Package 'pttstability'

July 23, 2025

Title Particle-Takens Stability

Version 1.4

Description Includes a collection of functions presented in ``Measuring stability in ecological systems without static equilibria'' by Clark et al. (2022) <doi:10.1002/ecs2.4328> in Ecosphere. These can be used to estimate the parameters of a stochastic state space model (i.e. a model where a time series is observed with error). The goal of this package is to estimate the variability around a deterministic process, both in terms of observation error - i.e. variability due to imperfect observations that does not influence system state - and in terms of process noise - i.e. stochastic variation in the actual state of the process. Unlike classical methods for estimating variability, this package does not necessarily assume that the deterministic state is fixed (i.e. a fixed-point equilibrium), meaning that variability around a dynamic trajectory can be estimated (e.g. stochastic fluctuations during predator-prey dynamics).

Depends R (>= 3.4)

Imports graphics, stats

Suggests BayesianTools

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Collate 'bayesfun.R' 'data.R' 'fake_data.R' 'logit_funs.R' 'particlefilter.R' 'pttstability_man.R' 'smapping_functions.R'

NeedsCompilation no

Author Adam Clark [aut, cre] (ORCID: <https://orcid.org/0000-0002-8843-3278>)

Maintainer Adam Clark <adam.tclark@gmail.com>

Repository CRAN

Date/Publication 2024-01-09 15:00:02 UTC

Contents

`colfun0` *default colonization function*

Description

Simulates colonization events - events occur as a binomial random process with probability ilogit(p), and populations are seeded with abundance exp(A).

Usage

```
colfun0(co, xt)
```

Arguments

`co`	a numeric vector of length two (p, A), specifying the logit-transformed colonization probability when abundance is zero, and the log-transformed abundance observed immediately after a colonization event
`xt`	a number or numeric vector of abundances at time t, before colonization has occurred

Value

a numeric, including number or numeric vector of length xt, with predicted abundances after colonization has occurred

---

`dat` *Microcosm experimental data*

Description

A dataset containing the abundances of Chlamydomonas terricola growing in a multi-species community. Includes 17 time steps covering 463 days in two treatments: LSA ('low' temperature, 'stable' oscillations, and 'absence' of predators) and LVA ('low' temperature, 'variable' oscillations, and 'absence' of predators).

Usage

```
dat
```

Format

A data frame with 271 rows and 4 variables:

treatment Experimental treatment

number Replicate number

time Day of experiment

Chlamydomonas.terricola Species abundance

Source

Burgmer & Hillebrand 2011, Oikos 120:922-933.

`density_fun0` *Default density function for prior*

Description

Default density function, following the syntax for priors in the BayesianTools package. Uses flat priors for all paramters, within the given interval. Density function integrates to one.

Usage

```
density_fun0(param, minv, maxv)
```

Arguments

param	a vector model parameters
minv	a vector of minimum values for the interval
maxv	a vector of maximum values for the interval

Value

returns log likelihood of parameters given priors.

---

detfun0 *default deterministic function*

Description

Simulates deterministic component of Ricker model, of the form xt+1 = xt exp(exp(sdet[1])*(1-xt/exp(sdet[2])))

Usage

```
detfun0(sdet, xt, time = NULL, ...)
```

Arguments

sdet	a numeric vector of length two, specifying growth rate and carrying capacity
xt	a number or numeric vector of abundances at time t
time	the timestep - defaults to NULL (i.e. not used)
...	additional arguments, for compatability with other usages of the function - values are not used in this implementation

Value

a number or numeric vector of length xt, with predicted abundances at time t+1

detfun0_sin *deterministic function with time-varying carrying capacity*

Description

Simulates deterministic component of Ricker model, of the form xt+1 = xt exp(exp(sdet[1])*(1-xt/K)) where K varies with time as (sin(time/2)+exp(sdet[2])+0.5)*(2/3). Function is calibrated such that for exp(sdet[2]) = 1, mean(K) = 1.

