Package 'destim'

October 30, 2020

Type Package

Title R package for mobile devices position estimation

Version 0.1.0

Author David Salgado <david.salgado.fernandez@ine.es>, Luis Sanguiao Sande <luis.sanguiao.sande@ine.es>

Maintainer Bogdan Oancea <bogdan.oancea@gmail.com>

Description R package for mobile devices position estimation

License GPL3, EUPL

Imports Rcpp,
Matrix,
raster,
RColorBrewer,
ggplot2

LinkingTo Rcpp, RcppEigen

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Suggests knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

`addconstraint` *Adds constraints to the model.*

Description

The specified constraints are added to the model. If parameter `ct` is a vector, it is expected to be a set of transition probabilities indexed as in field transitions of the model. In this case the constraint added is the equality between the referred probabilities of transition. If parameter `ct` is a matrix, it is expected to be a system of additional linear equalities that the model must fulfill. Thus, the new equations are added to the field constraints of the model. While it is possible to use a matrix to add equality constraints, it is not recommended because of performance. Previous constraints of the model are preserved.

Usage

```
addconstraint(x, ct)
```

Arguments

`x`	A HMM object.
`ct`	The additional constraints, which can be either a matrix or a vector (see details).

Value

A HMM object similar to the input but with the additional constraints.

See Also

HMM, addtransition

Examples

```
model <- HMM(3)
model <- addtransition(model, c(1,2))
model <- addtransition(model, c(2,3))
model <- addtransition(model, c(3,1))
transitions(model)
constraints(model)
model <- addconstraint(model,c(2,4,5))
constraints(model)
```

`addtransition` *Adds a transition to the model.*

Description

The specified transition is added to the model as a transition with non zero probability. Since the transition probabilities from the initial state of the newly specified transition still have to sum up to one, the correspondent constraint is modified accordingly. It is not recommended to use this function to define a big model, as it is much slower than specifying all transitions in advance.

Usage

```
addtransition(x, t)
```

Arguments

`x`	A HMM object.
`t`	The transition, as a two dimensional integer vector. The first element is the number of the initial state and the second one the number of the final state.

Value

A HMM object similar to the input but with the additional transition.

See Also

HMM, addconstraint

Examples

```
model <- HMM(3)
model <- addtransition(model, c(1,2))
model <- addtransition(model, c(2,3))
model <- addtransition(model, c(3,1))
transitions(model)
constraints(model)
```

constraints *Matrix of constraints.*

Description

Returns the matrix of constraints from a HMM object.

Usage

```
constraints(x)

## S3 method for class 'HMM'
constraints(x)
```

Arguments

`x`	the HMM object.

Value

A row major sparse matrix as in `HMM`.

See Also

HMM, nconstraints, transitions

Examples

```
model <- HMMrectangle(3,3)
constraints(model)
nconstraints(model)
nrow(constraints(model)) # should agree
```

createEM *Creates the events matrix.*

Description

Creates the events matrix for a rectangular grid according to the location of the towers and the S function.

Usage

```
createEM(size, towers, S)
```

Arguments

`size`	The number of rows and columns in the grid.
`towers`	The towers(antenna) positions. This parameter is a matrix with two rows and the number of columns equals to the number of antennas. On the first row we have the X coordinate of the towers and on the secnd row the Y coordinate.
`S`	A function to compute the signal strength.

Value

The events matrix.

`destim` *A package for mobile devices position estimation using HMM.*

Description

This package contains functions to compute the posterior location probability for each device over a grid of tiles covering the geographical area under consideration. It uses Hidden Markov Models. The theory behind the method is described in detail in WPI Deliverable 3 and in the paper *An end-to-end statistical process with mobile network data for Official Statistics*. For an example on how to use this package please read example1 and example2.

Details

destim: A package for mobile devices position estimation.

