Package: dsmmR (via r-universe)

September 6, 2024

Type Package

Title Estimation and Simulation of Drifting Semi-Markov Models

Version 1.0.6

Date 2024-08-07

Description Performs parametric and non-parametric estimation and simulation of drifting semi-Markov processes. The definition of parametric and non-parametric model specifications is also possible. Furthermore, three different types of drifting semi-Markov models are considered. These models differ in the number of transition matrices and sojourn time distributions used for the computation of a number of semi-Markov kernels, which in turn characterize the drifting semi-Markov kernel. For the parametric model estimation and specification, several discrete distributions are considered for the sojourn times: Uniform, Poisson, Geometric, Discrete Weibull and Negative Binomial. The non-parametric model specification makes no assumptions about the shape of the sojourn time distributions. Semi-Markov models are described in: Barbu, V.S., Limnios, N. (2008) <doi:10.1007/978-0-387-73173-5>. Drifting Markov models are described in: Vergne, N. (2008) <doi:10.2202/1544-6115.1326>. Reliability indicators of Drifting Markov models are described in: Barbu, V. S., Vergne, N. (2019) <doi:10.1007/s11009-018-9682-8>. We acknowledge the DATALAB Project <https://lmrs-num.math.cnrs.fr/projet-datalab.html> (financed by the European Union with the European Regional Development fund (ERDF) and by the Normandy Region) and the HSMM-INCA Project (financed by the French Agence Nationale de la Recherche (ANR) under grant ANR-21-CE40-0005).

License GPL

URL https://github.com/Mavrogiannis-Ioannis/dsmmR

BugReports https://github.com/Mavrogiannis-Ioannis/dsmmR/issues

Imports DiscreteWeibull

Suggests utils, knitr, rmarkdown, testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation yes

Depends R (>= 3.5.0)

Config/testthat/edition 3

Repository https://mavrogiannis-ioannis.r-universe.dev

RemoteUrl https://github.com/mavrogiannis-ioannis/dsmmr

RemoteRef HEAD

RemoteSha 13ec4afcb37e7e538b488d1f343a9cfce15e82c2

Contents

`dsmmR-package` *dsmmR : Estimation and Simulation of Drifting Semi-Markov Models*

Description

Performs parametric and non-parametric estimation and simulation of drifting semi-Markov processes. The definition of parametric and non-parametric model specifications is also possible. Furthermore, three different types of drifting semi-Markov models are considered. These models differ in the number of transition matrices and sojourn time distributions used for the computation of a number of semi-Markov kernels, which in turn characterize the drifting semi-Markov kernel.

Details

Introduction

The difference between the Markov models and the semi-Markov models concerns the modelling of the sojourn time distributions. The Markov models (in discrete time) are modelled by a sojourn time following the Geometric distribution. The semi-Markov models are able to have a sojourn time distribution of arbitrary shape. The further difference with a *drifting* semi-Markov model, is that we have $d+1$ (arbitrary) sojourn time distributions and $d+1$ transition matrices (Model 1), where d is defined as the polynomial degree. Through them, we compute $d+1$ semi-Markov kernels. In this work, we also consider the possibility for obtaining these semi-Markov kernels with $d+1$ transition matrices and 1 sojourn time distribution (Model 2) or $d+1$ sojourn time distributions and 1 transition matrix (Model 3).

Definition

Drifting semi-Markov processes are particular non-homogeneous semi-Markov chains for which the drifting semi-Markov kernel $q_{\frac{t}{n}}(u,v,l)$ is defined as the probability that, given at the instance t the previous state is u, the next state state v will be reached with a sojourn time of l:

$$q_{\frac{t}{n}}(u,v,l) = P(J_t = v, X_t = l | J_{t-1} = u),$$

where n is the model size, defined as the length of the embedded Markov chain $(J_t)_{t\in\{0,\ldots,n\}}$ minus the last state, where J_t is the state at the instant t and $X_t = S_t - S_{t-1}$ is the sojourn time of the state J_{t-1}.

The drifting semi-Markov kernel $q_{\frac{t}{n}}$ is a linear combination of the product of $d+1$ semi-Markov kernels $q_{\frac{i}{d}}$, where every semi-Markov kernel is the product of a transition matrix p and a sojourn time distribution f. We define the situation when both p and f are "drifting" between $d+1$ fixed points of the model as Model 1, and thus we will use the exponential (1) as a way to refer to the drifting semi-Markov kernel $q_{\frac{t}{n}}^{(1)}$ and corresponding semi-Markov kernels $q_{\frac{i}{d}}^{(1)}$ in this case. For Model 2, we allow the transition matrix p to drift but not the sojourn time distributions f, and for Model 3 we allow the sojourn time distributions f to drift but not the transition matrix p. The exponential (2) or (3) will be used for signifying Model 2 or Model 3, respectively. In the general case an exponential will not be used.

Model 1

Both p and f are drifting in this case. Thus, the drifting semi-Markov kernel $q_{\frac{t}{n}}^{(1)}$ is a linear combination of the product of $d+1$ semi-Markov kernels $q_{\frac{i}{d}}^{(1)}$, which are given by:

$$q_{\frac{i}{d}}^{(1)}(u,v,l) = p_{\frac{i}{d}}(u,v) f_{\frac{i}{d}}(u,v,l),$$

where for $i = 0,\ldots,d$ we have $d+1$ Markov transition matrices $p_{\frac{i}{d}}(u,v)$ of the embedded Markov chain $(J_t)_{t\in\{0,\ldots,n\}}$, and $d+1$ sojourn time distributions $f_{\frac{i}{d}}(u,v,l)$. Therefore, the drifting semi-Markov kernel is described as:

$$q_{\frac{t}{n}}^{(1)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, q_{\frac{i}{d}}^{(1)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, p_{\frac{i}{d}}(u,v) f_{\frac{i}{d}}(u,v,l),$$

where $A_i, i = 0,\ldots,d$ are $d+1$ polynomials with degree d, which satisfy the conditions:

$$\sum_{i=0}^{d} A_i(t) = 1,$$

$$A_i\left(\frac{nj}{d}\right) = 1_{\{i=j\}},$$

where the indicator function $1_{\{i=j\}} = 1$, if $i = j$, 0 otherwise.

Model 2

In this case, p is drifting and f **is not drifting**. Therefore, the drifting semi-Markov kernel is now described as:

$$q^{(2)}_{\frac{t}{n}}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, q^{(2)}_{\frac{i}{d}}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, p_{\frac{i}{d}}(u,v) f(u,v,l).$$

Model 3

In this case, f is drifting and p **is not drifting**. Therefore, the drifting semi-Markov Kernel is now described as:

$$q^{(3)}_{\frac{t}{n}}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, q^{(3)}_{\frac{i}{d}}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, p(u,v) f_{\frac{i}{d}}(u,v,l).$$

Parametric and non-parametric model specifications

In this package, we can define parametric and non-parametric drifting semi-Markov models.

For the *parametric* case, several discrete distributions are considered for the modelling of the sojourn times: Uniform, Geometric, Poisson, Discrete Weibull and Negative Binomial. This is done from the function `parametric_dsmm` which returns an object of the S3 class (`dsmm_parametric`, `dsmm`).

The *non-parametric* model specification concerns the sojourn time distributions when no assumptions are done about the shape of the distributions. This is done through the function called `nonparametric_dsmm()`, that returns an object of class (`dsmm_nonparametric`, `dsmm`).

It is also possible to proceed with a parametric or non-parametric estimation for a model on an existing sequence through the function `fit_dsmm()`, which returns an object with the S3 class (`dsmm_fit_parametric`, `dsmm`) or (`dsmm_fit_nonparametric`, `dsmm`) respectively, depending on the given argument `estimation = "parametric"` or `estimation = "nonparametric"` .

Therefore, the `dsmm` class acts like a wrapper class for drifting semi-Markov model specifications, while the classes `dsmm_fit_parametric`, `dsmm_fit_nonparametric`, `dsmm_parametric` and `dsmm_nonparametric` are exclusive to the functions that create the corresponding models, and inherit methods from the `dsmm` class.

In summary, based on an `dsmm` object it is possible to use the following methods:

- Simulate a sequence through the function `simulate.dsmm()`.
- Get the drifting semi-Markov kernel $q_{\frac{t}{n}}(u,v,l)$, for any choice of u, v, l or t, through the function `get_kernel()`.

Restrictions

The following restrictions must be satisfied for every drifting semi-Markov model:

- The drifting semi-Markov kernel $q_{\frac{t}{n}}(u,v,l)$, for every $t \in \{0,\dots,n\}$ and $u \in E$, has its sums over v and l, equal to 1:

$$\sum_{v\in E}\sum_{l=1}^{+\infty} q_{\frac{t}{n}}(u,v,l) = \sum_{v\in E}\sum_{l=1}^{+\infty} A_i(t)\, q_{\frac{i}{d}}(u,v,l) = 1.$$

- Therefore, we also get that for every $i \in \{0,\dots,d\}$ and $u \in E$, the semi-Markov kernel $q_{\frac{i}{d}}(u,v,l)$ has its sums over v and l equal to 1:

$$\sum_{v\in E}\sum_{l=1}^{+\infty} q_{\frac{i}{d}}(u,v,l) = 1.$$

- Lastly, like in semi-Markov models, we do not allow sojourn times equal to 0 or passing into the same state:

$$q_{\frac{t}{n}}(u,v,0) = 0, \forall u,v \in E,$$
$$q_{\frac{t}{n}}(u,u,l) = 0, \forall u \in E, l \in \{1,\dots,+\infty\}.$$

Model specification restrictions

When we define a drifting semi-Markov model specification through the functions `parametric_dsmm` or `nonparametric_dsmm`, the following restrictions need to be satisfied.

Model 1

The semi-Markov kernels are equal to $q_{\frac{i}{d}}^{(1)}(u,v,l) = p_{\frac{i}{d}}(u,v) f_{\frac{i}{d}}(u,v,l)$. Therefore, $\forall u \in E$ the sums of $p_{\frac{i}{d}}(u,v)$ over v and the sums of $f_{\frac{i}{d}}(u,v,l)$ over l must be equal to 1:

$$\sum_{v\in E} p_{\frac{i}{d}}(u,v) = 1,$$

$$\sum_{l=1}^{+\infty} f_{\frac{i}{d}}(u,v,l) = 1.$$

Model 2

The semi-Markov kernels are equal to $q_{\frac{i}{d}}^{(2)}(u,v,l) = p_{\frac{i}{d}}(u,v) f(u,v,l)$. Therefore, $\forall u \in E$ the sums of $p_{\frac{i}{d}}(u,v)$ over v and the sums of $f(u,v,l)$ over l must be equal to 1:

$$\sum_{v\in E} p_{\frac{i}{d}}(u,v) = 1,$$

$$\sum_{l=1}^{+\infty} f(u,v,l) = 1.$$

Model 3

The semi-Markov kernels are equal to $q_{\frac{i}{d}}^{(3)}(u,v,l) = p(u,v) f_{\frac{i}{d}}(u,v,l)$. Therefore, $\forall u \in E$ the sums of $p(u,v)$ over v and the sums of $f_{\frac{i}{d}}(u,v,l)$ over l must be equal to 1:

$$\sum_{v\in E} p(u,v) = 1,$$

$$\sum_{l=1}^{+\infty} f_{\frac{i}{d}}(u,v,l) = 1.$$

Community Guidelines

For third parties wishing to contribute to the software, or to report issues or problems about the software, they can do so directly through the development github page of the package.