Usage

```
detfun0_sin(sdet, xt, time = NULL, ...)
```

Arguments

sdet	a numeric vector of length two, specifying growth rate and carrying capacity
xt	a number or numeric vector of abundances at time t
time	the timestep - defaults to NULL (i.e. not used)
...	additional arguments, for compatability with other usages of the function - values are not used in this implementation

Value

a number or numeric vector of length xt, with predicted abundances at time t+1

EDMfun0 *EDM deterministic function*

Description

Estimates future states of xt based on based behaviour

Usage

```
EDMfun0(smp_cf, yp, x, minest = 0, maxest = NULL, time)
```

Arguments

smp_cf	a matrix of s-map coefficients. Columns correspond to intercept and time lags, rows to observations. Final column corresponds to intercept term.
yp	a matrix of covariates to be multiplied by the smp_cf (typically time lags). Should have one fewer column than smp_cf.
x	observation at time-1, to be used to make the prediction.
minest	minimum value to return for prediction - defaults to 0.
maxest	maximum value to return for prediction - defaults to NULL (no maximum)
time	the time step (i.e. position in smp_cf) for the desired prediction. Prediction will be made based on observation in preceding time point (i.e. time-1).

Value

a number or numeric vector of length xt, with predicted abundances at time t+1

Source

Adapted from Ye, Sugihara, et al. (2015), PNAS 112:E1569-E1576.

`getcm` *Get rates*

Description

Calculates colonization rate, mortality rate, and expected mean occupancy time based on a time series

Usage

```
getcm(dat)
```

Arguments

`dat`	a numeric vector, including the timeseries

Value

a list including colonization and mortality probability per time step (pc and pm, respectively), and pocc, the expected fraction of time that the species will be present

---

`ilogit` *Inverse logit*

Description

Returns the inverse logit transformation of x

Usage

```
ilogit(x, ...)
```

Arguments

`x`	a number, vector, matrix, etc. to be transformed from (-inf, inf) to (0 1) by the inverse logit transform
`...`	additional arguments to be passed to plogis

Value

transformed result

indexsort *Sort output of particle filter*

Description

Sorts outputs of particle filter based on index - returns a sorted list of particles, based on the sampling trajectory through time. This is a somewhat more accurate estiamte of the true posterior than are the stepwise samples provided by the filter.

Usage

```
indexsort(fulltracemat, fulltraceindex, nsmp = NULL)
```

Arguments

fulltracemat	full output of particles from the particleFilterLL function
fulltraceindex	full output of particle indices from the particleFilterLL function
nsmp	number of particle paths to sample - defaults to NULL, which samples all paths

Value

an index-sorted matrix - each column shows the trajectory of a single particle

inv_fun0 *Default inverse transormation function*

Description

Takes in a matrix, where each column represents a parameter. Returns parameters in untransformed space. If length = 2, then in the order (obs1, proc1). If 3, then in the order (obs1, proc1, proc2). If 4, then in the order (obs1, obs2, proc1, proc2). If 6, then in the order (obs1, proc1, pcol1, pcol2, det1, det2) If 7, then in the order (obs1, proc1, proc2, pcol1, pcol2, det1, det2) If 8, then in the order (obs1, obs2, proc1, proc2, pcol1, pcol2, det1, det2)

Usage

```
inv_fun0(x)
```

Arguments

x	an nxm matrix with

Value

returns back-transformed values of parameters

`likelihood0` *Default likelihood function*

Description

Calculates likelihood of vector y given parameter values in param, based on the particleFilterLL function.

Usage

```
likelihood0(
  param,
  y = y,
  parseparam = parseparam0,
  N = 1000,
  detfun = detfun0,
  edmdat = NULL,
  obsfun = obsfun0,
  procfun = procfun0,
  neff = FALSE,
  lowerbound = (-999)
)
```