`emissions` *Emissions matrix 2description Returns the matrix of emissions from a HMM object.*

Description

Emissions matrix

2description Returns the matrix of emissions from a HMM object.

Usage

```
emissions(x)

## S3 method for class 'HMM'
emissions(x)
```

Arguments

`x`	the HMM object.

Value

A column major sparse matrix as in `HMM`.

See Also

HMM, emissions<-, getTM

Examples

```
model <- HMM(2)
emissions(model)<-diag(2)
emissions(model)
```

emissions<- *Set the emissions of the model*

Description

Sets the emissions of the model.

The number of rows must match the number of states. If a matrix is provided, it is converted to column major sparse matrix (`dgCMatrix`).

Usage

```
emissions(x) <- value
```

Arguments

`x`	A HMM model.
`value`	A (sparse column major) matrix with the likelihoods of each emission (column) conditioned on the state (row).

Value

Changes the emissions matrix in the model.

See Also

HMM, emissions

Examples

```
model <- HMMrectangle(10,10)
tws <- matrix(c(3.2, 6.1, 2.2, 5.7, 5.9, 9.3, 5.4,
4.0, 2.9, 8.6, 6.9, 6.2, 9.7, 1.3),
nrow = 2, ncol = 7)
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
emissions(model)<-createEM(c(10,10), tws, S)
emissions(model)
```

example1 *This is just an example of how to use* **destim**

Description

This is just an example of how to use **destim** to compute the posterior location probabilities of the devices.

Usage

```
example1()
```

References

https://github.com/MobilePhoneESSnetBigData

Examples

```
# Local probabilities of transition
mask <- matrix(c(0.00001, 0.00320, 0.00001,
                 0.00320, 0.98716, 0.00320,
                 0.00001, 0.00320, 0.00001), ncol = 3)
#Complete transition matrix
P <- createTM(c(20,20), mask)
#Towers position
data(towers)
# Function S for Tennekes's model
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
# Complete events matrix
E <- createEM(c(20,20), towers, S)
# Load detection events
data(events)
# Estimate filtered states
FS <- fstates(P, E, events)
# Estimate smooth states
SS <- sstates(P, E, events, FS)
# Load some required packages
library(raster)
library(RColorBrewer)
library(animation)
library(ggplot2)
# Create the animations
# Filtered states
saveGIF({
pal <- colorRampPalette(c("white","red"))
ani.options(interval = 0.02)
for (i in 1:198) {
 plot(raster(matrix(FS[,i], ncol = 20)), zlim = c(0,1), col = pal(100))
 ani.pause()
}},  movie.name = 'filteredest.gif')
# Smooth states
saveGIF({
pal <- colorRampPalette(c("white","red"))
ani.options(interval = 0.02)
for (i in 1:198) {
 plot(raster(matrix(SS[,i], ncol = 20)), zlim = c(0,1), col = pal(100))
 ani.pause()
}},  movie.name = 'smoothest.gif')
# The matrix GRID relates the states with coordinates
GRID <- matrix(1, nrow = 2, ncol = 400)
GRID[1,] <- rep(1:20,20)
GRID[2,] <- rep(1:20, each = 20)
# Calculate square distance mean
fdist <- sapply (1:198, function (x)
      sum(apply(GRID - matrix(c(x %/% 10 + 1,14), nrow = 2, ncol = 400), 2,norm,type = "2") * FS[,x]))
sdist <- sapply (1:198, function (x)
      sum(apply(GRID - matrix(c(x %/% 10 + 1,14), nrow = 2, ncol = 400),2,norm,type = "2") * SS[,x]))
dists <- data.frame(T = 1:198, fdist = fdist, sdist = sdist)
ggplot() +
geom_line(data = dists, aes(x = T, y = sdist), colour = "green") +
geom_line(data = dists, aes(x = T, y = fdist), colour = "red")
```

`example2` *This is just an example on how to use functions from* **destim**.

Description

This is just an example on how to use functions from **destim**.

