Package 'HiddenMarkov'

18 February 2008

Title Hidden Markov Models

Version 1.2-4

Date 2008-02-18

Author David Harte

Maintainer David Harte <david@statsresearch.co.nz>

Description Contains functions for the analysis of Discrete Time Hidden Markov Models, Markov Modulated GLMs and the Markov Modulated Poisson Process. It includes functions for simulation, parameter estimation, and the Viterbi algorithm. See the topic `Overview` for an introduction to the package, and `changes` for a list of recent changes. The algorithms are based of those of Walter Zucchini.

License GPL (>=2)

URL http://www.statsresearch.co.nz

R topics documented:

`BaumWelch` *Estimate Parameters Using Baum-Welch Algorithm*

Description

Estimates the parameters of a hidden Markov model. The Baum-Welch algorithm (Baum et al, 1970) referred to in the HMM literature is a version of the EM algorithm (Dempster et al, 1977). See Hartley (1958) for an earlier application of the EM methodology, though not referred to as such.

Usage

```
BaumWelch(object, control, ...)
## S3 method for class 'dthmm':
BaumWelch(object, control = bwcontrol(), ...)
## S3 method for class 'mmglm':
BaumWelch(object, control = bwcontrol(), ...)
## S3 method for class 'mmpp':
BaumWelch(object, control = bwcontrol(), ...)
```

Arguments

`object`	an object of class `"dthmm"`, `"mmglm"`, or `"mmpp"`.
`control`	a list of control settings for the iterative process. These can be changed by using the function `bwcontrol`.
`...`	other arguments.

Details

The initial parameter values used by the EM algorithm are those that are contained within the input `object`.

The code for the methods `"dthmm"`, `"mmglm"` and `"mmpp"` can be viewed by typing `BaumWelch.dthmm`, `BaumWelch.mmglm` or `BaumWelch.mmpp`, respectively, on the R command line.

Value

The output object (a `list`) with have the same class as the input, and will have the same components. The parameter values will be replaced by those estimated by this function. The object will also contain additional components.

An object of class `"dthmm"` will also contain

`u`	an $n \times m$ matrix containing estimates of the conditional expectations. See "Details" in `Estep`.
`v`	an $n \times m \times m$ array containing estimates of the conditional expectations. See "Details" in `Estep`.

LL	value of log-likelihood at the end.
iter	number of iterations performed.
diff	difference between final and previous log-likelihood.

References

Baum, L.E.; Petrie, T.; Soules, G. & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics* **41(1)**, 164–171. DOI: http://dx.doi.org/10.1214/aoms/1177697196

Dempster, A.P.; Laird, N.M. & Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J. Royal Statist. Society B* **39(1)**, 1–38.

Hartley, H.O. (1958). Maximum likelihood estimation from incomplete data. *Biometrics* **14(2)**, 174–194. DOI: http://dx.doi.org/10.2307/2527783

See Also

logLik, residuals, simulate, summary , neglogLik

---

bwcontrol *Control Parameters for the Baum Welch Algorithm*

Description

Creates a list of parameters that control the operation of BaumWelch.

Usage

```
bwcontrol(maxiter = 500, tol = 1e-05, prt = TRUE, posdiff = TRUE,
          converge = expression(diff < tol))
```

Arguments

maxiter	is the maximum number of iterations, default is 500.
tol	is the convergence criterion, default is 0.00001.
prt	is logical, and determines whether information is printed at each iteration; default is TRUE.
posdiff	is logical, and determines whether the iterative process stops if a negative log-likelihood difference occurs, default is TRUE.
converge	is an expression giving the convergence criterion. The default is the difference between successive values of the log-likelihood.

Examples

```
#    Increase the maximum number of iterations to 1000.
#    All other components will retain their default values.
a <- bwcontrol(maxiter=1000)
print(a)
```

changes *Changes Made to the Package*

Description

This page contains a listing of recent changes made to the package.

Details

1. Since we have included different classes of HMMs (see dthmm, mmglm and mmpp), it is much tidier to use an object orientated approach. This ensures that the functions across all models follow a more consistent naming convention, and also the argument list for the different model functions are more simple and consistent (see Overview). (14 Sep 2007)
2. The main tasks (model fitting, residuals, simulation, Viterbi, etc) can now be called by generic functions (see topic Overview). The package documentation has been rearranged so that these generic functions contain the documentation for all model types (e.g. see BaumWelch). (14 Sep 2007)
3. There are a number of functions, still contained in the package, that are obsolete. This is either because they do not easily fit into the current naming convention used to implement the more object orientated approach, or their argument list is slightly complicated. These functions have been grouped in the topics dthmm.obsolete and mmpp.obsolete. (14 Sep 2007)
4. There are various *second level* functions. For example, the model fitting is achieved by the generic BaumWelch function. However, this will call functions to do the E-step, M-step, forward and backward probabilities, and so on. At the moment, these *second level* functions have not been modified into an object orientated approach. It is not clear at this point whether this would be advantageous. If one went down this route, then one would probably group all of the E-step functions (for all models) under the same topic. If not, then it may be best to group all second level functions for each model under the same topic (e.g. forwardback, probhmm and Estep would be grouped together, being the second level functions for the dthmm model). (14 Sep 2007)
5. The original function called Viterbi has been renamed to Viterbihmm, and Viterbi is now a generic function. (14 Sep 2007)
6. Programming code that uses old versions of the functions should still work with this revised version of the package. However, you will get warning messages stating that certain functions are deprecated, and suggesting a possible alternative. To get a quick overview of the programming style, have a look at the examples in topic dthmm. (09 Nov 2007)
7. forwardback: for loops replaced by Fortran code; much faster. The corresponding R code is still contained within the function in case the Fortran has incompatibility issues. (23 Nov 2007)
8. forwardback.mmpp: for loops replaced by Fortran code. The corresponding R code is still contained within the function in case the Fortran has incompatibility issues. (24 Nov 2007)
9. Estep.mmpp: for loops replaced by Fortran code. Cuts time considerably. These loops in R used far more time than the forward and backward equations. The corresponding R code is still contained within the function in case the Fortran has incompatibility issues. (27 Nov 2007)
10. forwardback.mmpp, forwardback and Estep.mmpp: argument fortran added. (3 Dec 2007)

11. forwardback, forwardback.mmpp and Estep.mmpp: inclusion of all variable sized arrays in the Fortran subroutine call to be compatible with non gfortran compilers (3 Dec 2007); more added for calls to Fortran subroutines multi1 and multi2. (6 Dec 2007)
12. Estep.mmpp: error in Fortran code of loop 6; j1=0 to j1=1. (5 Dec 2007)
13. BaumWelch.mmpp: if (diff < 0) stop ... to if (diff < 0 & control$posdiff) stop ..., consistent with BaumWelch.dthmm. (11 Dec 2007)
14. logLik.dthmm, logLik.mmglm, logLik.mmpp: for loop replaced by Fortran code. (15 Feb 2008)
15. dthmm: argument discrete set automatically for known distributions, stops if not set for unknown distributions. (15 Feb 2008)
16. neglogLik, Pi2vector, vector2Pi, Q2vector, vector2Q: new functions providing an alternative means of calculating maximum likelihood parameter estimates. (18 Feb 2008)

Future Development

1. The functions residuals and Viterbi need methods for objects with class mmpp.
2. A number of the original functions have names that are too general. For example forwardback calculates the forward-backward probabilities, but only for the model dthmm. The corresponding function for the mmpp model is forwardback.mmpp. It would be more consistent to attach to these original functions a dthmm suffix.
3. The demonstration examples are all for dthmm. Also need some for mmglm and mmpp.

---

compdelta — *Compute Marginal Distribution of Stationary Markov Chain*

Description

Computes the marginal distribution of a *stationary* Markov chain with transition probability matrix Π. The m discrete states of the Markov chain are denoted by $1, \cdots, m$.

Usage

```
compdelta(Pi)
```

Arguments

Pi	is the $m \times m$ transition probability matrix of the Markov chain.

Details

If the Markov chain is stationary, then the marginal distribution δ satisfies

$$\delta = \delta\Pi\,.$$

Obviously,

$$\sum_{j}^{m} \delta_j = 1.$$

Value

A numeric vector of length m containing the marginal probabilities.

Examples

```
Pi <- matrix(c(1/2, 1/2,   0,   0,   0,
               1/3, 1/3, 1/3,   0,   0,
                 0, 1/3, 1/3, 1/3,   0,
                 0,   0, 1/3, 1/3, 1/3,
                 0,   0,   0, 1/2, 1/2),
             byrow=TRUE, nrow=5)

print(compdelta(Pi))
```

Demonstration *Demonstration Examples*

Description

Demonstration examples can be run by executing the code below.

