    -1.1
##
```

```
##                          state 5 state 6
## I((level == "2i") * 1)  -0.103  -0.105
##
##                          state 8 state 9
## I((level == "2i") * 1)    0.24  -0.777
##
## ------------------------------------------------------------
```

Everything checks out, so let's now fit the horn shark HHMM:

```
hhmm <- fitHMM(sharkData,
               hierStates=hierStates,
               hierDist=hierDist,
               Par0=Par0,
               hierBeta=hierBeta,
               hierDelta=hierDelta,
               DM=DM,
               fixPar=fixPar)

# transition probabilities for level1 and level2
trProbs12 <- getTrProbs(hhmm, covIndex=c(1,3))

# stationary distributions for level1 and level2
stats12 <- stationary(hhmm, covIndex=c(1,3))
```

The resulting estimates are again very similar to Adam *et al.* (2019), with the slight differences attributable to Adam *et al.* (2019) assuming the initial distributions for each level of the hierarchy are equal to the stationary distributions. Nevertheless, we can see that the estimated t.p.m. and stationary distributions are very similar:

```
### coarse scale ######################################
# t.p.m.
round(trProbs12$level1$gamma[,,1],3)

##          activity resting transit
## activity    0.423   0.221   0.357
## resting     0.082   0.543   0.375
## transit     0.121   0.285   0.594

# stationary distribution
round(stats12[[1]]$level1[1,],3)
```

```
## activity  resting  transit
##    0.153    0.371    0.477

#########################################################

### fine scale ########################################
# t.p.m.
lapply(trProbs12$level2,function(x) round(x$gamma[,,1],3))

## $activity
##       a1    a2    a3
## a1 0.948 0.052 0.000
## a2 0.044 0.944 0.011
## a3 0.000 0.028 0.972
##
## $resting
##       r1    r2    r3
## r1 0.963 0.037 0.000
## r2 0.041 0.907 0.052
## r3 0.000 0.069 0.931
##
## $transit
##       t1    t2    t3
## t1 0.961 0.039 0.000
## t2 0.030 0.954 0.016
## t3 0.000 0.040 0.960

# stationary distribution
lapply(stats12[[1]]$level2,function(x) round(x[1,],3))

## $activity
##    a1    a2    a3
## 0.377 0.443 0.181
##
## $resting
##    r1    r2    r3
## 0.385 0.349 0.265
##
## $transit
##    t1    t2    t3
## 0.353 0.459 0.188

#########################################################
```

3.11 African buffalo recharge dynamics

Here we demonstrate how to fit a discrete-time version of the African buffalo recharge dynamics model from Hooten *et al.* (2019) based only on surface water covariates. It is believed that water resources can strongly influence African buffalo space use, and surface water was therefore included in both the movement model (as distance to nearest surface water d) and the recharge function (as an indicator for being < 0.5 km to nearest surface water w). The model includes $N = 2$ states, where state 1 is the "recharged" state and state 2 is the "discharged" state. Conditional on the state $S_t \in \{1, 2\}$, the discrete-time analogue to the continuous-time model of Hooten *et al.* (2019) has the following bivariate normal random walk movement model for the locations ($\boldsymbol{\mu} = (\mu_x, \mu_y)$) at time t:

$$\boldsymbol{\mu}_t \mid S_t = s \sim \mathcal{N}\left(\boldsymbol{\mu}_{t-1} + D(\boldsymbol{\mu}_{t-1})\boldsymbol{\beta}^{\mu} I(s = 2), \sigma_s^2 \mathbf{I}\right),$$

where $I()$ is the indicator function, $\mathbf{I}$ is a 2×2 identity matrix, and $D(\boldsymbol{\mu}_t)$ is the negative gradient of d evaluated at location $\boldsymbol{\mu}_t$. Thus when the animal is in the charged state (i.e. $S_t = 1$), the movement model is a simple random walk. When the animal is in the discharged state (i.e. $S_t = 2$), the movement model includes a potential function surface based on distance to nearest surface water (for more on potential functions see Brillinger *et al.* 2012; Hooten *et al.* 2017, 2019, and sections 3.3 and 3.12). In terms of $\Gamma^{(t)}$, the model for the state-switching dynamics is simply:

$$\boldsymbol{\Gamma}^{(t)} = \begin{bmatrix} \frac{1}{(1+\exp(-g_t))} & \frac{\exp(-g_t)}{(1+\exp(-g_t))} \\ \frac{1}{(1+\exp(-g_t))} & \frac{\exp(-g_t)}{(1+\exp(-g_t))} \end{bmatrix}, \tag{11}$$

with recharge function

$$g_t = g_0 + \sum_{j=1}^{t} \theta_0 + w_j \theta_1,$$

where w_j is the distance to nearest water indicator covariate at location $\boldsymbol{\mu}_j$. Thus the probability of being in the "discharged" state decreases as the recharge function (g_t) increases. Note that there are no t.p.m. working parameter coefficients in Eq. 11 and, because $\gamma_{11} = \gamma_{21}$ and $\gamma_{12} = \gamma_{22}$, the state switching in this model is not Markov (i.e., the state at time t does not depend on the state at time $t - 1$).

In order to fit this model, we must first load the data and format the covariate

rasters:

```
library(raster)

## download buffalo data
load(url(paste0("https://github.com/henryrscharf/",
         "Hooten_et_al_EL_2018/raw/master/",
         "data/buffalo/buffalo_Cilla.RData")))