Notes

Automated tests are in place in order to aid the user with any false input made and, furthermore, to ensure that the functions used return the expected output. Moreover, through strict automated tests, it is made possible for the user to properly define their own `dsmm` objects and make use of them with the generic functions of the package.

Author(s)

Maintainer: Ioannis Mavrogiannis `<mavrogiannis.ioa@gmail.com>`

Authors:

- Vlad Stefan Barbu
- Ioannis Mavrogiannis
- Nicolas Vergne

References

Barbu, V. S., Limnios, N. (2008). Semi-Markov Chains and Hidden Semi-Markov Models Toward Applications - Their Use in Reliability and DNA Analysis. New York: Lecture Notes in Statistics, vol. 191, Springer.

Vergne, N. (2008). Drifting Markov models with Polynomial Drift and Applications to DNA Sequences. Statistical Applications in Genetics Molecular Biology 7 (1).

Barbu V. S., Vergne, N. (2019). Reliability and survival analysis for drifting Markov models: modelling and estimation. Methodology and Computing in Applied Probability, 21(4), 1407-1429.

T. Nakagawa and S. Osaki. (1975). The discrete Weibull distribution. IEEE Transactions on Reliability, R-24, 300-301.

Sanger, F., Coulson, A. R., Hong, G. F., Hill, D. F., & Petersen, G. B. (1982). Nucleotide sequence of bacteriophage λ DNA. Journal of molecular biology, 162(4), 729-773.

See Also

For the estimation of a drifting semi-Markov model given a sequence: fit_dsmm.

For drifting semi-Markov model specifications: parametric_dsmm, nonparametric_dsmm.

For the simulation of sequences: simulate.dsmm, create_sequence.

For the retrieval of the drifting semi-Markov kernel through a `dsmm` object: get_kernel.

create_sequence — *Simulate a sequence for states of choice.*

Description

This is a wrapper function around `sample()`.

Usage

```
create_sequence(states, len = 5000, probs = NULL, seed = NULL)
```

Arguments

`states`	Character vector of unique values. If given the value "DNA" the values c("a", "c", "g", "t") are given instead.
`len`	Optional. Positive integer with the default value equal to 5000.
`probs`	Optional. Numeric vector with values interpreted as probabilities for each of the states in `states`. Default value is equal to 1 over the number of states given, for every state.
`seed`	Optional. Object specifying the initialization of the random number generator (see more in `set.seed`).

Value

A character sequence of length `len`.

See Also

For the simulation of a sequence with a drifting semi-Markov kernel: simulate.dsmm.

The original function: `sample`.

About random number generation in R: `RNG`.

For the theoretical background of drifting semi-Markov models: dsmmR.

Examples

```
# This is equal to having the argument `probs = c(1/4, 1/4, 1/4, 1/4)`.
rand_dna_seq <- create_sequence(states = "DNA")
table(rand_dna_seq)

random_letters <- sample(letters, size = 5, replace = FALSE)
rand_dna_seq2 <- create_sequence(
    states = random_letters,
    probs = c(0.6, 0.3, 0.05, 0.025, 0.025),
    len = 10000)
table(rand_dna_seq2)
```

fit_dsmm *Estimation of a drifting semi-Markov chain*

Description

Estimation of a drifting semi-Markov chain, given one sequence of states. This estimation can be parametric or non-parametric and is available for the three types of drifting semi-Markov models.

Usage

```
fit_dsmm(
  sequence,
  degree,
  f_is_drifting,
  p_is_drifting,
  states = NULL,
  initial_dist = "unif",
  estimation = "nonparametric",
  f_dist = NULL
)
```

Arguments

`sequence` Character vector that represents a sequence of states from the state space E.

`degree` Positive integer that represents the polynomial degree d for the drifting semi-Markov model.

`f_is_drifting` Logical. Specifies if f is drifting or not.

`p_is_drifting` Logical. Specifies if p is drifting or not.

`states` Character vector that represents the state space E, with length equal to $s = |E|$. Default value is set equal to the sorted, unique states present in the given sequence.

`initial_dist` Optional. Character that represents the method to estimate the initial distribution.

- `"unif"` : The initial distribution of each state is equal to $1/s$ (default value).
- `"freq"` : The initial distribution of each state is equal to the frequency that it has in the sequence.

`estimation` Optional. Character. Represents whether the estimation will be nonparametric or parametric.

- `"nonparametric"` : The estimation will be non-parametric (default value).
- `"parametric"` : The estimation will be parametric.

`f_dist` Optional. It can be defined in two ways:

- If `estimation = "nonparametric"`, it is equal to `NULL` (default value).

- If `estimation = "parametric"`, it is a character array that specifies the distributions of the sojourn times, for every state transition. The list of possible values is: `["unif", "geom", "pois", "dweibull", "nbinom", NA]`. It can be defined in two ways:
  - If f **is not** drifting, it has dimensions of $s \times s$.
  - If f **is** drifting, it has dimensions of $s \times s \times (d+1)$ (see more in *Details, Parametric Estimation.*)

  It is defined similarly to the attribute `f_dist` in dsmm_parametric.

Details

This function estimates a drifting semi-Markov model in the parametric and non-parametric case. The parametric estimation can be achieved by the following steps:

1. We obtain the non-parametric estimation of the sojourn time distributions.
2. We estimate the parameters for the distributions defined in the attribute `f_dist` through the probabilities that were obtained in step 1.

Three different models are possible for to be estimated for each case. A normalization technique is used in order to correct estimation errors from small sequences. We will use the exponentials $(1), (2), (3)$ to distinguish between the drifting semi-Markov kernel $\widehat{q}_{\frac{t}{n}}$ and the semi-Markov kernels $\widehat{q}_{\frac{i}{d}}$ used in Model 1, Model 2, Model 3. More about the theory of drifting semi-Markov models in dsmmR.

Non-parametric Estimation

Model 1

When the transition matrix p of the embedded Markov chain $(J_t)_{t \in \{0,\dots,n\}}$ and the conditional sojourn time distribution f are both drifting, the drifting semi-Markov kernel can be estimated as:

$$\widehat{q}_{\frac{t}{n}}^{\,(1)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, \widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l),$$

$\forall t \in \{0,\dots,n\}, \forall u, v \in E, \forall l \in \{1,\dots,k_{max}\}$, where k_{max} is the maximum sojourn time that was observed in the sequence and $A_i, i = 0,\dots,d$ are $d+1$ polynomials with degree d (see dsmmR).

The semi-Markov kernels $\widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l), i = 0,\dots,d$, are estimated through Least Squares Estimation (LSE) and are obtained after solving the following system, $\forall t \in \{0,\dots,n\}$, $\forall u, v \in E$ and $\forall l \in \{1,\dots,k_{max}\}$:

$$MJ = P,$$

where the matrices are written as:

- $M = (M_{ij})_{i,j \in \{0,\dots,d\}} = \left(\sum_{t=1}^{n} 1_u(t) A_i(t) A_j(t)\right)_{i,j \in \{0,\dots,d\}}$
- $J = (J_i)_{i \in \{0,\dots,d\}} = \left(\widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)\right)_{i \in \{0,\dots,d\}}$
- $P = (P_i)_{i \in \{0,\dots,d\}} = \left(\sum_{t=1}^{n} 1_{uvl}(t) A_i(t)\right)_{i \in \{0,\dots,d\}}$

and we use the following indicator functions:

- $1_u(t) = 1_{\{J_{t-1}=u\}} = 1$, if at t the previous state is u, 0 otherwise.
- $1_{uvl}(t) = 1_{\{J_{t-1}=u, J_t=v, X_t=l\}} = 1$, if at t the previous state is u with sojourn time l and next state v, 0 otherwise.

In order to obtain the estimations of $\widehat{p}_{\frac{i}{d}}(u,v)$ and $\widehat{f}_{\frac{i}{d}}(u,v,l)$, we use the following formulas:

$$\widehat{p}_{\frac{i}{d}}(u,v) = \sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l),$$

$$\widehat{f}_{\frac{i}{d}}(u,v,l) = \frac{\widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l)}{\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l)}.$$

Model 2

In this case, p is drifting and f is not drifting. Therefore, the estimated drifting semi-Markov kernel will be given by:

$$\widehat{q}_{\frac{t}{n}}^{(2)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, \widehat{q}_{\frac{i}{d}}^{(2)}(u,v,l),$$

$\forall t \in \{0,\ldots,n\}, \forall u,v \in E, \forall l \in \{1,\ldots,k_{max}\}$, where k_{max} is the maximum sojourn time that was observed in the sequence and $A_i, i = 0,\ldots,d$ are $d+1$ polynomials with degree d (see dsmmR). In order to obtain the estimators $\widehat{p}$ and $\widehat{f}$, we use the estimated semi-Markov kernels $\widehat{q}_{\frac{i}{d}}^{(1)}$ from Model 1. Since p is drifting, we define the estimation $\widehat{p}$ the same way as we did in Model 1. In total, we have the following estimations, $\forall u,v \in E, \forall l \in \{1,\ldots,k_{max}\}$:

$$\widehat{p}_{\frac{i}{d}}(u,v) = \sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l),$$

$$\widehat{f}(u,v,l) = \frac{\sum_{i=0}^{d} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l)}{\sum_{i=0}^{d}\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l)}.$$

Thus, the *estimated* semi-Markov kernels for Model 2, $\widehat{q}_{\frac{i}{d}}^{(2)}(u,v,l) = \widehat{p}_{\frac{i}{d}}(u,v)\widehat{f}(u,v,l)$, can be written with regards to the *estimated* semi-Markov kernels of Model 1, $\widehat{q}_{\frac{i}{d}}^{(1)}$, as in the following:

$$\widehat{q}_{\frac{i}{d}}^{(2)}(u,v,l) = \frac{(\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l))(\sum_{i=0}^{d} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l))}{\sum_{i=0}^{d}\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{(1)}(u,v,l)}.$$

Model 3

In this case, f is drifting and p is not drifting. Therefore, the estimated drifting semi-Markov kernel will be given by:

$$\widehat{q}_{\frac{t}{n}}^{(3)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, \widehat{q}_{\frac{i}{d}}^{(3)}(u,v,l),$$

$\forall t \in \{0, \ldots, n\}, \forall u, v \in E, \forall l \in \{1, \ldots, k_{max}\}$, where k_{max} is the maximum sojourn time that was observed in the sequence and $A_i, i = 0, \ldots, d$ are $d + 1$ polynomials with degree d (see dsmmR). In order to obtain the estimators $\widehat{p}$ and $\widehat{f}$, we use the estimated semi-Markov kernels estimated semi-Markov kernels $\widehat{q}_{\frac{i}{d}}^{\,(1)}$ from Model 1. Since f is drifting, we define the estimation $\widehat{f}$ the same way as we did in Model 1. In total, we have the following estimations, $\forall u, v \in E, \forall l \in \{1, \ldots, k_{max}\}$:

$$\widehat{p}\,(u,v) = \frac{\sum_{i=0}^{d}\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)}{d+1},$$

$$\widehat{f}_{\frac{i}{d}}(u,v,l) = \frac{\widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)}{\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)}.$$

Thus, the *estimated* semi-Markov kernels for Model 3, $\widehat{q}_{\frac{i}{d}}^{\,(3)}(u,v,l) = \widehat{p}\,(u,v)\widehat{f}_{\frac{i}{d}}(u,v,l)$, can be written with regards to the *estimated* semi-Markov kernels of Model 1, $\widehat{q}_{\frac{i}{d}}^{\,(1)}$, as in the following:

$$\widehat{q}_{\frac{i}{d}}^{\,(3)}(u,v,l) = \frac{\widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)\sum_{i=0}^{d}\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)}{(d+1)\sum_{l=1}^{k_{max}} \widehat{q}_{\frac{i}{d}}^{\,(1)}(u,v,l)}.$$

Parametric Estimation

In this package, the parametric estimation of the sojourn time distributions defined in the attribute `f_dist` is achieved as follows:

1. We obtain the non-parametric LSE of the sojourn time distributions f.
2. We estimate the parameters for the distributions defined in `f_dist` through the probabilities of f, estimated in previously in 1.