Examples

```
#   Model with class "dthmm" with the Beta distribution
demo("beta", package="HiddenMarkov")

#   Model with class "dthmm" with the Gamma distribution
demo("gamma", package="HiddenMarkov")

#   Model with class "dthmm" with the Log Normal distribution
demo("lnorm", package="HiddenMarkov")

#   Model with class "dthmm" with the Logistic distribution
demo("logis", package="HiddenMarkov")

#   Model with class "dthmm" with the Gaussian distribution
demo("norm", package="HiddenMarkov")
```

---

dthmm.obsolete *Discrete Time HMM - Obsolete Functions*

Description

These functions are obsolete and will ultimately be removed from the package. Please change to the object orientated versions: BaumWelch, residuals, simulate or Viterbi.

Usage

```
Baum.Welch(x, Pi, delta, distn, pm, pn = NULL, nonstat = TRUE,
           maxiter = 500, tol = 1e-05, prt = TRUE,
           posdiff = (distn[1]!="glm"))
residualshmm(x, Pi, delta, distn, pm, pn = NULL, discrete = FALSE)
sim.hmm(n, initial, Pi, distn, pm, pn = NULL)
sim.hmm1(n, initial, Pi, distn, pm)
sim.markov(n, initial, Pi)
Viterbihmm(x, Pi, delta, distn, pm, pn = NULL)
```

Arguments

`x`	is a vector of length n containing the observed process.
`n`	length of process.
`initial`	integer, being the initial hidden Markov state $(1, \cdots, m)$.
`Pi`	is the $m \times m$ transition probability matrix of the hidden Markov chain.
`delta`	is the marginal probability distribution of the m hidden states at the first time point.
`distn`	is a character string with the distribution name, e.g. `"norm"` or `"pois"`. If the distribution is specified as `"wxyz"` then a distribution function called `"pwxyz"` should be available, in the standard R format (e.g. `pnorm` or `ppois`).
`pm`	is a list object containing the (Markov dependent) parameter values associated with the distribution of the observed process (see `dthmm`).
`pn`	is a list object containing the observation dependent parameter values associated with the distribution of the observed process (see `dthmm`).
`discrete`	is logical, and is `TRUE` if `distn` is a discrete distribution.
`nonstat`	is logical, `TRUE` if the homogeneous Markov chain is assumed to be non-stationary, default. See "Details" below.
`maxiter`	is the maximum number of iterations, default is 500.
`tol`	is the convergence criterion, being the difference between successive values of the log-likelihood; default is 0.00001.
`prt`	is logical, and determines whether information is printed at each iteration; default is `TRUE`.
`posdiff`	is logical, and determines whether the iterative process stops if a negative log-likelihood difference occurs.

Details

The function `sim.hmm1` will run faster for cases where the argument `pn` is `NULL`.

dthmm *Discrete Time HMM Object*

Description

Creates a discrete time hidden Markov model object with class "dthmm".

Usage

```
dthmm(x, Pi, delta, distn, pm, pn = NULL, discrete = NULL,
      nonstat = TRUE)
```

Arguments

x	is a vector of length n containing the observed process. Alternatively, x could be specified as NULL, meaning that the data will be added later (e.g. simulated).
Pi	is the $m \times m$ transition probability matrix of the homogeneous hidden Markov chain.
delta	is the marginal probability distribution of the m hidden states at the first time point.
distn	is a character string with the distribution name, e.g. "norm" or "pois". If the distribution is specified as "wxyz" then a distribution function called "pwxyz" should be available, in the standard R format (e.g. pnorm or ppois).
pm	is a list object containing the (Markov dependent) parameter values associated with the distribution of the observed process (see below).
pn	is a list object containing the observation dependent parameter values associated with the distribution of the observed process (see below).
discrete	is logical, and is TRUE if distn is a discrete distribution. Set automatically for distributions already contained in the package.
nonstat	is logical, TRUE if the homogeneous Markov chain is assumed to be non-stationary, default. See "Details" below.

Value

A list object with class "dthmm", containing the above arguments as named components.

Notation

1. MacDonald & Zucchini (1997) use t to denote the *time*, where $t = 1, \cdots, T$. To avoid confusion with other uses of t and T in R we use $i = 1, \cdots, n$.
2. We denote the observed sequence as $\{X_i\},\ i = 1, \cdots, n$; and the hidden Markov chain as $\{C_i\},\ i = 1, \cdots, n$.
3. The history of the observed process up to time i is denoted by $X^{(i)}$, i.e.

$$X^{(i)} = (X_1, \cdots, X_i)$$

   where $i = 1, \cdots, n$. Similarly for $C^{(i)}$.
4. The hidden Markov chain has m states denoted by $1, \cdots, m$.

5. The Markov chain transition probability matrix is denoted by Π, where the (j,k)th element is

$$\pi_{jk} = \Pr\{C_{i+1} = k \,|\, C_i = j\}$$

for all i (i.e. all time points), and $j, k = 1, \cdots, m$.

6. The Markov chain is assumed to be *homogeneous*, i.e. for each j and k, π_{jk} is constant over time.

7. The Markov chain is said to be *stationary* if the marginal distribution is the same over time, i.e. for each j, $\delta_j = \Pr\{C_i = j\}$ is constant for all i. The marginal distribution is denoted by $\delta = (\delta_1, \cdots, \delta_m)$.

List Object pm

The list object `pm` contains parameter values for the probability distribution of the observed process that are dependent on the hidden Markov state. These parameters are generally required to be estimated. See "Modifications" in topic `Mstep` when some do not require estimation.

Assume that the hidden Markov chain has m states, and that there are ℓ parameters that are dependent on the hidden Markov state. Then the list object `pm` should contain ℓ *named* vector components each of length m. The names are determined by the required probability distribution.

For example, if `distn == "norm"`, the arguments names must coincide with those used by the functions `dnorm` or `rnorm`, which are `mean` and `sd`. Each must be specified in either `pm` or `pn`. If they both vary according to the hidden Markov state then `pm` should have the *named* components `mean` and `sd`. These are both vectors of length m containing the means and standard deviations of the observed process when the hidden Markov chain is in each of the m states. If, for example, `sd` was "time" dependent, then `sd` would be contained in `pn` (see below).

If `distn == "pois"`, then `pm` should have one component named `lambda`, being the parameter name in the function `dpois`. Even if there is only one parameter, the vector component should still be within a list and named.

List Object pn

The list object `pn` contains parameter values of the probability distribution for the observed process that are dependent on the observation number or "time". These parameters are assumed to be *known*.

Assume that the observed process is of length n, and that there are ℓ parameters that are dependent on the observation number or time. Then the list object `pn` should contain ℓ *named* vector components each of length n. The names, as in `pm`, are determined by the required probability distribution.

For example, in the observed process we may count the number of successes in a *known* number of Bernoulli trials, i.e. the number of Bernoulli trials is known at each time point, but the probability of success varies according to a hidden Markov state. The `prob` parameter of `rbinom` (or `dbinom`) would be specified in `pm` and the `size` parameter would specified in `pn`.

One could also have a situation where the observed process was Gaussian, with the means varying according to the hidden Markov state, but the variances varying non-randomly according to the observation number (or vice versa). Here `mean` would be specified within `pm` and `sd` within `pn`. Note that a given parameter can only occur within *one* of `pm` or `pn`.

Complete Data Likelihood

The "complete data likelihood", L_c, is

$$L_c = \Pr\{X_1 = x_1, \cdots, X_n = x_n, C_1 = c_1, \cdots, C_n = c_n\}\,.$$

This can be shown to be

$$\Pr\{X_1 = x_1 \,|\, C_1 = c_1\} \Pr\{C_1 = c_1\} \prod_{i=2}^{n} \Pr\{X_i = x_i \,|\, C_i = c_i\} \Pr\{C_i = c_i \,|\, C_{i-1} = c_{i-1}\},$$

and hence, substituting model parameters, we get

$$L_c = \delta_{c_1} \pi_{c_1 c_2} \pi_{c_2 c_3} \cdots \pi_{c_{n-1} c_n} \prod_{i=1}^{n} \Pr\{X_i = x_i \,|\, C_i = c_i\},$$

and so

$$\log L_c = \log \delta_{c_1} + \sum_{i=2}^{n} \log \pi_{c_{i-1} c_i} + \sum_{i=1}^{n} \log \Pr\{X_i = x_i \,|\, C_i = c_i\}.$$

Hence the "complete data likelihood" is split into three terms: the first relates to parameters of the marginal distribution (Markov chain), the second to the transition probabilities, and the third to the distribution parameters of the observed random variable. When the Markov chain is non-stationary, each term can be maximised separately.

When the hidden Markov chain is assumed to be stationary, $\delta = \Pi'\delta$ (see topic compdelta), and then the first two terms of $\log L_c$ determine the transition probabilities Π. This raises more complicated numerical problems, as the first term is effectively a constraint. We deal with this in a slightly ad-hoc manner by effectively disregarding the first term, which is assumed to be relatively small. In the M-step, the transition matrix is determined by the second term, then δ is estimated using the relation $\delta = \delta\Pi$.