## download distance to water covariate raster
load(url(paste0("https://github.com/henryrscharf/",
                "Hooten_et_al_EL_2018/raw/master/",
                "data/buffalo/dist2sabie.RData")))
names(dist2sabie) <- "dist2sabie"

## standardize dist2sabie based on slope of gradient
dist2sabie_scaled <- dist2sabie / mean(values(terrain(dist2sabie,
                                                      opt = "slope")),
                                       na.rm = T)

# calculate gradient
D_scaled <- ctmcmove::rast.grad(dist2sabie_scaled)

## W (recharge function covariates)
# near_sabie = indicator for <500m from water
intercept <- raster(dist2sabie)
values(intercept) <- 1
W <- stack(list("intercept" = intercept,
                "near_sabie" = dist2sabie < 0.5e3))
W_names <- names(W)

## orthogonalize W based on locations ----
W_ortho <- W
W_path <- extract(x = W, y = matrix(buffalo_proj@coords, ncol = 2))
obstimes <- as.numeric(buffalo_proj$POSIX) / 3600 # numeric hours
W_tilde <- apply(W_path * c(0, diff(obstimes)), 2, cumsum)
W_tilde_svd <- svd(W_tilde)
W_tilde_proj_mat <- W_tilde_svd$v %*% diag(W_tilde_svd$d^(-1))
W_mat <- as.matrix(W)
W_mat_proj <- W_mat %*% W_tilde_proj_mat
for(layer in 1:ncol(W_mat)){
  values(W_ortho[[layer]]) <- W_mat_proj[, layer]
```

```
  names(W_ortho[[layer]]) <- paste0("svd", layer)
}
```

Note that to (presumably) help with numerical stability, Hooten *et al.* (2019) orthogonalized the recharge function covariates as above; the resulting recharge function is now $g_t = g_0 + \sum_{j=1}^{t} w^*_{1,j}\theta_1 + w^*_{2,j}\theta_2$, where $w^*_{1,j}$ and $w^*_{2,j}$ are the transformed intercept and distance indicator covariates, respectively.

The buffalo track data were collected from a GPS collar, but the roughly hourly observations were not perfectly regular.We will therefore use `crawlWrap` to predict the track at regular 15 min intervals (the average interval used by Hooten *et al.* 2019) and assume a conservative 50 m isotropic error ellipse for the measurement error model.

```
lnError <- crawl::argosDiag2Cov(50,50,0) # 50m isotropic error ellipse
buffaloData <- data.frame(ID = 1,
                          time = obstimes,
                          x = buffalo_proj@coords[, 1],
                          y = buffalo_proj@coords[, 2],
                          ln.sd.x = lnError$ln.sd.x,
                          ln.sd.y = lnError$ln.sd.y,
                          error.corr = lnError$error.corr)

crwOut <- crawlWrap(buffaloData,
                    theta = c(6.5,-.1),
                    fixPar = c(1,1,NA,NA),
                    err.model = list(x = ~ln.sd.x-1,
                                     y = ~ln.sd.y-1,
                                     rho = ~error.corr),
                    timeStep = 0.25, # predict at 15 min time steps
                    attempts = 10)
```

Now we're ready to specify the recharge model. We'll first fit the model to the best predicted track from `crawlWrap` and then use this model fit to specify starting values for a multiple imputation analysis:

```
spatialCovs <- list(W_intercept = W_ortho$svd1,
                    W_near_sabie = W_ortho$svd2,
                    dist2sabie = dist2sabie,
                    D.x = D_scaled$rast.grad.x,
```

```
                    D.y = D_scaled$rast.grad.y)

# best predicted track data
hmmData <- prepData(crwOut,
                    spatialCovs = spatialCovs,
                    altCoordNames = "mu")
head(hmmData[,c("ID","time", "mu.x", "mu.y",
                "W_intercept","W_near_sabie","dist2sabie",
                "D.x","D.y")])

##   ID     time     mu.x     mu.y   W_intercept W_near_sabie dist2sabie
## 1  1 313344.4 384560.8 -2770752 -0.0001873081 -0.003616337   1323.424
## 2  1 313344.7 384560.2 -2770750 -0.0001873081 -0.003616337   1323.424
## 3  1 313344.9 384559.7 -2770749 -0.0001873081 -0.003616337   1323.424
## 4  1 313345.2 384559.3 -2770747 -0.0001873081 -0.003616337   1323.424
## 5  1 313345.4 384559.1 -2770744 -0.0001873081 -0.003616337   1323.424
## 6  1 313345.7 384559.1 -2770742 -0.0001873081 -0.003616337   1323.424
##          D.x       D.y
## 1 -0.8709203 0.2576337
## 2 -0.8709203 0.2576337
## 3 -0.8709203 0.2576337
## 4 -0.8709203 0.2576337
## 5 -0.8709203 0.2576337
## 6 -0.8709203 0.2576337

nbStates <- 2
stateNames <- c("charged", "discharged")
dist <- list(mu = "rw_mvnorm2") # bivariate normal random walk