Usage

```
example2()
```

Examples

```
#' #Towers position
data(towers)
# Function S for Tennekes's model
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
# Complete events matrix
E <- createEM(c(20,20), towers, S)
# Load detection events
data(events)
# Probabilities of transition from maximum
#likelihood
lhood <- function(delta) logLik(createTM(c(20,20),
           mask = matrix(c((pi/4)*delta^2, delta,
           (pi/4)*delta^2, delta, 1 - 4*delta - pi * delta^2,
           delta, (pi/4)*delta^2, delta, (pi/4)*delta^2),
           ncol = 3)), E, events)
delta <- optimize(lhood, interval = c(0, 0.1))$minimum
#Complete transition matrix
P <- createTM(c(20,20),
           mask = matrix(c((pi/4)*delta^2, delta,
           (pi/4)*delta^2, delta, 1 - 4*delta - pi * delta^2,
           delta, (pi/4)*delta^2, delta, (pi/4)*delta^2),
           ncol = 3))
```

```
# Estimate observed states (no need)
OS <- ostates(P, E, events)
# Estimate filtered states
FS <- fstates(P, E, events)
# Estimate smooth states
SS <- sstates(P, E, events, FS)
# Load some required packages
library(raster)
library(RColorBrewer)
library(animation)
library(ggplot2)
# Create the animations
# Observed states
saveGIF({
 pal <- colorRampPalette(c("#00000000","#000000FF"), alpha = TRUE)
 ani.options(interval = 0.02)
 for (i in 1:198) {
  plot(raster(cbind(matrix(0,ncol = 13, nrow = 20), matrix(1,ncol=1,nrow=20), matrix(0,ncol=6,nrow=20))), bre
   plot(raster(matrix(OS[,i], ncol = 20)), zlim = c(0,1), col = pal(100), add = TRUE)
   ani.pause()
 }},  movie.name = 'obsest.gif')
# Filtered states
saveGIF({
 pal <- colorRampPalette(c("#00000000","#000000FF"), alpha = TRUE)
 ani.options(interval = 0.02)
 for (i in 1:198) {
  plot(raster(cbind(matrix(0,ncol = 13, nrow = 20), matrix(1,ncol=1,nrow=20), matrix(0,ncol=6,nrow=20))), bre
   plot(raster(matrix(FS[,i], ncol = 20)), zlim = c(0,1), col = pal(100), add = TRUE)
   ani.pause()
 }},  movie.name = 'filteredest.gif')
# Smooth states
saveGIF({
 pal <- colorRampPalette(c("#00000000","#000000FF"), alpha = TRUE)
 ani.options(interval = 0.02)
 for (i in 1:198) {
  plot(raster(cbind(matrix(0,ncol = 13, nrow = 20), matrix(1,ncol=1,nrow=20), matrix(0,ncol=6,nrow=20))), bre
   plot(raster(matrix(SS[,i], ncol = 20)), zlim = c(0,1), col = pal(100), add = TRUE)
   ani.pause()
 }},  movie.name = 'smoothest.gif')
# The matrix GRID relates the states with coordinates
GRID <- matrix(1, nrow = 2, ncol = 400)
GRID[1,] <- rep(1:20,20)
GRID[2,] <- rep(1:20, each = 20)
# Calculate square distance mean
fdist <- sapply (1:198, function (x)
      sum(apply(GRID - matrix(c(x %/% 10 + 1,14), nrow = 2, ncol = 400), 2,norm,type = "2") * FS[,x]))
sdist <- sapply (1:198, function (x)
      sum(apply(GRID - matrix(c(x %/% 10 + 1,14), nrow = 2, ncol = 400),2,norm,type = "2") * SS[,x]))
dists <- data.frame(T = 1:198, fdist = fdist, sdist = sdist)
ggplot() +
geom_line(data = dists, aes(x = T, y = sdist), colour = "green") +
geom_line(data = dists, aes(x = T, y = fdist), colour = "red")
```

`fit` *Fits a HMM model*

Description

Fits the transition probabilities of the model by maximum likelihood. The transition probabilities are fitted by ML, subject to the linear constraints specified in the model. The argument retrain can be used to avoid local minima. It is possible to specify additional non linear constraints, passing the suitable arguments to the optimizer.