The available distributions for the modelling of the conditional sojourn times of the drifting semi-Markov model, defined from the argument `f_dist`, have their parameters estimated through the following formulas:

- Geometric (p):
  $f(x) = p(1-p)^{x-1}$, where $x = 1, 2, \ldots, k_{max}$. We estimate the probability of success $\widehat{p}$ as such:
  $$\widehat{p} = \frac{1}{E(X)}$$
- Poisson (λ):
  $f(x) = \frac{\lambda^{x-1}exp(-\lambda)}{(x-1)!}$, where $x = 1, 2, \ldots, k_{max}$. We estimate $\widehat{\lambda} > 0$ as such:
  $$\widehat{\lambda} = E(X)$$
- Negative binomial (α, p):
  $f(x) = \frac{\Gamma(x+\alpha-1)}{\Gamma(\alpha)(x-1)!}p^{\alpha}(1-p)^{x-1}$, where $x = 1, 2, \ldots, k_{max}$. Γ is the Gamma function, p is the probability of success and $\alpha \in (0, +\infty)$ is the parameter describing the number of successful

trials, or the dispersion parameter (the shape parameter of the gamma mixing distribution). We estimate them as such:

$$\widehat{p} = \frac{E(X)}{Var(X)},$$
$$\widehat{\alpha} = E(X)\frac{\widehat{p}}{1-\widehat{p}} = \frac{E(X)^2}{Var(X)-E(X)}.$$

- Discrete Weibull of type 1 (q, β):
  $f(x) = q^{(x-1)^\beta} - q^{x^\beta}$, where $x = 1, 2, \ldots, k_{max}$, q is the first parameter with $0 < q < 1$ and $\beta \in (0, +\infty)$ the second parameter. We estimate them as such:

$$\widehat{q} = 1 - f(1),$$
$$\widehat{\beta} = \frac{\sum_{i=2}^{k_{max}} \log_i(\log_{\widehat{q}}(\sum_{j=1}^{i} f(j)))}{k_{max}-1}.$$

  Note that we require $k_{max} \geq 2$ for estimating $\widehat{\beta}$.
- Uniform (n): $f(x) = 1/n$ where $x = 1, 2, \ldots, n$, for n a positive integer. We use a numerical method to obtain an estimator for $\widehat{n}$ in this case.

Value

Returns an object of S3 class `(dsmm_fit_nonparametric, dsmm)` or `(dsmm_fit_parametric, dsmm)`. It has the following attributes:

- `dist` : List. Contains 2 or 3 arrays.
  - If `estimation = "nonparametric"` we have 2 arrays:
    - `p_drift` or `p_notdrift`, corresponding to whether the defined p transition matrix is drifting or not.
    - `f_drift` or `f_notdrift`, corresponding to whether the defined f sojourn time distribution is drifting or not.
  - If `estimation = "parametric"` we have 3 arrays:
    - `p_drift` or `p_notdrift`, corresponding to whether the defined p transition matrix is drifting or not.
    - `f_drift_parametric` or `f_notdrift_parametric`, corresponding to whether the defined f sojourn time distribution is drifting or not.
    - `f_drift_parameters` or `f_notdrift_parameters`, which are the defined f sojourn time distribution parameters, depending on whether f is drifting or not.
- `emc` : Character vector that contains the **embedded Markov chain** $(J_t)_{t\in\{0,\ldots,n\}}$ **of the original sequence**. It is this attribute of the object that describes the size of the model n. Last state is also included, for a total length of $n+1$, but it is not used for any calculation.
- `soj_times` : Numerical vector that contains the sojourn times spent for each state in `emc` before the jump to the next state. Last state is also included, for a total length of $n+1$, but it is not used for any calculation.
- `initial_dist` : Numerical vector that contains an estimation for the initial distribution of the realized states in `sequence`. It always has values between 0 and 1.

- `states` : Character vector. Passing down from the arguments. It contains the realized states given in the argument `sequence`.
- `s` : Positive integer that contains the length of the character vector given in the attribute `states`, which is equal to $s = |E|$.
- `degree` : Positive integer. Passing down from the arguments. It contains the polynomial degree d considered for the drifting of the model.
- `k_max` : Numerical value that contains the maximum sojourn time, which is the maximum value in `soj_times`, excluding the last state.
- `model_size` : Positive integer that contains the size of the drifting semi-Markov model n, which is equal to the length of the embedded Markov chain $(J_t)_{t\in\{0,\dots,n\}}$, minus the last state. It has a value of `length(emc) - 1`, for `emc` as defined above.
- `f_is_drifting` : Logical. Passing down from the arguments. Specifies if f is drifting or not.
- `p_is_drifting` : Logical. Passing down from the arguments. Specifies if p is drifting or not.
- `Model` : Character. Possible values:
  - `"Model_1"` : Both p and f are drifting.
  - `"Model_2"` : p is drifting and f is not drifting.
  - `"Model_3"` : f is drifting and p is not drifting.
- `estimation` : Character. Specifies whether parametric or nonparametric estimation was used.
- `A_i` : Numerical Matrix. Represents the polynomials $A_i(t)$ with degree d that were used for solving the system $MJ = P$. Used for the methods defined for the object. Not printed when viewing the object.
- `J_i` : Numerical Array. Represents the estimated semi-Markov kernels of the first model $(\widehat{q}^{\,(1)}_{\frac{i}{d}}(u,v,l))_{i\in\{0,\dots,d\}}$ that were obtained after solving the system $MJ = P$. Not printed when viewing the object.

References

V. S. Barbu, N. Limnios. (2008). semi-Markov Chains and Hidden semi-Markov Models Toward Applications - Their Use in Reliability and DNA Analysis. New York: Lecture Notes in Statistics, vol. 191, Springer.

Vergne, N. (2008). Drifting Markov models with Polynomial Drift and Applications to DNA Sequences. Statistical Applications in Genetics Molecular Biology 7 (1).

Barbu V. S., Vergne, N. (2019). Reliability and survival analysis for Drifting Markov models: modelling and estimation. Methodology and Computing in Applied Probability, 21(4), 1407-1429.

T. Nakagawa and S. Osaki. (1975). The discrete Weibull distribution. IEEE Transactions on Reliability, R-24, 300-301.

See Also

For the theoretical background of drifting semi-Markov models: dsmmR.

For sequence simulation: simulate.dsmm and create_sequence.

For drifting semi-Markov model specification: parametric_dsmm, nonparametric_dsmm

For the retrieval of the drifting semi-Markov kernel: get_kernel.

