Package 'FVDDPpkg'

May 7, 2026

Type Package

Title Implement Fleming-Viot-Dependent Dirichlet Processes

Version 0.1.2

Description A Bayesian Nonparametric model for the study of time-evolving frequencies, which has become renowned in the study of population genetics. The model consists of a Hidden Markov Model (HMM) in which the latent signal is a distribution-valued stochastic process that takes the form of a finite mixture of Dirichlet Processes, indexed by vectors that count how many times each value is observed in the population. The package implements methodologies presented in Ascolani, Lijoi and Ruggiero (2021) <doi:10.1214/20-BA1206> and Ascolani, Lijoi and Ruggiero (2023) <doi:10.3150/22-BEJ1504> that make it possible to study the process at the time of data collection or to predict its evolution in future or in the past.

License LGPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.3

Imports Rcpp, Rdpack

LinkingTo Rcpp

RdMacros Rdpack

Suggests rmarkdown, knitr

VignetteBuilder knitr

NeedsCompilation yes

Author Stefano Damato [aut, cre]

Maintainer Stefano Damato <stefano.damato@idsia.ch>

Repository CRAN

Date/Publication 2024-07-09 16:10:12 UTC

Contents

`approx.propagate` *Approximate the propagation of a Fleming-Viot latent signal*

Description

Approximate the propagation of a Fleming-Viot latent signal

Usage

```
approx.propagate(fvddp, delta.t, N)
```

Arguments

`fvddp`	An instance of class generated via `initialize()`. In order to perform the propagation, the FVDDP has to be fed some data using `update()`, at least once.
`delta.t`	The time of the propagation.
`N`	The amount of samples to be drawn in order to perform the approximation.

Value

A object of class `fvddp`. Since this function is a Monte-Carlo based approximation of `propagate()`, the outputs are similar.

References

Ascolani F, Lijoi A, Ruggiero M (2021). "Predictive inference with Fleming–Viot-driven dependent Dirichlet processes." *Bayesian Analysis*, **16**(2), 371 – 395. doi:10.1214/20BA1206.

See Also

`approx.propagate()` for a (slower) exact computation.

Examples

```
#a first example
FVDDP = initialize(theta = 1, sampling.f = function(x) rpois(x, 3),
                   density.f = function(x) dpois(x, 3), atomic = TRUE)
FVDDP = update(FVDDP, c(4,5))
approx.propagate(FVDDP, 0.2, 10000)

#another example; it does not matter wether P0 is atomic or not
FVDDP=initialize(theta = 3, function(x) rnorm(x, -1, 3),
                 function(x) dnorm(x, -1, 3), atomic = FALSE)
FVDDP = update(FVDDP, c(-1.145, 0.553, 0.553, 0.553))
approx.propagate(FVDDP, 0.6, 10000)
```

`approx.smooth` *Approximate the smoothing distribution of a Fleming-Viot latent signal*

Description

Approximate the smoothing distribution of a Fleming-Viot latent signal

Usage

```
approx.smooth(fvddp.past, fvddp.future, t.past, t.future, y.new, N)
```

Arguments

`fvddp.past`	An instance of class `fvddp`, progressively updated ad propagated with data referring to past times via `update()` and `propagate()` (or its approximate version, `approx.propagate()`).
`fvddp.future`	Same as `fvddp.past`, but in this case the propagation has been performed with time data from times later than the one to be estimated. Its hyperparameters must be equals to the ones of `fvddp.past`.
`t.past`	The time between the last collection of data (in the past) and the time at which the smoothing is performed.
`t.future`	Same as `t.past`, but in this case it is referred to the future. `t.future` is positive too.
`y.new`	The data collected at the time the smoothing is performed.
`N`	the amount of samples to be drawn in order to perform the approximation.

Value

An object of class `fvddp`, with the same hyperparmeters as `fvddp.past` and `fvddp.future`. Since this function is a Monte-Carlo based approximation of `smooth()`, the outputs are similar.