Usage

```
fit(x, e, init = FALSE, method = "solnp", retrain = 1, ...)
```

Arguments

`x`	A HMM model.
`e`	A vector with the observed events. It admits missing values.
`init`	Logical specifying whether the initial state found in x is going to be used. Defaults to FALSE, which means that steady state initialization will be used instead.
`method`	The optimization algorithm to be used. Defaults to `solnp` from package **Rsolnp**. The other possible choice is `constrOptim` from package **stats**.
`retrain`	The times the optimizer will be launched with different initial parameters. The model with higher likelihood will be returned.
`...`	Arguments to be passed to the optimizer.

Value

The fitted model.

See Also

logLik, initparams, minparams

Examples

```
model <- HMMrectangle(20,20)
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
emissions(model) <- createEM(c(20,20), towers, S)
model <- initparams(model)
model <- minparams(model)
logLik(model,events)
model <- fit(model,events)
logLik(model,events)
```

`fstates` *This function returns the filtered states whenever an observation is present and predicted states otherwise.*

Description

This function returns the filtered states whenever an observation is present and predicted states otherwise.

Usage

```
fstates(P, E, e)
```

Arguments

`P`	Transition matrix P(will go to i | is in j)
`E`	Event observation matrix P(detection event j | is in i)
`e`	Sequence of observation events. Since we are taking time increase small, it is expected to have mostly missing values. The first value is expected to have an observation.

Value

The filtered states.

`getTC` *Gets the column of the transition matrix.*

Description

Gets the column of the transition matrix corresponding to a point from a rectangular grid according to the mask

Usage

```
getTC(point, size, mask)
```

Arguments

`point`	We are setting the transition probabilities starting from this point. It corresponds to the column (point[2] - 1) * size[1] + point[1] of the transition matrix.
`size`	Size of the grid. It is expected to be a 2 dimensional vector, containing the number of rows and columns respectively.
`mask`	Transition probabilities to the contiguous tiles. It is expected to be a 3x3 matrix where the (2,2) element represents the probability of staying in the current tile.

getTM *Transition matrix.*

Description

Returns the transition matrix from a HMM object. The transition matrix is represented as row major. This way its transpose matrix which is used to left multiply the state is column major.

Usage

```
getTM(x)

## S3 method for class 'HMM'
getTM(x)
```

Arguments

`x`	the HMM object.

Value

A row major sparse matrix which is the transition matrix of the model.

See Also

HMM, linkemissions

Examples

```
model <- HMM(2)
model <- addtransition(model,c(1,2))
model <- initparams(model)
getTM(model)
```

gettransmatrix *Transformation matrix*

Description

Returns the transformation matrix that transforms the minimal parameters into the probabilities of transition. The transformation matrix allows obtains the probabilities of transition from the minimal set of parameters. If we append an one at the end of the vector of parameters, the product of this matrix by such vector is the probabilities of transition vector.

Usage

```
gettransmatrix(x)

## S3 method for class 'HMM'
gettransmatrix(x)
```

Arguments

`x`	the HMM object.

Value

A matrix.

See Also

minparams, ptransition, rparams, fit

Examples

```
model <- HMMrectangle(3,3)
model <- initparams(model)
model <- minparams(model)
# Should be close to zero
range(ptransition(model) - gettransmatrix(model) %*% c(rparams(model), 1))
```

`HMM` *Class constructor for Hidden Markov models*

Description

Creates a HMM object, as specified.

The HMM object contains five fields: states, transitions, constraints, emissions and parameters.

