Package 'timedeppar'

July 22, 2025

Type Package

Title Infer Constant and Stochastic, Time-Dependent Model Parameters

Version 1.0.3

Date 2023-08-28

Author Peter Reichert <peter.reichert@emeriti.eawag.ch>

Maintainer Peter Reichert <peter.reichert@emeriti.eawag.ch>

Description Infer constant and stochastic, time-dependent parameters to consider intrinsic stochasticity of a dynamic model and/or to analyze model structure modifications that could reduce model deficits.
The concept is based on inferring time-dependent parameters as stochastic processes in the form of Ornstein-Uhlenbeck processes jointly with inferring constant model parameters and parameters of the Ornstein-Uhlenbeck processes.
The package also contains functions to sample from and calculate densities of Ornstein-Uhlenbeck processes.
References:
Tomassini, L., Reichert, P., Kuensch, H.-R. Buser, C., Knutti, R. and Borsuk, M.E. (2009), A smoothing algorithm for estimating stochastic, continuous-time model parameters and its application to a simple climate model, Journal of the Royal Statistical Society: Series C (Applied Statistics) 58, 679-704, <doi:10.1111/j.1467-9876.2009.00678.x>
Reichert, P., and Mieleitner, J. (2009), Analyzing input and structural uncertainty of nonlinear dynamic models with stochastic, time-dependent parameters. Water Resources Research, 45, W10402, <doi:10.1029/2009WR007814>
Reichert, P., Ammann, L. and Fenicia, F. (2021), Potential and challenges of investigating intrinsic uncertainty of hydrological models with time-dependent, stochastic parameters. Water Resources Research 57(8), e2020WR028311, <doi:10.1029/2020WR028311>
Reichert, P. (2022), timedeppar: An R package for inferring stochastic, time-dependent model parameters, in preparation.

License GPL-3

Depends mvtnorm

URL https://gitlab.com/p.reichert/timedeppar

BugReports https://gitlab.com/p.reichert/timedeppar/-/issues

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-08-28 10:50:02 UTC

Contents

`calc.acceptfreq` *Calculate apparent acceptance frequency of time-dependent parameter*

Description

This function calculates the apparent acceptance frequency from a potentially thinned Markov chain sample.

Usage

```
calc.acceptfreq(x, n.burnin)
```

Arguments

`x`	results from the function `infer.timedeppar` of class `timedeppar`.
`n.burnin`	number of (unthinned) burnin points of the Markov chain to omit from analysis.

Value

list two-column matrices with time and apparent acceptance frequencies

calc.logpdf — *Calculate log pdf values (prior, internal, posterior) from an object of type* timedeppar

Description

This function calculated log priors, log pdf of Ornstein-Uhlenbeck time dependent parameters, and log posterior from an object of type timedeppar produced by the function infer.timedeppar.

Usage

```
calc.logpdf(x, param, verbose)
```

Arguments

x	results from the function infer.timedeppar of class timedeppar.
param	list of parameter lists and vectors extracted from an object of class timedeppar using the function get.param.
verbose	boolean indicator for writing the results to the console (default is not to do it).

Value

numeric vecctor of values of the different log pdfs.

get.label — *Construct a plot label from expressions for variables and units*

Description

This function produces an expression to label plots from expressions for variables and units and from character strings.

Usage

```
get.label(var.name, labels = NA, units = NA, t1 = "", t2 = "", t3 = "")
```

Arguments

var.name	name of the variable.
labels	named vector of expressions encoding variable names with greek letters, sub- and superscripts.
units	named vector of expressions encoding units with sub- and superscripts.
t1	optional text (see below).
t2	optional text (see below).
t3	optional text (see below).

Value

expression of a label of the form: t1 label t2 [unit] t3
label and unit are extracted from the vectors `labels` and `units` by using the compontents named `var.name`

`get.param` *Extract parameter list and process parameter vectors from an object of type* `timedeppar`

Description

This function extracts an element of the stored Markov chain from an object of type `timedeppar` produced by the function `infer.timedeppar` and converts it to a format that facilitates re-evaluation of the posterior, evaluation of the underlying model, and sampling from the Ornstein-Uhlenbeck process. In particular, the constant and time-dependent parameters are provided in the same list format as supplied as `param.ini` to the function `infer.timedeppar`. In addition, the parameters of the Ornstein-Uhlenbeck process(es) of the time-dependent parameters are provided in different formats (see details below under `Value`.

