Package ‘HiddenMarkov’

November 1, 2017

Title Hidden Markov Models

Version 1.8-11

Date 2017-11-01

Author David Harte

Maintainer David Harte <d.s.harte@gmail.com>

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 ``HiddenMarkov'' for an introduction to the package, and ``Change Log'' for a list of recent changes. The algorithms are based of those of Walter Zucchini.

Suggests parallel

License GPL (>= 2)

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-11-01 16:45:51 UTC

R topics documented:

`HiddenMarkov-package` *Overview of Package HiddenMarkov*

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 `mmglm1`.

Markov Modulated Generalised Linear Longitudinal Model: is described under the topic `mmglmlong1`.

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. We have referred to some of these other functions as “2nd level” functions, for example see the topic `mmpp-2nd-level-functions`.

Organisation of Topics in the Package

Cited References: anywhere in the manual are only listed within this topic.

General Documentation: topics summarising general structure are indexed under the keyword "documentation" in the Index.

Acknowledgement

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

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: 10.1214/aoms/1177697196

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

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. URL: `http://www.jstor.org/stable/2984875`

Elliott, R.J.; Aggoun, L. & Moore, J.B. (1994). *Hidden Markov Models: Estimation and Control.* Springer-Verlag, New York. doi: 10.1007/9780387848549

Harte, D. (2006). *Mathematical Background Notes for Package "HiddenMarkov".* Statistics Research Associates, Wellington. URL: `ftp://ftp.gns.cri.nz/pub/davidh/sslib/other/notes.pdf`

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

Klemm, A.; Lindemann, C. & Lohmann, M. (2003). Modeling IP traffic using the batch Markovian arrival process. *Performance Evaluation* **54(2)**, 149–173. doi: 10.1016/S01665316(03)000671

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

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

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: 10.1109/5.18626

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: 10.1109/LSP.2006.871709

Ryden, T. (1994). Parameter estimation for Markov modulated Poisson processes. *Stochastic Models* **10(4)**, 795–829. doi: 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: 10.1016/01679473(95)000259

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

BaumWelch *Estimation 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 'mmglm0'
BaumWelch(object, control = bwcontrol(), ...)
## S3 method for class 'mmglm1'
BaumWelch(object, control = bwcontrol(), ...)
## S3 method for class 'mmglmlong1'
BaumWelch(object, control = bwcontrol(), PSOCKcluster=NULL,
          tmpfile=NULL, ...)
## S3 method for class 'mmpp'
BaumWelch(object, control = bwcontrol(), ...)
```

Arguments

`object`	an object of class "`dthmm`", "`mmglm0`", "`mmglm1`", "`mmglmlong1`", or "`mmpp`".
`control`	a list of control settings for the iterative process. These can be changed by using the function `bwcontrol`.
`PSOCKcluster`	see section below called "Parallel Processing".
`tmpfile`	name of a file (.Rda) into which estimates are written at each 10th iteration. The model object is called `object`. If `NULL` (default), no file is created.
`...`	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`", "`mmglm0`", "`mmglm1`","`mmglmlong1`" and "`mmpp`" can be viewed by appending `BaumWelch.dthmm`, `BaumWelch.mmglm0`, `BaumWelch.mmglm1`, `BaumWelch.mmglmlong1` or `BaumWelch.mmpp`, respectively, to `HiddenMarkov:::`, on the R command line; e.g. `HiddenMarkov:::dthmm`. The three colons are needed because these method functions are not in the exported NAMESPACE.

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.

Parallel Processing

In longitudinal models, the forward and backward equations need to be calculated for each individual subject. These can be done independently, the results being concatenated to be used in the E-step. If the argument PSOCKcluster is set, subjects are divided equally between each node in the cluster for the calculation of the forward and backward equations. This division is very basic, and assumes that all nodes run at a roughly comparable speed.

If the communication between nodes is slow and the dataset is small, then the time taken to allocate the work to the various nodes may in fact take more time than simply using one processor to perform all of the calculations.

The required steps in initiating parallel processing are as follows.

```
#   load the "parallel" package
library(parallel)

#   define the SNOW cluster object, e.g. a SOCK cluster
#   where each node has the same R installation.
cl <- makePSOCKcluster(c("localhost", "horoeka.localdomain",
                         "horoeka.localdomain", "localhost"))

#   A more general setup: Totara is Fedora, Rimu is Debian:
#   Use 2 processors on Totara, 1 on Rimu:
totara  <- list(host="localhost",
                rscript="/usr/lib/R/bin/Rscript",
                snowlib="/usr/lib/R/library")
rimu    <- list(host="rimu.localdomain",
                rscript="/usr/lib/R/bin/Rscript",
                snowlib="/usr/local/lib/R/site-library")
cl <- makeCluster(list(totara, totara, rimu), type="SOCK")

#   then define the required model object
#   say the model object is called x
```

```
BaumWelch(x, PSOCKcluster=cl)