The field states contains a character vector with the names of the states. If the constructor is given a number S, it sets the names as follows: as.character(1:S). An additional field, called coordinates is provided too, were in the future the geolocation of the states will be specified. Note that state determines geolocation, but different states might share geolocation.

The field transitions contain a matrix which is a list of the transitions with non-zero probability. It is a two row integer matrix where each column represents the transition from first row state to second row state. The columns of the matrix are ordered by first row and then by second row. This order corresponds to a row major representation of the transition matrix. The states are referenced in the same order as they appear in field states. While (number of states)^2 transitions are possible, a much smaller number is expected. It defaults to still transitions for all states.

The field constraints is the augmented matrix of the system of linear equalities that the model must fulfill. The variables of the system correspond to the probabilities of transition, in the same order as in field transitions. It is a row major sparse matrix. The first rows should have equalities between pairs of transition probabilities, which are rows with just two non zero elements. Next, we have the sum up to one conditions, which are rows with constant term equal to one. Finally, the remaining constraints are expected to have constant term different from one (otherwise multiply the constraint by a constant). This structure, allows an efficient treatment of constraints that are equalities between pairs of transition probabilities.They are expected to be the most frequent constraints.

The field emissions consists in a matrix that contains the emission probabilities, where the number of rows is the number of states and each column correspond to a possible output. EM is a column major sparse matrix. Unlike usual, the emission probabilities are fixed, do not have parameters to estimate.

The field parameters contain additional information about the probabilities of transition and the initial state of the model. Also some auxiliary information to reduce the number of parameters of the model. See `initparams`, `minparams` and `initsteady`.

Usage

```
HMM(...)

## S3 method for class 'integer'
HMM(S, TL, CT, EM = NULL, checks = TRUE)

## S3 method for class 'numeric'
HMM(S, ...)

## S3 method for class 'character'
HMM(S, ...)
```

Arguments

`S`	Number or names of states. It can be either a numeric or a character.
`TL`	Matrix of integers that lists non-zero transitions. The matrix corresponds to the field transitions of the object (see details).
`CT`	Matrix of constraints. It corresponds to the field constraints of the object (see details).
`EM`	Matrix of emissions. It corresponds to the field emissions of the object (see details).

Value

A HMM object.

See Also

initparams, minparams, initsteady

Examples

```
model1 <- HMM(5)
model2 <- HMM(c("a","b","c"),
           TL = matrix(c(1, 1,
                         1, 2,
                         2, 1,
                         2, 2,
                         2, 3,
                         3, 2,
                         3, 3), nrow = 2))
nstates(model1)
ntransitions(model1)
nstates(model2)
ntransitions(model2)
```

`HMMrectangle` *Basic HMM grid model.*

Description

Creates a basic rectangular grid model as specified. This model is a rectangular grid where the only transitions allowed are those between contiguous tiles. Moreover, horizontal and vertical transition probabilities are equal for all tiles. Diagonal transition probabilities are equal between them too, but different from the former. These constraints mean that there are only two parameters to estimate. The emissions field is left unassigned.

Usage

```
HMMrectangle(x, y)
```

Arguments

`x`	length of the rectangle in tiles.
`y`	width of the rectangle in tiles.

Value

A HMM object.

See Also

emissions, minparams

Examples

```
model <- HMMrectangle(3,3)
nstates(model)
ntransitions(model)
nconstraints(model)
```

`initparams` *Initializer for HMM objects*

Description

Sets initial parameters for a HMM object, as specified. The field parameters of the HMM object, which includes both initial state and transition probabilities, is initialized at random.

The initial states probabilities are set to an uniform (0,1) distribution and then divided by their sum.

The initial probabilities of transition are also set to an uniform (0,1) and in this case, projected on the constrained space. After the projection some probability might result greater than one or less than zero. Those probabilities are then set to uniform (0,1) again and the process is repeated until all probabilities of transition are in (0,1) and the constraints are satisfied.

