

Package: funHMM (via r-universe)

September 24, 2026

Type Package

Title Hidden Markov Models for Functional Data

Version 0.1.0

Description Fits hidden Markov models to time-ordered sequences of curves, such as sample paths of stochastic processes or smoothed functional observations, without projecting the curves onto a finite basis. The emission functions are Onsager-Machlup functionals of Gaussian measures on function spaces, which allows for Brownian motion with drift, fractional Brownian motion, Ornstein-Uhlenbeck processes and non-parametric state means under a choice of Cameron-Martin norm. The Baum-Welch and Viterbi algorithms are implemented in C. Methods are described in Kashlak, Loliencar and Heo (2023) <<https://jmlr.org/papers/v24/22-0685.html>>.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports stats, graphics, grDevices

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation yes

RoxygenNote 7.2.3

Config/testthat/edition 3

Author Adam B Kashlak [aut, cre]

Maintainer Adam B Kashlak <kashlak@ualberta.ca>

Repository <https://cran.r-universe.dev>

Date/Publication 2026-09-24 05:03:24 UTC

RemoteUrl <https://github.com/cran/funHMM>

RemoteRef HEAD

RemoteSha d0388a300ae8a9197bc177d2c03e94e51d2632a6

Contents

ari	2
plot.thmm	3
predict.thmm	4
rbm	5
rbridge	6
rmarkov	6
rou	7
rthmm	8
simulate.thmm	9
thmm	10
Index	14

ari	<i>Adjusted Rand index</i>
-----	----------------------------

Description

Measures the agreement between two labellings of the same objects, e.g. the decoded states of a `thmm()` fit and the true states of a simulation. A value of one indicates identical partitions (up to relabelling) and values near zero indicate agreement no better than chance.

Usage

`ari(x, y)`

Arguments

`x, y` Two vectors of labels of the same length.

Value

A single number.

References

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2, 193–218.

Examples

```
ari(c(1, 1, 2, 2, 3), c(2, 2, 1, 1, 3)) # 1: identical up to relabelling
ari(c(1, 1, 2, 2), c(1, 2, 1, 2))      # negative: worse than chance
```

plot.thmm

*Plot a fitted THMM***Description**

Plots the observed curves coloured by their decoded (Viterbi) state. For the non-parametric model the fitted mean curves are overlaid; for Brownian motion with drift the fitted linear drifts are overlaid.

Usage

```
## S3 method for class 'thmm'
plot(
  x,
  which = seq_len(dim(x$data)[3L]),
  col = NULL,
  lwd.mean = 3,
  legend = TRUE,
  ...
)
```

Arguments

x	A fitted "thmm" object.
which	For multi-dimensional curves, which coordinate(s) to plot.
col	Colours for the states (a vector of length nstates).
lwd.mean	Line width for the overlaid state means (0 suppresses them).
legend	Logical; add a legend?
...	Further arguments passed to graphics::matplot() .

Value

x, invisibly.

Examples

```
set.seed(3)
tt <- seq(0, 1, length.out = 40)
sim <- rthmm(60, init.prob = c(1, 0), trans = matrix(c(.8, .2, .2, .8), 2),
  type = "nonpar", par = rbind(tt, 1 - tt), sigma = 0.3)
fit <- thmm(sim$data, 2, type = "nonpar", norm = "L2", start = "kmeans")
plot(fit)
```

predict.thmm

*Decode new curves with a fitted THMM***Description**

Runs the forward-backward and Viterbi algorithms on a new sequence of curves using the parameters of a fitted model.

Usage

```
## S3 method for class 'thmm'
predict(object, newdata = NULL, type = c("states", "posterior", "loglik"), ...)
```

Arguments

object	A fitted "thmm" object.
newdata	Curves in the same format as the data used for fitting (same grid length and dimension). Defaults to the training data.
type	What to return: the Viterbi "states", the "posterior" state probabilities, or the "loglik" of the new sequence.
...	Unused.

Value

An integer vector of states, a matrix of posterior probabilities, or a single number.