#   stop the R jobs on the slave machines
stopCluster(cl)
```

Note that the communication method does not need to be SOCKS; see the **parallel** package documentation, topic `makeCluster`, for other options. Further, if some nodes are on other machines, the firewalls may need to be tweaked. The master machine initiates the R jobs on the slave machines by communicating through port 22 (use of security keys are needed rather than passwords), and subsequent communications through port 10187. Again, these details can be tweaked in the options settings within the **parallel** package.

References

Cited references are listed on the HiddenMarkov manual page.

See Also

`logLik`, `residuals`, `simulate`, `summary`, `neglogLik`

---

bwcontrol *Control Parameters for 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)
```

Change Log *Changes Made to Package HiddenMarkov*

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, mmglm0 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 HiddenMarkov). (14 Sep 2007)
2. The main tasks (model fitting, residuals, simulation, Viterbi, etc) can now be called by generic functions (see topic HiddenMarkov). 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.mmglm0`, `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)
17. `dthmm`: argument `nonstat` was not set correctly. (21 Jun 2008)
18. Hyperlinks on package vignettes page. (22 Jun 2008)
19. `mmpp`: argument `nonstat` was not set correctly. (23 Jun 2008)
20. The manual pages `HiddenMarkov-dthmm-deprecated` and `HiddenMarkov-mmpp-deprecated` have been given a keyword of "internal". This hides them from the listing of package functions. (3 Jul 2008)
21. All cited references are now only listed in the topic `HiddenMarkov`. (3 Jul 2008)
22. `neglogLik`: argument `updatep` has been renamed to `pmap`. (9 Jul 2008)
23. `neglogLik`: format of this function changed to be consistent with that in package **PtProcess**. Argument `p` renamed as `params`. (07 Aug 2008)
24. `mmglm0`: remove some LaTeX specific formatting to be compatible with R 2.9.0. (26 Jan 2009)
25. `Viterbi`: Correct hyperlink to base function `which.max`. (10 Oct 2009)
26. Tidied HTML representation of equations in manual pages. (15 Dec 2009)
27. `mmglm`: Renamed to `mmglm0`, new version `mmglm1`. See manual page for more details. (5 Jan 2010)
28. `mmglmlong1`: new function for longitudinal data. (18 Jan 2010)
29. `dthmm`: clarify argument `distn` on manual page, and nature of parameter estimates when the Markov chain is stationary. (04 Feb 2010)
30. `BaumWelch.mmglmlong1`: new argument `tmpfile` added. (13 Feb 2010)
31. `Viterbi`: Methods added for objects of class `"mmglm1"` and `"mmglmlong1"`. (29 Jul 2010)
32. `logLik`: Method added for object of class `"mmglmlong1"`. (30 Jul 2010)
33. `forwardback.dthmm`, `forwardback.mmpp`: New argument `"fwd.only"`. (30 Jul 2010)
34. `logLik.dthmm`: Calls `forwardback.dthmm` to perform calculations. (30 Jul 2010)
35. `logLik.mmpp`: Calls `forwardback.mmpp` to perform calculations. (30 Jul 2010)
36. `Viterbi`: Now generates an error message when applied to objects of class `"mmpp"`. Method not currently available. (03 Aug 2010)
37. `"Viterbi.mmglm1"`: Fixed bug with number of Bernoulli trials specification when using a binomial family. (05 Aug 2010)

38. `residuals.dthmm`, `probhmm`: Modify for greater efficiency and generality to accommodate more general models. Arguments of `probhmm` have also been changed. (05 Aug 2010)
39. `residualshmm`: Made defunct, incompatible with revised `probhmm`. (05 Aug 2010)
40. `residuals`: Methods added for objects of class `"mmglm1"` and `"mmglmlong1"`. Generates an error message when applied to objects of class `"mmpp"`, currently no method available. (05 Aug 2010)
41. Add CITATION file. (24 Sep 2010)
42. `makedistn`: Change `eval(parse(text=paste(x, " list(log=log))", sep="")))` to `eval(parse(text=paste(x, " list(log.p=log)))", sep="")))`. (19 Dec 2010)
43. `pglm`, `pmmglm`: Change all `log` arguments to `log.p`. (19 Dec 2010)
44. Revise examples in `/tests` directory. (02 May 2011)
45. Implement very basic NAMESPACE and remove file `/R/zzz.R`. (5 Nov 2011)
46. List functions explicitly in NAMESPACE. (19 Dec 2011)
47. `mmglm` and `neglogLik`: Restrict the number of iterations in examples on manual pages to minimise time during package checks. (19 Dec 2011)
48. `modify.func`: New function to allow the user to modify package functions in the NAMESPACE set-up. This function violates the CRAN policy as users are not supposed to change the NAMESPACE on such packages. Some examples where it is required to modify package functions will not work, for example, the second example in `Mstep`. (7 Mar 2012)
49. `modify.func`: Function removed. See the second example in `Mstep` for a work-around when package functions need to be modified. (14 Apr 2012)
50. `Mstep`: Revised documentation about distribution requirements and ways to include other distributions into the software framework. (14 Apr 2012)
51. The package **snow** has been superseded by **parallel**, changed where needed. In `BaumWelch.mmglmlong1` arguments `makeSOCKcluster` and `SNOWcluster` renamed to `makePSOCKcluster` and `PSOCKcluster`, respectively. Functions `dmmglm` and `pmmglm` added to exported namespace (required for parallel processing). (13 Aug 2014)
52. `BaumWelch.mmglmlong1`: Call to `clusterApply` and `clusterExport` changed to `parallel::clusterApply` and `parallel::clusterExport`, respectively. (25 Sep 2014)
53. Fix error in `inst/CITATION` file. (21 Jan 2015)
54. Added to NAMESPACE: functions imported from **stats**. (06 Jul 2015)
55. `HiddenMarkov`: Add DOI to some references, rename topic to appear first in table of contents. (16 Oct 2015)
56. Fortran warning: in file `src/extract.f`, `integer` definitions should precede `double precision` definitions. (29 Aug 2016)
57. Fix NOTES in `R CMD check --as-cran`: `Found no calls to: 'R_registerRoutines', 'R_useDynamicSymbols'` (17 Jun 2017)
58. `simulate.mchain`: Change `if (sum(object$delta)!=1)` to `if (!isTRUE(all.equal(sum(object$delta), 1)))`. (21 Oct 2017)
59. `simulate.mmpp`: Change `if (sum(object$delta)!=1)` to `if (!isTRUE(all.equal(sum(object$delta), 1)))`. (27 Oct 2017)
60. Clarify various points in documentation. (27 Oct 2017)