Examples

```
# Create a random sequence
sequence <- create_sequence("DNA", len = 2000, seed = 1)
## Alternatively, we could obtain a sequence as follows:
## > data("lambda", package = "dsmmR")
## > sequence <- c(lambda)
states <- sort(unique(sequence))
degree <- 3


# ===========================================================================
# Nonparametric Estimation.
# Fitting a random sequence under distributions of unknown shape.
# ===========================================================================


# ---------------------------------------------------------------------------
# Both p and f are drifting - Model 1.
# ---------------------------------------------------------------------------

obj_model_1 <- fit_dsmm(sequence = sequence,
                        degree = degree,
                        f_is_drifting = TRUE,
                        p_is_drifting = TRUE,
                        states = states,
                        initial_dist = "freq",
                        estimation = "nonparametric", # default value
                        f_dist = NULL # default value
                        )

cat(paste0("We fitted a sequence with ", obj_model_1$Model, ",\n",
           "model size: n = ", obj_model_1$model_size, ",\n",
           "length of state space: s = ", obj_model_1$s, ",\n",
           "maximum sojourn time: k_max = ", obj_model_1$k_max, " and\n",
           "polynomial (drifting) Degree: d = ", obj_model_1$degree, ".\n"))

# Get the drifting p and f arrays.
p_drift <- obj_model_1$dist$p_drift
f_drift <- obj_model_1$dist$f_drift

cat(paste0("Dimension of p_drift: (s, s, d + 1) = (",
           paste(dim(p_drift), collapse = ", "), ").\n",
           "Dimension of f_drift: (s, s, k_max, d + 1) = (",
           paste(dim(f_drift), collapse = ", "), ").\n"))

# We can even check the embedded Markov chain and the sojourn times
# directly from the returned object, if we wish to do so.
# This is achieved through the `base::rle()` function, used on `sequence`.
model_emc <- obj_model_1$emc
model_sojourn_times <- obj_model_1$soj_times

# ---------------------------------------------------------------------------
```

```
# Fitting the sequence when p is drifting and f is not drifting - Model 2.
# ---------------------------------------------------------------------------


obj_model_2 <- fit_dsmm(sequence = sequence,
                        degree = degree,
                        f_is_drifting = FALSE,
                        p_is_drifting = TRUE)


cat(paste0("We fitted a sequence with ", obj_model_2$Model, ".\n"))


# Get the drifting p and non-drifting f arrays.
p_drift_2 <- obj_model_2$dist$p_drift
f_notdrift <- obj_model_2$dist$f_notdrift


all.equal.numeric(p_drift, p_drift_2) # p is the same as in Model 1.


cat(paste0("Dimension of f_notdrift: (s, s, k_max) = (",
           paste(dim(f_notdrift), collapse = ", "), ").\n"))



# ---------------------------------------------------------------------------
# Fitting the sequence when f is drifting and p is not drifting - Model 3.
# ---------------------------------------------------------------------------

obj_model_3 <- fit_dsmm(sequence = sequence,
                        degree = degree,
                        f_is_drifting = TRUE,
                        p_is_drifting = FALSE)

cat(paste0("We fitted a sequence with ", obj_model_3$Model, ".\n"))
# Get the drifting f and non-drifting p arrays.
p_notdrift <- obj_model_3$dist$p_notdrift
f_drift_3 <- obj_model_3$dist$f_drift
all.equal.numeric(f_drift, f_drift_3) # f is the same as in Model 1.
cat(paste0("Dimension of f_notdrift: (s, s) = (",
           paste(dim(p_notdrift), collapse = ", "), ").\n"))

# ===========================================================================
# Parametric Estimation
# Fitting a random sequence under distributions of known shape.
# ===========================================================================
### Comments
### 1.  For the parametric estimation it is recommended to use a common set
###     of distributions while only the parameters (of the sojourn times)
###     are drifting. This results in (generally) higher accuracy.
### 2.  This process is similar to that used in `dsmm_parametric()`.
```

```
s <- length(states)
# Getting the distributions for the states.
# Rows correspond to previous state `u`.
# Columns correspond to next state `v`.
f_dist_1 <- matrix(c(NA,        "unif",     "dweibull", "nbinom",
                     "pois",      NA,        "pois",     "dweibull",
                     "geom",     "pois",      NA,        "geom",
                     "dweibull", 'geom',     "pois",      NA),
                   nrow = s, ncol = s, byrow = TRUE)
f_dist <- array(f_dist_1, dim = c(s, s, degree + 1))
dim(f_dist)

# ---------------------------------------------------------------------------
# Both p and f are drifting - Model 1.
# ---------------------------------------------------------------------------

obj_fit_parametric <- fit_dsmm(sequence = sequence,
                               degree = degree,
                               f_is_drifting = TRUE,
                               p_is_drifting = TRUE,
                               states = states,
                               initial_dist = 'unif',
                               estimation = 'parametric',
                               f_dist = f_dist)
cat("The class of `obj_fit_parametric` is : (",
    paste0(class(obj_fit_parametric), collapse = ', '), ").\n")

# Estimated parameters.
f_params <- obj_fit_parametric$dist$f_drift_parameters

# The drifting sojourn time distribution parameters.
f_0 <- f_params[,,,1]
f_1.3 <- f_params[,,,2]
f_2.3 <- f_params[,,,3]
f_1 <- f_params[,,,4]

params <- paste0('q = ', round(f_params["c", "t", 1, ], 3),
                 ', beta = ', round(f_params["c", "t", 2, ], 3))
f_names <- c("f_0", paste0("f_", 1:(degree-1), "/", degree), "f_1")
all_names <- paste(f_names, ":", params)
cat("The drifting of the parameters for passing from \n",
    "`u` = 'c' to `v` = 't' under a discrete Weibull distribution is:",
    "\n", all_names[1], "\n", all_names[2],
    "\n", all_names[3], "\n", all_names[4])

# ---------------------------------------------------------------------------
# f is not drifting, only p is drifting - Model 2.
# ---------------------------------------------------------------------------

obj_fit_parametric_2 <- fit_dsmm(sequence = sequence,
                                 degree = degree,
                                 f_is_drifting = FALSE,
```

```
                                  p_is_drifting = TRUE,
                                  initial_dist = 'unif',
                                  estimation = 'parametric',
                                  f_dist = f_dist_1)

cat("The class of `obj_fit_parametric_2` is : (",
    paste0(class(obj_fit_parametric_2), collapse = ', '), ").\n")
# Estimated parameters.
f_params_2 <- obj_fit_parametric_2$dist$f_notdrift_parameters

params_2 <- paste0('q = ', round(f_params_2["c", "t", 1], 3),
                   ', beta = ', round(f_params_2["c", "t", 2], 3))

cat("Not-drifting parameters for passing from ",
    "`u` = 'c' to `v` = 't' \n under a discrete Weibull distribution are:\n",
    paste("f :", params_2))


# ============================================================================
# `simulate()` and `get_kernel()` can be used for the two objects,
# `dsmm_fit_nonparametric` and `dsmm_fit_parametric`.
# ============================================================================


sim_seq_nonparametric <- simulate(obj_model_1, nsim = 10)
str(sim_seq_nonparametric)


kernel_drift_parametric <- get_kernel(obj_fit_parametric, klim = 10)
str(kernel_drift_parametric)
```

---

get_kernel *Obtain the Drifting semi-Markov kernel*

Description

This is a generic method that computes and returns the drifting semi-Markov kernel.

Usage

```
get_kernel(obj, t, u, v, l, klim = 100)
```

Arguments

obj	An object that inherits from the S3 classes dsmm, dsmm_fit_parametric, or dsmm_fit_nonparametric, dsmm_nonparametric or dsmm_parametric.
t	Optional, but recommended. Positive integer specifying the instance t of the visited states.

`u` Optional. Can be either of the two options below:

- Character specifying the previous state u, e.g. `u = "a"`.
- Positive integer, specifying a state in the state space E. For example, if $E = \{a, c, g, t\}$ and `u = 1`, it corresponds to the state a, if `u = 2`, it corresponds to the state c.

`v` Optional. Can be either of the two options below:

- Character specifying the next state v, e.g. `v = "c"`.
- Positive integer, specifying a state in the state space E. For example, if $E = \{a, c, g, t\}$ and `v = 3`, it corresponds to the state c, if `v = 4`, it corresponds to the state t.

`l` Optional. Positive integer specifying the sojourn time l that is spent in the previous state u.

`klim` Optional. Positive integer. Used only when `obj` inherits from the S3 classes `dsmm_parametric` or `dsmm_fit_parametric`. Specifies the time horizon used to approximate the $d + 1$ sojourn time distributions if f is drifting, or just 1 sojourn time distribution if f is *not drifting*. Default value is 100.

A larger value will result in a considerably larger kernel, which has dimensions of $s \times s \times klim \times (n + 1)$, which will increase the memory requirements and will slow down considerably the `simulate.dsmm()` method. However, this will lead to better estimations through `fit_dsmm()`. (dsmm_parametric, fit_dsmm, simulate.dsmm)

Details

The drifting semi-Markov kernel is given as the probability that, given at the instance t the previous state is u, the next state state v will be reached with a sojourn time of l:

$$q_{\frac{t}{n}}(u, v, l) = P(J_t = v, X_t = l | J_{t-1} = u),$$

where n is the model size, defined as the length of the embedded Markov chain $(J_t)_{t\in\{0,\dots,n\}}$ minus the last state, J_t is the visited state at the instant t and $X_t = S_t - S_{t-1}$ is the sojourn time of the state J_{t-1}. Specifically, it is given as the sum of a linear combination:

$$q_{\frac{t}{n}}(u, v, l) = \sum_{i=0}^{d} A_i(t)\, q_{\frac{i}{d}}(u, v, l),$$

where $A_i, i = 0, \dots, d$ are $d + 1$ polynomials with degree d that satisfy certain conditions (see dsmmR) and $q_{\frac{i}{d}}(u, v, l), i = 0, \dots, d$ are $d + 1$ semi-Markov kernels. Three possible model specifications are described below. We will use the exponentials $(1), (2), (3)$ to distinguish between the drifting semi-Markov kernel $q_{\frac{t}{n}}$ and the semi-Markov kernels $q_{\frac{i}{d}}$ used in Model 1, Model 2 and Model 3.

Model 1

In this case, both p and f are "drifting" between $d+1$ fixed points of the model, hence the "drifting" in drifting semi-Markov models. Therefore, the semi-Markov kernels $q_{\frac{i}{d}}^{(1)}$ are equal to:

$$q_{\frac{i}{d}}^{(1)}(u, v, l) = p_{\frac{i}{d}}(u, v) f_{\frac{i}{d}}(u, v, l),$$

where for $i = 0, \ldots, d$ we have $d+1$ Markov Transition matrices $p_{\frac{i}{d}}(u,v)$, and $d+1$ sojourn time distributions $f_{\frac{i}{d}}(u,v,l)$, where d is the polynomial degree.

Thus, the drifting semi-Markov kernel will be equal to:

$$q_{\frac{t}{n}}^{(1)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, q_{\frac{i}{d}}^{(1)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, p_{\frac{i}{d}}(u,v) f_{\frac{i}{d}}(u,v,l)$$

Model 2

In this case, p is drifting and f **is not drifting**. Therefore, the semi-Markov kernels $q_{\frac{i}{d}}^{(2)}$ are equal to:

$$q_{\frac{i}{d}}^{(2)}(u,v,l) = p_{\frac{i}{d}}(u,v) f(u,v,l).$$

Thus, the drifting semi-Markov kernel will be equal to:

$$q_{\frac{t}{n}}^{(2)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, q_{\frac{i}{d}}^{(2)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, p_{\frac{i}{d}}(u,v) f(u,v,l)$$

Model 3

In this case, f is drifting and p **is not drifting**.

Therefore, the semi-Markov kernels $q_{\frac{i}{d}}^{(3)}$ are now described as:

$$q_{\frac{i}{d}}^{(3)}(u,v,l) = p(u,v) f_{\frac{i}{d}}(u,v,l).$$

Thus, the drifting semi-Markov kernel will be equal to:

$$q_{\frac{t}{n}}^{(3)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, q_{\frac{i}{d}}^{(3)}(u,v,l) = \sum_{i=0}^{d} A_i(t)\, p(u,v) f_{\frac{i}{d}}(u,v,l)$$

Value

An array with dimensions of $s \times s \times k_{max} \times (n+1)$, giving the value of the drifting semi-Markov kernel $q_{\frac{t}{n}}(u,v,l)$ for the corresponding (u,v,l,t). If any of u, v, l or t are specified, we obtain the element of the array for their given value.

See Also

For the objects required to calculate this kernel: fit_dsmm, parametric_dsmm, nonparametric_dsmm.

For sequence simulation through this kernel: simulate.dsmm.

For the theoretical background of drifting semi-Markov models: dsmmR.

Examples

```
# Setup.
states <- c("Rouen", "Bucharest", "Samos", "Aigio", "Marseille")
emc <- create_sequence(states, probs = c(0.3, 0.1, 0.1, 0.3, 0.2))
obj_model_2 <- fit_dsmm(
    sequence = emc,
    states = states,
    degree = 3,
    f_is_drifting = FALSE,
    p_is_drifting = TRUE
)

# Get the kernel.
kernel_model_2 <- get_kernel(obj_model_2)
cat(paste0("If no further arguments are made, kernel has dimensions ",
           "for all u, v, l, t:\n",
           "(s, s, k_max, n + 1) = (",
           paste(dim(kernel_model_2), collapse = ", "), ")"))

# Specifying `t`.
kernel_model_2_t <- get_kernel(obj_model_2, t = 100)
# kernel_model_2_t[ , , , t = 100]
cat(paste0("If we specify t, the kernel has dimensions for ",
           "all the remaining u, v, l:\n(s, s, k_max) = (",
           paste(dim(kernel_model_2_t), collapse = ", "), ")"))

# Specifying `t` and `u`.
kernel_model_2_tu <- get_kernel(obj_model_2, t = 2, u = "Aigio")
# kernel_model_2_tu["Aigio", , , t = 2]
cat(paste0("If we specify t and u, the kernel has dimensions for ",
           "all the remaining v, l:\n(s, k_max) = (",
           paste(dim(kernel_model_2_tu), collapse = ", "), ")"))

# Specifying `t`, `u` and `v`.
kernel_model_2_tuv <- get_kernel(obj_model_2, t = 3,
                                 u = "Rouen", v = "Bucharest")
# kernel_model_2_tuv["Rouen", "Bucharest", , t = 3]
cat(paste0("If we specify t, u and v, the kernel has dimensions ",
           "for all l:\n(k_max) = (",
           paste(length(kernel_model_2_tuv), collapse = ", "), ")"))

# It is possible to ask for any valid combination of `u`, `v`, `l` and `t`.
```

---

is.dsmm *Check if an object has a valid* dsmm *class*

Description

Checks for the validity of the specified attributes and the inheritance of the S3 class dsmm. This class acts like a parent class for the classes dsmm_fit_nonparametric, dsmm_fit_parametric, dsmm_parametric and dsmm_nonparametric.