References

Elliott, R.J.; Aggoun, L. & Moore, J.B. (1994). *Hidden Markov Models: Estimation and Control.* Springer-Verlag, New York.

Harte, D. (2006). *Mathematical Background Notes for Package "HiddenMarkov".* Statistics Research Associates, Wellington. URL: http://homepages.paradise.net.nz/david.harte/SSLib/Manuals/notes.pdf.

MacDonald, I.L. & Zucchini, W. (1997). *Hidden Markov and Other Models for Discrete-valued Time Series.* Chapman and Hall/CRC, Boca Raton.

Rabiner, L.R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* **77(2)**, 257–286. DOI: http://dx.doi.org/10.1109/5.18626.

Zucchini, W. (2005). *Hidden Markov Models Short Course, 3–4 April 2005.* Macquarie University, Sydney.

Examples

```
#-----  Test Gaussian Distribution -----

Pi <- matrix(c(1/2, 1/2,   0,
               1/3, 1/3, 1/3,
                 0, 1/2, 1/2),
             byrow=TRUE, nrow=3)

delta <- c(0, 1, 0)

x <- dthmm(NULL, Pi, delta, "norm",
           list(mean=c(1, 6, 3), sd=c(0.5, 1, 0.5)))
```

```
x <- simulate(x, nsim=1000)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(summary(y))
print(logLik(y))
hist(residuals(y))

#   check parameter estimates
print(sum(y$delta))
print(y$Pi %*% rep(1, ncol(y$Pi)))

#-----  Test Poisson Distribution  -----

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

delta <- c(0, 1)

x <- dthmm(NULL, Pi, delta, "pois", list(lambda=c(4, 0.1)),
           discrete = TRUE)

x <- simulate(x, nsim=1000)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(summary(y))
print(logLik(y))
hist(residuals(y))

#   check parameter estimates
print(sum(y$delta))
print(y$Pi %*% rep(1, ncol(y$Pi)))

#-----  Test Exponential Distribution  -----

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

delta <- c(0, 1)

x <- dthmm(NULL, Pi, delta, "exp", list(rate=c(2, 0.1)))

x <- simulate(x, nsim=1000)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(summary(y))
print(logLik(y))
hist(residuals(y))
```

```
#   check parameter estimates
print(sum(y$delta))
print(y$Pi %*% rep(1, ncol(y$Pi)))

#-----  Test Beta Distribution  -----

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

delta <- c(0, 1)

x <- dthmm(NULL, Pi, delta, "beta", list(shape1=c(2, 6), shape2=c(6, 2)))

x <- simulate(x, nsim=1000)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(summary(y))
print(logLik(y))
hist(residuals(y))

#   check parameter estimates
print(sum(y$delta))
print(y$Pi %*% rep(1, ncol(y$Pi)))

#-----  Test Binomial Distribution  -----

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

delta <- c(0, 1)

#   vector of "fixed & known" number of Bernoulli trials
pn <- list(size=rpois(1000, 10)+1)

x <- dthmm(NULL, Pi, delta, "binom", list(prob=c(0.2, 0.8)), pn,
           discrete=TRUE)

x <- simulate(x, nsim=1000)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(summary(y))
print(logLik(y))
hist(residuals(y))

#   check parameter estimates
print(sum(y$delta))
print(y$Pi %*% rep(1, ncol(y$Pi)))

#-----  Test Gamma Distribution  -----

Pi <- matrix(c(0.8, 0.2,
```

```
                0.3, 0.7),
              byrow=TRUE, nrow=2)

delta <- c(0, 1)

pm <- list(rate=c(4, 0.5), shape=c(3, 3))

x <- seq(0.01, 10, 0.01)
plot(x, dgamma(x, rate=pm$rate[1], shape=pm$shape[1]),
     type="l", col="blue", ylab="Density")
points(x, dgamma(x, rate=pm$rate[2], shape=pm$shape[2]),
       type="l", col="red")

x <- dthmm(NULL, Pi, delta, "gamma", pm)

x <- simulate(x, nsim=1000)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(summary(y))
print(logLik(y))
hist(residuals(y))

#   check parameter estimates
print(sum(y$delta))
print(y$Pi %*% rep(1, ncol(y$Pi)))
```

`Estep` *E Step of EM Algorithm*

Description

Performs the *expectation* step of the EM algorithm for a `dthmm` process. This function is called by the `BaumWelch` function. The Baum-Welch algorithm referred to in the HMM literature is a version of the EM algorithm.

Usage

```
Estep(x, Pi, delta, distn, pm, pn = NULL)
```

Arguments

`x`	is a vector of length n containing the observed process.
`Pi`	is the current estimate of the $m \times m$ transition probability matrix of the hidden Markov chain.
`distn`	is a character string with the distribution name, e.g. `"norm"` or `"pois"`. If the distribution is specified as `"wxyz"` then a probability (or density) function called `"dwxyz"` should be available, in the standard R format (e.g. `dnorm` or `dpois`).
`pm`	is a list object containing the current (Markov dependent) parameter estimates associated with the distribution of the observed process (see `dthmm`).

`pn`	is a list object containing the observation dependent parameter values associated with the distribution of the observed process (see `dthmm`).
`delta`	is the current estimate of the marginal probability distribution of the m hidden states.

Details

Let u_{ij} be one if $C_i = j$ and zero otherwise. Further, let v_{ijk} be one if $C_{i-1} = j$ and $C_i = k$, and zero otherwise. Let $X^{(n)}$ contain the complete observed process. Then, given the current model parameter estimates, the returned value `u[i,j]` is

$$\widehat{u}_{ij} = \mathrm{E}[u_{ij} \mid X^{(n)}] = \Pr\{C_i = j \mid X^{(n)} = x^{(n)}\},$$

and `v[i,j,k]` is

$$\widehat{v}_{ijk} = \mathrm{E}[v_{ijk} \mid X^{(n)}] = \Pr\{C_{i-1} = j, C_i = k \mid X^{(n)} = x^{(n)}\},$$

where $j, k = 1, \cdots, m$ and $i = 1, \cdots, n$.

Value

A `list` object is returned with the following components.

`u`	an $n \times m$ matrix containing estimates of the conditional expectations. See "Details".
`v`	an $n \times m \times m$ array containing estimates of the conditional expectations. See "Details".
`LL`	the current value of the log-likelihood.

Author(s)

The algorithm has been taken from Zucchini (2005).

References

Zucchini, W. (2005). *Hidden Markov Models Short Course, 3–4 April 2005.* Macquarie University, Sydney.

See Also

`BaumWelch`, `Mstep`

`forwardback` *Forward and Backward Probabilities*

Description

These functions calculate the forward and backward probabilities for a `dthmm` process, as defined in MacDonald & Zucchini (1997, Page 60).

Usage

```
backward(x, Pi, distn, pm, pn = NULL)
forward(x, Pi, delta, distn, pm, pn = NULL)
forwardback(x, Pi, delta, distn, pm, pn = NULL, fortran = TRUE)
```

Arguments

`x`	is a vector of length n containing the observed process.
`Pi`	is the $m \times m$ transition probability matrix of the hidden Markov chain.
`delta`	is the marginal probability distribution of the m hidden states.
`distn`	is a character string with the distribution name, e.g. `"norm"` or `"pois"`. If the distribution is specified as `"wxyz"` then a probability (or density) function called `"dwxyz"` should be available, in the standard R format (e.g. `dnorm` or `dpois`).
`pm`	is a list object containing the current (Markov dependent) parameter estimates associated with the distribution of the observed process (see `dthmm`).
`pn`	is a list object containing the observation dependent parameter values associated with the distribution of the observed process (see `dthmm`).
`fortran`	logical, if `TRUE` (default) use the Fortran code, else use the R code.

Details

Denote the $n \times m$ matrices containing the forward and backward probabilities as A and B, respectively. Then the (i, j)th elements are

$$\alpha_{ij} = \Pr\{X_1 = x_1, \cdots, X_i = x_i, C_i = j\}$$

and

$$\beta_{ij} = \Pr\{X_{i+1} = x_{i+1}, \cdots, X_n = x_n \,|\, C_i = j\}\,.$$

Further, the diagonal elements of the product matrix AB' are all the same, taking the value of the log-likelihood.

Value

The function `forwardback` returns a list with two matrices containing the forward and backward probabilities, `logalpha` and `logbeta`, respectively, and the log-likelihood (`LL`).

The functions `backward` and `forward` return a matrix containing the forward and backward probabilities, `logalpha` and `logbeta`, respectively.

Author(s)

David Harte, 2005. The algorithm has been taken from Zucchini (2005).

References

MacDonald, I.L. & Zucchini, W. (1997). *Hidden Markov and Other Models for Discrete-valued Time Series.* Chapman and Hall/CRC, Boca Raton.

Zucchini, W. (2005). *Hidden Markov Models Short Course, 3–4 April 2005.* Macquarie University, Sydney.