Future Development

1. The functions Viterbi and residuals need methods for objects of 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 mmglm1, mmglmlong1 and mmpp.

---

compdelta — *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` *Discrete Time HMM Object (DTHMM)*

Description

Creates a discrete time hidden Markov model object with class `"dthmm"`. The observed process is univariate.

Usage

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

Arguments

`x`	is a vector of length n containing the univariate 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 abbreviated distribution name. Distributions provided by the package are Beta ("beta"), Binomial ("binom"), Exponential ("exp"), GammaDist ("gamma"), Lognormal ("lnorm"), Logistic ("logis"), Normal ("norm"), and Poisson ("pois"). See topic Mstep, Section "Modifications and Extensions", to extend to other distributions.
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 section "Stationarity" 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.

Stationarity

When the hidden Markov chain is assumed to be non-stationary, the complete data likelihood has a neat structure, in that δ only occurs in the first term, Π only occurs in the second term, and the parameters associated with the observed probabilities only occur in the third term. Hence, the likelihood can easily be maximised by maximising each term individually. In this situation, the estimated parameters using `BaumWelch` will be the "exact" maximum likelihood estimates.

When the hidden Markov chain is assumed to be stationary, $\delta = \Pi'\delta$ (see topic `compdelta`), and then the first two terms of the complete data likelihood determine the transition probabilities Π. This raises more complicated numerical problems, as the first term is effectively a constraint. In our implementation of the EM algorithm, 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 $\delta = \delta\Pi$. Hence, using the `BaumWelch` function will only provide approximate maximum likelihood estimates. Exact solutions can be calculated by directly maximising the likelihood function, see first example in `neglogLik`.

References

Cited references are listed on the HiddenMarkov manual page.

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 for DTHMM*

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

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

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

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

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

Cited references are listed on the HiddenMarkov manual page.

See Also

BaumWelch, Mstep

`forwardback` *Forward and Backward Probabilities of DTHMM*

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)
forwardback.dthmm(Pi, delta, prob, fortran = TRUE, fwd.only = FALSE)
```

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`).
`prob`	an $n \times m$ matrix containing the observation probabilities or densities (rows) by Markov state (columns).
`fortran`	logical, if `TRUE` (default) use the Fortran code, else use the R code.
`fwd.only`	logical, if `FALSE` (default) calculate both forward and backward probabilities; else calculate and return only forward probabilities and log-likelihood.

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 (log) probabilities, `logalpha` and `logbeta`, respectively, and the log-likelihood (`LL`).

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

Author(s)

The algorithm has been taken from Zucchini (2005).

References

Cited references are listed on the HiddenMarkov manual page.

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)
```

---

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 'mmglm0'
logLik(object, fortran=TRUE, ...)
## S3 method for class 'mmglm1'
logLik(object, fortran=TRUE, ...)
## S3 method for class 'mmglmlong1'
logLik(object, fortran=TRUE, ...)
## S3 method for class 'mmpp'
logLik(object, fortran=TRUE, ...)
```

Arguments

`object`	an object with class "`dthmm`", "`mmglm0`", "`mmglm1`", "`mmglmlong1`" 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`", "`mmglm0`", "`mmglm1`","`mmglmlong1`" and "`mmpp`" can be viewed by appending `logLik.dthmm`, `logLik.mmglm0`, `logLik.mmglm1`, `logLik.mmglmlong1` or `logLik.mmpp`, respectively, to `HiddenMarkov:::`, on the R command line; e.g. `HiddenMarkov:::dthmm`. The three colons are needed because these method functions are not in the exported NAMESPACE.

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

These functions create Markov modulated generalised linear model objects. ***These functions are in development and may change***, see "Under Development" below.

Usage