Usage

```
is.dsmm(obj)
```

Arguments

obj	Arbitrary R object.

Value

TRUE or FALSE.

See Also

is.dsmm_fit_nonparametric, is.dsmm_fit_nonparametric, is.dsmm_parametric, is.dsmm_nonparametric

---

is.dsmm_fit_nonparametric

Check if an object has a valid `dsmm_fit_nonparametric` *class*

Description

Checks for the validity of the specified attributes and the inheritance of the S3 class `dsmm_fit_nonparametric`. This class inherits methods from the parent class `dsmm`.

Usage

```
is.dsmm_fit_nonparametric(obj)
```

Arguments

obj	Arbitrary R object.

Value

TRUE or FALSE.

See Also

is.dsmm, is.dsmm_fit_parametric, is.dsmm_nonparametric, is.dsmm_parametric is.dsmm_fit_parametric *Check if an object has a valid* `dsmm_fit_parametric` *class*

Description

Checks for the validity of the specified attributes and the inheritance of the S3 class `dsmm_fit_parametric`. This class inherits methods from the parent class `dsmm`.

Usage

```
is.dsmm_fit_parametric(obj)
```

Arguments

`obj`	Arbitrary `R` object.

Value

TRUE or FALSE.

See Also

is.dsmm, is.dsmm_fit_nonparametric, is.dsmm_parametric, is.dsmm_nonparametric

---

is.dsmm_nonparametric *Check if an object has a valid* `dsmm_nonparametric` *class*

Description

Checks for the validity of the specified attributes and the inheritance of the S3 class `dsmm_nonparametric`. This class inherits methods from the parent class `dsmm`.

Usage

```
is.dsmm_nonparametric(obj)
```

Arguments

`obj`	Arbitrary `R` object.

Value

TRUE or FALSE.

See Also

is.dsmm, is.dsmm_fit_nonparametric, is.dsmm_fit_parametric, is.dsmm_parametric

is.dsmm_parametric — *Check if an object has a valid* dsmm_parametric *class*

Description

Checks for the validity of the specified attributes and the inheritance of the S3 class dsmm_parametric. This class inherits methods from the parent class dsmm.

Usage

```
is.dsmm_parametric(obj)
```

Arguments

obj	Arbitrary R object.

Value

TRUE or FALSE.

See Also

is.dsmm, is.dsmm_fit_parametric, is.dsmm_fit_nonparametric, is.dsmm_nonparametric

lambda — *lambda genome*

Description

Contains the complete genome of the Escherichia phage Lambda.

Usage

```
data("lambda", package = "dsmmR")
data(lambda, package = "dsmmR") # equivalent.
# The following requires the package to be loaded,
# e.g. through `library(dsmmR)`.
data("lambda")
data(lambda)
```

Format

A vector object of type "character" and length of 48502. It has class of "Rdata".

References

Sanger, F., Coulson, A. R., Hong, G. F., Hill, D. F., & Petersen, G. B. (1982). Nucleotide sequence of bacteriophage λ DNA. Journal of molecular biology, 162(4), 729-773.

See Also

data

Examples

```
data("lambda", package = "dsmmR")
class(lambda)
sequence <- c(lambda) # Convert to "character" class
str(sequence)
```

---

nonparametric_dsmm *Non-parametric Drifting semi-Markov model specification*

Description

Creates a non-parametric model specification for a drifting semi-Markov model. Returns an object of class (`dsmm_nonparametric, dsmm`).

Usage

```
nonparametric_dsmm(
  model_size,
  states,
  initial_dist,
  degree,
  k_max,
  f_is_drifting,
  p_is_drifting,
  p_dist,
  f_dist
)
```

Arguments

`model_size`	Positive integer that represents the size of the drifting semi-Markov model n. It is equal to the length of a theoretical embedded Markov chain $(J_t)_{t \in \{0,\dots,n\}}$, without the last state.
`states`	Character vector that represents the state space E . It has length equal to $s = \|E\|$.
`initial_dist`	Numerical vector of s probabilities, that represents the initial distribution for each state in the state space E.

degree — Positive integer that represents the polynomial degree d for the drifting semi-Markov model.

k_max — Positive integer that represents the maximum sojourn time of choice, for the drifting semi-Markov model.

f_is_drifting — Logical. Specifies if f is drifting or not.

p_is_drifting — Logical. Specifies if p is drifting or not.

p_dist — Numerical array, that represents the probabilities of the transition matrix p of the embedded Markov chain $(J_t)_{t \in \{0,\dots,n\}}$ (it is defined the same way in the parametric_dsmm function). It can be defined in two ways:

- If p **is not** drifting, it has dimensions of $s \times s$.
- If p **is** drifting, it has dimensions of $s \times s \times (d+1)$ (see more in *Details, Defined Arguments*.)

f_dist — Numerical array, that represents the probabilities of the conditional sojourn time distributions f. 0 is allowed for state transitions that we do not wish to have a sojourn time distribution (e.g. all state transitions to the same state should have 0 as their value). It can be defined in two ways:

- If f **is not** drifting, it has dimensions of $s \times s \times k_{max}$.
- If f **is** drifting, it has dimensions of $s \times s \times k_{max} \times (d+1)$ (see more in *Details, Defined Arguments*.)

Details

Defined Arguments

For the non-parametric case, we explicitly define:

1. The *transition matrix* of the embedded Markov chain $(J_t)_{t \in \{0,\dots,n\}}$, given in the attribute p_dist:
   - If p **is not drifting**, it contains the values:
   $$p(u, v), \forall u, v \in E,$$
   given in an array with dimensions of $s \times s$, where the first dimension corresponds to the previous state u and the second dimension corresponds to the current state v.
   - If p **is drifting** then, for $i \in \{0, \dots, d\}$, it contains the values:
   $$p_{\frac{i}{d}}(u, v), \forall u, v \in E,$$
   given in an array with dimensions of $s \times s \times (d+1)$, where the first and second dimensions are defined as in the non-drifting case, and the third dimension corresponds to the $d+1$ different matrices $p_{\frac{i}{d}}$.
2. The *conditional sojourn time distribution*, given in the attribute f_dist:
   - If f **is not drifting**, it contains the values:
   $$f(u, v, l), \forall u, v \in E, \forall l \in \{1, \dots, k_{max}\},$$
   given in an array with dimensions of $s \times s \times k_{max}$, where the first dimension corresponds to the previous state u, the second dimension corresponds to the current state v, and the third dimension correspond to the sojourn time l.

- If f **is drifting** then, for $i \in \{0, \dots, d\}$, it contains the values:

$$f_{\frac{i}{d}}(u, v, l), \forall u, v \in E, \forall l \in \{1, \dots, k_{max}\},$$

given in an array with dimensions of $s \times s \times k_{max} \times (d + 1)$, where the first, second and third dimensions are defined as in the non-drifting case, and the fourth dimension corresponds to the $d + 1$ different arrays $f_{\frac{i}{d}}$.

Value

Returns an object of the S3 class `dsmm_nonparametric,dsmm`.

- `dist` : List. Contains 2 arrays, passing down from the arguments:
  - `p_drift` or `p_notdrift`, corresponding to whether the defined p transition matrix is drifting or not.
  - `f_drift` or `f_notdrift`, corresponding to whether the defined f sojourn time distribution is drifting or not.
- `initial_dist` : Numerical vector. Passing down from the arguments. It contains the initial distribution of the drifting semi-Markov model.
- `states` : Character vector. Passing down from the arguments. It contains the state space E.
- `s` : Positive integer. It contains the number of states in the state space, $s = |E|$, which is given in the attribute `states`.
- `degree` : Positive integer. Passing down from the arguments. It contains the polynomial degree d considered for the drifting of the model.
- `k_max` : Numerical value. Passing down from the arguments. It contains the maximum sojourn time, for the drifting semi-Markov model.
- `model_size` : Positive integer. Passing down from the arguments. It contains the size of the drifting semi-Markov model n, which represents the length of the embedded Markov chain $(J_t)_{t \in \{0, \dots, n\}}$, without the last state.
- `f_is_drifting` : Logical. Passing down from the arguments. Specifies if f is drifting or not.
- `p_is_drifting` : Logical. Passing down from the arguments. Specifies if p is drifting or not.
- `Model` : Character. Possible values:
  - `"Model_1"` : Both p and f are drifting.
  - `"Model_2"` : p is drifting and f is not drifting.
  - `"Model_3"` : f is drifting and p is not drifting.
- `A_i` : Numerical Matrix. Represents the polynomials $A_i(t)$ with degree d that are used for solving the system $MJ = P$. Used for the methods defined for the object. Not printed when viewing the object.

References

V. S. Barbu, N. Limnios. (2008). semi-Markov Chains and Hidden semi-Markov Models Toward Applications - Their Use in Reliability and DNA Analysis. New York: Lecture Notes in Statistics, vol. 191, Springer.

Vergne, N. (2008). Drifting Markov models with Polynomial Drift and Applications to DNA Sequences. Statistical Applications in Genetics Molecular Biology 7 (1).

Barbu V. S., Vergne, N. (2019). Reliability and survival analysis for drifting Markov models: modeling and estimation. Methodology and Computing in Applied Probability, 21(4), 1407-1429.

See Also

Methods applied to this object: simulate.dsmm, get_kernel.

For the parametric drifting semi-Markov model specification: parametric_dsmm.

For the theoretical background of drifting semi-Markov models: dsmmR.