See Also

`smooth()` for a (slower) exact computation

Examples

```
FVDDP = initialize(3, function(x) rbinom(x, 10, .2),
                   function(x) dbinom(x, 10, .2), TRUE)
FVDDP.PAST = update(FVDDP, c(2,3))
FVDDP.FUTURE = update(FVDDP, c(4))
FVDDP.FUTURE = propagate(FVDDP.FUTURE, 0.5)
FVDDP.FUTURE = update(FVDDP.FUTURE, c(1))
approx.smooth(fvddp.past = FVDDP.PAST, fvddp.future = FVDDP.FUTURE,
              t.past = 0.4, t.future = 0.3, y.new = c(1,3), N = 20000)
```

`error.estimate` *Compare the performance of a Monte-Carlo estimate with respect to the exact result.*

Description

Compare the performance of a Monte-Carlo estimate with respect to the exact result.

Usage

```
error.estimate(fvddp.exact, fvddp.approx, remove.unmatched = FALSE)
```

Arguments

`fvddp.exact`	An instance of class `fvddp` obtained via smoothing (`smooth()`) or propagation (`propagate()`).
`fvddp.approx`	An instance of class `fvddp` obtained using the approximating algorithms for smoothing or propagation, with the same input as `fvddp.exact`.
`remove.unmatched`	Choose whether the weights associated to multiplicities that are in `fvddp.exact` but not in `fvddp.approx` should be removed in the computation (`TRUE`) or considered to be 0 (`FALSE`).

Value

A vector whose j-th element is the difference (in absolute value) between the weight of the j-th row of the matrix `M` of `fvddp.exact` and the weight of the row of the matrix `M` of `fvddp.approx` equal to it. The length depends on the value of `remove.unmathced`.

Examples

```
#iniialize the process
FVDDP = initialize(3, function(x) rgamma(x, 2,2),
                   function(x) dgamma(x, 2,2), FALSE)
FVDDP = update(FVDDP, c(rep(abs(rnorm(2,1, 4)), 2), rexp(2, 0.5)))
#perform n exact propagation and an approximate one
EXACT = propagate(FVDDP, 0.7)
APPROX = approx.propagate(FVDDP, 0.7, 10000)
#measure the error with this function
error.estimate(fvddp.exact = EXACT, fvddp.approx = APPROX, TRUE)

#in order to smoot, create and propagate the signal from the past and from the future
FVDDP=initialize(3, function(x) rbinom(x, 10, 0.2),
                 function(x) dbinom(x, 10, 0.2), TRUE)
FVDDP.PAST = update(FVDDP, c(2,3))
FVDDP.FUTURE = update(FVDDP, c(4))
FVDDP.FUTURE = propagate(FVDDP.FUTURE, 0.5)
FVDDP.FUTURE = update(FVDDP.FUTURE, c(1))
#compute an exact and an approximate smoothing
EXACT = smooth(FVDDP.PAST, FVDDP.FUTURE, 0.4, 0.3, c(1,3))
APPROX = approx.smooth(FVDDP.PAST, FVDDP.FUTURE, 0.4, 0.3, c(1,3), 20000)
#compute the error again
error.estimate(fvddp.exact = EXACT, fvddp.approx = APPROX)
```

`initialize` *Initialize Fleming-Viot dependent Dirichlet Processes by setting hyperparameters*

Description

Initialize Fleming-Viot dependent Dirichlet Processes by setting hyperparameters

Usage

```
initialize(theta, sampling.f, density.f, atomic)
```

Arguments

`theta`	The intensity of the centering measure, in the sense of Bayesian Nonparametrics.
`sampling.f`	A function to sample from the centering. Its unique argument must be the amount of values to be drawn.
`density.f`	A function to compute the value of the density function or mass function of the centering. It has to be consistent with `sampling.f`.
`atomic`	A boolean value stating whether the centering is atomic or not.

Value

A list containing the input (renamed as `theta`, `P0.sample`, `P0.density`, and `is.atomic`) and three empty slots that will contain the information once the FVDDP is updated with data. In particular, they are:

- `y.star`: a vector of unique values
- `M`: a matrix of multiplicities, represented as row vectors
- `w`: a vector of weights associated to each row of the matrix of multiplicities. Such list repesents a n object of the `fvddp` class.