Arguments

`param`	An unformatted vector of parameters, to be passed to parseparam function.
`y`	A numeric vector of observed values, from which the likelihood of parameters and functions will be determined.
`parseparam`	A function for transforming the vector param into a form that can be read by particleFilterLL. See particleFilterLL for details.
`N`	Number of particles to simulate. Defaults to 1e3.
`detfun`	A function that simulates deterministic dynamics, which takes in arguments sdet (parameters for deterministic model, taken from pars$proc), and xt, observed abundances at time t. Returns estimated abundances at time t+1 based on deterministic function (either a parametric function or an EDM function). Defaults to detfun0.
`edmdat`	A list including arguments to be passed to S_map_Sugihara1994 - see S_map_Sugihara1994 help file for details. Alternatively, the user can provide a matrix of pre-computed S-map coefficients, in element "smp_cf". Default for edmdat is NULL, which implies that EDM will not be applied - instead, a detfun and pars$det must be included.
`obsfun`	The observation error function to be used: defaults to obsfun0
`procfun`	The process noise function to be used: defaults to procfun0

neff	Should effective sample size be used to scale likelihood? Defaults to FALSE. TRUE uses automatic sample size, based on correlations in y. Otherwise, can be any positive number.
lowerbound	Lower bound for log likelihood. Filter will be re-run if the value falls below this threshold. NOTE - this option may induce a bias in the resulting likelihood (and subsequent parameter) estimates. Should only be set if the lower limit is indicative of filter failure (e.g. if all particles) are degenerate. Defaults to (-Inf) - i.e. no lower limit.

Value

Log likelihood generated by particleFilterLL function

`likelihood_EDM_piecewise`

calculate likelihood for piecewise data

Description

Calculates likelihoods across several segments of data - e.g. multiple plots from a single experiment. See documentation for particleFilterLL_piecewise for examples of use.

Usage

```
likelihood_EDM_piecewise(
  param,
  y,
  libuse_y,
  smap_coefs,
  Euse,
  tuse,
  N,
  colpar = c(logit(1e-06), log(0.1))
)
```

Arguments

param	parameters to be passed to likelihood0 function
y	the time series to be analyzed
libuse_y	a matrix with two columns, specifying the start end end positions of segments within vector y
smap_coefs	a matrix of s-mapping coefficients
Euse	embedding dimension for the s-mapping analysis
tuse	theta for s-mapping analysis
N	number of particles
colpar	parameters to be passed to the colfun0 - defaults to c(logit(1e-6), log(0.1))

Value

summed log likelihood across all segments

`logit` *Logit*

Description

Returns the logit transformation of x

Usage

```
logit(x, ...)
```

Arguments

`x`	a number, vector, matrix, etc. to be transformed from (0, 1) to (-inf inf) by the logit transform
`...`	additional arguments to be passed to plogis

Value

transformed result - impossible values are replaced with NA, without warnings

---

`logitnormal_imode` *Get inverse logit-normal mode*

Description

Returns a mean for a logit normal such that the mode will be centered around mu

Usage

```
logitnormal_imode(mu, sd)
```

Arguments

`mu`	the value around which the mode should be centered (in logit space)
`sd`	the standard deviation of the logit distribution (in logit space)

Value

the proposed mean for the distribution

lognormal_imode *Get inverse log-normal mode*

Description

Returns a mean for a lognormal such that the mode will be centered around mu

Usage

```
lognormal_imode(mu, sd)
```

Arguments

mu	the value around which the mode should be centered (in log space)
sd	the standard deviation of the lognormal distribution (in log space)

Value

the proposed mean for the distribution

makeblock *Make an embedding block from timeseries data*

Description

Returns a matrix X, where columns are time-delayed embeddings of Y, with number of embeddings specified by embedding dimension E. See help file for the S_map_Sugihara1994 function for examples.

Usage

```
makeblock(Y, E, lib = NULL)
```

Arguments

Y	a timeseries vector from which to build the embedding.
E	a positive integer, specifying the embedding dimension
lib	an optional matrix of library positions, for specifying cases where Y is a composite timeseries made up of multiple separate observations (e.g. spatial replicates). Matrix should have two columns, with the first row in each column specifying the start of the timeseries section, and the second column specifying the end.

Value

a matrix of time-delayed embeddings

`makedynamics_general` *Simulate general time series*

Description

Simulates a time series following a user-defined deterministic function, observation function, process noise function, and colonization function.