# pseudo-design matrix for mu
DM <- list(mu=matrix(c("mu.x_tm1",         0,    0,0,0,0,
                       "mu.x_tm1",         0,"D.x",0,0,0,
                                  0,"mu.y_tm1",    0,0,0,0,
                                  0,"mu.y_tm1","D.y",0,0,0,
                                  0,         0,    0,1,0,0,
                                  0,         0,    0,0,1,0,
                                  0,         0,    0,0,0,1,
                                  0,         0,    0,0,0,1,
                                  0,         0,    0,1,0,0,
                                  0,         0,    0,0,1,0),
                     5*nbStates,
                     6,byrow=TRUE,
                     dimnames=list(c(paste0("mean.",
```

```
                                        rep(c("x_","y_"),
                                            each=nbStates),
                                        1:nbStates),
                                  paste0("sigma.",
                                        rep(c("x_","xy_","y_"),
                                            each=nbStates),
                                        1:nbStates)),
                                c("x:x_tm1",
                                  "y:y_tm1",
                                  "xy:D",
                                  "sigma_1:(Intercept)",
                                  "sigma_2:(Intercept)",
                                  "sigma_12:(Intercept)"))))

# starting values
Par0=list(mu=c(1, 1, 0, log(85872.66), log(37753.53), 0))
g0 <- 0 # recharge function at time 0
theta <- c(0,0,0) # recharge function parameters

## specify recharge formula
# note that theta formula requires an 'intercept' term
formula <- ~ recharge(g0 = ~1,
                      theta = ~W_intercept+W_near_sabie)

## remove Markov property
betaRef <- c(1,1) # make state 1 the reference state
betaCons <- matrix(c(1,2),2,2) # 1 -> 1 = 2 -> 1 and 1 -> 2 = 2 -> 2

## set fixed parameters
fixPar <- list(mu = c(Par0$mu[1:2],NA,NA,NA,Par0$mu[6]),
               beta = matrix(c(0,-1,0,-1),2,2),
               delta = c(0.5,0.5),
               theta = c(0,NA,NA)) # fix extra 'intercept' term to zero

# check recharge model specification
checkPar0(hmmData, nbStates = nbStates, dist = dist,
          formula = formula, Par0 = Par0,
          beta0 = list(beta = fixPar$beta,
                       g0 = g0,
                       theta = theta),
          delta0 = fixPar$delta, fixPar = fixPar,
          DM = DM, betaRef = betaRef, betaCons = betaCons,
```

```
          stateNames = stateNames)

##
## Regression coeffs for mu parameters:
## ------------------------------------
##      x:x_tm1 y:y_tm1 xy:D sigma_1:(Intercept) sigma_2:(Intercept)
## [1,]       1       1    0            11.36062            10.53883
##      sigma_12:(Intercept)
## [1,]                    0
##
## Regression coeffs for the transition probabilities (beta):
## ----------------------------------------------------------
##             1 -> 2 2 -> 2
## (Intercept)      0      0
## recharge        -1     -1
##
## Initial distribution:
## ---------------------
##    charged discharged
##        0.5        0.5
##
## Initial recharge parameter (g0):
## --------------------------------
## (Intercept)
##           0
##
## Recharge function parameters (theta):
## -------------------------------------
##  (Intercept)  W_intercept W_near_sabie
##            0            0            0


# fit to best predicted path
buffaloFit <- fitHMM(hmmData, nbStates = nbStates, dist = dist,
                     formula = formula, Par0 = Par0,
                     beta0 = list(g0=g0,
                                  theta=theta),
                     fixPar = fixPar,
                     DM = DM, betaRef = betaRef, betaCons = betaCons,
                     stateNames = stateNames,
                     mvnCoords = "mu",
                     optMethod = "Nelder-Mead",
                     control = list(maxit=1000))
```

```
## ============================================================================
## Fitting HMM with 2 states and 1 data stream
## ----------------------------------------------------------------------------
##  mu ~ rw_mvnorm2(mean.x:  custom, mean.y:  custom, sigma.x:  custom,
sigma.xy:  custom, sigma.y:  custom)
##
##  Transition probability matrix formula:  ~recharge(g0 = ~1, theta = ~W_intercept
+ W_near_sabie)
##
##  Initial distribution formula:  ~1
## ============================================================================
## DONE

# extract starting values
bestPar <- getPar(buffaloFit)
```

There are several things worth noting in the code above. When specifying normal random walk models using a pseudo-design matrix, we must include terms for the previous location in `mean.x` and `mean.y` (in this case "mu.x_tm1" and "mu.y_tm1", respectively), and we'll typically fix the corresponding coeffcients to 1 using `fixPar`. As in Hooten *et al.* (2019), we assume that the coefficients for the negative gradient (D) are equal in the x- and y-directions (i.e., $\beta_x^\mu = \beta_y^\mu$; this constraint is specified in the third column of `DM` above) and that the state-dependent `sigma.x` and `sigma.y` are equal. Similar to random effects models (section 3.9), note that for recharge models `beta0` must now be specified as a list (consisting of objects named `beta`, `g0`, and/or `theta`). Hooten *et al.* (2019) assumed state transitions were non-Markov (Eq. 11), and we can accomplish this by setting the reference states for the t.p.m. working scale parameters to state 1 (i.e., `betaRef <- c(1,1)`) and setting the columns within each row to be equal (i.e., `betaCons <- matrix(c(1,2),2,2)`). With `formula <- ~ recharge(g0 = ~1, theta = ~W_intercept+W_near_sabie)`, the resulting state transition probability matrix at time t is:

$$\mathbf{\Gamma}^{(t)} = \begin{bmatrix} \frac{1}{(1+\exp(\beta_0+g_t\beta_1))} & \frac{\exp(\beta_0+g_t\beta_1)}{(1+\exp(\beta_0+g_t\beta_1))} \\ \frac{1}{(1+\exp(\beta_0+g_t\beta_1))} & \frac{\exp(\beta_0+g_t\beta_1)}{(1+\exp(\beta_0+g_t\beta_1))} \end{bmatrix},$$

where

$$g_t = g_0 + \sum_{j=1}^{t} \theta_0 + \text{W\_intercept}_j \theta_1 + \text{W\_near\_sabie}_j \theta_2.$$

Using `fixPar`, we fixed $\beta_0 = 0$ and $\beta_1 = -1$. Thus we have removed the Markov property from the state-switching dynamics and ensured that the probability of being in the "discharged" state (state 2) decreases as the recharge function (g_t) increases as in Eq. 11. We also fix $\theta_0 = 0$ because we do not need the required intercept term in this particular case (an orthogonalized intercept term "W_intercept" is already included as a covariate). Because we have removed the Markov property from the state-switching dynamics, the initial distribution has no effect on the likelihood; we therefore fixed it (arbitrarily) to `delta=c(0.5,0.5)`.

Now that we have our starting values ("bestPar"), let's fit 28 imputations of the position process using `MIfitHMM`:

```
buffaloFits <- MIfitHMM(crwOut, nSims=28,
                        spatialCovs = spatialCovs,
                        mvnCoords="mu", altCoordNames = "mu",
                        nbStates=nbStates, dist=dist, formula=formula,
                        Par0=bestPar$Par, beta0=bestPar$beta,
                        fixPar=fixPar, DM=DM,
                        betaRef=betaRef, betaCons=betaCons,
                        stateNames = stateNames,
                        retryFits = 3, retrySD=list(mu=c(0,0,3,0,0,0),
                                                    g0=1,
                                                    theta=c(0,1,1)),
                        optMethod = "Nelder-Mead",
                        control = list(maxit=100000))

plot(buffaloFits,plotCI=TRUE,ask=FALSE)
plotSpatialCov(buffaloFits,dist2sabie)

# plot estimates and CIs for Pr(discharged) at each time step
trProbs <- getTrProbs(buffaloFits, getCI=TRUE)
plot(trProbs$est[1,2,],type="l", ylim=c(0,1),
     ylab="Pr(discharged)", xlab="t",
     col=c("#E69F00", "#56B4E9")[buffaloFits$miSum$Par$states])
arrows(1:dim(trProbs$est)[3],
       trProbs$lower[1,2,],
       1:dim(trProbs$est)[3],
```

```
        trProbs$upper[1,2,],
        length=0.025, angle=90, code=3,
        col=c("#E69F00", "#56B4E9")[buffaloFits$miSum$Par$states],
        lwd=1.3)
abline(h=0.5,lty=2)
```

As in Hooten *et al.* (2019), we found that the buffalo spent a majority of time steps in the discharged state (73%, 95% CI: 70 − 76%) and thus needed to recharge regularly near water resources. With estimated $g_0 = -0.27$ (95% CI: $-0.75 - 0.2$), $\theta_1 = 1.43$ (95% CI: $-0.61 - 3.48$), and $\theta_2 = 2.52$ (95% CI: $1.47 - 3.57$), the estimated recharge function and transition probabilities (Figure 18) look very similar to those reported by Hooten *et al.* (2019). However, Hooten *et al.* (2019) found some evidence that the buffalo orients toward surface water when in the discharged state, but our discrete-time formulation did not find evidence of such biased movement ($\beta^{\mu} = 0.88$, 95% CI: $-3.2 - 4.96$). This difference could be attributable to several factors, including our formulation being in discrete time (instead of continuous time), our use of a 2-stage multiple imputation approach based on the CTCRW (instead of a single-stage model), and the absence of prior distributions in our non-Bayesian model. Nevertheless, inferences about recharge and state-switching dynamics are essentially the same between our discrete-time formulation and the continuous-time model of Hooten *et al.* (2019).

3.12 Simulating constrained movement

In section 3.3 we briefly demonstrated how potential functions can be used within a bivariate normal random walk to model loggerhead turtle movements relative to ocean surface currents. Here we'll show how this approach can be used in `simData` to simulate movement data subject to barriers or other constraints (e.g. land for marine animals). To accomplish this, we'll rely on the `forest` raster that is automatically loaded with `momentuHMM`. We'll start by pretending that `forest` cells with values > 0 are "land" and all others are "water". Then we'll create a new raster named `boundary` containing the shortest distance from land to water using `raster::distance`:

```
boundary <- forest
boundary[boundary>0] <- NA
boundary <- raster::distance(boundary)
names(boundary) <- "boundary"
```

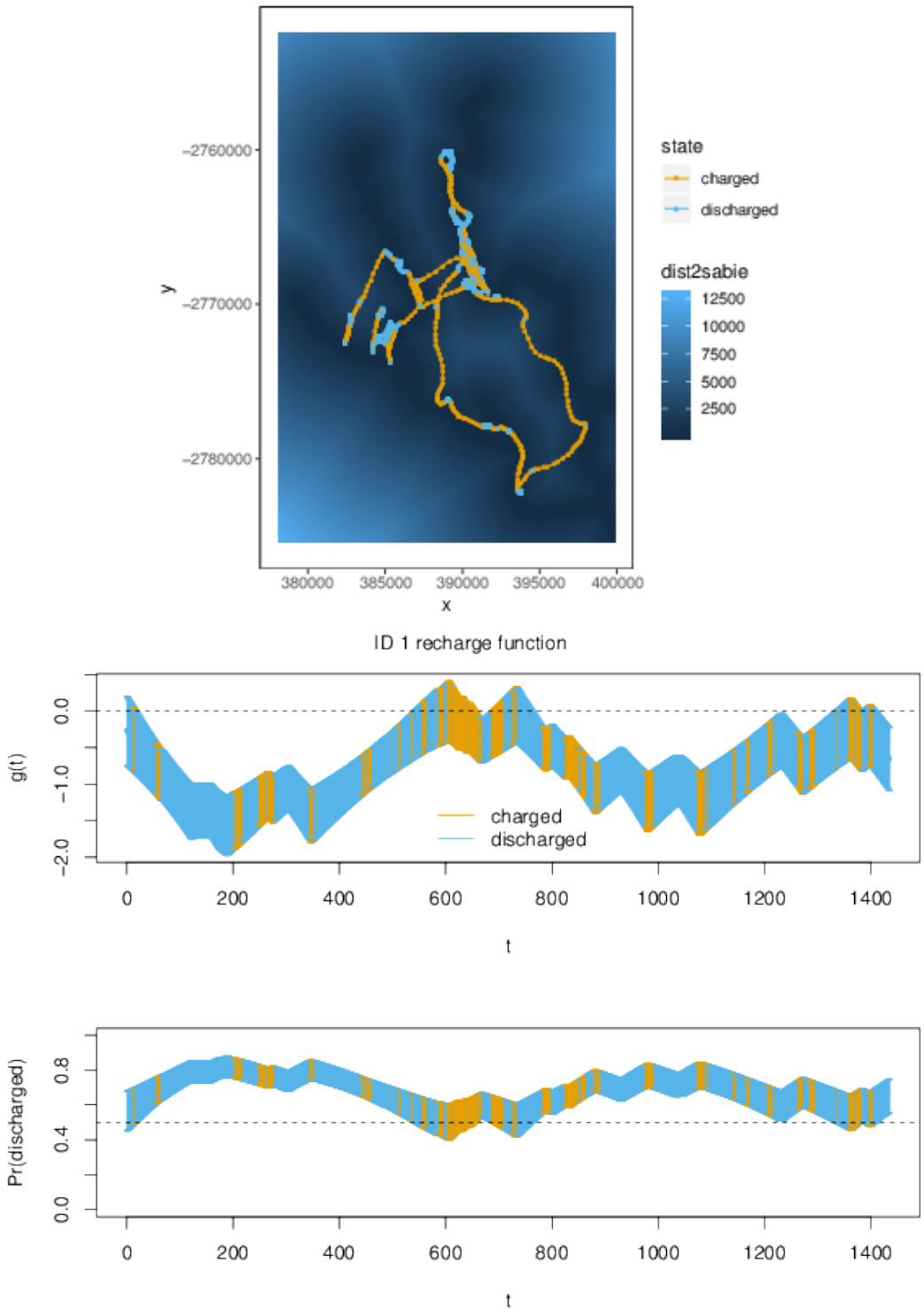


Figure 18. *African buffalo estimated states (top), recharge function (middle), and (non-Markov) transition probability to the discharged state (bottom) at each time step* (t).

Next we'll calculate the gradients of the potential function surface in the x- and y-directions using `ctmcmove::rast.grad`:

```
# boundary needs to have a CRS for ctmcmove::rast.grad
proj4string(boundary) <- sp::CRS("+init=epsg:4326")

# compute gradients in x- and y- directions
grad <- ctmcmove::rast.grad(boundary)
grad.x <- grad$rast.grad.x
grad.y <- grad$rast.grad.y
```

Now we're ready to simulate our bivariate normal random walk model including the gradients as covariates:

```
dist <- list(mu="rw_mvnorm2") # bivariate normal random walk
DM <- list(mu=list(mean.x=~mu.x_tm1+crw(mu.x_tm1,lag=1)+grad.x,
                   mean.y=~mu.y_tm1+crw(mu.y_tm1,lag=1)+grad.y,
                   sigma.x=~1,
                   sigma.xy=~1,
                   sigma.y=~1))

# specify parameters on working parameter scale
Par <- list(mu=c(1,0.75,-750000,1,0.75,-1500,log(100000),0,log(100000)))
names(Par$mu) <- c("mu.x_tm1","crw(mu.x_tm1,lag=1)","grad.x",
                   "mu.y_tm1","crw(mu.y_tm1,lag=1)","grad.y",
                   "sigma.x","sigma.xy","sigma.y")