```
mmglm0(x, Pi, delta, family, link, beta, glmformula = formula(y~x1),
       sigma = NA, nonstat = TRUE, msg = TRUE)
mmglm1(y, Pi, delta, glmfamily, beta, Xdesign,
       sigma = NA, nonstat = TRUE, size = NA, msg = TRUE)
mmglmlong1(y, Pi, delta, glmfamily, beta, Xdesign, longitude,
           sigma = NA, nonstat = TRUE, size = NA, msg = TRUE)
```

Arguments

x	a dataframe containing the observed variable (i.e. the response variable in the generalised linear model) and the covariate. The function `mmglm0` requires that the response variable 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. The functions `mmglm1` and `mmglmlong1` do not have these naming restrictions.
y	numeric vector, response variable. In the case of binomial, it is the number of successes (see argument `size`).
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"`.
glmfamily	a `family` object defining the glm family and link function. It is currently restricted to Gaussian, Poisson, Binomial or Gamma models with the standard link functions provided by `glm`.

Xdesign	a $nN \times p$ design matrix, where p is the number of parameters in the linear predictor, N is the number of subjects ($N = 1$ in mmglm1), and n is the number of observations for each subject (*assumed to be the same*).
beta	a $p \times m$ matrix containing parameter values, used as initial values during estimation. In the case of the simple regression model of mmglm0, $p = 2$. In the case of mmglm1 and mmglmlong1, p is the number of columns of Xdesign.
glmformula	the only model formula for mmglm0 is y~x1. Note that the functions mmglm1 and mmglmlong1 do not have this restriction, however, in those cases, the model formula is currently implicitly defined through Xdesign.
sigma	if family == "gaussian", then it is the variance; if family == "Gamma", then it is 1/sqrt(shape). It is of length m for each Markov state.
nonstat	is logical, TRUE if the homogeneous Markov chain is assumed to be non-stationary, default.
longitude	a vector the same length as y identifying the subject for each observation. The observations must be grouped by subject, and ordered by "time" within subject.
size	is number of Bernoulli trials in each observation when the glm family is binomial. It is the same length as y.
msg	is logical, suppress messages about developmental status.

Details

This family of models is similar in nature to those of the class dthmm, in that both classes have the distribution of the observed variable being "modulated" by the changing hidden Markov state. They differ slightly in the mechanism. This family assumes that the mean of the observation distribution can be expressed as a linear model of other known variables, but it is the parameters in the linear predictor that are being modulated by the hidden Markov process, thus causing the changes in the observed means. The linear model is assumed to be a generalised linear model as described by McCullagh & Nelder (1989).

The function mmglm0 is a very simple trivial case where the linear predictor is of the form $\beta_0 + \beta_1 x_1$. The version mmglm1 does not have this limitation. The model formula for mmglm1 is defined implicitly through the structure of the specified design matrix. The model mmglmlong1 is similar to mmglm1 but can be applied to longitudinal observations. Models of the form given by mmglm1 are assumed to have one time series, and from a theoretical perspective, one would be interested in the asymptotic properties of the parameter estimates as the series length gets very large. In the longitudinal case (mmglmlong1), the series of observations per individual is probably very small (< 10), and hence interest is in the asymptotic properties as the number of individuals becomes large. Note that in the longitudinal case, the number of observations per individual is assumed to be the same. The responses are assumed to be conditionally independent given the value of the Markov chain and the explanatory variables in the linear predictor.

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, Sec. 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 `"mmglm0"`, containing the above arguments as named components.

Under Development

These functions are still being developed. In previous releases of the package ($<$ 1.3), there was only one function called `mmglm`. This has been renamed to `mmglm0`. The most recent version is `mmglm1` along with `mmglmlong1` which has flexibility to include longitudinal data. Further development versions will be numbered sequentially. The name `mmglm` has been reserved for the final stable version, at which point the numbered versions will become deprecated.

References

Cited references are listed on the HiddenMarkov manual page.

Examples

```
#--------------------------------------------------------
#     Gaussian with identity link function
#         using mmglm0

delta <- c(0,1)

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

beta <- matrix(c(0.1, -0.1,
                 1.0,  5.0),
               byrow=TRUE, nrow=2)

x <- mmglm0(NULL, Pi, delta, family="gaussian", link="identity",
            beta=beta, sigma=c(1, 2))