Usage

```
get.param(x, ind.sample)
```

Arguments

`x`	results from the function `infer.timedeppar` of class `timedeppar`.
`ind.sample`	index of the stored (potentially thinned) Markov chain defining which parameters to reconstruct. Default is to extract the parameters corresponding to the maximum posterior solution.

Value

list with the following elements:
`param`: list of constant and time-dependent model parameters,
`param.ou.estim`: vector of estimated process parameters of all time-dependent parameters,
`param.ou.fixed`: vector of fixed process parameters of all time-dependent parameters,
`param.ou`: matrix of Ornstein-Uhlenbeck parameters for all time-dependent parameters; columns are mean, sd and gamma of the processes, rows are the time-dependent parameter(s).
`logpdf`: corresponding lopdf values (posterior, intermediate densities, and priors),
`ind.timedeppar`: indices of `param` at which parameters are time-dependent.
`ind.sample`: sample index.
`ind.chain`: corresponding index of the Markov chain.

`get.parsamp` *Get a sample of lists of constant and time-dependent parameters from inference results of* `infer.timedeppar`

Description

This function produces a sample of parameter sets for past and potentially future time points based on the results of class `timedeppar` generated by Bayesian inference with the function `infer.timedeppar`. For time points used for inference, the sample is a sub-samble of the Markov chain, for future time points of time-dependent parameters it is a random sample based on the corresponding Ornstein-Uhlenbeck parameters and constrained at there initial point to the end point of the sub-sample.

Usage

```
get.parsamp(x, samp.size = 1000, n.burnin = 0, times.new = numeric(0))
```

Arguments

`x`	results from the function `infer.timedeppar` of class `timedeppar`.
`samp.size`	size of the produced sample constructed from the Markov chain stored in the object of class `timedeppar` omitting the adaptation and burnin phases.
`n.burnin`	number of Markov chain points to omit for density and pairs plots (number of omitted points is max(control$n.adapt,n.burnin)).
`times.new`	vector of time points to predict for. If no time points are provided, sampling is only from the inference Markov chain; if time points are provided, they need to be increasing and start with a larger value than the time points used for inference. In the latter case, time-dependent parameters are sampled for the future points and appended to the inferred part of the time-dependend parameter.

Value

list of
`param.maxpost`: list of constant and time-dependent parameters corresponding to the maximum posterior solution for inference (no extrapolation to the future).
`param.maxlikeli`: list of constant and time-dependent parameters corresponding to the solution with maximum observation likelihood found so far.
`param.list`: list of length `samp.size` containing lists of constant and time-dependent parameters; for time-dependent parameters sub-sample of the Markov chain for past time points, sample from Ornstein-Uhlenbeck processes conditioned at the initial point for future time points (see argument `times.new`).
`param.const`: sub-sample of constant parameters.
`param.timedep`: list of sub-samples of time-dependent parameters.
`param.ou`: sub-sample of Ornstein-Uhlebeck parameters of the time-dependent parameter(s).
`ind.timedeppar`: indices of time-dependent parameters in the parameter lists.
`ind.sample`: indices of the stored, thinned sample defining the sub-sample.
`ind.chain`: indices of the original non-thinned Markov chain defining the sub-sample.

dot.args: ... arguments passed to infer.timedeppar; to be re-used for new model evaluations.

---

infer.timedeppar *Jointly infer constant and time-dependent parameters of a dynamic model given time-series data*

Description

This function draws a Markov Chain from the posterior of constant and time-dependent parameters (following Ornstein-Uhlenbeck processes) of a dynamic model. The dynamic model is specified by a function that calculates the log likelihood for given time-series data. The Ornstein-Uhlenbeck processes of time-dependent processes are characterized by there mean (mean), standard deviation (sd), and a rate parameter (gamma) that quantifies temporal correlation.

Usage

```
infer.timedeppar(
  loglikeli = NULL,
  loglikeli.keepstate = FALSE,
  param.ini = list(),
  param.range = list(),
  param.log = logical(0),
  param.logprior = NULL,
  param.ou.ini = numeric(0),
  param.ou.fixed = numeric(0),
  param.ou.logprior = NULL,
  task = c("start", "continue", "restart"),
  n.iter = NA,
  cov.prop.const.ini = NA,
  cov.prop.ou.ini = NA,
  scale.prop.const.ini = NA,
  scale.prop.ou.ini = NA,
  control = list(),
  res.infer.timedeppar = list(),
  verbose = 0,
  file.save = "",
  ...
)
```

Arguments

loglikeli	function that calculates the log likelihood of the model for given constant or time-dependent parameters and given observational data.