References

Papaspiliopoulos O, Ruggiero M (2014). "Optimal filtering and the dual process." *Bernoulli*, **20**(4). doi:10.3150/13bej548.

Papaspiliopoulos O, Ruggiero M, Spanò D (2016). "Conjugacy properties of time-evolving Dirichlet and gamma random measures." *Electronic Journal of Statistics*, **10**(2), 3452 – 3489. doi:10.1214/16EJS1194.

Examples

```
#initiization with an atomic measure (Pois(3))
initialize(theta = 1, sampling.f = function(x) rpois(x, 3),
           density.f = function(x) dpois(x, 3), atomic = TRUE)

#initialization with a non-atomic measure (N(-1, 3))
initialize(theta = 3, sampling.f = function(x) rnorm(x, -1, 3),
           density.f = function(x) dnorm(x, -1, 3), atomic = FALSE)
```

---

polya.sample *Sampling via Polya Urn scheme*

Description

Sampling via Polya Urn scheme

Usage

```
polya.sample(n, theta, v = c(), sampling.f)
```

Arguments

`n`	The amount of samples to be drawn.
`theta`	The intensity, in the sense of Bayesian Statistics
`v`	A vector of values, considered to be already drawn from the Polya scheme.
`sampling.f`	A function to sample new values. Its unique argument must express the number of values to draw.

Value

A vector containing n values extracted.

Examples

```
polya.sample(10, 2, c(0,1), function(x) rbeta(x,1,1))
```

`posterior.sample` *Draw values from the posterior distribution FVDDP*

Description

Draw values from the posterior distribution FVDDP

Usage

```
posterior.sample(fvddp, N)
```

Arguments

`fvddp`	The instance of class `fvddp` the values are drawn from.
`N`	The amount of values to draw.

Value

A vector of length `N` of values drawn either from the centering of the FVDDP (the input) or from the empirical probability measure generated by past observations. The difference between this function and `predictive.struct()` is that in this case the process is not update with respect to any drawn value.

Examples

```
#create a dummy process and sample some values from it
FVDDP = initialize(7, function(x) rbeta(x, 3,3),
                   function(x) dgamma(x, 3,3), FALSE)
FVDDP = update(FVDDP, rep(0:1, 2))
posterior.sample(fvddp = FVDDP, N = 100)
```

`predictive.struct` *Use the predictive structure of the FVDDP to sequentially draw values adn update*

Description

Use the predictive structure of the FVDDP to sequentially draw values adn update

Usage

```
predictive.struct(fvddp, N)
```

Arguments

`fvddp`	The instance of class `fvddp` the values are drawn from.
`N`	The amount of values to draw.

Value

A vector of length `N` of values obtained using the predictive structure. Precisely, after that any observation is drawn (either from the centering measure or from past observations) the input `fvddp` is modified as if the function `update()` is called, with the new value as second argument.

References

Ascolani F, Lijoi A, Ruggiero M (2021). "Predictive inference with Fleming–Viot-driven dependent Dirichlet processes." *Bayesian Analysis*, **16**(2), 371 – 395. doi:10.1214/20BA1206.

Examples

```
#create a dumy process and expoit the predictive structure
FVDDP = initialize(7, function(x) rbeta(x, 3,3),
                   function(x) dgamma(x, 3,3), FALSE)
FVDDP = update(FVDDP, rep(0:1, 2))
predictive.struct(fvddp = FVDDP, N = 100)
```

`print.fvddp` *Print hyperparameters and values from Fleming-Viot Dependent Dirichlet Processes*

Description

Print hyperparameters and values from Fleming-Viot Dependent Dirichlet Processes

Usage

```
## S3 method for class 'fvddp'
print(x, ...)
```

Arguments

x	The fvddp object to be printed.
...	Optional arguments for summary methods.

Value

A list of the hyperparameters of the process, i.e. theta, P0.sample, Po.density, and is.atomic. Moreover, if the process is still empty, this will be printed; if otherwise it has been updated (via update()), then the values in y.star, M and w will be printed.