# simulate and plot
simBound <- simData(nbStates=1, obsPerAnimal = 10000, dist=dist, Par=Par,
                    DM=DM, spatialCovs=list(grad.x = grad.x,
                                            grad.y = grad.y),
                    mvnCoords="mu", initialPosition=c(25000,75000))
plot(simBound,dataNames=c("mu.x","mu.y"),ask=FALSE)
plot(boundary$boundary)
points(simBound$mu.x,simBound$mu.y,type="l")
```

The model specified in `DM` is identical to Eq. 10, except we have replaced the ocean surface current velocities with the gradients for shortest distance to water in the x- and y-directions (`grad.x` and `grad.y`, respectively). As we can see in Figure 19, the track is repelled from "land" and tends to stay in the "water". This strong attraction to

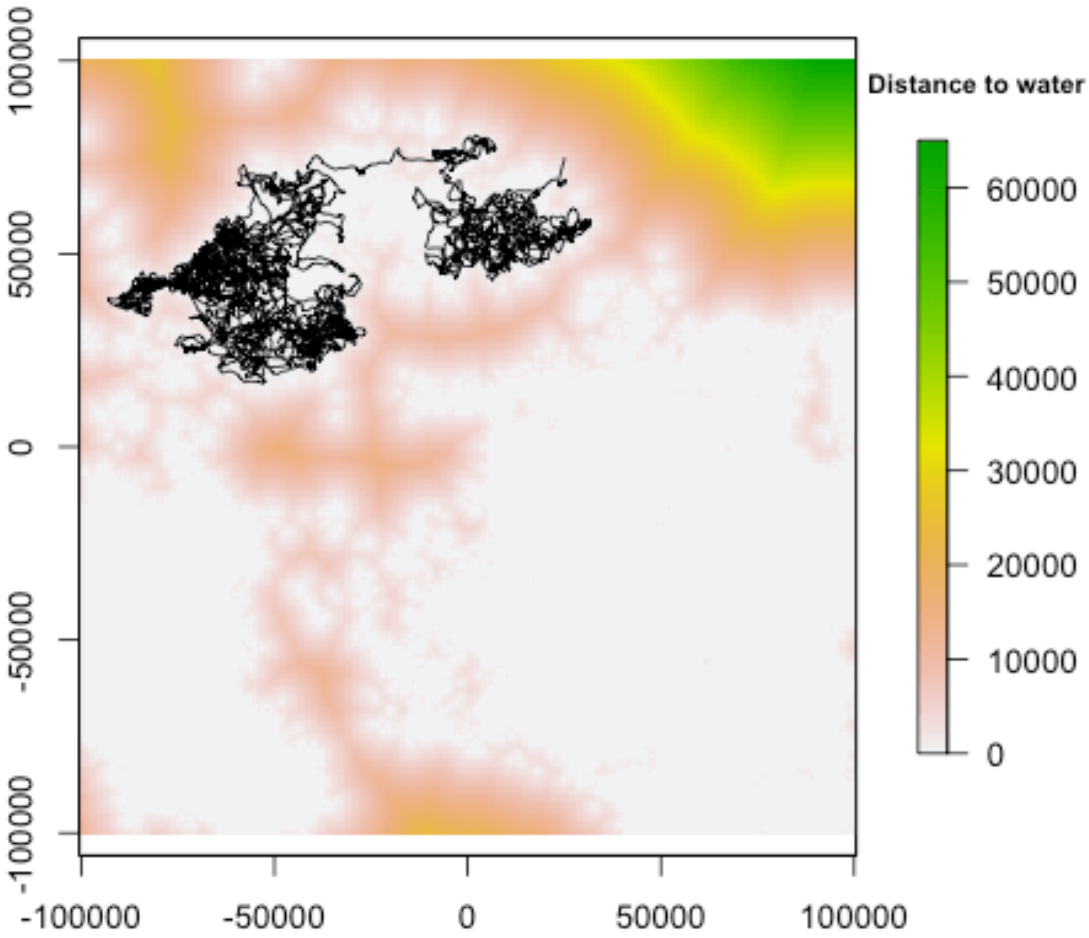


Figure 19. *Simulated track from a bivariate normal random walk model with movements repulsed from "land" using a potential function.*

water owes to the large negative values for the coefficients corresponding to `grad.x` and `grad.y`.

4 Discussion

Here we have introduced version 1.5.0 of the R package `momentuHMM` and demonstrated some of its capabilities for conducting multivariate HMM analyses with animal location, auxiliary biotelemetry, and environmental data. The package allows for fitting (and simulating from) a suite of biased and correlated random walk movement process models (e.g. McClintock *et al.* 2012), can be used for an unlimited number of data streams and latent behavior states, includes multiple imputation methods to account for measurement error, temporal irregularity, and other forms of missing data that would otherwise be prohibitive to maximum likelihood analysis, and integrates seamlessly with rasters to facilitate spatio-temporal covariate modelling. Because the package incorporates biased random walks, it can also be used to implement group dynamic models (e.g. Langrock *et al.* 2014). The package therefore greatly expands on available software and facilitates the incorporation of more ecological and behavioral realism for hypothesis-driven analyses of animal movement that account for many of the challenges commonly associated with telemetry data. While many of the features of `momentuHMM` were motivated by animal movement data, we note that the package is not limited to location data and can be used for analyzing any type of data that is amenable to (multivariate) HMMs.

Model fitting in `momentuHMM` is relatively fast because the forward algorithm (Eq. 1) is coded in C++. Because multiple imputations are completely parallelizable, with sufficient processing power computation times for analyses that account for measurement error, temporal irregularity, or other forms of missing data need not be longer than that required to fit a single HMM. However, computation times will necessarily be longer as the number of states and/or parameters increase. For example, `momentuHMM` required about 1 hr to fit a single HMM with $N = 6$ states, seven data streams, and $T = 7414$ time steps (McClintock 2017).

As in any maximum likelihood analysis based on numerical optimization, computation times will also depend on the starting values (`Par0` and `beta0`). Specifying "good" starting values is arguably the most challenging aspect of model fitting in

momentuHMM, particularly for the working scale coefficients when using covariates. The getPar, getPar0, getParDM, and checkPar0 functions are designed to help with the specification of starting values, and the retryFits argument in crawlWrap, fitHMM, and MIfitHMM will re-optimize based on random perturbations of the parameters to help explore the likelihood surface and diagnose convergence to local maxima. Optimization for the circular-linear regression link function (tan(mean/2); see Table 2) in particular can be prone to local minima, so users are encouraged to explore a range of starting values when fitting these models.

While momentuHMM includes functions for drawing realizations of the position process based on the CTCRW model of Johnson *et al.* (2008), this is but one of many methods for performing the first stage of multiple imputation. Realizations of the position process from any movement model that accounts for measurement error and/or temporal irregularity (e.g. Calabrese *et al.* 2016; Gurarie *et al.* 2017) could be passed to MIfitHMM for HMM-type analyses in the second stage. Multiple imputation methods also need not be limited to these telemetry error scenarios. For example, conventional missing data could also be imputed using standard techniques (Rubin & Schenker 1986), thereby allowing the investigation of non-random mechanisms for missingness that can be problematic if left unaccounted for in HMMs.