See Also

logLik

Examples

```
#    Set Parameter Values

Pi <- matrix(c(1/2, 1/2,   0,   0,   0,
               1/3, 1/3, 1/3,   0,   0,
                 0, 1/3, 1/3, 1/3,   0,
                 0,   0, 1/3, 1/3, 1/3,
                 0,   0,   0, 1/2, 1/2),
             byrow=TRUE, nrow=5)

p <- c(1, 4, 2, 5, 3)
delta <- c(0, 1, 0, 0, 0)

#------   Poisson HMM   ------

x <- dthmm(NULL, Pi, delta, "pois", list(lambda=p), discrete=TRUE)

x <- simulate(x, nsim=10)

y <- forwardback(x$x, Pi, delta, "pois", list(lambda=p))

# below should be same as LL for all time points
print(log(diag(exp(y$logalpha) %*% t(exp(y$logbeta)))))
print(y$LL)

#------   Gaussian HMM   ------

x <- dthmm(NULL, Pi, delta, "norm", list(mean=p, sd=p/3))

x <- simulate(x, nsim=10)

y <- forwardback(x$x, Pi, delta, "norm", list(mean=p, sd=p/3))

# below should be same as LL for all time points
print(log(diag(exp(y$logalpha) %*% t(exp(y$logbeta)))))
print(y$LL)
```

HiddenMarkov-internal

Internally Used Functions

Description

This page lists internally used functions. They should not be required by most users.

Usage

```
## S3 method for class 'mmpp':
residuals(object, ...)
```

```
as.dthmm(object)
as.mmglm(object)

makedensity(distn)
makedensity1(distn)
makedistn(distn)

getj(x, j)

dglm(x, x1, beta0, beta1, sigma, family, link, size = NA,
     log = FALSE)
pglm(q, x1, beta0, beta1, sigma, family, link, size = NA,
     log = FALSE)
```

Details

The function `makedensity` is used to reparameterise various R probability (or density) functions (e.g. `dnorm` and `dpois`) into a format with a standard argument list. Similarly, `makedistn` reparameterises a distribution function.

The function `getj` is used to extract the jth element from each vector component in a list object.

The function `as.dthmm` coerces an object with class `"mmglm"` to an object with class `"dthmm"`. Similarly, the function `as.mmglm` coerces an object with class `"dthmm"` to an object with class `"mmglm"` (if possible).

The functions `dglm` and `pglm` calculate the density and probability, respectively, for an observation given a generalised linear model.

---

logLik *Log Likelihood of Hidden Markov Model*

Description

Provides methods for the generic function `logLik`.

Usage

```
## S3 method for class 'dthmm':
logLik(object, fortran=TRUE, ...)
## S3 method for class 'mmglm':
logLik(object, fortran=TRUE, ...)
## S3 method for class 'mmpp':
logLik(object, fortran=TRUE, ...)
```

Arguments

`object`	an object with class `"dthmm"`, `"mmglm"` or `"mmpp"`.
`fortran`	logical, if `TRUE` (default) use the Fortran code, else use the R code.
`...`	other arguments.

Details

The methods provided here will always recalculate the log-likelihood even if it is already contained within the `object`. This enables the user to change parameter or data values within the `object` and recalculate the log-likelihood for the revised configuration.

The code for the methods `"dthmm"`, `"mmglm"` and `"mmpp"` can be viewed by typing `logLik.dthmm`, `logLik.mmglm` or `logLik.mmpp`, respectively, on the R command line.

Value

Returns the value of the log-likelihood.

Examples

```
Pi <- matrix(c(1/2, 1/2,   0,
               1/3, 1/3, 1/3,
                 0, 1/2, 1/2),
             byrow=TRUE, nrow=3)

x <- dthmm(NULL, Pi, c(0,1,0), "norm",
           list(mean=c(1, 6, 3), sd=c(1, 0.5, 1)))

x <- simulate(x, nsim=100)

print(logLik(x))
```

---

mchain *Markov Chain Object*

Description

Creates a Markov chain object with class `"mchain"`. It does not simulate data.

Usage

```
mchain(x, Pi, delta, nonstat = TRUE)
```

Arguments

`x`	is a vector of length n containing the observed process, else it is specified as `NULL`. This is used when there are no data and a process is to be simulated.
`Pi`	is the $m \times m$ transition probability matrix of the Markov chain.
`delta`	is the marginal probability distribution of the m state Markov chain at the first time point.
`nonstat`	is logical, `TRUE` if the homogeneous Markov chain is assumed to be non-stationary, default. See "Details" below.

Value

A `list` object with class `"mchain"`, containing the above arguments as named components.

Examples

```
Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

#    Create a Markov chain object with no data (NULL)
x <- mchain(NULL, Pi, c(0,1))

#    Simulate some data
x <- simulate(x, nsim=2000)

#   estimate transition probabilities
estPi <- table(x$mc[-length(x$mc)], x$mc[-1])
rowtotal <- estPi %*% matrix(1, nrow=nrow(Pi), ncol=1)
estPi <- diag(as.vector(1/rowtotal)) %*% estPi
print(estPi)
```

---

mmglm *Markov Modulated GLM Object*

Description

Creates a Markov modulated generalised linear model object with class "mmglm".

Usage

```
mmglm(x, Pi, delta, family, link, beta, glmformula = formula(y~x1),
      sigma = NA, nonstat = TRUE)
```

Arguments

x	a dataframe containing the observed variable (i.e. the response variable in the generalised linear model) and the covariate. Currently, the response variable must be named y and the covariate x1. Alternatively, x could be specified as NULL, meaning that the data will be added later (e.g. simulated). See Details below for the binomial case.
Pi	is the $m \times m$ transition probability matrix of the hidden Markov chain.
delta	is the marginal probability distribution of the m hidden states at the first time point.
family	character string, the GLM family, one of "gaussian", "poisson", "Gamma" or "binomial".
link	character string, the link function. If family == "binomial", then one of "logit", "probit" or "cloglog"; else one of "identity", "inverse" or "log".
beta	a $2 \times m$ matrix containing parameter estimates. The first row contains the m constants in the linear predictor for each Markov state, and the second row contains the linear regression coefficient in the linear predictor for each Markov state.
glmformula	currently the only model formula is y~x1.

sigma	if family == "gaussian", then it is the variance; if family == "Gamma", then it is 1/sqrt(shape); for each Markov state.
nonstat	is logical, TRUE if the homogeneous Markov chain is assumed to be non-stationary, default.

Details

This model assumes that the observed responses are ordered in time, together with a covariate at each point. The model is based on a simple regression model within the glm framework (see McCullagh & Nelder, 1989), but where the coefficients β_0 and β_1 in the linear predictor vary according to a hidden Markov state. The responses are assumed to be conditionally independent given the value of the Markov chain.

If family == "binomial" then the response variable y is interpreted as the number of successes. The dataframe x must also contain a variable called size being the number of Bernoulli trials. This is different to the format used by the function glm where y would be a matrix with two columns containing the number of successes and failures, respectively. The different format here allows one to specify the number of Bernoulli trials *only* so that the number of successes or failures can be simulated later.

When the density function of the response variable is from the exponential family (Charnes et al, 1976, Eq. 2.1), the likelihood function (Charnes et al, 1976, Eq. 2.4) can be maximised by using iterative weighted least squares (Charnes et al, 1976, Eq. 1.1 and 1.2). This is the method used by the R function glm. In this Markov modulated version of the model, the third term of the complete data log-likelihood, as given in Harte (2006, §2.3), needs to be maximised. This is simply the sum of the individual log-likelihood contributions of the response variable weighted by the Markov state probabilities calculated in the E-step. This can also be maximised using iterative least squares by passing these additional weights (Markov state probabilities) into the glm function.

Value

A list object with class "mmglm", containing the above arguments as named components.

References

Charnes, A.; Frome, E.L. & Yu, P.L. (1976). The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *JASA* **71(353)**, 169–171. DOI: http://dx.doi.org/10.2307/2285762

Harte, D. (2006). *Mathematical Background Notes for Package "HiddenMarkov"*. Statistics Research Associates, Wellington. URL: http://homepages.paradise.net.nz/david.harte/SSLib/Manuals/notes.pdf.

McCullagh, P. & Nelder, J.A. (1989). *Generalized Linear Models (2nd Edition)*. Chapman and Hall, London.