Usage

```
initparams(x)
```

Arguments

x	A HMM object.

Value

An initialized HMM object.

See Also

HMM, minparams, initsteady

Examples

```
model <- HMMrectangle(3,3)
model <- initparams(model)
range(constraints(model) %*% c(ptransition(model), -1)) # It should be close to zero
```

---

initsteady   *Sets the initial state to the steady state*

Description

The initial a priori distribution is set to the steady state of the transition matrix.

The Markov Chain is expected to be irreducible and aperiodic. The first because otherwise the devices would not have freedom of movement. The second because some probabilities from one state to itself are expected to be non zero. This implies that there exists one unique steady state.

The steady state is computed by solving the sparse linear system (TM - I)x = 0, where TM is the matrix of transitions I is identity and x the steady state. As it is an homogeneous system, and because of the uniqueness of the steady state, the solution is a one dimensional vector space, and the generator does not have any coordinate equal to zero. Then the last coordinate is set to 1 / number of states, so the sparse linear system becomes inhomogeneous with unique solution. Finally the solution is normalized so that the components of x sum up to 1.

Usage

```
initsteady(x)
```

Arguments

x	A HMM object.

Value

The same HMM object of the input with its initial state set to steady state.

See Also

HMM, initparams, minparams

Examples

```
model <- HMM(2)
model <- addtransition(model, c(1,2))
model <- addtransition(model, c(2,1))
model <- initparams(model)
istates(model)
model <- initsteady(model)
istates(model)
(istates(model) %*% getTM(model))
```

`istates` *Initial state probabilities.*

Description

Returns the initial state probabilities from a HMM object. The object has to be initialized with `initparams`, which generates a random initial state. The vector of probabilities follows the same order as the states, so `ptransition(model)[i]` is the probability of state `i`. Of course, the probabilities sum up to one.

Usage

```
istates(x)

## S3 method for class 'HMM'
istates(x)
```

Arguments

`x`	the HMM object.

Value

A numeric vector with the probabilities.

See Also

initparams, fit, nstates, initsteady

Examples

```
model <- HMM(2)
model <- addtransition(model,c(1,2))
model <- addtransition(model,c(2,1))
model <- initparams(model)
istates(model)
sum(istates(model)) # should be one
```

`istates<-` *Set the initial states*

Description

Sets the initial distribution of states.

The length must match the number of states, and the sum of the vector must be one.

Usage

```
istates(x) <- value
```

Arguments

`x`	A HMM model.
`value`	A numeric vector with a probability for each state that represents the initial distribution of states.

Value

Changes the initial distribution of states in the model.

See Also

HMM, initparams, initsteady, fit

Examples

```
model <- HMMrectangle(3,3)
model <- initparams(model)
istates(model)
istates(model) <- (1:9) / sum(1:9)
istates(model)
```

`logLik` *Minus logLikelihood*

Description

Returns the minus logarithm of the likelihood given a model and a set of observations.

A slightly modified version of the forward algorithm is used to compute the likelihood, to avoid store unneeded data. The sign is changed because it is usual to minimize instead maximize.

Usage

```
logLik(...)

## S3 method for class 'HMM'
logLik(x, e)
```

Arguments

`x`	A HMM model.
`e`	A vector with the observed events. It admits missing values.

Value

The minus logarithm of the likelihood of the events given the model.

See Also

fit, HMM, initparams

Examples

```
model <- HMMrectangle(20,20)
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
emissions(model) <- createEM(c(20,20), towers, S)
model <- initparams(model)
model <- minparams(model)
logLik(model,events)
```

`minparams` *Reparametrizes a HMM model with a minimal set of parameters.*

Description

Finds a minimal set of parameters that fully determine the probabilities of transition.

This function avoids to solve a high dimensional optimization problem with many constraints, parametrizing the probabilities of transition with as few parameters as possible: the number of degrees of freedom.