Examples

```
set.seed(2)
A <- matrix(c(.9, .1, .1, .9), 2)
sim <- rthmm(80, init.prob = c(1, 0), trans = A, type = "bmwd",
            par = c(-3, 3), len = 50)
fit <- thmm(sim$data, 2, type = "bmwd")
new <- rthmm(40, init.prob = c(1, 0), trans = A, type = "bmwd",
            par = c(-3, 3), len = 50)
table(predict(fit, new$data), new$states)
```

rbm

*Simulate sample paths of (fractional) Brownian motion with drift***Description**

Simulates n sample paths of $Y_\tau = c\tau + \sigma W_\tau^H$ on the grid $\tau_k = k/\text{len}$, $k = 1, \dots, \text{len}$, where W^H is fractional Brownian motion with Hurst parameter hurst (standard Brownian motion for $\text{hurst} = 0.5$).

Usage

```
rbm(n, len = 100L, drift = 0, hurst = 0.5, sigma = 1)
```

Arguments

<code>n</code>	Number of sample paths.
<code>len</code>	Number of grid points per path.
<code>drift</code>	Drift coefficient; a single number or a vector of length n giving one drift per path.
<code>hurst</code>	Hurst parameter in $(0, 1)$.
<code>sigma</code>	Diffusion coefficient.

Details

For $\text{hurst} \neq 0.5$ the paths are generated from the covariance function $(\tau_1^{2H} + \tau_2^{2H} - |\tau_1 - \tau_2|^{2H})/2$ using a Cholesky factor of the $\text{len} \times \text{len}$ covariance matrix, so large values of len are slow.

Value

An $n \times \text{len}$ matrix with one path per row.

Examples

```
x <- rbm(5, len = 100, drift = c(-2, -1, 0, 1, 2))
matplot(t(x), type = "l", lty = 1)
y <- rbm(3, len = 100, hurst = 0.8)
```

rbridge	<i>Simulate smooth Brownian bridge noise</i>
---------	--

Description

Simulates smooth random curves from a truncated Karhunen-Loeve expansion of the Brownian bridge, $\epsilon(\tau) = \sqrt{2} \sum_{k=1}^K Z_k \sin(k\pi\tau)/(k\pi)$ with Z_k independent $N(0, \sigma^2)$. This is the error process used in the non-parametric simulations of Kashlak, Loliencar and Heo (2023).

Usage

```
rbridge(n, len = 100L, sigma = 1, nterms = 16L)
```

Arguments

n	Number of curves.
len	Number of grid points $\tau_k = k/\text{len}$.
sigma	Standard deviation of the coefficients.
nterms	Number of terms K in the expansion.

Value

An $n \times \text{len}$ matrix with one curve per row.

Examples

```
e <- rbridge(5, len = 100, sigma = 0.4)
matplot(t(e), type = "l", lty = 1)
```

rmarkov	<i>Simulate a Markov chain</i>
---------	--------------------------------

Description

Simulate a Markov chain

Usage

```
rmarkov(n, init.prob, trans)
```

Arguments

n	Length of the chain.
init.prob	Initial state probabilities.
trans	Transition matrix (rows sum to one).

Value

An integer vector of states in `1:length(init.prob)`.

Examples

```
rmarkov(20, c(1, 0), matrix(c(.9, .1, .1, .9), 2))
```

rou

Simulate sample paths of the Ornstein-Uhlenbeck process

Description

Simulates $dY = \theta(\mu - Y) d\tau + \sigma dW$, $Y_0 = 0$, by an Euler scheme on the grid $\tau_k = k/\text{len}$.

Usage

```
rou(n, len = 100L, mean = 0, rate = 1, sigma = 1)
```

Arguments

n	Number of sample paths.
len	Number of grid points per path.
mean	Long-run mean μ ; a single number or a vector of length n.
rate	Mean-reversion rate θ ; a single number or a vector of length n.
sigma	Diffusion coefficient.

Value

An $n \times \text{len}$ matrix with one path per row.