The parameters are passed as a named list in the first argument of the function. The list elements are either scalar values representing constant parameters or two-column matrices with columns for time points and values for time-dependent parameters. If the argument `loglikeli.keepstate` is `FALSE` no further arguments are needed (but can be provided, see below). In this case, the function should return the log likelihood as a single double value.
If the argument `loglikeli.keepstate` is `TRUE`, the second argument provides the time range over which a time-dependent parameter was changed or NA if the full simulation time has to be evaluated, and the third argument provides the state of the functon at the last successful call. This allows the function to only calculate and return modifications to that previous state. In this case, the function has to return a list with the log likelihood value as its first element and the current state of the function as the second argument. This state can be an R variable of an arbitrary data type. The version from the last accepted MCMC step will be returnde at the next call.
Further arguments provided to `infer.timedeppar` will be passed to this function.

`loglikeli.keepstate`
: boolean to indicate which kind of interface to the likelihood function is used. See argument `loglikeli` for details.

`param.ini`
: named list of initial vallues of parameters to be estimated. scalar initial values for constant parameters, two-column matrices for time and parameter values for time-dependent parameters (values of time-dependent parameters may be NA and are then drawn from the Ornstein-Uhlenbeck process). The list `param.ini` needs to be a legal and complete first element of the function passed by the argument `loglikeli`. For each time-dependent parameter with name `<name>` initial values or fixed value of the parameters `<name>_mean`, `<name>_sd` and `<name>_gamma` must be provided in the arguments `param.ou.ini` or `param.ou.fixed`, respectively. These parameters represent the mean, the asymptotic standard deviation, and the rate parameter of the Ornstein-Uhlenbeck process. If these parameters are given in the argument `param.ou.ini`, they are used as initial condition of the inference process and the parameters are estimated, if they are given in the argument `param.ou.fixed`, they are assumed to be given and are kept fixed.

`param.range`
: named list of ranges (2 element vectors with minimum and maximum) of parameters that are constrained (non-logarithmic for all parameters)

`param.log`
: named vector of logicals indicating if inference should be done on the log scale (parameters are still given and returned on non-log scales). For time-dependent parameters, selecting this option implies the use of a lognormal marginal for the Ornstein-Uhlenbeck process. This means that the parameter is modelled as exp(Ornstein-Uhlenbeck), but mean and standard devistion of the process are still on non-log scales.

`param.logprior`
: function to calculate the (joint) log prior of all estimaged constant parameters of the model. The function gets as its argument a named vector of the values of the estimaged constant parameters to allow the function to identify for which parameters a joint prior is required in the current setting).

`param.ou.ini`
: named vector of initial values of parameters of the Ornstein-Uhlenbeck processes of time-dependent parameters; see description of argument `param.ini`.

param.ou.fixed named vector of values of parameters of the Ornstein-Uhlenbeck processes of time-dependent parameters that are kept fixed rather than being extimated. If all process parameters are kept fixed, these names are <name>_mean, <name>_sd and <name>_gamma for each time-dependent parameter with name <name>; see description of the argument param.ini. The values specified in param.fixed are ignored if the parameter is also given in the argument param.ini; in this case it is estimated.

param.ou.logprior function to calculate the (joint) log prior of all estimated parameters of the Ornstein-Uhlenbeck processes of a single time-dependent parameter. The function gets as its argument a named vector of the values of the process parameters to estimated. These names are a subset of <name>_mean, <name>_sd and <name>_gamma; see description of the argument param.ini. The function has to work for each time-dependent parameter by being sensitive to the parameter names.

task Which task to perform (default value: "start"):
"start": start an inference process from scratch based on the arguments of the function. The argument res.infer.timedeppar is ignored.
"continue": continue a Markov chain from a previous call to infer.timedeppar. The results of a previous call have to be provided as the argument res.infer.timedeppar in the form of an object of type timedeppar. To guarantee convergence of the chain, all numerical specifications including the final state of the chain are taken from the object provided by res.infer.timedeppar and the actual arguments of the function are ignored except n.iter which specifies the number of iterations to be added to the chain.
"restart": A new chain is started from the last point of a previous chain except if the argument param.ini is provided. The results of a previous call have to be provided as the argument res.infer.timedeppar in the form of an object of type timedeppar. Likelihood, prior pdf functions, initial proposal covariance matrices, scales and control parameters are taken from the previous chain unless explicitly provided.

n.iter number of iterations of the Markov chain to be performed (default value: 10000).