A pivoted QR decomposition of the constraints matrix is done, to get both the free parameters and the matrix that transforms them back into the probabilities of transition.

Many constraints are expected to be equalities between two probabilities of transition, so the function is optimized for this special kind of constraints.

Usage

```
minparams(x)
```

Arguments

`x`	A HMM object.

Value

The same HMM object of the input with some additional fields that store the new parameters and the matrix that transforms these parameters in the probabilities of transition.

See Also

rparams, rparams<-

Examples

```
model <- HMMrectangle(2,2)
model <- initparams(model)
ntransitions(model)
nconstraints(model)
model <- minparams(model)
rparams(model)
range(ptransition(model) -
  gettransmatrix(model) %*% c(rparams(model), 1))
```

---

nconstraints *Number of constraints.*

Description

Returns the number of constraints from a HMM object.

Usage

```
nconstraints(x)

## S3 method for class 'HMM'
nconstraints(x)
```

Arguments

x	the HMM object.

Value

An integer with the number of constraints of the model.

See Also

constraints, nstates, ntransitions

Examples

```
model <- HMM(5)
nconstraints(model)
model <- HMMrectangle(3, 3)
nconstraints(model)
```

nstates *Number of states.*

Description

Returns the number of states from a HMM object.

Usage

```
nstates(x)

## S3 method for class 'HMM'
nstates(x)
```

Arguments

x	the HMM object.

Value

An integer with the number of states of the model.

See Also

ntransitions, nconstraints

Examples

```
model <- HMM(5)
nstates(model)
```

ntransitions *Number of transitions.*

Description

Returns the number of possible transitions from a HMM object.

Usage

```
ntransitions(x)

## S3 method for class 'HMM'
ntransitions(x)
```

Arguments

x	the HMM object.

Value

An integer with the number of possible transitions of the model.

See Also

transitions, nstates, nconstraints

Examples

```
model <- HMM(5)
ntransitions(model)
model <- addtransition(model, c(1,2))
ntransitions(model)
```

---

ptransition *Probabilities of transition.*

Description

Returns the probabilities of transition from a HMM object. The object has to be initialized with `initparams`, otherwise it will return numeric(0). The order is row major.

Usage

```
ptransition(x)

## S3 method for class 'HMM'
ptransition(x)
```

Arguments

x	the HMM object.

Value

A numeric vector with the probabilities of transition.

See Also

HMM, initparams, transitions, ntransitions

Examples

```
model <- HMM(2)
model <- addtransition(model,c(1,2))
model <- addtransition(model,c(2,1))
model <- initparams(model)
ptransition(model)
```

rparams — *Reduced parameters.*

Description

Returns the values of the minimal set of parameters. The minimal set of parameters are selected by `minparams`. They are a few probabilities of transition that determine the remaining ones because of the constraints. They are used to fit the model.

Usage

```
rparams(x)

## S3 method for class 'HMM'
rparams(x)
```

Arguments

`x`	the HMM object.

Value

A numeric vector with the values of the parameters.

See Also

minparams, ptransition, gettransmatrix, fit

Examples

```
model <- HMMrectangle(3,3)
model <- initparams(model)
model <- minparams(model)
rparams(model)
ntransitions(model)
length(rparams(model)) # A much smaller parameter space!
```

rparams<- — *Set reduced parameters*

Description

Sets the parameters selected by `minparams` function.

The function `minparams` selects a minimal set of parameters, that fully determine the transition probabilities. This function sets those parameters and recalculates all transition probabilities from them.

The model is initialized with `initparams` and `minparams` when required.

Usage

```
rparams(x) <- value
```

Arguments

`x`	A HMM model.
`value`	A numeric vector with the new parameters.

Value

Changes parameters$reducedparams$params and parameters$transitions in the object x.