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

#   Increase maxiter below to achieve convergence
#   Has been restricted to minimise time of package checks
y <- BaumWelch(x, bwcontrol(maxiter=2))
```

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

print(summary(y))
print(logLik(y))


#-----------------------------------------------------
#    Gaussian with log link function
#         using mmglm1

n <- 1000

#   the range of x needs changing according to the glmfamily
x <- seq(-0.9, 1.5, length.out=n)

colour <- c("blue", "green", "red")
colnum <- rep(1:3, n/3+1)[1:n] - 1

data <- data.frame(x=x, colour=colour[colnum+1])

#   will simulate response variable, not required in formula
#   design matrix only depends on RHS of formula
glmformula <- formula( ~ x + I(x^2) + colour)
glmfamily <- gaussian(link="log")
Xdesign <- model.matrix(glmformula, data=data)

# --- Parameter Values and Simulation ---

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

delta <- c(1, 0)

sd <- c(1.2, 1)

beta <- matrix(c(-1, -1.2,
                 -2, -1.8,
                  3,  2.8,
                  1,  0.8,
                  2,  2.2),
               ncol=ncol(Pi), nrow=ncol(Xdesign), byrow=TRUE)

y <- mmglm1(NULL, Pi, delta, glmfamily, beta, Xdesign, sigma=sd)

y <- simulate(y, seed=5)

# --- Estimation ---

#   Increase maxiter below to achieve convergence
```

```
#   Has been restricted to minimise time of package checks
tmp <- BaumWelch(y, bwcontrol(posdiff=FALSE, maxiter=2))
print(summary(tmp))


#-------------------------------------------------
#    Binomial with logit link function
#         using mmglm1

#   n = series length
n <- 1000

#   the range of x need changing according to the glmfamily
x <- seq(-1, 1.5, length.out=n)

colour <- c("blue", "green", "red")
colnum <- rep(1:3, n/3+1)[1:n] - 1

data <- data.frame(x=x, colour=colour[colnum+1])

glmformula <- formula( ~ x + I(x^2) + colour)
glmfamily <- binomial(link="logit")
Xdesign <- model.matrix(glmformula, data=data)

# --- Parameter Values and Simulation ---

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

delta <- c(1, 0)

beta <- matrix(c(-1, -1.2,
                 -2, -1.8,
                  3,  2.8,
                  1,  0.8,
                  2,  2.2),
               ncol=ncol(Pi), nrow=ncol(Xdesign), byrow=TRUE)

y <- mmglm1(NULL, Pi, delta, glmfamily, beta, Xdesign, sigma=sd,
            size=rep(100, n))

#   each element of y$y is the number of successes in 100 Bernoulli trials
y <- simulate(y, seed=5)


# --- Estimation ---

#   Increase maxiter below to achieve convergence
#   Has been restricted to minimise time of package checks
tmp <- BaumWelch(y, bwcontrol(posdiff=FALSE, maxiter=2))
print(summary(tmp))
```

```
#--------------------------------------------------
#    Gaussian with log link function, longitudinal data
#         using mmglmlong1

#   n = series length for each subject
#   N = number of subjects
n <- 5
N <- 1000

#   the range of x need changing according to the glmfamily
x <- seq(-0.9, 1.5, length.out=n)

colour <- c("blue", "green", "red")
colnum <- rep(1:3, n/3+1)[1:n] - 1

data <- data.frame(x=x, colour=colour[colnum+1])

#   will simulate response variable, not required in formula
#   design matrix only depends on RHS of formula
glmformula <- formula( ~ x + I(x^2) + colour)
glmfamily <- gaussian(link="log")
Xdesign0 <- model.matrix(glmformula, data=data)

#    multiple subjects
Xdesign <- NULL
for (i in 1:N) Xdesign <- rbind(Xdesign, Xdesign0)

# --- Parameter Values and Simulation ---

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

delta <- c(0.5, 0.5)

sd <- c(1.2, 1)

beta <- matrix(c(-1, -1.2,
                 -2, -1.8,
                  3,  2.8,
                  1,  0.8,
                  2,  2.2),
               ncol=ncol(Pi), nrow=ncol(Xdesign), byrow=TRUE)

y <- mmglmlong1(NULL, Pi, delta, glmfamily, beta, Xdesign, sigma=sd,
                longitude=rep(1:N, each=n))

y <- simulate(y, seed=5)

# --- Estimation ---

#    Note: the "Not run" blocks below are not run during package checks
```

```
#    as the makePSOCKcluster definition is specific to my network,
#    modify accordingly if you want parallel processing.

cl <- NULL
## Not run:
if (require(parallel)){
    cl <- makePSOCKcluster(c("localhost", "horoeka.localdomain",
                             "horoeka.localdomain", "localhost"))
}
## End(Not run)

#   Increase maxiter below to achieve convergence
#   Has been restricted to minimise time of package checks
tmp <- BaumWelch(y, bwcontrol(posdiff=FALSE, maxiter=2),
                 PSOCKcluster=cl)

## Not run:
if (!is.null(cl)){
    stopCluster(cl)
    rm(cl)
}
## End(Not run)

print(summary(tmp))


#--------------------------------------------------
#    Binomial with logit link function, longitudinal data
#         using mmglmlong1

#   n = series length for each subject
#   N = number of subjects
n <- 10
N <- 100

#   the range of x need changing according to the glmfamily
x <- seq(-1, 1.5, length.out=n)

colour <- c("blue", "green", "red")
colnum <- rep(1:3, n/3+1)[1:n] - 1

data <- data.frame(x=x, colour=colour[colnum+1])

glmformula <- formula( ~ x + I(x^2) + colour)
glmfamily <- binomial(link="logit")
Xdesign0 <- model.matrix(glmformula, data=data)

#    multiple subjects
Xdesign <- NULL
for (i in 1:N) Xdesign <- rbind(Xdesign, Xdesign0)

# --- Parameter Values and Simulation ---
```