cov.prop.const.ini scaled covariance matrix of the proposal distribution for the Metropolis step of constant parameters. The proposal distribution of the Metropolis step is a normal distribution centered at the last point of the chain with a covariance matrix equal to scale.prop.const^2 * cov.prop.const. Note that if param.log is TRUE for a parameter, then the proposal is evaluated at the log scale of the parameter. During the adaptation phase, the covariance matrix is periodically adapted to the covariance matrix of the current sample and the scale to get a reasonable acceptance rate. After the adaptation phase, both variables are kept constant to guarantee convergence.

cov.prop.ou.ini list of scaled covariance matrices of the proposal distributions for the Metropolis step of the parameters of the Ornstein-Uhlenbeck processes of time-dependent parameters. The proposal distribution of the Metropolis step for the process parameters of the time-dependent parameter i is a normal distribution centered at the last point of the chain with a covariance matrix equal to cov.prop.ou[[i]]

* scale.prop.ou[i]^2. Note that if param.log is TRUE for a parameter, then the proposal for the mean is evaluated at the log scale of the parameter. This is anyway the case for the standard deviation and the rate parameter of the Ornstein-Uhlenbeck process. During the adaptation phase, the covariance matrices are periodically adapted to the covariance matrix of the current sample and the scale to get a reasonable acceptance rate. After the adaptation phase, both variables are kept constant to guarantee convergence.

scale.prop.const.ini
: scale factor for the covariance matrix of the Metropolis step for constant parameters with a proposal distribution equal to a normal distribution centered at the previous point of the chain and a covariance matrix equal to scale.prop.const^2 * cov.prop.const. During the adaptation phase, the covariance matrix is periodically adapted to the covariance matrix of the current sample and the scale to get a reasonable acceptance rate. After the adaptation phase, both variables are kept constant to guarantee convergence.

scale.prop.ou.ini
: vector of scale factors for the covariance matrices of the Metropolis step for parameters of Ornstein-Uhlenbeck processes with a proposal distribution equal to a normal distribution centered at the previous point of the chain and a covariance matrix for the time dependent parameter i equal to cov.prop.ou[[i]] * scale.prop.ou[i]^2. During the adaptation phase, the covariance matrix is periodically adapted to the covariance matrix of the current sample and the scale to get a reasonable acceptance rate. After the adaptation phase, both variables are kept constant to guarantee convergence.

control
: list of control parameters of the algorithm:

- n.interval: number of sub-intervals into which the time domain is splitted to infer the time-dependent parameters; either scalar for universal choice for all parameters or named vector for parameter-specific choices (default value: 50; this number must be increased if the acceptance rates of the time-dependent parameters are very low, it can be decreased if they are high);
- min.internal: minimum number of internal points in an interval (default value: 1; may be increased if time resolution is high).
- splitmethod: method used for random splitting of time domain into sub-intervals. Possible values: "modunif": modification of uniform intervals; "random": random split (higher variability in inverval lengths); "weighted": weighted random split leading to shorter intervals where the acceptance frequency is low; "autoweights": use weighted random split but adjusts weights adaptively. (default value: "modunif").
- interval.weights: numerical vector or named list of numerical vectors (by time-dependent parameter) of weights for sampling interval boundaries (the length(s) of the vector(s) must be equal to the time series length in the parameter specification). The weight vectors do not have to be normalized. The weights are used if the parameter splitmethod is equal to "weighted" or as optional initial weights if splitmethod is equal to "autoweights".
- n.autoweighting: number of past iterations to consider for weight calculation for splitmethod "autoweights" (default value: 1000). Note that the calculation of weights only starts after n.autoweights iterations and

that only stored points are considerd so that the number of points considered is equal to `n.autoweighting/thin`.