Usage

```
makedynamics_general(
  n = 1000,
  n0 = 0.1,
  pdet = c(log(3), log(1)),
  proc = c(log(1)),
  obs = c(log(1)),
  pcol = c(logit(0.2), log(1)),
  detfun = detfun0,
  procfun = procfun0,
  obsfun = obsfun0,
  colfun = colfun0,
  doplot = FALSE
)
```

Arguments

`n`	number of timesteps to simulate
`n0`	starting population size
`pdet`	a numeric vector of parameters for the deterministic function
`proc`	a numeric vector of parameters for the process noise function
`obs`	a numeric vector of parameters for the observation error function
`pcol`	a numeric vector of parameters for the colonization function
`detfun`	A function that simulates deterministic dynamics, which takes in arguments sdet (parameters for deterministic model, taken from pars$proc), and xt, observed abundances at time t. Returns estimated abundances at time t+1 based on deterministic function (either a parametric function or an EDM function). Defaults to detfun0.
`procfun`	A function that simulates process noise, which takes in arguments sp (parameters for process noise function, taken from pars$proc) and xt (abundances prior to process noise). Returns abundances after process noise has occurred. Defaults to procfun0.
`obsfun`	An observation function, which takes in up to five variables, including so (a vector of parameter values, inherited from pars$obs), yt (a number, showing observed abundance at time t), xt (predicted abundances), binary value "inverse",

and number "N". If inverse = TRUE, then function should simulate N draws from the observation function, centered around value yt. If inverse = FALSE, then function should return log probability denisty of observed value yt given predicted values in xt. Defaults to obsfun0.

`colfun` A function simulating colonization events, that takes in two arguments: co, a vector of parameter values taken from pars$pcol, and xt, a number or numeric vector of abundances at time t, before colonization has occurred. Returns predicted abundances after colonization has occurred. Defaults to colful0.

`doplot` a logical specifying wether output should be plotted - defaults to FALSE

Value

An n-by-3 dataframe of states, including obs (observed values), truth (true values), and noproc (values without process noise)

Examples

```
#run function
datout<-makedynamics_general(n=2e4, proc = c(-2,log(1.2)))

#show regression of variance vs. mean for binned data
datout_ps<-datout[datout$true>0 & datout$noproc>0,]
#bins
sq<-seq(0, quantile(datout$true, 0.95), length=50)
ctd<-cut(datout_ps$noproc, sq)
#calculate mean and variance by bin
tdat<-data.frame(mu=(sq[-1]+sq[-length(sq)])/2,
     var=tapply((datout_ps$true-datout_ps$noproc)^2, ctd, mean))
#plot result
plot(log(tdat$mu), log(tdat$var), xlab="mu", ylab="var")
#show regression
summary(mod<-lm(log(var)~log(mu), tdat)); abline(mod, col=2)
```

`obsfun0` *default observation noise function*

Description

Two options: If inverse=FALSE, calculates the log probability density of observation yt based on true state xt and observation error. Otherwise, simulates N random observations of yt. Observation error follows a Gaussian distribution truncated at zero, using a Tobit distribution. Note that probability density is calculated based on a Tobit distribution, with lower boundary zero.

Usage

```
obsfun0(
  so,
  yt,
```

```
  xt = NULL,
  inverse = FALSE,
  N = NULL,
  minsd = 0.01,
  time = NULL
)
```

Arguments

so	a numeric vector of length one, specifying either log-transformed standard deviation of the observation error as a fraction of the observation, or two log-transformed parameters of the form sd=exp(B0)+exp(B1)*x.
yt	a number, representing a potential observed value of xt
xt	a number or numeric vector of "true" (or simulated) abundances at time t, from which the likelihood of yt will be calculated - defaults to NULL for inverse=TRUE
inverse	a logical specifying whether inverse (i.e. random number generator) function should be implemented - defaults to FALSE
N	number of draws from the random number generator, if inverse=TRUE - defaults to NULL
minsd	minimum observation error allowed (e.g. if observation = 0), to prevent log likelihoods of -infinity - defaults to 0.01
time	the timestep - defaults to NULL (i.e. not used)

Value

If inverse=FALSE, returns a list including LL, a number or numeric vector of length xt, with predicted log likelihoods of observation yt, and wts, a number or vector with weights corresponding to the relative likelihood of each observation (after accounting for variable continuous vs. discrete probability distributions). If inverse = FALSE, returns N random draws from the observation function.