Examples

```
delta <- c(0,1)

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

#-----------------------------------------------------------
#     Poisson with log link function
```

```
x <- mmglm(NULL, Pi, delta, family="poisson", link="log",
           beta=rbind(c(0.1, -0.1), c(1, 5)))

x <- simulate(x, nsim=5000, seed=10)

y <- BaumWelch(x)

hist(residuals(y))
print(summary(y))
print(logLik(y))

#-------------------------------------------------------
#     Binomial with logit link function

x <- mmglm(NULL, Pi, delta, family="binomial", link="logit",
           beta=rbind(c(0.1, -0.1), c(1, 5)))

x <- simulate(x, nsim=5000, seed=10)

y <- BaumWelch(x)

hist(residuals(y))
print(summary(y))
print(logLik(y))

#-------------------------------------------------------
#     Gaussian with identity link function

x <- mmglm(NULL, Pi, delta, family="gaussian", link="identity",
           beta=rbind(c(0.1, -0.1), c(1, 5)), sigma=c(1, 2))

x <- simulate(x, nsim=5000, seed=10)

y <- BaumWelch(x)

hist(residuals(y))
print(summary(y))
print(logLik(y))

#-------------------------------------------------------
#     Gamma with log link function

x <- mmglm(NULL, Pi, delta, family="Gamma", link="log",
           beta=rbind(c(2, 1), c(-2, 1.5)), sigma=c(0.2, 0.1))

x1 <- seq(0.01, 0.99, 0.01)
plot(x1, exp(x$beta[1,2] + x$beta[2,2]*x1), type="l",
     xlim=c(0,1),ylim=c(0, 10), col="red", lwd=3)
points(x1, exp(x$beta[1,1] + x$beta[2,1]*x1), type="l",
       col="blue", lwd=3)

x <- simulate(x, nsim=1000, seed=10)

points(x$x$x1, x$x$y)

x$beta[2,] <- c(-3, 4)
y <- BaumWelch(x, bwcontrol(posdiff=FALSE))
```

```
hist(residuals(y))
print(summary(y))
print(logLik(y))
```

`mmpp.misc` *Markov Modulated Poisson Process - 2nd Level Functions*

Description

These functions have not been put into a generic format, but are called by generic functions.

Usage

```
forwardback.mmpp(tau, Q, delta, lambda, fortran = TRUE)
Estep.mmpp(tau, Q, delta, lambda, fortran = TRUE)
```

Arguments

`tau`	vector containing the interevent times. Note that the first event is at time zero.
`Q`	the infinitesimal generator matrix of the Markov process.
`lambda`	a vector containing the Poisson rates.
`delta`	is the marginal probability distribution of the m hidden states at time zero.
`fortran`	logical, if `TRUE` (default) use the Fortran code, else use the R code.

Details

These functions use the algorithm given by Ryden (1996) based on eigenvalue decompositions.

`mmpp.obsolete` *Markov Modulated Poisson Process - Obsolete Functions*

Description

These functions are obsolete and will ultimately be removed from the package. Please change to the revised versions: `BaumWelch`, `Estep.mmpp`, `forwardback.mmpp`, `simulate` or `logLik`.

Usage

```
backward0.mmpp(tau, Q, lambda)
forward0.mmpp(tau, Q, delta, lambda)

logLikmmpp(tau, Q, delta, lambda)

Estep0.mmpp(tau, Q, delta, lambda)

Baum.Welch.mmpp(tau, Q, delta, lambda, nonstat = TRUE,
                maxiter = 500, tol = 1e-05, prt = TRUE,
                converge = expression(diff < tol))
```

```
Baum.Welch0.mmpp(tau, Q, delta, lambda, nonstat = TRUE,
                 maxiter = 500, tol = 1e-05, prt = TRUE,
                 converge = expression(diff < tol))

sim.mmpp(n, initial, Q, lambda)
```

Arguments

tau	vector containing the interevent times. Note that the first event is at time zero.
Q	the infinitesimal generator matrix of the Markov process.
lambda	a vector containing the Poisson rates.
delta	is the marginal probability distribution of the m hidden states at time zero.
n	number of Poisson events to be simulated.
initial	integer, being the initial hidden Markov state $(1, \cdots, m)$.
nonstat	is logical, TRUE if the homogeneous Markov chain is assumed to be non-stationary, default.
maxiter	is the maximum number of iterations, default is 500.
tol	is the convergence criterion, being the difference between successive values of the log-likelihood; default is 0.00001.
prt	is logical, and determines whether information is printed at each iteration; default is TRUE.
converge	is an expression giving the convergence criterion.

Details

The functions with a suffix of zero are non-scaled, and hence will have numerical problems for series containing larger numbers of events; and are *much* slower.

These functions use the algorithm given by Ryden (1996) based on eigenvalue decompositions.

---

mmpp *Markov Modulated Poisson Process Object*

Description

Creates a Markov modulated Poisson process model object with class "mmpp".

Usage

```
mmpp(tau, Q, delta, lambda, nonstat = TRUE)
```

Arguments

tau	vector containing the *event times*. Note that the first event is at time zero. Alternatively, tau could be specified as NULL, meaning that the data will be added later (e.g. simulated).
Q	the infinitesimal generator matrix of the Markov process.
delta	is the marginal probability distribution of the m hidden states at time zero.
lambda	a vector containing the Poisson rates.
nonstat	is logical, TRUE if the homogeneous Markov process is assumed to be non-stationary, default.

Details

The Markov modulated Poisson process is based on a hidden Markov process in continuous time. The initial state probabilities (at time zero) are specified by `delta` and the transition rates by the `Q` matrix. The rate parameter of the Poisson process (`lambda`) is determined by the current state of the hidden Markov process. Within each state, the Poisson process is homogeneous (constant rate parameter). A Poisson event is assumed to occur at time zero and at the end of the observation period, however, state transitions of the Markov process do not necessarily coincide with Poisson events. For more details, see Ryden (1996).

Value

A `list` object with class `"mmpp"`, containing the above arguments as named components.

References

Klemm, A.; Lindemann, C. & Lohmann, M. (2003). Modeling IP traffic using the batch Markovian arrival process. *Performance Evaluation* **54(2)**, 149–173. DOI: http://dx.doi.org/10.1016/S0166-5316(03)00067-1

Roberts, W.J.J.; Ephraim, Y. & Dieguez, E. (2006). On Ryden's EM algorithm for estimating MMPPs. *IEEE Signal Processing Letters* **13(6)**, 373–376. DOI: http://dx.doi.org/10.1109/LSP.2006.871709

Ryden, T. (1994). Parameter estimation for Markov modulated Poisson processes. *Stochastic Models* **10(4)**, 795–829. DOI: http://dx.doi.org/10.1080/15326349408807323

Ryden, T. (1996). An EM algorithm for estimation in Markov-modulated Poisson processes. *Computational Statistics & Data Analysis* **21(4)**, 431–447. DOI: http://dx.doi.org/10.1016/0167-9473(95)00025-9

Examples

```
Q <- matrix(c(-2,  2,
               1, -1),
            byrow=TRUE, nrow=2)/10

#    NULL indicates that we have no data at this point
x <- mmpp(NULL, Q, delta=c(0, 1), lambda=c(5, 1))

x <- simulate(x, nsim=5000, seed=5)

y <- BaumWelch(x)

print(summary(y))

#    log-likelihood using initial parameter values
print(logLik(x))

#    log-likelihood using estimated parameter values
print(logLik(y))
```

`Mstep` *M Step of EM Algorithm*

Description

Performs the *maximisation* step of the EM algorithm for a `dthmm` process. This function is called by the `BaumWelch` function. The Baum-Welch algorithm used in the HMM literature is a version of the EM algorithm.

Usage

```
Mstep.beta(x, cond, pm, pn, maxiter = 200)
Mstep.binom(x, cond, pm, pn)
Mstep.exp(x, cond, pm, pn)
Mstep.gamma(x, cond, pm, pn, maxiter = 200)
Mstep.glm(x, cond, pm, pn, family, link)
Mstep.lnorm(x, cond, pm, pn)
Mstep.logis(x, cond, pm, pn, maxiter = 200)
Mstep.norm(x, cond, pm, pn)
Mstep.pois(x, cond, pm, pn)
```

Arguments

`x`	is a vector of length n containing the observed process.
`cond`	is an object created by `Estep`.
`family`	character string, the GLM family, one "gaussian", "poisson", "Gamma" or "binomial".
`link`	character string, the link function. If `family == "Binomial"`, then one of "logit", "probit" or "cloglog"; else one of "identity", "inverse" or "log".
`pm`	is a list object containing the current (Markov dependent) parameter estimates associated with the distribution of the observed process (see `dthmm`). These are only used as initial values if the algorithm within the `Mstep` is iterative.
`pn`	is a list object containing the observation dependent parameter values associated with the distribution of the observed process (see `dthmm`).
`maxiter`	maximum number of Newton-Raphson iterations.

Details

The functions `Mstep.beta`, `Mstep.binom`, `Mstep.exp`, `Mstep.gamma`, `Mstep.lnorm`, `Mstep.logis`, `Mstep.norm` and `Mstep.pois` perform the maximisation step for the Beta, Binomial, Exponential, Gamma, Log Normal, Logistic, Normal and Poisson distributions, respectively. Each function has the same argument list, even if specific arguments are redundant, because the functions are called from within other functions in a generic like manner. Specific notes for some follow.