- `offset.weighting`: offset used to caluclate weights from apparent acceptance frequencies for `splitmethod` `"autoweights"` (default value: 0.05).
- `n.widening`: number grid points used to widen areas of high weight for `splitmethod` `"autoweights"` (default value: 10).
- `n.timedep.perstep`: number of updates of the time-dependent parameter(s) before updating the constant parameters (default value: 1).
- `n.const.perstep`: number of Markov chain steps for the constant parameters to be performed between updating the time-dependent parameters (default value: 1).
- `n.adapt`: number of iterations of the Markov chain during which adaptation is made (default value: 2000; only during this phase, the covariance matrix and the scaling factors are adapted).
- `n.adapt.scale`: number of iterations after which the acceptance rate is checked for potentially adapting the scaling factor (default value: 30).
- `n.adapt.cov`: number of iterations of the Markov chain, after which the covariance matrix of the proposal distribution is adapted (default value: 900; 0 means no adaptation of the covariance matrix; note that after `control$n.adapt` iterations adaptation is turned off; for this reason, after the last multiple of `n.adapt.cov` below `n.adapt` there should be sufficient iterations left to adapt the scaling factors).
- `f.reduce.cor`: factor by which sample correlations are reduced when constructing the covariance matrix of the proposal distribution (default value: 0.90).
- `f.accept.decscale`: acceptance rate below which the proposal scaling factor is decreased during the adaptation phase (default value: 0.05).
- `f.accept.incscale`: acceptance rate above which the proposal scaling factor is increased during the adaptation phase (default value: 0.30).
- `f.max.scalechange`: max. factor for changing proposal distribution scale from reference (default value: 10; reference is either initial value or modified value when the covariance matrix was adapted).
- `f.sample.cov.restart`: fraction of previous samples to be used to calculate the covariance matrix of proposal distribution when restarting inference (default value: 0.3; the last part of the samples is used).
- `thin`: thinning for storing Markov chain results (default value: 1).
- `n.save`: number of iterations after which the results are (periodically) saved (default value: 1000).
- `save.diag`: save diagnostic information about acceptance ratio, acceptance, and interval lengths for inference of the time-dependent parameters.

`res.infer.timedeppar`
: results of a previous call to this function. These results are ignored if the argument `task` is equal to `"start"`, but it is needed for the tasks `"continue"` and `"restart"`.

`verbose`
: integer parameter indicating the level of progress reporting:
0: no reporting;

	1: reporting of thinned and accepted Markov Chain steps and of adapted proposal covariance matrices; 2: reporting of proposals and accepted steps before thinning.
`file.save`	if non-empty string, the intermediate results are saved to this file as variable `res` in a workspace after every `control$n.save` iterations (the extension `.RData` will be appended to the file name).
`...`	additional parameters passed to the function `loglikeli`.

Value

class of type `timedeppar` with the following elements:

- `package`: package timedeppar: version and date,
- `func`: function called (infer.timedeppar),
- `date`: date of call,
- `dot.args`: arguments passed to the likelihood function (included for reproducibility of results),
- `task`: task that was performed (`start`, `restart` or `continue`),
- `file`: name of file to which output was written,
- `param.ini`: initial values of likelihood parameters (constant and time-dependent),
- `param.ou.ini`: initial values of Ornstein-Uhlenbeck process parameters that are estimated,
- `param.ou.fixed`: values of Ornstein-Uhlenbeck process parameters that are not estimated,
- `loglikeli`: function that was passed to calculate the log likelihood of the observations,
- `loglikeli.keepstate`: boolean indicating whether or not the state from the previous run should be kept (this allows only partial time evaluation when only part of the input was replaced),
- `param.logprior`: function that was passed to calculate the joint log prior of the constant likelihood parameters,
- `param.ou.logprior`: function that was passed to calculate the joint log prior of the estimated Ornstein-Uhlenbeck process parameters (in case of multiple Ornstein-Uhlenbeck processes the function has to return the prior for the correct process; this can be identified by the names of the argument),
- `param.range`: parameter ranges,
- `param.log`: named logical vector of indicators for log inference,
- `control`: named list of control parameters as passed to the call (or read from a previous call),
- `n.iter`: number of iterations peformed (note that the size of the sample will be n.iter/control$thin),
- `sample.diag`: list of samples of proposals, log acceptance ratios, and interval lengths of time-dependent parameters (only available if the control variable `save.diag` is set to `TRUE`),
- `sample.param.timedep`: list of samples of time dependent parameters (first row contains time points),
- `sample.param.ou`: sample of Ornstein-Uhlenbeck process parameters,
- `sample.param.const`: sample of constant parameters,

- `sample.logpdf`: sample of prior, Ornstein-Uhlenbeck and posterior pdf,
- `acceptfreq.constpar`: acceptance frequency of constant parameters after adaptation phase,
- `acceptfreq.oupar`: acceptance frequencies of Ornstein-Uhlenbeck process parameters after adaptation phase,
- `acceptfreq.timedeppar`: acceptance frequencies of time-depenent parameters,
- `param.maxpost`: parameters at the maximum posterior (constant and time-dependent parameters),
- `param.ou.maxpost`: Ornstein-Uhlenbeck process parameters at the maximum posterior,
- `cov.prop.const`: final covariance matrix used for proposal distribution of constant parameters,
- `cov.prop.ou`: list of final covariance matrices used for proposal distribution of Ornstein-Uhlenbeck process paramemters,
- `scale.prop.const`: final scale of proposal distribution of constant parameters,
- `scale.prop.ou`: final scale of proposal distribution of Ornstein-Uhlenbeck process parameters,
- `sys.time`: run time used for the previous inference job.