---

parseparam0 *Parse parameters*

Description

Takes in a vector of 3 or 6 parameters, and puts them into a list of the format expected by the particleFilterLL function.

Usage

```
parseparam0(
  param,
  colparam = c(logit(0.2), log(0.1)),
  detparam = c(log(1.2), log(1))
)
```

Arguments

param	List of paramters, of length 2, 3, 4, 6, 7, or 8. If 2, then in the order (obs1, proc1). If 3, then in the order (obs1, proc1, proc2). If 4, then in the order (obs1, obs2, proc1, proc2). If 6, then in the order (obs1, proc1, pcol1, pcol2, det1, det2) If 7, then in the order (obs1, proc1, proc2, pcol1, pcol2, det1, det2) If 8, then in the order (obs1, obs2, proc1, proc2, pcol1, pcol2, det1, det2) Note that if param is of length 2 or 3, then detparam and colparam must be supplied. See obsfun0, procfun0, and detfun0 for more details.
colparam	Optional vector of length two, including parameters for the colonization function.
detparam	Optional vector of length two, including paramters for the deterministic function.

Value

a formatted list of parameters

---

particleFilterLL *particle filter*

Description

General function for caluclating the log-likeihood of a stochastic discrete-time model, based on a noisy observation of time-series y. Returns estimates of true values of y, as well as for process noise, observation error, colonization rates, and extinction rates. Function is adapted from the R code of Knape and Valpine (2012), Ecology 93:256-263.

Usage

```
particleFilterLL(
  y,
  pars,
  N = 1000,
  detfun = detfun0,
  procfun = procfun0,
  obsfun = obsfun0,
  colfun = colfun0,
  edmdat = NULL,
  dotraceback = FALSE,
  fulltraceback = FALSE
)
```

Arguments

y	A numeric vector of observed values, from which the likelihood of parameters and functions will be determined.
pars	A list of parameter values. Must include elements obs (observation error parameters), proc (process noise parameters), and pcol (colonization parameters), which are passed on the their respecive functions, described below. If edmdat=NULL, then element det (deterministic process parameters) must be included.
N	Number of particles to simulate. Defaults to 1e3.
detfun	A function that simulates deterministic dynamics, which takes in arguments sdet (parameters for deterministic model, taken from pars$proc), and xt, observed abundances at time t. Returns estimated abundances at time t+1 based on deterministic function (either a parametric function or an EDM function). Defaults to detfun0.
procfun	A function that simulates process noise, which takes in arguments sp (parameters for process noise function, taken from pars$proc) and xt (abundances prior to process noise). Returns abundances after process noise has occurred. Defaults to procfun0.
obsfun	An observation function, which takes in up to five variables, including so (a vector of parameter values, inherited from pars$obs), yt (a number, showing observed abundance at time t), xt (predicted abundances), binary value "inverse", and number "N". If inverse = TRUE, then function should simulate N draws from the observation function, centered around value yt. If inverse = FALSE, then function should return log probability denisty of observed value yt given predicted values in xt. Defaults to obsfun0.
colfun	A function simulating colonization events, that takes in two arguments: co, a vector of parameter values taken from pars$pcol, and xt, a number or numeric vector of abundances at time t, before colonization has occurred. Returns predicted abundances after colonization has occurred. Defaults to colful0.
edmdat	A list including arguments to be passed to the S_map_Sugihara1994 function - see S_map_Sugihara1994 help file for details. Alternatively, the user can provide a matrix of pre-computed S-map coefficients, in element "smp_cf". Default for edmdat is NULL, which implies that EDM will not be applied - instead, a detfun and pars$det must be included.
dotraceback	A logical, indicating whether estimated values and demographic rates should be reported - defaults to FALSE
fulltraceback	A logical, indicating whether full matrix of particles for all time steps should be returned.