Examples

```
# Setup.
states <- c("AA", "AC", "CC")
s <- length(states)
d <- 2
k_max <- 3

# ===========================================================================
# Defining non-parametric drifting semi-Markov models.
# ===========================================================================

# ---------------------------------------------------------------------------
# Defining distributions for Model 1 - both p and f are drifting.
# ---------------------------------------------------------------------------

# `p_dist` has dimensions of: (s, s, d + 1).
# Sums over v must be 1 for all u and i = 0, ..., d.
p_dist_1 <- matrix(c(0,   0.1, 0.9,
                     0.5, 0,   0.5,
                     0.3, 0.7, 0),
                   ncol = s, byrow = TRUE)

p_dist_2 <- matrix(c(0,   0.6, 0.4,
                     0.7, 0,   0.3,
                     0.6, 0.4, 0),
                   ncol = s, byrow = TRUE)

p_dist_3 <- matrix(c(0,   0.2, 0.8,
                     0.6, 0,   0.4,
                     0.7, 0.3, 0),
                   ncol = s, byrow = TRUE)

# Get `p_dist` as an array of p_dist_1, p_dist_2 and p_dist_3.
p_dist <- array(c(p_dist_1, p_dist_2, p_dist_3),
                dim = c(s, s, d + 1))

# `f_dist` has dimensions of: (s, s, k_max, d + 1).
# First f distribution. Dimensions: (s, s, k_max).
# Sums over l must be 1, for every u, v and i = 0, ..., d.
f_dist_1_l_1 <- matrix(c(0,   0.2, 0.7,
                         0.3, 0,   0.4,
                         0.2, 0.8, 0),
                       ncol = s, byrow = TRUE)

f_dist_1_l_2 <- matrix(c(0,   0.3,  0.2,
```

```
                         0.2, 0,    0.5,
                         0.1, 0.15, 0),
                       ncol = s, byrow = TRUE)

f_dist_1_l_3 <- matrix(c(0,   0.5,  0.1,
                         0.5, 0,    0.1,
                         0.7, 0.05, 0),
                       ncol = s, byrow = TRUE)
# Get f_dist_1
f_dist_1 <- array(c(f_dist_1_l_1, f_dist_1_l_2, f_dist_1_l_3),
                  dim = c(s, s, k_max))

# Second f distribution. Dimensions: (s, s, k_max)
f_dist_2_l_1 <- matrix(c(0,   1/3, 0.4,
                         0.3, 0,   0.4,
                         0.2, 0.1, 0),
                       ncol = s, byrow = TRUE)

f_dist_2_l_2 <- matrix(c(0,   1/3, 0.4,
                         0.4, 0,   0.2,
                         0.3, 0.4, 0),
                       ncol = s, byrow = TRUE)

f_dist_2_l_3 <- matrix(c(0,   1/3, 0.2,
                         0.3, 0,   0.4,
                         0.5, 0.5, 0),
                       ncol = s, byrow = TRUE)

# Get f_dist_2
f_dist_2 <- array(c(f_dist_2_l_1, f_dist_2_l_2, f_dist_2_l_3),
                  dim = c(s, s, k_max))

# Third f distribution. Dimensions: (s, s, k_max)
f_dist_3_l_1 <- matrix(c(0,    0.3, 0.3,
                         0.3,  0,   0.5,
                         0.05, 0.1, 0),
                       ncol = s, byrow = TRUE)

f_dist_3_l_2 <- matrix(c(0,   0.2, 0.6,
                         0.3, 0,   0.35,
                         0.9, 0.2, 0),
                       ncol = s, byrow = TRUE)

f_dist_3_l_3 <- matrix(c(0,    0.5, 0.1,
                         0.4,  0,   0.15,
                         0.05, 0.7, 0),
                       ncol = s, byrow = TRUE)

# Get f_dist_3
f_dist_3 <- array(c(f_dist_3_l_1, f_dist_3_l_2, f_dist_3_l_3),
                  dim = c(s, s, k_max))

# Get f_dist as an array of f_dist_1, f_dist_2 and f_dist_3.
```

```
f_dist <- array(c(f_dist_1, f_dist_2, f_dist_3),
                dim = c(s, s, k_max, d + 1))

# ---------------------------------------------------------------------------
# Non-Parametric object for Model 1.
# ---------------------------------------------------------------------------

obj_nonpar_model_1 <- nonparametric_dsmm(
    model_size = 8000,
    states = states,
    initial_dist = c(0.3, 0.5, 0.2),
    degree = d,
    k_max = k_max,
    p_dist = p_dist,
    f_dist = f_dist,
    p_is_drifting = TRUE,
    f_is_drifting = TRUE
)

# p drifting array.
p_drift <- obj_nonpar_model_1$dist$p_drift
p_drift

# f distribution.
f_drift <- obj_nonpar_model_1$dist$f_drift
f_drift

# ---------------------------------------------------------------------------
# Defining Model 2 - p is drifting, f is not drifting.
# ---------------------------------------------------------------------------

# p_dist has the same dimensions as in Model 1: (s, s, d + 1).
p_dist_model_2 <- array(c(p_dist_1, p_dist_2, p_dist_3),
                        dim = c(s, s, d + 1))

# f_dist has dimensions of: (s,s,k_{max}).
f_dist_model_2 <- f_dist_2


# ---------------------------------------------------------------------------
# Non-Parametric object for Model 2.
# ---------------------------------------------------------------------------

obj_nonpar_model_2 <- nonparametric_dsmm(
    model_size = 10000,
    states = states,
    initial_dist = c(0.7, 0.1, 0.2),
    degree = d,
    k_max = k_max,
    p_dist = p_dist_model_2,
    f_dist = f_dist_model_2,
    p_is_drifting = TRUE,
    f_is_drifting = FALSE
```

```
)

# p drifting array.
p_drift <- obj_nonpar_model_2$dist$p_drift
p_drift

# f distribution array.
f_notdrift <- obj_nonpar_model_2$dist$f_notdrift
f_notdrift


# ---------------------------------------------------------------------------
# Defining Model 3 - f is drifting, p is not drifting.
# ---------------------------------------------------------------------------


# `p_dist` has dimensions of: (s, s, d + 1).
p_dist_model_3 <- p_dist_3


# `f_dist` has the same dimensions as in Model 1: (s, s, d + 1).
f_dist_model_3 <- array(c(f_dist_1, f_dist_2, f_dist_3),
                        dim = c(s, s, k_max, d + 1))


# ---------------------------------------------------------------------------
# Non-Parametric object for Model 3.
# ---------------------------------------------------------------------------

obj_nonpar_model_3 <- nonparametric_dsmm(
    model_size = 10000,
    states = states,
    initial_dist = c(0.3, 0.4, 0.3),
    degree = d,
    k_max = k_max,
    p_dist = p_dist_model_3,
    f_dist = f_dist_model_3,
    p_is_drifting = FALSE,
    f_is_drifting = TRUE
)

# p distribution matrix.
p_notdrift <- obj_nonpar_model_3$dist$p_notdrift
p_notdrift

# f distribution array.
f_drift <- obj_nonpar_model_3$dist$f_drift
f_drift

# ===========================================================================
# Using methods for non-parametric objects.
# ===========================================================================
```

```
kernel_parametric <- get_kernel(obj_nonpar_model_3)
str(kernel_parametric)

sim_seq_par <- simulate(obj_nonpar_model_3, nsim = 50)
str(sim_seq_par)
```

`parametric_dsmm` *Parametric Drifting semi-Markov model specification*

Description

Creates a parametric model specification for a drifting semi-Markov model. Returns an object of class `(dsmm_parametric, dsmm)`.

Usage

```
parametric_dsmm(
  model_size,
  states,
  initial_dist,
  degree,
  f_is_drifting,
  p_is_drifting,
  p_dist,
  f_dist,
  f_dist_pars
)
```

Arguments

`model_size`	Positive integer that represents the size of the drifting semi-Markov model n. It is equal to the length of a theoretical embedded Markov chain $(J_t)_{t\in\{0,\ldots,n\}}$, without the last state.
`states`	Character vector that represents the state space E. It has length equal to $s = \|E\|$.
`initial_dist`	Numerical vector of s probabilities, that represents the initial distribution for each state in the state space E.
`degree`	Positive integer that represents the polynomial degree d for the drifting semi-Markov model.
`f_is_drifting`	Logical. Specifies if f is drifting or not.
`p_is_drifting`	Logical. Specifies if p is drifting or not.
`p_dist`	Numerical array, that represents the probabilities of the transition matrix p of the embedded Markov chain $(J_t)_{t\in\{0,\ldots,n\}}$ (it is defined the same way in the nonparametric_dsmm function). It can be defined in two ways: • If p **is not** drifting, it has dimensions of $s \times s$.

- If p **is** drifting, it has dimensions of $s \times s \times (d+1)$ (see more in *Details, Defined Arguments*.)

`f_dist` Character array, that represents the discrete sojourn time distribution f of our choice. `NA` is allowed for state transitions that we do not wish to have a sojourn time distribution (e.g. all state transition to the same state should have `NA` as their value). The list of possible values is: `["unif", "geom", "pois", "dweibull", "nbinom", NA]`. It can be defined in two ways:

- If f **is not** drifting, it has dimensions of $s \times s$.
- If f **is** drifting, it has dimensions of $s \times s \times (d+1)$ (see more in *Details, Defined Arguments*.)

`f_dist_pars` Numerical array, that represents the parameters of the sojourn time distributions given in `f_dist`. `NA` is allowed, in the case that the distribution of our choice does not require a parameter. It can be defined in two ways:

- If f **is not** drifting, it has dimensions of $s \times s \times 2$, specifying **two** possible parameters required for the discrete distributions.
- If f **is** drifting, it has dimensions of $s \times s \times 2 \times (d+1)$, specifying **two** possible parameters required for the discrete distributions, but for every single one of the $i = 0, \ldots, d$ sojourn time distributions $f_{\frac{i}{d}}$ that are required. (see more in *Details, Defined Arguments*.)

Details

Defined Arguments

For the parametric case, we explicitly define:

1. The *transition matrix* of the embedded Markov chain $(J_t)_{t \in \{0,\ldots,n\}}$, given in the attribute `p_dist`:
   - If p **is not drifting**, it contains the values:
   $$p(u, v), \forall u, v \in E,$$
   given in an array with dimensions of $s \times s$, where the first dimension corresponds to the previous state u and the second dimension corresponds to the current state v.
   - If p **is drifting**, for $i \in \{0, \ldots, d\}$, it contains the values:
   $$p_{\frac{i}{d}}(u, v), \forall u, v \in E,$$
   given in an array with dimensions of $s \times s \times (d+1)$, where the first and second dimensions are defined as in the non-drifting case, and the third dimension corresponds to the $d+1$ different matrices $p_{\frac{i}{d}}$.
2. The *conditional sojourn time distribution*, given in the attribute `f_dist`:
   - If f **is not drifting**, it contains the discrete distribution *names* (as characters or `NA`), given in an array with dimensions of $s \times s$, where the first dimension corresponds to the previous state u, the second dimension corresponds to the current state v.
   - If f **is drifting**, it contains the discrete distribution *names* (as characters or `NA`) given in an array with dimensions of $s \times s \times (d+1)$, where the first and second dimensions are defined as in the non-drifting case, and the third dimension corresponds to the $d+1$ different arrays $f_{\frac{i}{d}}$.

3. The *conditional sojourn time distribution parameters*, given in the attribute `f_dist_pars`:
   - If f **is not drifting**, it contains the *numerical values* (or `NA`) of the corresponding distributions defined in `f_dist`, given in an array with dimensions of $s \times s$, where the first dimension corresponds to the previous state u, the second dimension corresponds to the current state v.
   - If f **is drifting**, it contains the *numerical values* (or `NA`) of the corresponding distributions defined in `f_dist`, given in an array with dimensions of $s \times s \times (d + 1)$, where the first and second dimensions are defined as in the non-drifting case, and the third dimension corresponds to the $d + 1$ different arrays $f_{\frac{i}{d}}$.