`Mstep.beta` The R functions for the Beta distribution have arguments `shape1` and `shape2`; and the density also has `ncp`. We only use `shape1` and `shape2`, i.e. `ncp` is assumed to be zero. Different combinations of `"shape1"` and `"shape2"` can be "time" dependent (specified in `pn`) and Markov dependent (specified in `pm`). However, each should only be specified in one (see topic `dthmm`).

`Mstep.binom` The `size` argument of the binomial distribution should always be specified in the `pn` argument (see topic `dthmm`).

`Mstep.gamma` The R functions for the Gamma distribution have arguments `shape`, `rate` and `scale`. Since `scale` is redundant, we only use `shape` and `rate`. Different combinations of `"shape"` and `"rate"` can be "time" dependent (specified in `pn`) and Markov dependent (specified in `pm`). However, each should only be specified in one (see topic `dthmm`).

`Mstep.lnorm` Different combinations of `"meanlog"` and `"sdlog"` can be "time" dependent (specified in `pn`) and Markov dependent (specified in `pm`). However, each should only be specified in one (see topic `dthmm`).

`Mstep.logis` Different combinations of `"location"` and `"scale"` can be "time" dependent (specified in `pn`) and Markov dependent (specified in `pm`). However, each should only be specified in one (see topic `dthmm`).

`Mstep.norm` Different combinations of `"mean"` and `"sd"` can be "time" dependent (specified in `pn`) and Markov dependent (specified in `pm`). However, each should only be specified in one (see topic `dthmm`).

Value

A list object with the same structure as `pm` (see topic `dthmm`).

Modifications

Consider a distribution with two parameters where both parameters are Markov dependent, but one is known and the other requires estimation. For example, consider the Gaussian distribution. Say we know the Markov dependent means, but we need to estimate the standard deviations. Since both parameters are Markov dependent, they both need to be specified in the `pm` argument. The estimation of the distribution specific parameters takes place in the Mstep, in this case `Mstep.norm`. To achieve what we want, we need to modify this function. In this case it is relatively easy (see code in "Examples" below. From the function `Mstep.norm`, take the code under the section `if (all(nms==c("mean", "sd")))`, i.e. both of the parameters are Markov dependent. However, replace the line where the mean is estimated to `mean <- pm$mean`, i.e. leave it as was initially specified. Then `source` this revised function so that is found by R in preference to the standard version in the package.

One needs to take a little more care when dealing with a distributions like the beta, where the cross derivatives of the log likelihood between the parameters, i.e. $\partial^2 \log L/(\partial\alpha_1\partial\alpha_2)$ are non-zero.

Note

The Mstep functions can be used to estimate the maximum likelihood parameters from a simple sample. See the example below where this is done for the logistic distribution.

See Also

`BaumWelch`, `Estep`

Examples

```
#    Fit logistic distribution to a simple single sample

#    Simulate data
n <- 20000
location <- -2
scale <- 1.5
```

```
x <- rlogis(n, location, scale)

#    give each datum equal weight
cond <- NULL
cond$u <- matrix(rep(1/n, n), ncol=1)

#    calculate maximum likelihood parameter estimates
#    start iterations at values used to simulate
print(Mstep.logis(x, cond,
                  pm=list(location=location,
                          scale=scale)))

#   Example with Gaussian Observations
#   Assume that both mean and sd are Markov dependent, but the means
#   are known and sd requires estimation (See "Modifications" above).

Mstep.norm <- function(x, cond, pm, pn){
    nms <- sort(names(pm))
    n <- length(x)
    m <- ncol(cond$u)
    if (all(nms==c("mean", "sd"))){
        mean <- pm$mean
        sd <- sqrt(apply((matrix(x, nrow = n, ncol=m) -
                  matrix(mean,
                  nrow = n, ncol=m, byrow=TRUE))^2 * cond$u, MARGIN=2,
                  FUN=sum)/apply(cond$u, MARGIN = 2, FUN = sum))
        return(list(mean=mean, sd=sd))
    }
}

Pi <- matrix(c(1/2, 1/2,   0,
               1/3, 1/3, 1/3,
                 0, 1/2, 1/2),
             byrow=TRUE, nrow=3)
p1 <- c(1, 6, 3)
p2 <- c(0.5, 1, 0.5)
n <- 1000

pm <- list(mean=p1, sd=p2)

x <- dthmm(NULL, Pi, c(0, 1, 0), "norm", pm)

x <- simulate(x, n)

#    use above parameter values as initial values
y <- BaumWelch(x)

print(y$delta)
print(y$pm)
print(y$Pi)
```

`neglogLik` *Negative Log-Likelihood*

Description

Calculates the log-likelihood multiplied by negative one. It is in a format that can be used with the functions nlm and optim, providing an alternative to the BaumWelch algorithm for maximum likelihood parameter estimation.

Usage

```
neglogLik(p, object, updatep)
```

Arguments

p	a vector of revised parameter values.
object	an object of class "dthmm", "mmglm", or "mmpp".
updatep	a user provided function mapping the revised parameter values p into the appropriate locations in object.

Details

This function is in a format that can be used with the two functions nlm and optim (see Examples below). This provides alternative methods of estimating the maximum likelihood parameter estimates to the EM provided by BaumWelch including Newton type methods and grid searches. It can also be used to restrict estimation to a subset of parameters.

The EM algorithm is very stable when starting from poor initial values but convergence is very slow in close proximity to the solution. Newton type methods are very sensitive to initial conditions but converge much more quickly in close proximity to the solution. This suggests initially using the EM and then switching to Newton type methods (see Examples below).

The function nlm requires the parameters over which the function is to be maximised to be specified as a vector. Some functions are provided to partially achieve this (see topic Transform.Parameters).

Value

Value of the log-likelihood.

See Also

nlm, optim, Transform.Parameters, BaumWelch

Examples

```
#    Simulate an example dataset

Pi <- matrix(c(0.8, 0.1, 0.1,
               0.1, 0.6, 0.3,
               0.2, 0.3, 0.5),
             byrow=TRUE, nrow=3)

delta <- c(0, 1, 0)

x <- dthmm(NULL, Pi, delta, "exp", list(rate=c(5, 3, 1)))
x <- simulate(x, nsim=5000, seed=5)

#-------------------------------------------------------
#   Fully estimate both Pi and rate
```

```
allmap <- function(y, p){
    #    maps vector back to Pi and rate
    m <- sqrt(length(p))
    y$Pi <- vector2Pi(p[1:(m*(m-1))])
    y$pm$rate <- exp(p[(m^2-m+1):(m^2)])
    return(y)
}

#    Start using the EM algorithm
x1 <- BaumWelch(x, control=bwcontrol(maxiter=1000, tol=0.01))

#    use above as initial values for the nlm function
#    map parameters to a single vector, fixed delta
p <- c(Pi2vector(x1$Pi), log(x1$pm$rate))

#    complete estimation using nlm
z <- nlm(neglogLik, p, object=x, updatep=allmap,
         print.level=2, gradtol=0.000001, iterlim=500)

#    dthmm object with estimated parameter values from nlm
x2 <- allmap(x, z$estimate)

#    compare log-likelihoods
print(logLik(x))
print(logLik(x1))
print(logLik(x2))

#   print final parameter estimates
print(summary(x2))

#-------------------------------------------------------
#   Estimate only the off diagonal elements in the matrix Pi
#   Hold all others as in the simulation

#   This function maps the changeable parameters into the
#   dthmm object - done within the function neglogLik
#   The logit-like transform removes boundaries

offdiagmap <- function(y, p){
    #   rows must sum to one
    invlogit <- function(eta)
        exp(eta)/(1+exp(eta))
    y$Pi[1,2] <- (1-y$Pi[1,1])*invlogit(p[1])
    y$Pi[1,3] <- 1-y$Pi[1,1]-y$Pi[1,2]
    y$Pi[2,1] <- (1-y$Pi[2,2])*invlogit(p[2])
    y$Pi[2,3] <- 1-y$Pi[2,1]-y$Pi[2,2]
    y$Pi[3,1] <- (1-y$Pi[3,3])*invlogit(p[3])
    y$Pi[3,2] <- 1-y$Pi[3,1]-y$Pi[3,3]
    return(y)
}

z <- nlm(neglogLik, c(0, 0, 0), object=x, updatep=offdiagmap,
         print.level=2, gradtol=0.000001)

#    x1 contains revised parameter estimates
x1 <- offdiagmap(x, z$estimate)
```

```
#    print revised values of Pi
print(x1$Pi)

#    print log-likelihood using original and revised values
print(logLik(x))
print(logLik(x1))

#-------------------------------------------------------
#   Fully estimate both Q and lambda for an MMPP Process

Q <- matrix(c(-8,  5,  3,
               1, -4,  3,
               2,  5, -7),
            byrow=TRUE, nrow=3)/25
lambda <- c(5, 3, 1)
delta <- c(0, 1, 0)

#    simulate some data
x <- mmpp(NULL, Q, delta, lambda)
x <- simulate(x, nsim=5000, seed=5)

allmap <- function(y, p){
    #    maps vector back to Pi and rate
    m <- sqrt(length(p))
    y$Q <- vector2Q(p[1:(m*(m-1))])
    y$lambda <- exp(p[(m^2-m+1):(m^2)])
    return(y)
}

#    Start by using the EM algorithm
x1 <- BaumWelch(x, control=bwcontrol(maxiter=1000, tol=0.01))

#    use above as initial values for the nlm function
#    map parameters to a single vector, fixed delta
p <- c(Q2vector(x1$Q), log(x1$lambda))

#    complete estimation using nlm
z <- nlm(neglogLik, p, object=x, updatep=allmap,
         print.level=2, gradtol=0.000001, iterlim=500)

#    mmpp object with estimated parameter values from nlm
x2 <- allmap(x, z$estimate)

#    compare log-likelihoods
print(logLik(x))
print(logLik(x1))
print(logLik(x2))

#   print final parameter estimates
print(summary(x2))
```

`Overview` *Overview of Package*

Description

In this topic we give an overview of the package.