Value

LL (total log likelihood), LLlst (log likelihood for each time step), Nest (mean estimated state), Nsd (standard deviation of estimated state), Nest_noproc (mean estimated state at time t+1 without process error), Nsd_noproc (standard deviation of estimated state at time t+1 without process error), fulltracemat (full traceback of particle paths), fulltracemat_noproc (full traceback of particle paths at time t+1 without process noise), and fulltraceindex (index positions for the particle traces over time)

Source

Adapted from Knape and Valpine (2012), Ecology 93:256-263.

---

particleFilterLL_piecewise

run particle filter across piecewise data

Description

Calculates likelihoods across several segments of data - e.g. multiple plots from a single experiment. Requires several implicitely defined variables to run:

Usage

```
particleFilterLL_piecewise(
  param,
  N,
  y,
  libuse_y,
  smap_coefs,
  Euse,
  tuse,
  colpar = c(logit(1e-06), log(0.1)),
  nsmp = 1,
  lowerbound = -999,
  maxNuse = 512000
)
```

Arguments

param	parameters to be passed to parseparam0 function
N	number of particles
y	the time series to be analyzed
libuse_y	a matrix with two columns, specifying the start end end positions of segments within vector y
smap_coefs	a matrix of s-mapping coefficients
Euse	embedding dimension for the s-mapping analysis
tuse	theta for s-mapping analysis
colpar	parameters to be passed to the colfun0 - defaults to c(logit(1e-6), log(0.1))
nsmp	number of sample particle trajectories to return - defaults to 1
lowerbound	minimum accepted likelihood - used to automatically select number of particles. Defaults to -999
maxNuse	maximum number of particles to simulate - defaults to 512000

Value

results from particle filter - including mean estimates (Nest) and standard deviations (Nsd), across particles, and sample particle trajectories with (Nsmp) and without (Nsmp_noproc) process noise