Examples

```
x <- rou(4, len = 100, mean = c(-2, 0, 2, 4), rate = c(2, 4, 8, 20))
matplot(t(x), type = "l", lty = 1)
```

rthmm

*Simulate data from a topological hidden Markov model***Description**

Generates a hidden state sequence from a Markov chain and, conditional on the states, a sequence of curves from one of the emission models of [thmm\(\)](#).

Usage

```
rthmm(
  n,
  init.prob,
  trans,
  type = c("bmwd", "ou", "nonpar"),
  par,
  len = 100L,
  hurst = 0.5,
  sigma = 1,
  nterms = 16L
)
```

Arguments

<code>n</code>	Number of time steps (curves).
<code>init.prob</code>	Initial state probabilities.
<code>trans</code>	Transition matrix.
<code>type</code>	Emission model, see thmm() .
<code>par</code>	State parameters in the format described in thmm() : a vector (or <code>nstates</code> x <code>d</code> matrix) of drifts for "bmwd", an <code>nstates</code> x 2 matrix of means and rates for "ou", or an <code>nstates</code> x <code>len</code> matrix (or <code>nstates</code> x <code>len</code> x <code>d</code> array) of mean curves for "nonpar".
<code>len</code>	Number of grid points per curve (taken from <code>par</code> for "nonpar").
<code>hurst</code>	Hurst parameter for "bmwd".
<code>sigma</code>	Noise scale: the diffusion coefficient for "bmwd" and "ou", or the coefficient standard deviation passed to rbridge() for "nonpar".
<code>nterms</code>	Number of terms passed to rbridge() for "nonpar".

Value

A list with components `data` (an `n` x `len` matrix, or an `n` x `len` x `d` array) and `states` (integer vector of true states).

Examples

```
A <- matrix(0.09, 5, 5) + 0.55 * diag(5) # matrix A1 of the paper
sim <- rthmm(200, init.prob = c(1, 0, 0, 0, 0), trans = A,
            type = "bmwd", par = c(-4, -2, 0, 2, 4), len = 100)
matplot(t(sim$data), type = "l", lty = 1, col = sim$states + 1)
```

simulate.thmm

Simulate from a fitted THMM

Description

Simulate from a fitted THMM

Usage

```
## S3 method for class 'thmm'
simulate(object, nsim = 1L, seed = NULL, ...)
```

Arguments

object	A fitted "thmm" object.
nsim	Number of curves (time steps) to simulate.
seed	Optional random seed.
...	Further arguments passed to rthmm() , e.g. sigma or nterms.

Value

A list with components data and states, see [rthmm\(\)](#).

Examples

```
set.seed(4)
sim <- rthmm(60, init.prob = c(1, 0), trans = matrix(c(.8, .2, .2, .8), 2),
            type = "bmwd", par = c(-3, 3), len = 30)
fit <- thmm(sim$data, 2, type = "bmwd")
new <- simulate(fit, nsim = 10)
dim(new$data)
```

thmm

*Fit a topological hidden Markov model***Description**

Fits a hidden Markov model to a time-ordered sequence of curves (functional observations) using the Baum-Welch algorithm, with emission functions given by Onsager-Machlup functionals as described in Kashlak, Loliencar and Heo (2023). The most likely state sequence is then decoded with the Viterbi algorithm. The heavy lifting is done in C.