Classes of Hidden Markov Models Analysed

The classes of models currently fitted by the package are listed below. Each are defined within an object that contains the data, current parameter values, and other model characteristics.

Discrete Time Hidden Markov Model: is described under the topic `dthmm`. This model can be simulated or fitted to data by defining the required model structure within an object of class `"dthmm"`.

Markov Modulated Generalised Linear Model: is described under the topic `mmglm`. This model can be simulated or fitted to data by defining the required model structure within an object of class `"mmglm"`.

Markov Modulated Poisson Process: is described under the topic `mmpp`. This model can be simulated or fitted to data by defining the required model structure within an object of class `"mmpp"`.

Main Tasks Performed by the Package

The main tasks performed by the package are listed below. These can be achieved by calling the appropriate generic function.

Simulation of HMMs: can be performed by the function `simulate`.

Parameter Estimation: can be performed by the functions `BaumWelch` (EM algorithm), or `neglogLik` together with `nlm` or `optim` (Newton type methods or grid searches).

Model Residuals: can be extracted with the function `residuals`.

Model Summary: can be extracted with the function `summary`.

Log-Likelihood: can be calculated with the function `logLik`.

Prediction of the Markov States: can be performed by the function `Viterbi`.

All other functions in the package are called from within the above generic functions, and only need to be used if their output is specifically required.

Acknowledgement

Many of the functions contained in the package are based on those of Walter Zucchini (2005).

References

Zucchini, W. (2005). *Hidden Markov Models Short Course, 3–4 April 2005.* Macquarie University, Sydney.

probhmm *Conditional Distribution Function*

Description

Calculates the distribution function at each point for a dthmm process given the complete observed process except the given point.

Usage

```
probhmm(x, Pi, delta, distn, pm, pn = NULL, adj = 0, ...)
```

Arguments

x	is a vector of length n containing the observed process.
Pi	is the $m \times m$ transition probability matrix of the hidden Markov chain.
delta	is the marginal probability distribution of the m hidden states at the first time point.
distn	is a character string with the distribution name, e.g. "norm" or "pois". If the distribution is specified as "wxyz" then a distribution function called "pwxyz" should be available, in the standard R format (e.g. pnorm or ppois).
pm	is a list object containing the (Markov dependent) parameter values associated with the distribution of the observed process (see dthmm).
pn	is a list object containing the observation dependent parameter values associated with the distribution of the observed process (see dthmm).
adj	zero or one, being an adjustment for discrete distributions.
...	other arguments. This allows more complicated objects with the required arguments, but with redundant components also, to be passed to this function.

Details

Let $X^{(-i)}$ denote the entire process, except with the point X_i removed. The distribution function at the point X_i is $\Pr\{X_i \leq x_i \,|\, X^{(-i)} = x^{(-i)}\}$. This R function calculates the distribution function for each point X_i for $i = 1, \cdots, n$. This is done by using the forward and backward probabilities before and after the ith point, respectively.

In the programming code, note the subtraction of the mean. This is to stop underflow when the exponential is taken. Removal of the mean is automatically compensated for by the fact that the same factor is removed in both the numerator and denominator.

Value

A vector containing the probability.

References

Zucchini, W. (2005). *Hidden Markov Models Short Course, 3–4 April 2005.* Macquarie University, Sydney.

See Also

residuals

`residuals` *Residuals of Hidden Markov Models*

Description

Provides methods for the generic function `residuals`. There is currently no method for objects of class "`mmpp`".

Usage

```
## S3 method for class 'dthmm':
residuals(object, ...)
## S3 method for class 'mmglm':
residuals(object, ...)
```

Arguments

`object`	an object with class `dthmm` or `mmglm`. There is not yet a method for `mmpp`.
`...`	other arguments.

Details

For objects of class "`dthmm`" or "`mmglm`" the calculated residuals are pseudo residuals. Under satisfactory conditions they have an approximate standard normal distribution.

Initially the function `probhmm` is called. If the model fits satisfactorily, the returned values should be approximately uniformly distributed. Hence by applying the function `qnorm`, the resultant "residuals" should have an approximate standard normal distribution.

When the distribution (observed) is discrete an adjustment is made. However, if relatively few of the possible outcomes are observed, the pseudo residuals will be rather poorly described by the standard normal distribution; see the Poisson example below.

The code for the methods "`dthmm`" and "`mmglm`" can be viewed by typing `residuals.dthmm` or `residuals.mmglm`, respectively, on the R command line.

Value

A vector containing the pseudo residuals.

Examples

```
#   Example Using Beta Distribution

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

n <- 2000

x <- dthmm(NULL, Pi, c(0,1), "beta",
           list(shape1=c(2, 6), shape2=c(6, 2)))

x <- simulate(x, nsim=n, seed=5)
```

```
y <- residuals(x)

w <- hist(y, main="Beta HMM: Pseudo Residuals")
z <- seq(-3, 3, 0.01)
points(z, dnorm(z)*n*(w$breaks[2]-w$breaks[1]), col="red", type="l")
box()

qqnorm(y, main="Beta HMM: Q-Q Plot of Pseudo Residuals")
abline(a=0, b=1, lty=3)
abline(h=seq(-2, 2, 1), lty=3)
abline(v=seq(-2, 2, 1), lty=3)

#-----------------------------------------------
#   Example Using Gaussian Distribution

Pi <- matrix(c(1/2, 1/2,   0,   0,   0,
               1/3, 1/3, 1/3,   0,   0,
                 0, 1/3, 1/3, 1/3,   0,
                 0,   0, 1/3, 1/3, 1/3,
                 0,   0,   0, 1/2, 1/2),
             byrow=TRUE, nrow=5)

x <- dthmm(NULL, Pi, c(0, 1, 0, 0, 0), "norm",
           list(mean=c(1, 4, 2, 5, 3), sd=c(0.5, 1, 1, 0.5, 0.1)))

n <- 2000
x <- simulate(x, nsim=n, seed=5)

y <- residuals(x)

w <- hist(y, main="Gaussian HMM: Pseudo Residuals")
z <- seq(-3, 3, 0.01)
points(z, dnorm(z)*n*(w$breaks[2]-w$breaks[1]), col="red", type="l")
box()

qqnorm(y, main="Gaussian HMM: Q-Q Plot of Pseudo Residuals")
abline(a=0, b=1, lty=3)
abline(h=seq(-2, 2, 1), lty=3)
abline(v=seq(-2, 2, 1), lty=3)

#-----------------------------------------------
#   Example Using Poisson Distribution

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

x <- dthmm(NULL, Pi, c(0, 1), "pois",
           list(lambda=c(1, 5)), discrete=TRUE)

n <- 2000
x <- simulate(x, nsim=n, seed=5)

y <- residuals(x)

w <- hist(y, main="Poisson HMM: Pseudo Residuals")
```

```
z <- seq(-3, 3, 0.01)
points(z, dnorm(z)*n*(w$breaks[2]-w$breaks[1]), col="red", type="l")
box()

qqnorm(y, main="Poisson HMM: Q-Q Plot of Pseudo Residuals")
abline(a=0, b=1, lty=3)
abline(h=seq(-2, 2, 1), lty=3)
abline(v=seq(-2, 2, 1), lty=3)
```

`simulate` *Simulate Various HMM Processes*

Description

These functions provide methods for the generic function `simulate`.