Examples

```
# load data
data(dat)
# sort by index
dat = dat[order(dat$treatment, dat$number, dat$time),]


# make list of starting and ending positions for each replicate in the dat list
libmat<-NULL
trtmat<-data.frame(trt=as.character(sort(unique(dat$treatment))))
datnum<-1:nrow(dat)

for(i in 1:nrow(trtmat)) {
  ps1<-which(dat$treatment==trtmat$trt[i])
  replst<-sort(unique(dat$number[ps1]))

  for(j in 1:length(replst)) {
    ps2<-which(dat$number[ps1]==replst[j])
    libmat<-rbind(libmat, data.frame(trt=trtmat$trt[i], rep=replst[j],
      start=min(datnum[ps1][ps2]), end=max(datnum[ps1][ps2])))
  }
}

## run particle filter
# select treatment to analyse: enter either "LSA" or "LSP"
trtuse<-"HSP"
# extract library positions for treatment
libuse<-as.matrix(libmat[libmat$trt==trtuse,c("start", "end")])
# save abundance data to variable y
yps<-which(dat$treatment==trtuse)
y<-dat[,"Chlamydomonas.terricola"][yps]
libuse_y<-libuse-min(libuse)+1 # translate positions in dat to positions in y vector
y<-y/sd(y) # standardize to mean of one
timesteps<-dat$time[yps]

# get S-mapping parameters
sout<-data.frame(E = 2:4, theta = NA, RMSE = NA)
for(i in 1:nrow(sout)) {
  optout = optimize(f = function(x) {S_map_Sugihara1994(Y = y, E = sout$E[i],
      theta = x, lib = libuse_y)$RMSE}, interval = c(0,10))
  sout$theta[i] = optout$minimum
  sout$RMSE[i] = optout$objective
}
tuse<-sout$theta[which.min(sout$RMSE)] # find theta (nonlinerity) parameter
euse<-sout$E[which.min(sout$RMSE)] # find embedding dimension
spred<-S_map_Sugihara1994(Y = y, E = euse, theta = tuse, lib = libuse_y)
```

```
# set priors (log-transformed Beta_obs, Beta_proc1, and Beta_proc2)
minvUSE_edm<-c(log(0.001), log(0.001))  # lower limits
maxvUSE_edm<-c(log(2), log(2)) # upper limits


## Not run:
  ## Run filter
  # Commented-out code: Install BayesianTools package if needed
  #install.packages("BayesianTools")
  set.seed(2343)
  require(BayesianTools)
  density_fun_USE_edm<-function(param) density_fun0(param = param,
    minv = minvUSE_edm, maxv=maxvUSE_edm)
 sampler_fun_USE_edm<-function(x) sampler_fun0(n = 1, minv = minvUSE_edm, maxv=maxvUSE_edm)
  prior_edm <- createPrior(density = density_fun_USE_edm, sampler = sampler_fun_USE_edm,
                           lower = minvUSE_edm, upper = maxvUSE_edm)
  niter<-5e3 # number of steps for the MCMC sampler
  N<-2e3 # number of particles
  smap_coefs<-process_scof(spred$C) # coefficients from s-mapping routine

  # likelihood and bayesian set-ups for EDM functions
  likelihood_EDM_piecewise_use<-function(x) {
    # default values for filter - see likelihood_EDM_piecewise documentation for details
   # note that colpar are set near zero because we expect no colonisation into a closed microcosm.
    likelihood_EDM_piecewise(param=x, y, libuse_y, smap_coefs, euse, tuse, N,
                             colpar = c(logit(1e-06), log(0.1)))
  }

  bayesianSetup_EDM <- createBayesianSetup(likelihood = likelihood_EDM_piecewise_use,
     prior = prior_edm)

  # run MCMC optimization (will take ~ 5 min)
  out_EDM <- runMCMC(bayesianSetup = bayesianSetup_EDM,
     settings = list(iterations=niter, consoleUpdates=20))
  burnin<-floor(niter/5) # burnin period
  plot(out_EDM, start=burnin) # plot MCMC chains
  gelmanDiagnostics(out_EDM, start=burnin) # calculate Gelman statistic
  summary(out_EDM, start=burnin) # coefficient summary

  ## extract abundance estimate from particle filter
  # use final estimate from MCMC chain
  smp_EDM<-(getSample(out_EDM, start=floor(niter/5)))
 tmp<-particleFilterLL_piecewise(param = smp_EDM[nrow(smp_EDM),], N=N, y = y, libuse_y = libuse_y,
                                 smap_coefs = smap_coefs, Euse = euse, tuse = tuse)
  # mean estimated abundance
  simout<-tmp$Nest
  # sd estimated abundance
  sdout<-tmp$Nsd
  # sample from true particle trajectory
  simout_smp<-tmp$Nsmp
  # sample from true particle trajectory pre-process noise
  simout_smp_noproc<-tmp$Nsmp_noproc
```

```
  plot(timesteps, simout, xlab="Time", ylab="Abundance")
  abline(h=0, lty=3)

## End(Not run)
```

---

process_scof *Process S-mapping coefficients*

Description

Processes s-mapping coefficients from S_map_Sugihara1994 into a matrix of form C1, C2, C3, ... C0, where C0 is the intercept, C1 is the current time step t, C2 is timestep t-1, C3 is timestep t-2, and so on. Rows correspond to the time step used to produce the prediction, e.g. row 4 is used to calculate predicted value for time step 5. This is the format expected by the EDMfun0 function. See help file for the S_map_Sugihara1994 function for examples.

Usage

```
process_scof(smap_coefs)
```

Arguments

smap_coefs	a matrix of s-map coefficients, taken from the S_map_Sugihara1994 function.