See Also

minparams, rparams, initparams

Examples

```
model <- HMMrectangle(3,3)
rparams(model)<-c(0.3, 0.03)
ptransition(model)
```

`scpstates` *Returns ξ like in the Baum-Welch algorithm.*

Description

Returns the smooth joint probability mass function for consecutive states, which is usually called ξ in the Baum-Welch algorithm. Smooth states are marginal but as they are far to be independent it is convenient to have some information about their dependence. This function returns the joint probability mass function for two time consecutive states, conditional on the observations. This agrees with the so called ξ from the Baum-Welch algorithm.

It is returned as a matrix, so that the said joint probability for time instants i - 1 and i are the columns from i - 1 times the number of states plus one, to i times the number of states.

Usage

```
scpstates(...)

## S3 method for class 'HMM'
scpstates(x, e)
```

Arguments

`x`	A HMM model.
`e`	A vector with the observed events. It admits missing values.

Value

A sparse matrix. The number of rows is the number of states, and the number of columns is the number of states times the number of observed events minus one. Each full row square slice of the output matrix corresponds to a joint probability mass function, so it sums up to one.

See Also

HMM, sstates, backward

Examples

```
model <- HMMrectangle(10,10)
tws <- matrix(c(3.2, 6.1, 2.2, 5.7, 5.9, 9.3, 5.4,
4.0, 2.9, 8.6, 6.9, 6.2, 9.7, 1.3),
nrow = 2, ncol = 7)
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
emissions(model)<-createEM(c(10,10), tws, S)
obs <- c(1,2,NA,NA,NA,NA,7,7)
model <- fit(model, obs)
scpstates(model, obs)
```

`setsnames<-` *Set the names of the states.*

Description

Sets the names of the states.

The length of the character vector must match the number of states of the model.

Usage

```
setsnames(x) <- value
```

Arguments

`x`	A HMM model.
`value`	A character vector with the names.

Value

Changes states names in the object x.

See Also

HMM

Examples

```
model <- HMM(3)
setsnames(model) <- c("a","b","c")
model$states$names
```

`sstates` *Smooth states*

Description

Returns the smooth states from the forward-backward algorithm.

Smooth states are the marginal of the state conditional on the observations, for each time. This agrees with the so called γ from the Baum-Welch algorithm.

It is returned as a matrix, so that the smooth state for time instant i is the column i of the matrix.

Usage

```
sstates(...)

## S3 method for class 'HMM'
sstates(x, e)
```

Arguments

x	A HMM model.
e	A vector with the observed events. It admits missing values.

Value

A sparse matrix. The number of rows is the number of states, and the number of columns is the number of observed events. Each column of the output matrix corresponds to the probability mass function for the state, so it sums up to one.

See Also

HMM, scpstates, backward

Examples

```
model <- HMMrectangle(10,10)
tws <- matrix(c(3.2, 6.1, 2.2, 5.7, 5.9, 9.3, 5.4,
4.0, 2.9, 8.6, 6.9, 6.2, 9.7, 1.3),
nrow = 2, ncol = 7)
S <- function(x) if (x > 5) return(0) else return(20*log(5/x))
emissions(model)<-createEM(c(10,10), tws, S)
obs <- c(1,2,NA,NA,NA,NA,7,7)
model <- fit(model, obs)
sstates(model, obs)
```

`transitions` *Transitions.*

Description

Returns the list of possible transitions from a HMM object. Each column represents a transition, the first row is the initial state and the second row is the final state. The transitions are ordered, first on the initial state and then on the final state. Any transition not listed in the matrix is supposed to be not possible (zero probability).

Usage

```
transitions(x)

## S3 method for class 'HMM'
transitions(x)
```

Arguments

`x`	the HMM object.

Value

An integer matrix with two rows as in `HMM`.

See Also

HMM, ntransitions, constraints, ptransition

Examples

```
model <- HMM(2)
transitions(model)
model <- addtransition(model,c(1,2))
model <- addtransition(model,c(2,1))
transitions(model)
```

Index