Usage

```
thmm(
  data,
  nstates,
  type = c("bmwd", "ou", "nonpar"),
  norm = c("W21", "L2", "W22"),
  hurst = 0.5,
  sigma = NULL,
  par = NULL,
  trans = NULL,
  init.prob = NULL,
  start = c("kmeans", "random"),
  nstart = 1L,
  max.iter = 500L,
  min.iter = 10L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

data	A numeric matrix with one curve per row (n curves observed on len equally spaced grid points on the unit interval), or a 3-d array of dimension n x len x d for d-dimensional curves. Rows are assumed to be in time order: row t is the observation emitted at time t.
nstates	Number of hidden states.
type	Emission model, see Details. One of "bmwd" (Brownian motion with drift, including fractional Brownian motion), "ou" (Ornstein-Uhlenbeck process) or "nonpar" (non-parametric state means under a user-chosen Cameron-Martin norm).
norm	Cameron-Martin norm for type = "nonpar": "W21" (Sobolev norm of the first derivative, the norm of the standard Wiener space), "L2" or "W22" (Sobolev norm of the second derivative).

hurst	Hurst parameter of the driving fractional Brownian motion for type = "bmwd". The default 0.5 is ordinary Brownian motion. The Onsager-Machlup functional is derived for hurst in (1/4, 1).
sigma	Scale (diffusion coefficient) of the driving Gaussian measure. All log-emission functions are divided by sigma^2, so sigma acts as a temperature: small values make the emissions dominate the transition probabilities. If NULL (the default) it is estimated from the realised quadratic variation of the curves, see Details.
par	Optional starting values for the state parameters. For "bmwd" a vector of nstates drifts (or an nstates x d matrix); for "ou" an nstates x 2 matrix whose columns are the long-run mean and the mean-reversion rate; for "nonpar" an nstates x len matrix (or nstates x len x d array) of mean curves. If NULL, starting values are generated according to start.
trans	Optional starting transition matrix (nstates x nstates, rows summing to one). Defaults to the uniform matrix.
init.prob	Optional starting initial state probabilities. Defaults to uniform.
start	How to generate starting values when par is NULL: "kmeans" (the default) runs k-means on the per-curve sufficient statistics and takes one re-estimation step from the resulting hard clustering; "random" picks nstates observations at random. The k-means start is markedly more robust to local optima, in particular for the Ornstein-Uhlenbeck model.
nstart	Number of random starts. The fit with the largest final log-likelihood is returned. Ignored (set to one) when par is supplied.
max.iter, min.iter	Maximum and minimum number of Baum-Welch iterations.
tol	Convergence tolerance: iterations stop once the absolute change in log-likelihood is below tol * max(1, loglik).
verbose	Logical; print the log-likelihood at each iteration?

Details

Let $O_1, \dots, O_n$ be the observed curves and let $b_j(O_t)$ denote the emission function of state j . Writing $\dot{O}_t$ for the derivative of the curve, the models are:

"bmwd" Brownian motion with drift, $dY = c_j d\tau + \sigma dW$. Up to a term that does not depend on c_j , $\log b_j(O_t) = -V(D_t - c_j)^2 / (2\sigma^2)$ where $D_t = \Gamma(2v + 2) / \Gamma(v + 1) \int_0^1 \tau^v \dot{O}_t(\tau) d\tau$, $v = 1/2 - \text{hurst}$ and $V = \Gamma(v + 1)^2 / \{\Gamma(2v + 1)\Gamma(2v + 2)\}$. For $\text{hurst} = 0.5$ this reduces to $D_t = O_t(1) - O_t(0)$ and $V = 1$. The drift is re-estimated as the posterior-weighted mean of D_t . For d-dimensional curves the coordinates are treated as independent Brownian motions with a d-vector of drifts per state.

"ou" Ornstein-Uhlenbeck process $dY = \theta_j(\mu_j - Y) d\tau + \sigma dW$, parametrised internally by $b_0 = \theta_j \mu_j$ and $b_1 = \theta_j$. Then $\log b_j(O_t) = \{b_0 A_t - b_1 E_t - b_0^2/2 + b_0 b_1 B_t - b_1^2 C_t/2\} / \sigma^2 + b_1/2$ with $A_t = O_t(1) - O_t(0)$, $B_t = \int O_t$, $C_t = \int O_t^2$ and $E_t = \int O_t \circ dO_t$ (Stratonovich). Because this is quadratic in (b_0, b_1) the re-estimation step has a closed form, with b_1 constrained to be non-negative.

"nonpar" Non-parametric mean curves h_j with $\log b_j(O_t) = -|O_t - h_j|_H^2 / (2\sigma^2)$ where $|\cdot|_H$ is the chosen Cameron-Martin norm, approximated by a Riemann sum on the observation grid: $\int (O - h)^2$ for "L2", $\int (\dot{O} - \dot{h})^2$ for "W21" and $\int (\ddot{O} - \ddot{h})^2$ for "W22". The mean curves are re-estimated as posterior-weighted averages of the observed curves.

All forward and backward probabilities are computed on the log scale. Iteration stops when the relative change in log-likelihood drops below `tol` (after at least `min.iter` iterations) or after `max.iter` iterations. Note that the "log-likelihood" is built from Onsager-Machlup functionals rather than densities and may be positive.

Automatic choice of sigma. When `sigma = NULL` the diffusion coefficient is estimated from second differences of the curves (the realised quadratic variation, which does not depend on the drift or mean curves). For "bmwd" and "ou" data generated from the standard model this returns a value close to one; for the non-parametric model it returns the scale of the noise in the chosen norm. For smooth curves under the "L2" norm the estimate can be very small, in which case the emissions dominate the transition probabilities and the fit behaves like a k-means clustering that respects time ordering. Supplying a larger sigma smooths the state assignments.

Value

An object of class "thmm": a list with components

<code>par</code>	Estimated state parameters, in the format described for the <code>par</code> argument.
<code>trans</code>	Estimated transition matrix.
<code>init.prob</code>	Estimated initial state probabilities.
<code>states</code>	Integer vector: the Viterbi (most likely) state sequence.
<code>posterior</code>	$n \times nstates$ matrix of posterior state probabilities.
<code>loglik</code>	Final log-likelihood (of the returned parameters).
<code>loglik.trace</code>	Log-likelihood at every iteration.
<code>viterbi.loglik</code>	Log-likelihood of the Viterbi path.
<code>iter, converged</code>	Number of iterations and convergence flag.
<code>nstates, type, norm, hurst, sigma, sigma.auto</code>	Model settings.
<code>data</code>	The data, as an $n \times len \times d$ array.
<code>call</code>	The matched call.

References

Kashlak, A. B., Loliencar, P. and Heo, G. (2023). Topological Hidden Markov Models. *Journal of Machine Learning Research*, 24(340), 1–49. <https://jmlr.org/papers/v24/22-0685.html>

See Also

`rthmm()` to simulate data, `predict.thmm()` to decode new sequences, `plot.thmm()`.

Examples