References

Reichert, P. timedeppar: An R package for inferring stochastic, time-dependent model parameters in preparation, 2020.

Reichert, P., Ammann, L. and Fenicia, F. Potential and challenges of investigating intrinsic uncertainty of hydrological models with time-dependent, stochastic parameters. *Water Resources Research* 57(8), e2020WR028311, 2021. doi:10.1029/2020WR028311

Reichert, P. and Mieleitner, J. Analyzing input and structural uncertainty of nonlinear dynamic models with stochastic, time-dependent parameters. *Water Resources Research*, 45, W10402, 2009. doi:10.1029/2009WR007814

Tomassini, L., Reichert, P., Kuensch, H.-R. Buser, C., Knutti, R. and Borsuk, M.E. A smoothing algorithm for estimating stochastic, continuous-time model parameters and its application to a simple climate model. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 58, 679-704, 2009. doi:10.1111/j.14679876.2009.00678.x

See Also

`plot.timedeppar` for visualizing results.
`calc.acceptfreq` for calculating (apparent) acceptance frequencies.
`calc.logpdf` for calculating log pdf values (prior, internal, posterior) from the results.
`get.param` for extracting individual parameters from the Markov chain.
`get.parsamp` for extracting subsamples of the Markov chain.
`readres.timedeppar` for reading saved results from a previous run.
`randOU` for sampling from an Ornstein-Uhlenbeck process.
`logpdfOU` for calculating the probability density of a sample from an Ornstein-Uhlenbeck process.

Examples

```
# Simple example for re-inferring parameters of an Ornstein-Uhlenbeck process
# with observational noise from synthetically generated data
# ------------------------------------------------------------------------------

# load package:
if ( !require("timedeppar") ) { install.packages("timedeppar"); library(timedeppar) }

# choose model parameters:
y_mean      <-  0
y_sd        <-  1
y_gamma     <- 10
obs_sd      <-  0.2

# choose control parameters of numerical algorithm:
n.iter      <-   100   # this is just to demonstrate how it works and is compatible
# with the computation time requirements for examples in CRAN
# n.iter      <- 50000   # please go for a sample size like this
#                        # for getting a reasonable sample
n.interval  <-    25   # increase if rejection frequency of stoch. par. too high
fract.adapt <- 0.4
n.adapt     <- floor(fract.adapt*n.iter)

# synthetically generate data:
set.seed(123)
data <- randOU(mean=y_mean,sd=y_sd,gamma=y_gamma,t=seq(from=0,to=2,length.out=101))
data$yobs <- data$y + rnorm(nrow(data),mean=0,sd=obs_sd)

# define observational likelihood:
loglikeli <- function(param,data)
{
 # get parameter y at time points of observations::
 y <- param$y
 if ( is.matrix(y) | is.data.frame(y) ) y <- approx(x=y[,1],y=y[,2],xout=data[,1])$y
 # calculate likelihood:
 loglikeli <- sum(dnorm(data[,"yobs"],mean=y,sd=param$obs_sd,log=TRUE))
 # return result:
 return(loglikeli)
}

# sample from the posterior of y, mu_y, sd_y and sd_obs assuming a uniform prior:
res <- infer.timedeppar(loglikeli      = loglikeli,
                        param.ini      = list(y=randOU(mean=y_mean,sd=y_sd,gamma=y_gamma,
                                                       t=seq(from=0,to=2,length.out=501)),
                                              obs_sd=obs_sd),
                        param.log      = c(y=FALSE,obs_sd=TRUE),
                        param.ou.ini   = c(y_mean=0,y_sd=1),
                        param.ou.fixed = c(y_gamma=10),
                        n.iter         = n.iter,
                        control        = list(n.interval = n.interval,
                                              n.adapt    = n.adapt),
                        data = data)
```

```
# plot results using pre-defined options:
# pdf(paste0("infer_OU_",n.iter,"_",n.adapt,"_original.pdf"),width=8,height=12)
plot(res,
    labels=expression(y       = y,
                      y_mean  = mu[y],
                      y_sd    = sigma[y],
                      y_gamma = gamma[y]))
# dev.off()

# plot time series and data:
# pdf(paste0("infer_OU_",n.iter,"_",n.adapt,"_comparison.pdf"),width=8,height=6)
t <- res$sample.param.timedep$y[1,]
sample <- res$sample.param.timedep$y[(1+n.adapt+1):nrow(res$sample.param.timedep$y),]
q <- apply(sample,2,quantile,probs=c(0.025,0.5,0.975))
plot(numeric(0),numeric(0),type="n",xaxs="i",yaxs="i",
    xlim=range(t),ylim=2.5*c(-1,1),xlab="t",ylab="y")
polygon(c(t,rev(t),t[1]),c(q[1,],rev(q[3,]),q[1,1]),
       col="grey80",border=NA)
lines(t,q[2,])
lines(data$t,data$y,col="red")
points(data$t,data$yobs,pch=19,cex=0.8)
legend("bottomright",
      legend=c("original process","noisy data","inferred median","inferred 95% range"),
    lwd=c(1,NA,1,5),lty=c("solid",NA,"solid","solid"),col=c("red","black","black","grey80"),
      pch=c(NA,19,NA,NA),cex=0.8)
# dev.off()
```