There remain many potential avenues for refining and extending the capabilities of momentuHMM. Computation times could likely be improved by further optimizing the R and C++ code for speed. Notable extensions include hidden semi-Markov models and random effects on data stream probability distribution parameters (Zucchini *et al.* 2016). We would also like to incorporate additional parameters for change-point thresholds and the locations of activity centers instead of requiring that they be pre-specified (and potentially compared using AIC or other model selection criteria) as in grey seal example. Lastly, it is relatively straightforward to add additional probability distributions, and we are pleased to do so upon request. Practitioners interested in additional features for momentuHMM are encouraged to contact the authors.

Acknowledgments

We are grateful to R. Scott, B. Godley, M. Godfrey, J. Sudre, and North Carolina Aquariums for providing the data used in our turtle example. We are also grateful to

the many authors who made their data publicly available for use in our examples (Wall *et al.* 2014; Pirotta *et al.* 2018; Isojunno *et al.* 2017; Leos-Barajas *et al.* 2017; Adam *et al.* 2019). The findings and conclusions in this vignette are those of the author(s) and do not necessarily represent the views of the National Marine Fisheries Service, NOAA. Any use of trade, product, or firm names does not imply an endorsement by the US Government.

References

Adam, T., Griffiths, C.A., Leos-Barajas, V., Meese, E.N., Lowe, C.G., Blackwell, P.G., Righton, D. & Langrock, R. (2019) Joint modelling of multi-scale animal movement data using hierarchical hidden markov models. *Methods in Ecology and Evolution*.

Beyer, H.L., Morales, J.M., Murray, D. & Fortin, M.J. (2013) The effectiveness of Bayesian state-space models for estimating behavioural states from movement paths. *Methods in Ecology and Evolution*, **4**, 433–441.

Brillinger, D.R., Preisler, H.K., Ager, A.A. & Kie, J. (2012) The use of potential functions in modelling animal movement. *Selected Works of David Brillinger*, pp. 385–409. Springer.

Calabrese, J.M., Fleming, C.H. & Gurarie, E. (2016) ctmm: an R package for analyzing animal relocation data as a continuous-time stochastic process. *Methods in Ecology and Evolution*, **7**, 1124–1132.

Cornelissen, G. (2014) Cosinor-based rhythmometry. *Theoretical Biology and Medical Modelling*, **11**, 16.

Costa, D.P., Robinson, P.W., Arnould, J.P., Harrison, A.L., Simmons, S.E., Hassrick, J.L., Hoskins, A.J., Kirkman, S.P., Oosthuizen, H., Villegas-Amtmann, S. *et al.* (2010) Accuracy of argos locations of pinnipeds at-sea estimated using fastloc gps. *PloS one*, **5**, e8677.

DeRuiter, S.L., Langrock, R., Skirbutas, T., Goldbogen, J.A., Calambokidis, J., Friedlaender, A.S. & Southall, B.L. (2017) A multivariate mixed hidden Markov model to analyze blue whale diving behaviour during controlled sound exposures. *The Annals of Applied Statistics*, **11**, 362–392.

Gilbert, P. & Varadhan, R. (2016) *numDeriv: Accurate Numerical Derivatives*. R package version 2016.8-1.

Glur, C. (2018) *data.tree: General Purpose Hierarchical Data Structure*. R package version 0.7.8.

Gurarie, E., Fleming, C.H., Fagan, W.F., Laidre, K.L., Hernández-Pliego, J. & Ovaskainen, O. (2017) Correlated velocity models as a fundamental unit of animal movement: synthesis and applications. *Movement Ecology*, **5**, 13.

Hijmans, R.J. (2016a) *geosphere: Spherical Trigonometry*. R package version 1.5-5.

Hijmans, R.J. (2016b) *raster: Geographic Data Analysis and Modeling*. R package version 2.5-8.

Hooten, M.B., Johnson, D.S., McClintock, B.T. & Morales, J.M. (2017) *Animal Movement: Statistical Models for Telemetry Data*. CRC Press.

Hooten, M.B., Scharf, H.R. & Morales, J.M. (2019) Running on empty: recharge dynamics from animal movement data. *Ecology letters*, **22**, 377–389.

Isojunno, S., Sadykova, D., DeRuiter, S., CurÃ©, C., Visser, F., Thomas, L., Miller, P.J.O. & Harris, C.M. (2017) Individual, ecological, and anthropogenic influences on activity budgets of long-finned pilot whales. *Ecosphere*, **8**, e02044.

Johnson, D.S. (2017) *crawl: Fit Continuous-Time Correlated Random Walk Models to Animal Movement Data*. R package version 2.1.1.

Johnson, D.S., London, J.M., Lea, M.A. & Durban, J.W. (2008) Continuous-time correlated random walk model for animal telemetry data. *Ecology*, **89**, 1208–1215.