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

delta <- c(0.5, 0.5)

beta <- matrix(c(-1, -1.2,
                 -2, -1.8,
                  3,  2.8,
                  1,  0.8,
                  2,  2.2),
               ncol=ncol(Pi), nrow=ncol(Xdesign), byrow=TRUE)

y <- mmglmlong1(NULL, Pi, delta, glmfamily, beta, Xdesign, sigma=sd,
                longitude=rep(1:N, each=n), size=rep(200, N*n))

y <- simulate(y, seed=5)

# --- Estimation ---

#   Increase maxiter below to achieve convergence
#   Has been restricted to minimise time of package checks
tmp <- BaumWelch(y, bwcontrol(posdiff=FALSE, maxiter=1))
print(summary(tmp))
```

`mmglm-2nd-level-functions` *Markov Modulated Generalised Linear Model - 2nd Level Functions*

Description

These functions will generally not be called directly by the user.

Usage

```
Estep.mmglm1(object, fortran=TRUE)
Mstep.mmglm1(object, u)
```

Arguments

`object`	an object with class `"mmglm"` or `"mmglmlong"`
`u`	a matrix of weights by Markov state and observation used in fitting the generalised linear model.
`fortran`	logical, if `TRUE` (default) use the Fortran code in the forward-backward equations, else use the R code.

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

Cited references are listed on the HiddenMarkov manual page.

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))
```

`mmpp-2nd-level-functions` *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, fwd.only = FALSE)
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.
`fwd.only`	logical, if `FALSE` (default) calculate both forward and backward probabilities; else calculate and return only forward probabilities and log-likelihood.

Details

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

References

Cited references are listed on the HiddenMarkov manual page.

Mstep *M-Step of EM Algorithm for DTHMM*

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 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".
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, shape2 and 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 R functions for the Binomial Distribution have arguments size and prob. 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 GammaDist 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 The R functions for the Lognormal Distribution have arguments meanlog and sdlog. 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 The R functions for the Logistic Distribution have arguments location and scale. 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 The R functions for the Normal Distribution have arguments mean and sd. 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 and Extensions

The **HiddenMarkov** package calls the associated functions belonging to the specified probability distribution in a generic way. For example, if the argument distn in dthmm is "xyz", it will expect to find functions pxyz, dxyz, and Mstep.xyz. And if simulations are to be performed, it will require rxyz. In this section we describe the required format for the distribution related functions pxyz, dxyz, and rxyz; and for the function Mstep.xyz required for the M-step in the EM algorithm.

Consider the examples below of distribution related functions and their arguments. Note that the probability functions all have a first argument of q, and the last two arguments are all the same, with the same default values. Similarly, the density functions have a first argument of x, and the last argument is the same, with the same defaults. The arguments in the middle are peculiar to the given distribution, *one argument for each distribution parameter*. Note that the observed process x is *univariate*.

```
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
pbeta(q, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)

dnorm(x, mean = 0, sd = 1, log = FALSE)
```

```
dbeta(x, shape1, shape2, ncp = 0, log = FALSE)
dpois(x, lambda, log = FALSE)
dbinom(x, size, prob, log = FALSE)

rnorm(n, mean = 0, sd = 1)
rbeta(n, shape1, shape2, ncp = 0)
rpois(n, lambda)
rbinom(n, size, prob)
```

The functions pxyz (distribution function), dxyz (density) and rxyz (random number generator) must be consistent with the conventions used in the above examples. The software will deduce the distribution argument names from what is specified in pm and pn, and it will call these functions assuming that their argument list is consistent with those described above. The functions pxyz and dxyz are used in the forward and backward equations.

The functions dxyz, pxyz and rxyz *must* also behave in the same vectorised way as dnorm. For example, if x is a vector, and mean and sd are scalars, then dnorm(x, mean, sd) calculates the density for each element in x using the scalar values of mean and sd; thus the returned value is the same length as x. Alternatively, if x is a scalar and mean and sd are vectors, both of the same length, then the returned value is the same length as mean and is the density of x evaluated at the corresponding pairs of values of mean and sd. The third possibility is that x and one of the distribution parameters, say sd, are vectors of the same length, and mu is a scalar. Then the returned vector will be of the same length as x, where the ith value is the density at x[i] with mean mean and standard deviation sd[i]. Note that the functions for the Multinomial distribution do not have this behaviour. Here the vector x contains the counts for *one* multinomial experiment, so the vector is used to characterise the multivariate character of the random variable rather than multiple univariate realisations. Further, the distribution parameters (i.e. category probabilities) are characterised as one vector rather than a sequence of separate function arguments.