`logpdfOU` *Calculate log pdf of an Ornstein-Uhlenbeck process*

Description

This function calculates the log pdf of a realization of an Ornstein-Uhlenbeck process with given parameters. The calculation can be done for an Ornstein-Uhlenbeck process with a random start value, for a process conditional on the start value, or for a process conditional on start and end values. The function includes the option of performing the calculation for lognormal marginal generated by exponential tranformation.

Usage

```
logpdfOU(t, y, mean = 0, sd = 1, gamma = 1, cond = 0, log = FALSE)
```

Arguments

`t`	vector of time points at which the OU process is available.
`y`	vector of y-values corresponing to the t-values (note that t and y need to be of the same length).
`mean`	asymptotic mean of the process.

sd	asymptotic standard deviation of the process.
gamma	rate coefficient for return to the mean
cond	conditioning: 0 indicates no conditioning, 1 conditioning to start value, and 2 conditioning to start and end value
log	if true, the log pdf of the log of y is calculated (mean and sd are interpreted in y, not in log(y) units)

Value

the function returns the log pdf

Examples

```
OU <- randOU(mean=0,sd=1,gamma=1,t=0:1000/1000)
logpdfOU(OU$t,OU$y,mean=0,sd=1,gamma=1)
```

---

plot.timedeppar *Plot results of time-dependent parameter estimation*

Description

This function plot Markov chains and marginal densities of constant parameters, distributions of time dependent parameters, and Markov chains and marginal densities of time-dependent parameters at selected points in time.

Usage

```
## S3 method for class 'timedeppar'
plot(
  x,
  type = c("traces", "marginals", "summary", "pairs", "time-series", "accept"),
  chains.at = numeric(0),
  labels = NA,
  units = NA,
  prob.band = 0.9,
  max.diag.plots = 100,
  xlim.ts = numeric(0),
  n.burnin = 0,
  nrow = 4,
  nrow.constpar = NA,
  nrow.timedeppar = NA,
  nrow.diagnostics = NA,
  ...
)
```

Arguments

x	results from the function infer.timedeppar of class timedeppar or list of such results for comparing multiple chains (in the latter case you have to call explicitly plot.timedeppar rather than being able to do the generic call plot as the list of results is not an object of class timedeppar).
type	vector of plot types: "traces" or "marginals": traces and 1d marginals of Markov chains of constant parameters, of time-dependent parameters at certain time points (see argument chains.at), chains of log posterior and log observational likelihood values. For selected outputs only, specify traces.constpar, traces.timedeppar, traces.logposterior. "summary": print summary of acceptance rates and maximum log posterior and log likelihood values. "pairs": scatterplot matrix of posterior sample of constant parameters. "time-series": uncertainty range and median time series of time-dependent parameters. "accept": time series of apparent acceptance frequencies (at the level of thinning). "realizations": realizations of time-dependent parameters to check for burnin.. "diagnostics": plot diagnostics for inference of time-dependent parameters (note that the plot file could become very large to follow the inference steps).
chains.at	vector of time points at which chains and marginals of time-dependent parameters should be plotted if "traces" or "marginals" is contained in the vector argument type (default: none [numeric(0)]).
labels	optional named vector of expressions to label variables in the plots (names of the expression have to correspond to the variable names as used by the program, expressions can have special symbols, e.g. expression(a=alpha,b=beta,c1=gamma[1])).
units	optional named vector of expressions to add units to variables in the plots (names of the expression have to correspond to the variable names as used by the program, expressions can have special symbols, e.g. expression(a=m^3/s,b=h^-1,c1=m)).
prob.band	probability defining the width of the uncertainty bands plotted for output variables (default value: 0.9)
max.diag.plots	maximum number of diagnostic plots of inference steps
xlim.ts	optional range of time values for time-series plot
n.burnin	number of Markov chain points to omit for density and pairs plots (number of omitted points is max(control$n.adapt,n.burnin)).
nrow	number of plot rows per page (except for pairs plot).
nrow.constpar	number of plot rows per page for traces and marginals (default is nrow).
nrow.timedeppar	number of plot rows per page for time-dependent parameters (default is nrow).
nrow.diagnostics	number of plot rows per page for diagnostics plots (default is nrow).
...	additional arguments passed to the plotting function.