Sojourn time distributions

In this package, the available distributions for the modeling of the conditional sojourn times, of the drifting semi-Markov model, used through the argument `f_dist`, are the following:

- Uniform (n):
  $f(x) = 1/n$, for $x = 1, 2, \ldots, n$, where n is a positive integer. This can be specified through the following:
  - `f_dist` = `"unif"`
  - `f_dist_pars` = (n, `NA`) (n as defined here).
- Geometric (p):
  $f(x) = p(1 - p)^{x-1}$, for $x = 1, 2, \ldots,$ where $p \in (0, 1)$ is the probability of success. This can be specified through the following:
  - `f_dist` = `"geom"`
  - `f_dist_pars` = (p, `NA`) (p as defined here).
- Poisson (λ):
  $f(x) = \frac{\lambda^{x-1} exp(-\lambda)}{(x-1)!}$, for $x = 1, 2, \ldots,$ where $\lambda > 0$. This can be specified through the following:
  - `f_dist` = `"pois"`
  - `f_dist_pars` = (λ, `NA`)
- Negative binomial (α, p):
  $f(x) = \frac{\Gamma(x+\alpha-1)}{\Gamma(\alpha)(x-1)!} p^{\alpha}(1 - p)^{x-1}$, for $x = 1, 2, \ldots,$ where Γ is the Gamma function, $\alpha \in (0, +\infty)$ is the parameter describing the target for number of successful trials, or the dispersion parameter (the shape parameter of the gamma mixing distribution). p is the probability of success, $0 < p < 1$.
  - `f_dist` = `"nbinom"`
  - `f_dist_pars` = (α, p) (p as defined here)
- Discrete Weibull of type 1 (q, β):
  $f(x) = q^{(x-1)^{\beta}} - q^{x^{\beta}}$, for $x = 1, 2, \ldots,$ with $q \in (0, 1)$ is the first parameter (probability) and $\beta \in (0, +\infty)$ is the second parameter. This can be specified through the following:
  - `f_dist` = `"dweibull"`
  - `f_dist_pars` = (q, β) (q as defined here)

From these discrete distributions, by using `"dweibull", "nbinom"` we require two parameters. It's for this reason that the attribute `f_dist_pars` is an array of dimensions $s \times s \times 2$ if f **is not drifting** or $s \times s \times 2 \times (d + 1)$ if f **is drifting**.

Value

Returns an object of the S3 class dsmm_parametric, dsmm. It has the following attributes:

- dist : List. Contains 3 arrays, passing down from the arguments:
  - p_drift or p_notdrift, corresponding to whether the defined p transition matrix is drifting or not.
  - f_drift_parametric or f_notdrift_parametric, corresponding to whether the defined f sojourn time distribution is drifting or not.
  - f_drift_parameters or f_notdrift_parameters, which are the defined f sojourn time distribution parameters, depending on whether f is drifting or not.
- initial_dist : Numerical vector. Passing down from the arguments. It contains the initial distribution of the drifting semi-Markov model.
- states : Character vector. Passing down from the arguments. It contains the state space E.
- s : Positive integer. It contains the number of states in the state space, $s = |E|$, which is given in the attribute states.
- degree : Positive integer. Passing down from the arguments. It contains the polynomial degree d considered for the drifting of the model.
- model_size : Positive integer. Passing down from the arguments. It contains the size of the drifting semi-Markov model n, which represents the length of the embedded Markov chain $(J_t)_{t\in\{0,\dots,n\}}$, without the last state.
- f_is_drifting : Logical. Passing down from the arguments. Specifies if f is drifting or not.
- p_is_drifting : Logical. Passing down from the arguments. Specifies if p is drifting or not.
- Model : Character. Possible values:
  - "Model_1" : Both p and f are drifting.
  - "Model_2" : p is drifting and f is not drifting.
  - "Model_3" : f is drifting and p is not drifting.
- A_i : Numerical matrix. Represents the polynomials $A_i(t)$ with degree d that are used for solving the system $MJ = P$. Used for the methods defined for the object. Not printed when viewing the object.

References

V. S. Barbu, N. Limnios. (2008). semi-Markov Chains and Hidden semi-Markov Models Toward Applications - Their Use in Reliability and DNA Analysis. New York: Lecture Notes in Statistics, vol. 191, Springer.

Vergne, N. (2008). Drifting Markov models with Polynomial Drift and Applications to DNA Sequences. Statistical Applications in Genetics Molecular Biology 7 (1).

Barbu V. S., Vergne, N. (2019). Reliability and survival analysis for drifting Markov models: modeling and estimation. Methodology and Computing in Applied Probability, 21(4), 1407-1429.

T. Nakagawa and S. Osaki. (1975). The discrete Weibull distribution. IEEE Transactions on Reliability, R-24, 300-301.

See Also

Methods applied to this object: simulate.dsmm, get_kernel.

For the non-parametric drifting semi-Markov model specification: nonparametric_dsmm.

For the theoretical background of drifting semi-Markov models: dsmmR.