The other calculation, that is specific to the chosen distribution, is the maximisation in the M-step. If we have distn="xyz", then there should be a function called Mstep.xyz. Further, it should have arguments (x, cond, pm, pn); see for example Mstep.norm. The parameters that are estimated within this function are named in a consistent way with those that are defined within the dthmm arguments pm and pn. Notice that the estimates of mean and sd in Mstep.norm are weighted by cond$u. The calculations for cond$u are performed in the E-step, and utilise the distribution related functions "dxyz" and "pxyz". The values of cond$u are essentially probabilities that the process belongs to the given Markov state, hence, when we calculate the distributional parameters (like mu and sd in Mstep.norm) we calculate weighted sums using the probabilities cond$u. This procedure can be shown to give the maximum likelihood estimates of mu and sd, and hence a similar weighting should be used for the distribution "xyz" (see Harte, 2006, for further mathematical detail). 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 nonzero. See Mstep.beta for further details.

Now consider a situation where we want to modify the way in which a normal distribution is fitted. Say we know the Markov dependent means, and we only want to estimate the standard deviations. Since both parameters are Markov dependent, they both need to be specified in the pm argument of dthmm. The estimation of the distribution specific parameters takes place in the M-step, 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. Unfortunately, one cannot easily modify the functions in a package namespace. The simple work-around here is to define a new distribution, say "xyz", then define a new function with the above changes called Mstep.xyz. However, the distribution related functions are just the same as those for the normal distribution, hence, define them as follows:

```
rxyz <- rnorm
dxyz <- dnorm
pxyz <- pnorm
qxyz <- qnorm
```

See the 2nd example below for full details.

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).
```

```
#   One way is to define a new distribution called "xyz", say.

Mstep.xyz <- function(x, cond, pm, pn){
    #   this function is a modified version of Mstep.norm
    #   here the mean is fixed to the values specified in pm$mean
    nms <- sort(names(pm))
    n <- length(x)
    m <- ncol(cond$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))
    }
}

#   define the distribution related functions for "xyz"
#   they are the same as those for the Normal distribution
rxyz <- rnorm
dxyz <- dnorm
pxyz <- pnorm
qxyz <- qnorm

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), "xyz", pm, discrete=FALSE)

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

#    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(params, object, pmap)
```

Arguments

params	a vector of revised parameter values.
object	an object of class "dthmm", "mmglm0", or "mmpp".
pmap	a user provided function mapping the revised (or restricted) 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 values, to that of the EM algorithm 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 relatively 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 algorithm and then switching to Newton type methods (see Examples below).

The maximisation of the model likelihood function can be restricted to be over a subset of the model parameters. Other parameters will then be fixed at the values stored in the model object. Let Θ denote the model parameter space, and let Ψ denote the parameter sub-space ($\Psi \subseteq \Theta$) over which the likelihood function is to be maximised. The argument params contains values in Ψ, and pmap is assigned a function that maps these values into the model parameter space Θ. See “Examples” below.

The mapping function assigned to pmap can also be made to impose restrictions on the domain of the parameter space Ψ so that the minimiser cannot jump to values such that $\Psi \not\subseteq \Theta$. For example, if a particular parameter must be positive, one can work with a transformed parameter that can take any value on the real line, with the model parameter being the exponential of this transformed parameter. Similarly a modified logit like transform can be used to ensure that parameter values remain within a fixed interval with finite boundaries. Examples of these situations can be found in the “Examples” below.

Some functions are provided in the topic Transform-Parameters that may provide useful components within the user provided function assigned to pmap.

Value

Value of the log-likelihood times negative one.

See Also

nlm, optim, Transform-Parameters, BaumWelch

Examples

```
#   Example where the Markov chain is assumed to be stationary

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)

#   start simulation in state 2
delta <- c(0, 1, 0)

x <- dthmm(NULL, Pi, delta, "exp", list(rate=c(5, 2, 0.2)), nonstat=FALSE)
x <- simulate(x, nsim=5000, seed=5)

#   Approximate solution using BaumWelch
x1 <- BaumWelch(x, control=bwcontrol(maxiter=10, tol=1e-5))


#   Exact solution using nlm

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

p <- c(Pi2vector(x$Pi), log(x$pm$rate))
#   Increase iterlim below to achieve convergence
#   Has been restricted to minimise time of package checks
z <- nlm(neglogLik, p, object=x, pmap=allmap,
         print.level=2, gradtol=0.000001, iterlim=2)
x2 <- allmap(x, z$estimate)


#   compare parameter estimates
print(summary(x))
print(summary(x1))
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

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)