randOU *Draw from an Ornstein-Uhlenbeck process*

Description

This function draws a realization of an Ornstein-Uhlenbeck process with a random start value (drawn from the marginal distribution), conditional of the start value, or conditional on both start and end values. The function includes the option of obtaining a lognormal marginal by exponential tranformation.

Usage

```
randOU(
  mean = 0,
  sd = 1,
  gamma = 1,
  t = 0:1000/1000,
  yini = NA,
  yend = NA,
  log = FALSE
)
```

Arguments

mean	asymptotic mean of the process.
sd	asymptotic standard deviation of the process
gamma	rate coefficient for return to the mean
t	vector of time points at which the process should be sampled (note: the value at t[1] will be the starting value yini, the value at t[length(t)] the end value yend if these are specified)
yini	start value of the process (NA indicates random with asymptotic mean and sd)
yend	end value of the process (NA indicates no conditioning at the end)
log	indicator whether the log of the variable should be an Ornstein-Uhlenbeck process (log=TRUE) rather than the variable itself (mean and sd are interpreted in original units also for log=TRUE)

Value

a data frame with t and y columns for time and for the realization of the Ornstein-Uhlenbeck process

Examples

```
plot(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000),type="l",ylim=2.5*c(-1,1))
abline(h=0)
lines(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000),col="red")
lines(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000),col="blue")
```

```
lines(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000),col="green")

plot(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000,yini=0,yend=0),type="l",ylim=2.5*c(-1,1))
abline(h=0)
lines(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000,yini=0,yend=0),col="red")
lines(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000,yini=0,yend=0),col="blue")
lines(randOU(mean=0,sd=1,gamma=1,t=0:1000/1000,yini=0,yend=0),col="green")
```

`randsplit` *Draw indices for a random split of a vector into intervals of the same mean length*

Description

This function draws indices for a random split of a vector into sub-vectors.

Usage

```
randsplit(
  n.grid,
  n.interval,
  method = c("modunif", "random", "weighted"),
  weights = numeric(0),
  offset = 0,
  min.internal = 2
)
```

Arguments

`n.grid`	number of grid points to divide into intervals
`n.interval`	number of intervals
`method`	method for random splitting: `modunif` modification of uniform intervals `random` random split (higher variability in inverval lengths) `weighted` random split with weights; non-normalized weights must be specified by the argument `weights`
`weights`	weights for choosing interval boundaries for method `weighted`; vector of length i2-i1+1 (does not need to be normalized and will be ignored for all methods except for method `weighted`)
`offset`	offset to shift subset of potential interval boundaries to draw from. To guarantee different intervals on subsequent calls, offset should be increased by one between subsequent calls for the same variable.
`min.internal`	minimum number of internal points between interval boundary points

Value

the function returns an index vector of length n+1 with the endpoint indices of the random intervals.

Examples

```
randsplit(100,10)
randsplit(100,10)
randsplit(100,10,method="random")
randsplit(100,10,method="weighted",weights=1:100)
for ( i in 1:10 ) print(randsplit(100,10,method="weighted",weights=1:100,offset=i))
```

`readres.timedeppar` *Reads an object of type* `timedeppar` *saved to a file by* `infer.timedeppar`

Description

This function read a workspace stored by the function `infer.timedeppar` and returns the object of type `timedeppar` that contains the intermediate or final results of the inference process

Usage

```
readres.timedeppar(file)
```

Arguments

`file`	file name of the workspace to be read.

Value

object of type `timedeppar` containing the intermediate or final results of the inference process or a list of length zero if the file was not found of no object called `res` of type `timedeppar` was found in the workspace.

Index