Examples

```
#a simple example
FVDDP = initialize(theta = 1, sampling.f = function(x) rpois(x, 3),
                   density.f = function(x) dpois(x, 3), atomic = TRUE)
FVDDP = update(FVDDP, c(4,5))
print(FVDDP)

#in case there are no data
FVDDP=initialize(theta = 3, function(x) rnorm(x, -1, 3),
                 function(x) dnorm(x, -1, 3), atomic = FALSE)
print(FVDDP)
```

---

propagate *Propagate the Fleming-Viot latent signal in time*

Description

Propagate the Fleming-Viot latent signal in time

Usage

```
propagate(fvddp, delta.t)
```

Arguments

fvddp	An instance of class generated via initialize(). In order to perform the propagation, the FVDDP has to be fed some data using update(), at least once.
delta.t	The non-negative time of the propagation. If 0, the returned process is the input.

Value

A list of the same class to the one given as an input (fvddp). The amount of rows of the matrix M, as well as the vector of weights, w, will increase. The hyperparameters will be the same.

References

Papaspiliopoulos O, Ruggiero M, Spanò D (2016). "Conjugacy properties of time-evolving Dirichlet and gamma random measures." *Electronic Journal of Statistics*, **10**(2), 3452 – 3489. doi:10.1214/16EJS1194.

See Also

`approx.propagate()` for a (faster) Monte-Carlo-based analogous.

Examples

```
FVDDP = initialize(1, function(x) rpois(x, 3),
                   function(x) dpois(x, 3), TRUE)
FVDDP = update(FVDDP, c(4,5))
propagate(FVDDP, 0.2)

FVDDP = initialize(3, function(x) rnorm(x, -1,3),
                   function(x) dnorm(x, -1, 3), FALSE)
FVDDP = update(FVDDP, c(-1.145, 0.553, 0.553, 0.553))
propagate(FVDDP, 0.6)
```

`prune` *Reduce the size of Fleming-Viot Dependent Dirichlet Processes*

Description

Reduce the size of Fleming-Viot Dependent Dirichlet Processes

Usage

```
prune(fvddp, eps)
```

Arguments

`fvddp`	An object of class `fvddp`, generated via `initialize()`.
`eps`	The value behold which the weights are removed, with their components of the mixture. `eps` has to be in the interval (0,1).

Value

An `fvddp` list of smaller size of the input. Precisely, the components whose weight goes below the treshold `eps` will be removed: the vector `w` and the matrix `M` will have a lower amount of rows; if the latter will include all-zero columns, they will be removed.

References

Ascolani F, Lijoi A, Ruggiero M (2023). "Smoothing distributions for conditional Fleming–Viot and Dawson–Watanabe diffusions." *Bernoulli*, **29**(2), 1410 – 1434. doi:10.3150/22BEJ1504.

Examples

```
#create a large process
FVDDP = initialize(3, function(x) rexp(x, 4),
                   function(x) dexp(x, 4), FALSE)
FVDDP = update(FVDDP, c(rep(rexp(1, 3), 7), rep(rexp(1, 5), 5), rexp(1, 7), 3))
FVDDP = propagate(FVDDP, 1)
prune(fvddp = FVDDP, eps = 1e-4)
```

`smooth` *Compute the smoothing distribution of the Fleming-Viot latent signal*

Description

Compute the smoothing distribution of the Fleming-Viot latent signal

Usage

```
smooth(fvddp.past, fvddp.future, t.past, t.future, y.new)
```

Arguments

`fvddp.past`	An instance of class `fvddp`, progressively updated ad propagated with data referring to past times via `update()` and `propagate()` (or its approximate version, `approx.propagate()`).
`fvddp.future`	Same as `fvddp.past`, but in this case the propagation has been performed with time data from times later than the one to be estimated. Its hyperparameters must be equals to the ones of `fvddp.past`.
`t.past`	The time between the last collection of data (in the past) and the time at which the smoothing is performed.
`t.future`	Same as `t.past`, but in this case it is referred to the future. `t.future` is positive too.
`y.new`	The data collected at the time the smoothing is performed.