Examples

```
# We can also define states in a flexible way, including spaces.
states <- c("Dollar $", " /1'2'3/ ", " Z E T A ", "O_M_E_G_A")
s <- length(states)
d <- 1

# ===========================================================================
# Defining parametric drifting semi-Markov models.
# ===========================================================================

# ---------------------------------------------------------------------------
# Defining the drifting distributions for Model 1.
# ---------------------------------------------------------------------------

# `p_dist` has dimensions of: (s, s, d + 1).
# Sums over v must be 1 for all u and i = 0, ..., d.

# First matrix.
p_dist_1 <- matrix(c(0,   0.1, 0.4, 0.5,
                     0.5, 0,   0.3, 0.2,
                     0.3, 0.4, 0,   0.3,
                     0.8, 0.1, 0.1, 0),
                   ncol = s, byrow = TRUE)

# Second matrix.
p_dist_2 <- matrix(c(0,   0.3, 0.6, 0.1,
                     0.3, 0,   0.4, 0.3,
                     0.5, 0.3, 0,   0.2,
                     0.2, 0.3, 0.5, 0),
                   ncol = s, byrow = TRUE)

# get `p_dist` as an array of p_dist_1 and p_dist_2.
p_dist_model_1 <- array(c(p_dist_1, p_dist_2), dim = c(s, s, d + 1))

# `f_dist` has dimensions of: (s, s, d + 1).
# First matrix.
f_dist_1 <- matrix(c(NA,         "unif", "dweibull", "nbinom",
                     "geom",       NA,    "pois",     "dweibull",
                     "dweibull", "pois",  NA,         "geom",
                     "pois",       NA,    "geom",      NA),
                   nrow = s, ncol = s, byrow = TRUE)


# Second matrix.
f_dist_2 <- matrix(c(NA,     "pois", "geom", "nbinom",
```

```
                     "geom",  NA,    "pois", "dweibull",
                     "unif", "geom",  NA,    "geom",
                     "pois", "pois", "geom",  NA),
                   nrow = s, ncol = s, byrow = TRUE)


# get `f_dist` as an array of `f_dist_1` and `f_dist_2`
f_dist_model_1 <- array(c(f_dist_1, f_dist_2), dim = c(s, s, d + 1))


# `f_dist_pars` has dimensions of: (s, s, 2, d + 1).
# First array of coefficients, corresponding to `f_dist_1`.
# First matrix.
f_dist_1_pars_1 <- matrix(c(NA,  5,  0.4, 4,
                            0.7, NA, 5,   0.6,
                            0.2, 3,  NA,  0.6,
                            4,   NA, 0.4, NA),
                          nrow = s, ncol = s, byrow = TRUE)
# Second matrix.
f_dist_1_pars_2 <- matrix(c(NA,  NA, 0.2, 0.6,
                            NA,  NA, NA,  0.8,
                            0.6, NA, NA,  NA,
                            NA,  NA, NA,  NA),
                          nrow = s, ncol = s, byrow = TRUE)

# Second array of coefficients, corresponding to `f_dist_2`.
# First matrix.
f_dist_2_pars_1 <- matrix(c(NA,  6,   0.4, 3,
                            0.7, NA,  2,   0.5,
                            3,   0.6, NA,  0.7,
                            6,   0.2, 0.7, NA),
                          nrow = s, ncol = s, byrow = TRUE)
# Second matrix.
f_dist_2_pars_2 <- matrix(c(NA, NA, NA, 0.6,
                            NA, NA, NA, 0.8,
                            NA, NA, NA, NA,
                            NA, NA, NA, NA),
                          nrow = s, ncol = s, byrow = TRUE)

# Get `f_dist_pars`.
f_dist_pars_model_1 <- array(c(f_dist_1_pars_1, f_dist_1_pars_2,
                               f_dist_2_pars_1, f_dist_2_pars_2),
                             dim = c(s, s, 2, d + 1))

# ---------------------------------------------------------------------------
# Parametric object for Model 1.
# ---------------------------------------------------------------------------

obj_par_model_1 <- parametric_dsmm(
    model_size = 10000,
    states = states,
    initial_dist = c(0.8, 0.1, 0.1, 0),
    degree = d,
```

```
    p_dist = p_dist_model_1,
    f_dist = f_dist_model_1,
    f_dist_pars = f_dist_pars_model_1,
    p_is_drifting = TRUE,
    f_is_drifting = TRUE
)

# p drifting array.
p_drift <- obj_par_model_1$dist$p_drift
p_drift

# f distribution.
f_dist_drift <- obj_par_model_1$dist$f_drift_parametric
f_dist_drift

# parameters for the f distribution.
f_dist_pars_drift <- obj_par_model_1$dist$f_drift_parameters
f_dist_pars_drift

# ---------------------------------------------------------------------------
# Defining Model 2 - p is drifting, f is not drifting.
# ---------------------------------------------------------------------------

# `p_dist` has the same dimensions as in Model 1: (s, s, d + 1).
p_dist_model_2 <- array(c(p_dist_1, p_dist_2), dim = c(s, s, d + 1))

# `f_dist` has dimensions of: (s, s).
f_dist_model_2 <- matrix(c( NA,       "pois",  NA,       "nbinom",
                            "geom",    NA,    "geom",    "dweibull",
                            "unif",   "geom",  NA,       "geom",
                            "nbinom", "unif", "dweibull", NA),
                         nrow = s, ncol = s, byrow = TRUE)

# `f_dist_pars` has dimensions of: (s, s, 2),
#  corresponding to `f_dist_model_2`.

# First matrix.
f_dist_pars_1_model_2 <- matrix(c(NA,  0.2, NA,  3,
                                  0.2, NA,  0.2, 0.5,
                                  3,   0.4, NA,  0.7,
                                  2,   3,   0.7, NA),
                                nrow = s, ncol = s, byrow = TRUE)

# Second matrix.
f_dist_pars_2_model_2 <- matrix(c(NA,  NA, NA,  0.6,
                                  NA,  NA, NA,  0.8,
                                  NA,  NA, NA,  NA,
                                  0.2, NA, 0.3, NA),
                                nrow = s, ncol = s, byrow = TRUE)


# Get `f_dist_pars`.
f_dist_pars_model_2 <- array(c(f_dist_pars_1_model_2,
```

```
                              f_dist_pars_2_model_2),
                            dim = c(s, s, 2))


# ---------------------------------------------------------------------------
# Parametric object for Model 2.
# ---------------------------------------------------------------------------

obj_par_model_2 <- parametric_dsmm(
    model_size = 10000,
    states = states,
    initial_dist = c(0.8, 0.1, 0.1, 0),
    degree = d,
    p_dist = p_dist_model_2,
    f_dist = f_dist_model_2,
    f_dist_pars = f_dist_pars_model_2,
    p_is_drifting = TRUE,
    f_is_drifting = FALSE
)

# p drifting array.
p_drift <- obj_par_model_2$dist$p_drift
p_drift

# f distribution.
f_dist_notdrift <- obj_par_model_2$dist$f_notdrift_parametric
f_dist_notdrift

# parameters for the f distribution.
f_dist_pars_notdrift <- obj_par_model_2$dist$f_notdrift_parameters
f_dist_pars_notdrift

# ---------------------------------------------------------------------------
# Defining Model 3 - f is drifting, p is not drifting.
# ---------------------------------------------------------------------------

# `p_dist` has dimensions of: (s, s).
p_dist_model_3 <- matrix(c(0,   0.1,  0.3,  0.6,
                           0.4, 0,    0.1,  0.5,
                           0.4, 0.3,  0,    0.3,
                           0.9, 0.01, 0.09, 0),
                         ncol = s, byrow = TRUE)

# `f_dist` has the same dimensions as in Model 1: (s, s, d + 1).
f_dist_model_3 <- array(c(f_dist_1, f_dist_2), dim = c(s, s, d + 1))


# `f_dist_pars` has the same dimensions as in Model 1: (s, s, 2, d + 1).
f_dist_pars_model_3 <- array(c(f_dist_1_pars_1, f_dist_1_pars_2,
                              f_dist_2_pars_1, f_dist_2_pars_2),
                            dim = c(s, s, 2, d + 1))

# ---------------------------------------------------------------------------
```

```
# Parametric object for Model 3.
# ---------------------------------------------------------------------------

obj_par_model_3 <- parametric_dsmm(
    model_size = 10000,
    states = states,
    initial_dist = c(0.3, 0.2, 0.2, 0.3),
    degree = d,
    p_dist = p_dist_model_3,
    f_dist = f_dist_model_3,
    f_dist_pars = f_dist_pars_model_3,
    p_is_drifting = FALSE,
    f_is_drifting = TRUE
)

# p drifting array.
p_notdrift <- obj_par_model_3$dist$p_notdrift
p_notdrift

# f distribution.
f_dist_drift <- obj_par_model_3$dist$f_drift_parametric
f_dist_drift

# parameters for the f distribution.
f_dist_pars_drift <- obj_par_model_3$dist$f_drift_parameters
f_dist_pars_drift

# ===========================================================================
# Parametric estimation using methods corresponding to an object
#     which inherits from the class `dsmm_parametric`.
# ===========================================================================

### Comments
### 1.  Using a larger `klim` and a larger `model_size` will increase the
###     accuracy of the model, with the need of larger memory requirements
###     and computational cost.
### 2.  For the parametric estimation it is recommended to use a common set
###     of distributions while only the parameters are drifting. This results
###     in higher accuracy.


# ---------------------------------------------------------------------------
# Defining the distributions for Model 1 - both p and f are drifting.
# ---------------------------------------------------------------------------

# `p_dist` has dimensions of: (s, s, d + 1).
# First matrix.
p_dist_1 <- matrix(c(0,   0.2, 0.4, 0.4,
                     0.5, 0,   0.3, 0.2,
                     0.3, 0.4, 0,   0.3,
                     0.5, 0.3, 0.2, 0),
                   ncol = s, byrow = TRUE)
```

```
# Second matrix.
p_dist_2 <- matrix(c(0,   0.3, 0.5, 0.2,
                     0.3, 0,   0.4, 0.3,
                     0.5, 0.3, 0,   0.2,
                     0.2, 0.4, 0.4, 0),
                   ncol = s, byrow = TRUE)

# get `p_dist` as an array of p_dist_1 and p_dist_2.
p_dist_model_1 <- array(c(p_dist_1, p_dist_2), dim = c(s, s, d + 1))

# `f_dist` has dimensions of: (s, s, d + 1).
# We will use the same sojourn time distributions.
f_dist_1 <- matrix(c( NA,        "unif",   "dweibull", "nbinom",
                     "geom",      NA,      "pois",     "dweibull",
                     "dweibull", "pois",    NA,        "geom",
                     "pois",     'nbinom', "geom",      NA),
                   nrow = s, ncol = s, byrow = TRUE)

# get `f_dist`
f_dist_model_1 <- array(f_dist_1, dim = c(s, s, d + 1))

# `f_dist_pars` has dimensions of: (s, s, 2, d + 1).
# First array of coefficients, corresponding to `f_dist_1`.
# First matrix.
f_dist_1_pars_1 <- matrix(c(NA,  7,  0.4, 4,
                            0.7, NA, 5,   0.6,
                            0.2, 3,  NA,  0.6,
                            4,   4,  0.4, NA),
                          nrow = s, ncol = s, byrow = TRUE)
# Second matrix.
f_dist_1_pars_2 <- matrix(c(NA,  NA,  0.2, 0.6,
                            NA,  NA,  NA,  0.8,
                            0.6, NA,  NA,  NA,
                            NA,  0.3, NA,  NA),
                          nrow = s, ncol = s, byrow = TRUE)

# Second array of coefficients, corresponding to `f_dist_2`.
# First matrix.
f_dist_2_pars_1 <- matrix(c(NA,  6,  0.5, 3,
                            0.5, NA, 4,   0.5,
                            0.4, 5,  NA,  0.7,
                            6,   5,  0.7, NA),
                          nrow = s, ncol = s, byrow = TRUE)
# Second matrix.
f_dist_2_pars_2 <- matrix(c(NA,  NA,  0.4, 0.5,
                            NA,  NA,  NA,  0.6,
                            0.5, NA,  NA,  NA,
                            NA,  0.4, NA,  NA),
                          nrow = s, ncol = s, byrow = TRUE)
# Get `f_dist_pars`.
f_dist_pars_model_1 <- array(c(f_dist_1_pars_1, f_dist_1_pars_2,
                               f_dist_2_pars_1, f_dist_2_pars_2),
                             dim = c(s, s, 2, d + 1))
```

```
# ---------------------------------------------------------------------------
# Defining the parametric object for Model 1.
# ---------------------------------------------------------------------------

obj_par_model_1 <- parametric_dsmm(
    model_size = 4000,
    states = states,
    initial_dist = c(0.8, 0.1, 0.1, 0),
    degree = d,
    p_dist = p_dist_model_1,
    f_dist = f_dist_model_1,
    f_dist_pars = f_dist_pars_model_1,
    p_is_drifting = TRUE,
    f_is_drifting = TRUE
)

cat("The object has class of (",
    paste0(class(obj_par_model_1),
           collapse = ', '), ").")


# ---------------------------------------------------------------------------
# Generating a sequence from the parametric object.
# ---------------------------------------------------------------------------

# A larger klim will lead to an increase in accuracy.
klim <- 20
sim_seq <- simulate(obj_par_model_1, klim = klim, seed = 1)


# ---------------------------------------------------------------------------
# Fitting the generated sequence under the same distributions.
# ---------------------------------------------------------------------------

fit_par_model1 <- fit_dsmm(sequence = sim_seq,
                           states = states,
                           degree = d,
                           f_is_drifting = TRUE,
                           p_is_drifting = TRUE,
                           estimation = 'parametric',
                           f_dist = f_dist_model_1)

cat("The object has class of (",
    paste0(class(fit_par_model1),
           collapse = ', '), ").")

cat("\nThe estimated parameters are:\n")
fit_par_model1$dist$f_drift_parameters
```

simulate.dsmm *Simulate a sequence given a drifting semi-Markov kernel.*

Description

Generic function that simulates a sequence under the rule of a drifting semi-Markov kernel. The number of simulated states is `nsim`, while the kernel is retrieved from the object `obj` via inheritance from the S3 class `dsmm`.

Usage

```
## S3 method for class 'dsmm'
simulate(
  object,
  nsim = NULL,
  seed = NULL,
  max_seq_length = NULL,
  klim = 100,
  ...
)
```

Arguments

`object`	An object of S3 class `dsmm`, `dsmm_fit_nonparametric`, `dsmm_nonparametric`, `dsmm_fit_parametric` or `dsmm_parametric`.
`nsim`	Optional. An integer specifying the number of simulations to be made from the drifting semi-Markov kernel. The maximum value of `nsim` is the model size which is specified in `obj`. This is also the default value. We define a special case for `nsim = 0`, where only the initial distribution is considered and only the simulation of its sojourn time will be made, without the next state.
`seed`	Optional. An integer specifying the initialization of the random number generator.
`max_seq_length`	Optional. A positive integer that will ensure the simulated sequence will not have a *maximum total length* greater than `max_seq_length` (however, it is possible for the total length to be *less* than `max_seq_length`).
`klim`	Optional. Positive integer. Passed down to `get_kernel` for the parametric object, with class `dsmm_parametric`. Default value is 100.
`...`	Optional. Attributes passed down from the `simulate` method.

Value

Returns the simulated sequence for the given drifting semi-Markov model. It is a character vector based on `nsim` simulations, with a maximum length of `max_seq_length`.

This sequence is not to be confused with the embedded Markov chain. The user can apply the `base::rle()` function on this simulated sequence, if he wishes to obtain the corresponding embedded Markov chain and the sojourn times.

See Also

About random number generation in R: RNG.

Fitting a model through a sequence from this function: fit_dsmm.

For the theoretical background of drifting semi-Markov models: dsmmR.

For obtaining the lengths and values of equals values in a vector: rle.

Examples

```
# Setup.
sequence <- create_sequence("DNA", len = 1000)
states <- sort(unique(sequence))
d <- 1
obj_model_3 <- fit_dsmm(sequence = sequence,
                        states = states,
                        degree = d,
                        f_is_drifting = TRUE,
                        p_is_drifting = FALSE)

# Using the method `simulate.dsmm()`.
simulated_seq <- simulate(obj_model_3, seed = 1)
short_sim <- simulate(obj = obj_model_3, nsim = 10, seed = 1)
cut_sim <- simulate(obj = obj_model_3, max_seq_length = 10, seed = 1)
str(simulated_seq)
str(short_sim)
str(cut_sim)

# To obtain the embedded Markov chain (EMC) and the corresponding sojourn times
# of any simulated sequence, we can simply use the `base::rle()` function.

sim_seq_emc <- base::rle(cut_sim)$values # embedded Markov chain
sim_seq_sojourn_times <- base::rle(cut_sim)$lengths # sojourn times
cat("Start of the simulated sequence: ", head(cut_sim),
    "...\nThe embedded Markov chain:       ", head(sim_seq_emc),
    "...\nThe sojourn times:               ", head(sim_seq_sojourn_times), "...")
```

Index