```
set.seed(1)
A <- matrix(0.1, 3, 3); diag(A) <- 0.8
sim <- rthmm(100, init.prob = c(1, 0, 0), trans = A,
            type = "bmwd", par = c(-4, 0, 4), len = 50)
fit <- thmm(sim$data, nstates = 3, type = "bmwd")
fit
table(fit$states, sim$states)
ari(fit$states, sim$states)

## Ornstein-Uhlenbeck curves with two states
sim <- rthmm(100, init.prob = c(1, 0), trans = matrix(c(.9, .1, .1, .9), 2),
            type = "ou", par = cbind(mean = c(-2, 2), rate = c(4, 4)),
            len = 50)
fit <- thmm(sim$data, nstates = 2, type = "ou")
fit$par

## non-parametric mean curves
tt <- seq(0, 1, length.out = 50)
mu <- rbind(sin(2 * pi * tt), cos(2 * pi * tt))
sim <- rthmm(100, init.prob = c(1, 0), trans = matrix(c(.9, .1, .1, .9), 2),
            type = "nonpar", par = mu, sigma = 0.4)
fit <- thmm(sim$data, nstates = 2, type = "nonpar", norm = "L2")
ari(fit$states, sim$states)
plot(fit)
```

Index

`ari`, [2](#)

`graphics::matplot()`, [3](#)

`plot.thmm`, [3](#)

`plot.thmm()`, [12](#)

`predict.thmm`, [4](#)

`predict.thmm()`, [12](#)

`rbm`, [5](#)

`rbridge`, [6](#)

`rbridge()`, [8](#)

`rmarkov`, [6](#)

`rou`, [7](#)

`rthmm`, [8](#)

`rthmm()`, [9](#), [12](#)

`simulate.thmm`, [9](#)

`thmm`, [10](#)

`thmm()`, [2](#), [8](#)