Value

The function returns an instance of class `fvddp` whose hyperparametrs are the same of `fvddp.past` and `fvddp.future`. The values of `y.star`and `M` are such that to represent the state of the FVDDP signal in the present time, i.e. the one Which is `t.past` away from the time at which `fvddp.past` i estimated, and is `t.future` away from the time at which `fvddp.future` is , estimated. Since the computation is usually extemely long, one can rely on the Monte-Carlo approximation provided by `approx.smooth()`.

References

Ascolani F, Lijoi A, Ruggiero M (2023). "Smoothing distributions for conditional Fleming–Viot and Dawson–Watanabe diffusions." *Bernoulli*, **29**(2), 1410 – 1434. doi:10.3150/22BEJ1504.

See Also

approx.smooth() for a (faster) approximation based on importance sampling.

Examples

```
#create wo process and sequentilly update and propagate them
FVDDP = initialize(3, function(x) rbinom(x, 10, .2),
                   function(x) dbinom(x, 10, .2), TRUE)
#for the past
FVDDP.PAST = update(FVDDP, c(2,3))
#for the future
FVDDP.FUTURE = update(FVDDP, c(4))
FVDDP.FUTURE = propagate(FVDDP.FUTURE, 0.5)
FVDDP.FUTURE = update(FVDDP.FUTURE, c(1))
#get a smoothed  FVDDP merging them (with new values too)
smooth(fvddp.past = FVDDP.PAST, fvddp.future = FVDDP.FUTURE,
       t.past= 0.4, t.future = 0.3, y.new = c(1,3))
```

---

summary.fvddp *Show the data contained within the Fleming-Viot Dependent Dirichlet Process*

Description

Show the data contained within the Fleming-Viot Dependent Dirichlet Process

Usage

```
## S3 method for class 'fvddp'
summary(object, ..., rows = FALSE, K = TRUE)
```

Arguments

object	An element of class fvddp, created via initialize().
...	Optional arguments for summary methods.
rows	Specify whether the rows must be printed. Useful in case M is large.
K	Specify whether the values of K, the amount of clusters for each row, must be printed.

Value

The function prints a base::data.frame() object (that is, of class "data.frame") where every row is a vector of multiplicities (according to the observations as in the names of the columns), with its associated weight.

Examples

```
#iniialize a simple process and show its summary
FVDDP = initialize(2, function(x) rgeom(x, .25),
                   function(x) dgeom(x, .25), TRUE)
FVDDP = update(FVDDP, rpois(4, 2))
FVDDP = propagate(FVDDP, 0.5)
summary(FVDDP)
```

`update` *Update the FVDDP when new observations are collected*

Description

Update the FVDDP when new observations are collected

Usage

```
update(fvddp, y.new)
```

Arguments

`fvddp`	An object of class `fvddp`; it can be created via `initialize()`.
`y.new`	A vector of new values to update the process.

Value

An object which is similar to the one given as an input. In particular, the multiplicities of `y.new` will be added to each row of `M`, and the weights `w` will be multiplied times the probability of drawing `y.new` form each row of the matrix `M` according to Polya urn sampling scheme.

References

Papaspiliopoulos O, Ruggiero M, Spanò D (2016). "Conjugacy properties of time-evolving Dirichlet and gamma random measures." *Electronic Journal of Statistics*, **10**(2), 3452 – 3489. doi:10.1214/16EJS1194.

Examples

```
#initialize and propagate a object
FVDDP = initialize(1, function(x) rpois(x, 3),
                   function(x) dpois(x, 3), TRUE)
update(fvddp = FVDDP, y.new = c(4,5))

#in this case, update after a propagation to see the diiffent effect of polya urn on the weights
FVDDP=initialize(3, function(x) rnorm(x, -1,3),
                 function(x) dnorm(x, -1, 3), FALSE)
FVDDP = update(FVDDP, c(-1.145, 0.553, 0.553))
FVDDP = propagate(FVDDP, 0.6)
update(fvddp = FVDDP, y.new = c(0.553, -0.316, -1.145))
```

Index