Jonsen, I.D., Flemming, J.M. & Myers, R.A. (2005) Robust state–space modeling of animal movement data. *Ecology*, **86**, 2874–2880.

Langrock, R., Hopcraft, G., Blackwell, P., Goodall, V., King, R., Niu, M., Patterson, T., Pedersen, M., Skarin, A. & Schick, R. (2014) Modelling group dynamic animal movement. *Methods in Ecology and Evolution*, **5**, 190–199.

Langrock, R., King, R., Matthiopoulos, J., Thomas, L., Fortin, D. & Morales, J.M. (2012) Flexible and practical modeling of animal telemetry data: hidden Markov models and extensions. *Ecology*, **93**, 2336–2342.

Leos-Barajas, V., Gangloff, E.J., Adam, T., Langrock, R., Van Beest, F.M., Nabe-Nielsen, J. & Morales, J.M. (2017) Multi-scale modeling of animal movement and general behavior data using hidden markov models with hierarchical structures. *Journal of Agricultural, Biological and Environmental Statistics*, **22**, 232–248.

McClintock, B.T. (2017) Incorporating telemetry error into hidden markov models of animal movement using multiple imputation. *Journal of Agricultural, Biological, and Environmental Statistics*, **22**, 249–269.

McClintock, B.T., Johnson, D.S., Hooten, M.B., Ver Hoef, J.M. & Morales, J.M. (2014) When to be discrete: the importance of time formulation in understanding animal movement. *Movement Ecology*, **2**, 21.

McClintock, B.T., King, R., Thomas, L., Matthiopoulos, J., McConnell, B.J. & Morales, J.M. (2012) A general discrete-time modeling framework for animal movement using multistate random walks. *Ecological Monographs*, **82**, 335–349.

McClintock, B.T., London, J.M., Cameron, M.F. & Boveng, P.L. (2017) Bridging the gaps in animal movement: hidden behaviors and ecological relationships revealed by integrated data streams. *Ecosphere*, **8**, e01751.

McClintock, B.T. & Michelot, T. (2018) momentuHMM: R package for generalized hidden Markov models of animal movement. *Methods in Ecology and Evolution*, **9**, 1518–1530.

McClintock, B.T., Russell, D.J., Matthiopoulos, J. & King, R. (2013) Combining individual animal movement and ancillary biotelemetry data to investigate population-level activity budgets. *Ecology*, **94**, 838–849.

McKellar, A.E., Langrock, R., Walters, J.R. & Kesler, D.C. (2014) Using mixed hidden Markov models to examine behavioral states in a cooperatively breeding bird. *Behavioral Ecology*, **26**, 148–157.

Michelot, T., Langrock, R., Bestley, S., Jonsen, I.D., Photopoulou, T. & Patterson, T.A. (2017) Estimation and simulation of foraging trips in land-based marine predators. *Ecology*, **98**, 1932–1944.

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

Morales, J.M., Haydon, D.T., Frair, J., Holsinger, K.E. & Fryxell, J.M. (2004) Extracting more out of relocation data: building movement models as mixtures of random walks. *Ecology*, **85**, 2436–2445.

Pirotta, E., Edwards, E.W.J., New, L. & Thompson, P.M. (2018) Central place foragers and moving stimuli: A hidden-state model to discriminate the processes affecting movement. *Journal of Animal Ecology*, **87**, 1116–1125.

R Core Team (2017) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rivest, L.P., Duchesne, T., Nicosia, A. & Fortin, D. (2016) A general angular regression model for the analysis of data on animal movement in ecology. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, **65**, 445–463.

Rubin, D.B. & Schenker, N. (1986) Multiple imputation for interval estimation from simple random samples with ignorable nonresponse. *Journal of the American Statistical Association*, **81**, 366–374.

Sarda-Espinosa, A. (2017) *dtwclust: Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance*. R package version 3.1.2.

Towner, A.V., Leos-Barajas, V., Langrock, R., Schick, R.S., Smale, M.J., Kaschke, T., Jewell, O.J.D. & Papastamatiou, Y.P. (2016) Sex-specific and individual preferences for hunting strategies in white sharks. *Functional Ecology*, **30**, 1397–1407.

Wall, J., Wittemyer, G., LeMay, V., Douglas-Hamilton, I. & Klinkenberg, B. (2014) Elliptical time-density model to estimate wildlife utilization distributions. *Methods in Ecology and Evolution*, **5**, 780–790.

Whoriskey, K., Auger-Méthé, M., Albertsen, C.M., Whoriskey, F.G., Binder, T.R., Krueger, C.C. & Mills Flemming, J. (2017) A hidden markov movement model for rapidly identifying behavioral states from animal tracks. *Ecology and Evolution*, **7**, 2112–2121.

Zucchini, W., MacDonald, I.L. & Langrock, R. (2016) *Hidden Markov Models for Time Series: An Introduction Using R*. CRC Press.