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, pmap=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=10, 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
#    Increase iterlim below to achieve convergence
#    Has been restricted to minimise time of package checks
z <- nlm(neglogLik, p, object=x, pmap=allmap,
         print.level=2, gradtol=0.000001, iterlim=5)

#    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))
```

---

probhmm *Conditional Distribution Function of DTHMM*

Description

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

Usage

```
probhmm(logalpha, logbeta, Pi, delta, cumprob)
```

Arguments

logalpha	an $n \times m$ matrix containing the logarithm of the forward probabilities.
logbeta	an $n \times m$ matrix containing the logarithm of the backward probabilities.
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.
`cumprob`	an $n \times m$ matrix where the (i, k)th element is $\Pr\{X_i \le x_i \,|\, C_k = c_k\}$.

Details

Let $X^{(-i)}$ denote the entire process, except with the point X_i removed. The distribution function at the point X_i is

$$\Pr\{X_i \le x_i \,|\, X^{(-i)} = x^{(-i)}\}\,.$$

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.

See Also

`residuals`

`residuals` *Residuals of Hidden Markov Model*

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 'mmglm0'
residuals(object, ...)
## S3 method for class 'mmglm1'
residuals(object, ...)
## S3 method for class 'mmglmlong1'
residuals(object, ...)
```

Arguments

`object`	an object with class `dthmm`, `mmglm0`, `mmglm1` or `mmglmlong1`.
`...`	other arguments.

Details

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.

A continuity adjustment is made when the observed distribution is discrete. In the case of count distributions (e.g. binomial and Poisson) where the observed count is close to or on the boundary of the domain (e.g. binomial or Poisson count is zero, or binomial count is "n"), the pseudo residuals will give a very poor indication of the models goodness of fit; see the Poisson example below.

The code for the methods "dthmm", "mmglm0", "mmglm1" and "mmglmlong1" can be viewed by appending residuals.dthmm, residuals.mmglm0, residuals.mmglm1 or residuals.mmglmlong1, respectively, to HiddenMarkov:::, on the R command line; e.g. HiddenMarkov:::dthmm. The three colons are needed because these method functions are not in the exported NAMESPACE.

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 Hidden Markov Process*

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 'mmglm0'
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mmglm1'
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'mmglmlong1'
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", "mmglm0", "mmglm1", "mmglmlong1" 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.mmglm0` 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", "mchain", "mmglm0", "mmglm1","mmglmlong1" and "mmpp" can be viewed by appending `simulate.dthmm`, `simulate.mchain`, `simulate.mmglm0`, `simulate.mmglm1`, `simulate.mmglmlong1` or `simulate.mmpp`, respectively, to `HiddenMarkov:::`, on the R command line; e.g. `HiddenMarkov:::dthmm`. The three colons are needed because these method functions are not in the exported NAMESPACE.

Value

The returned object has the same class as the input object and contains the components that were in the input object. Additional components depend on the class as below:

"dthmm": it will also have a vector `x` containing the simulated values;

"mmglm0": it will also contain a dataframe `x` about the glm; and

"mmpp": `tau` contains times of the simulated Poisson events (plus time 0), `ys` is a vector of states at the time of each event, `y` is the sequence of states visited, and `x` is the time spent in each state.

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 of Hidden Markov Model*

Description

Provides methods for the generic function `summary`.

Usage

```
## S3 method for class 'dthmm'
summary(object, ...)
## S3 method for class 'mmglm0'
summary(object, ...)
## S3 method for class 'mmglm1'
summary(object, ...)
## S3 method for class 'mmglmlong1'
```

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

Arguments

`object`	an object with class "dthmm", "mmglm0", "mmglm1", "mmglmlong1" or "mmpp".
`...`	other arguments.

Details

The code for the methods "dthmm", "mmglm0", "mmglm1","mmglmlong1" and "mmpp" can be viewed by appending `summary.dthmm`, `summary.mmglm0`, `summary.mmglm1`, `summary.mmglmlong1` or `summary.mmpp`, respectively, to `HiddenMarkov:::`, on the R command line; e.g. `HiddenMarkov:::dthmm`. The three colons are needed because these method functions are not in the exported NAMESPACE.

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 $m \times m$ 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 $m \times m$ 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*

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 'mmglm0'
Viterbi(object, ...)
## S3 method for class 'mmglm1'
Viterbi(object, ...)
## S3 method for class 'mmglmlong1'
Viterbi(object, ...)
```

Arguments

`object`	an object with class "`dthmm`", "`mmglm0`", "`mmglm1`" or "`mmglmlong1`".
`...`	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`", "`mmglm0`", "`mmglm1`" and "`mmglmlong1`" can be viewed by appending `Viterbi.dthmm`, `Viterbi.mmglm0`, `Viterbi.mmglm1` or `Viterbi.mmglmlong1`, respectively, to `HiddenMarkov:::`, on the R command line; e.g. `HiddenMarkov:::dthmm`. The three colons are needed because these method functions are not in the exported NAMESPACE.

Value

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

References

Cited references are listed on the HiddenMarkov manual page.

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