Usage

```
## S3 method for class 'dthmm':
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mchain':
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mmglm':
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mmpp':
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

`object`	an object with class `"dthmm"`, `"mchain"`, `"mmglm"` or `"mmpp"`
`nsim`	number of points to simulate.
`seed`	seed for the random number generator.
`...`	other arguments.

Details

Below details about particular methods are given where necessary.

`simulate.mmglm` If the covariate `x1` is `NULL`, then uniform (0,1) variables are generated as the values for `x1`. When the `family` is `"binomial"` and `size` is `NULL` (i.e. the number of Bernoulli trials are not specified), then they are simulated as `100+rpois(nsim, lambda=5)`.

The code for the methods `"dthmm"`, `"mmglm"` and `"mmpp"` can be viewed by typing `simulate.dthmm`, `simulate.mmglm` or `simulate.mmpp`, respectively, on the R command line.

Value

The returned object has the same class as the input object and contains the components that were in the input object. Where `object` is of class `"dthmm"` it will also have a vector `x` containing the simulated values; and when the class is `"mmglm"` `x` will be a dataframe. When the class is `"mmpp"` the times of the simulated Poisson events are contained in `tau`. Other variables are also added like the sequence of Markov states, and the time spent in each state (`"mmpp"`).

Examples

```
#    The hidden Markov chain has 5 states with transition matrix:

Pi <- matrix(c(1/2, 1/2,   0,   0,   0,
               1/3, 1/3, 1/3,   0,   0,
                 0, 1/3, 1/3, 1/3,   0,
                 0,   0, 1/3, 1/3, 1/3,
                 0,   0,   0, 1/2, 1/2),
             byrow=TRUE, nrow=5)

#-----------------------------------------
#   simulate a Poisson HMM

x <- dthmm(NULL, Pi, c(0, 1, 0, 0, 0), "pois",
           list(lambda=c(1, 4, 2, 5, 3)), discrete = TRUE)

x <- simulate(x, nsim=2000)

#    check Poisson means
for (i in 1:5) print(mean(x$x[x$y==i]))

#-----------------------------------------
#   simulate a Gaussian HMM

x <- dthmm(NULL, Pi, c(0, 1, 0, 0, 0), "norm",
           list(mean=c(1, 4, 2, 5, 3), sd=c(0.5, 1, 1, 0.5, 0.1)))

x <- simulate(x, nsim=2000)

#    check means and standard deviations
for (i in 1:5) print(mean(x$x[x$y==i]))
for (i in 1:5) print(sd(x$x[x$y==i]))
```

`summary` *Summary Methods for Hidden Markov Model Objects*

Description

Provides methods for the generic function `summary`.

Usage

```
## S3 method for class 'dthmm':
summary(object, ...)
## S3 method for class 'mmglm':
summary(object, ...)
## S3 method for class 'mmpp':
summary(object, ...)
```

Arguments

`object`	an object with class "`dthmm`", "`mmglm`" or "`mmpp`".
`...`	other arguments.

Details

The code for the methods "dthmm", "mmglm" and "mmpp" can be viewed by typing summary.dthmm, summary.mmglm or summary.mmpp, respectively, on the R command line.

Value

A list object with a reduced number of components, mainly the parameter values.

Examples

```
Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)

x <- dthmm(NULL, Pi, c(0, 1), "beta",
           list(shape1=c(2, 6), shape2=c(6, 2)))

x <- simulate(x, nsim=2000)

print(summary(x))
```

Transform.Parameters

Transform Transition or Rate Matrices to Vector

Description

These functions transform m by m transition probability matrices or Q matrices to a vector of length $m(m-1)$, and back. See Details.

Usage

```
Pi2vector(Pi)
vector2Pi(p)

Q2vector(Q)
vector2Q(p)
```

Arguments

Pi	an m by m transition probability matrix.
Q	an m by m rate matrix.
p	a vector of length $m(m-1)$.

Details

The function Pi2vector maps the m by m transition probability matrix of a discrete time HMM to a vector of length $m(m-1)$, and vector2Pi has the reverse effect. They use a logit like transformation so that the parameter space is on the whole real line thus avoiding hard boundaries which cause problems for many optimisation procedures (see neglogLik).

Similarly, the function `Q2vector` maps the m by m rate matrix Q of an MMPP process to a vector of length $m(m-1)$, and `vector2Q` has the reverse effect. They use a log transformation so that the parameter space is on the whole real line thus avoiding hard boundaries which cause problems for many optimisation procedures (see `neglogLik`).

Value

The functions `Pi2vector` and `Q2vector` return a vector of length $m(m-1)$, the function `vector2Pi` returns an m by m transition probability matrix, and `vector2Q` returns an m by m rate matrix Q.

See Also

`neglogLik`

Examples

```
Pi <- matrix(c(0.8, 0.1, 0.1,
               0.1, 0.6, 0.3,
               0.2, 0.3, 0.5),
             byrow=TRUE, nrow=3)

print(vector2Pi(Pi2vector(Pi)))

#-----------------------------------------------

Q <- matrix(c(-8,  5,  3,
               1, -4,  3,
               2,  5, -7),
            byrow=TRUE, nrow=3)

print(vector2Q(Q2vector(Q)))
```

Viterbi *Viterbi Algorithm for Hidden Markov Model Objects*

Description

Provides methods for the generic function `Viterbi`. This predicts the most likely sequence of Markov states given the observed dataset. There is currently no method for objects of class `"mmpp"`.

Usage

```
## S3 method for class 'dthmm':
Viterbi(object, ...)
## S3 method for class 'mmglm':
Viterbi(object, ...)
```

Arguments

`object`	an object with class `"dthmm"` or `"mmglm"`.
`...`	other arguments.

Details

The purpose of the Viterbi algorithm is to *globally decode* the underlying hidden Markov state at each time point. It does this by determining the sequence of states $(k_1^*, \cdots, k_n^*)$ which maximises the joint distribution of the hidden states given the entire observed process, i.e.

$$(k_1^*, \cdots, k_n^*) = \underset{k_1,\cdots,k_n \in \{1,2,\cdots,m\}}{\text{argmax}} \Pr\{C_1 = k_1, \cdots, C_n = k_n \,|\, X^{(n)} = x^{(n)}\}\,.$$

The algorithm has been taken from Zucchini (2005), however, we calculate sums of the logarithms of probabilities rather than products of probabilities. This lessens the chance of numerical underflow. Given that the logarithmic function is monotonically increasing, the *argmax* will still be the same. Note that *argmax* can be evaluated with the R function `which.max`.

Determining the *a posteriori* most probable state at time i is referred to as *local decoding*, i.e.

$$k_i^* = \underset{k \in \{1,2,\cdots,m\}}{\text{argmax}} \Pr\{C_i = k \,|\, X^{(n)} = x^{(n)}\}\,.$$

Note that the above probabilities are calculated by the function `Estep`, and are contained in `u[i,j]` (output from `Estep`), i.e. k_i^* is simply `which.max(u[i,])`.

The code for the methods `"dthmm"` and `"mmglm"` can be viewed by typing `Viterbi.dthmm` or `Viterbi.mmglm`, respectively, on the R command line.

Value

A vector of length n containing integers $(1, \cdots, m)$ representing the hidden Markov states for each node of the chain.

Examples

```
Pi <- matrix(c(1/2, 1/2,   0,   0,   0,
               1/3, 1/3, 1/3,   0,   0,
                 0, 1/3, 1/3, 1/3,   0,
                 0,   0, 1/3, 1/3, 1/3,
                 0,   0,   0, 1/2, 1/2),
             byrow=TRUE, nrow=5)
delta <- c(0, 1, 0, 0, 0)
lambda <- c(1, 4, 2, 5, 3)
m <- nrow(Pi)

x <- dthmm(NULL, Pi, delta, "pois", list(lambda=lambda), discrete=TRUE)
x <- simulate(x, nsim=2000)

#------  Global Decoding  ------

states <- Viterbi(x)
states <- factor(states, levels=1:m)

#  Compare predicted states with true states
#  p[j,k] = Pr{Viterbi predicts state k | true state is j}
p <- matrix(NA, nrow=m, ncol=m)
for (j in 1:m){
    a <- (x$y==j)
    p[j,] <- table(states[a])/sum(a)
}
print(p)
```

```
#------  Local Decoding  ------

#   locally decode at i=100

print(which.max(Estep(x$x, Pi, delta, "pois", list(lambda=lambda))$u[100,]))

#-------------------------------------------------
#   simulate a beta HMM

Pi <- matrix(c(0.8, 0.2,
               0.3, 0.7),
             byrow=TRUE, nrow=2)
delta <- c(0, 1)

y <- seq(0.01, 0.99, 0.01)
plot(y, dbeta(y, 2, 6), type="l", ylab="Density", col="blue")
points(y, dbeta(y, 6, 2), type="l", col="red")

n <- 100
x <- dthmm(NULL, Pi, delta, "beta",
           list(shape1=c(2, 6), shape2=c(6, 2)))
x <- simulate(x, nsim=n)

#   colour denotes actual hidden Markov state
plot(1:n, x$x, type="l", xlab="Time", ylab="Observed Process")
points((1:n)[x$y==1], x$x[x$y==1], col="blue", pch=15)
points((1:n)[x$y==2], x$x[x$y==2], col="red", pch=15)

states <- Viterbi(x)
#   mark the wrongly predicted states
wrong <- (states != x$y)
points((1:n)[wrong], x$x[wrong], pch=1, cex=2.5, lwd=2)
```

Index