Value

a matrix of s-mapping coefficients

---

procfun0 *default process noise function*

Description

Simulates effects of process noise following a Gaussian perturbation. Note that process noise only influences positive abundances (i.e. process noise cannot contribute to colonization)

Usage

```
procfun0(sp, xt, inverse = FALSE, time = NULL)
```

Arguments

sp	a numeric vector of length one or two, specifying either the log-transformed standard deviation of the process noise function, or an intercept and slope for calculating variance of process noise based on a power function of x, of the form var=exp(B0)*x^exp(B1)
xt	a number or numeric vector of abundances at time t, before process noise has occurred
inverse	a logical specifying whether the inverse (i.e. probability of drawing a value of zero given xt and sp) should be calcualted
time	the timestep - defaults to NULL (i.e. not used)

Value

a number or numeric vector of length xt, with predicted abundances after process noise has occurred

---

procfun_ct *continuous-time process noise function*

Description

Simulates effects of process noise following a Gaussian perturbation. Note that process noise only influences positive abundances (i.e. process noise cannot contribute to colonization)

Usage

```
procfun_ct(sp, xt, waiting_time = 1, time = NULL)
```

Arguments

sp	a numeric vector of length two or three, where terms 1-2 specify either the log-transformed standard deviation of the process noise function, or an intercept and slope for calculating variance of process noise based on a power function of x, of the form var=exp(B0)*x^exp(B1) The final term in the vector represents the recovery rate - i.e. the continuous time rate at which abundances recover from perturbation
xt	a number or numeric vector of abundances at time t, before process noise has occurred
waiting_time	average time between disturbance events: defaults to 1
time	the timestep - defaults to NULL (i.e. not used)

Value

a number or numeric vector of length xt, with predicted abundances after process noise has occurred

sampler_fun0 *Default sampler function for prior*

Description

Draws samples from a flat prior

Usage

```
sampler_fun0(n = 1, minv, maxv)
```

Arguments

n	number of random draws to take from the priors
minv	Vector of minimum values to return for each parameter
maxv	Vector of maximum values to return for each parameter

Value

returns random draws from the priors

sdproc_abstract *calculate estimated total variance*

Description

Function for estimating stochastic variation in linar process x as a function of relative growth rate and disturbance regime standard deviation.

Usage

```
sdproc_abstract(sd_proc, rgr, waiting_time = 1)
```

Arguments

sd_proc	standard deviation of the (Gaussian) disturbance process
rgr	relative growth rate of the linear process
waiting_time	average waiting time between (random exponentially distributed through time) disturbance events

Value

standard deviation of stochastic variability in x

S_map_Sugihara1994 *Apply S-mapping algorithm from Sugihara 1994*

Description

Carries out an S-mapping analysis, following the algorithm outlined in Sugihara (1994).

Usage

```
S_map_Sugihara1994(Y, E, theta, X = NULL, lib = NULL, trimNA = FALSE)
```

Arguments

Y	a timeseries vector from which to build the embedding.
E	a positive integer, specifying the embedding dimension
theta	a positive numeric scalar, specifying the nonlinearity parameter for the analysis. A value of 0 indicates a fully linear analysis; higher numbers indicate greater nonlinearity.
X	an optional matrix of time-delayed embeddings to use for the analysis
lib	an optional matrix of library positions, for specifying cases where Y is a composite timeseries made up of multiple separate observations (e.g. spatial replicates). Matrix should have two columns, with the first row in each column specifying the start of the timeseries section, and the second column specifying the end.
trimNA	a logical specifying whether NA values should be removed from Y and X - defaults to FALSE

Value

a list, including the timeseries used for S-mapping (Y), the delay embedding matrix used for S-mapping (X), a vector of predictions (Y_hat), a matrix of S-mapping coefficients (C), the standard errors for the S-mapping coefficients (C_SE), and goodness of fit metrics R-squared (R2) and root mean square error (RMSE).

Source

Sugihara, G. (1994). Nonlinear forecasting for the classification of natural time-series. Philos. Trans. R. Soc. -Math. Phys. Eng. Sci., 348, 477–495.

Examples

```
# create an example timeseries
n = 100
set.seed(1234)
datout<-makedynamics_general(n = n+2,
                             pdet=log(c(0.8,1)),
```

```
                              proc = -2.5,
                              detfun = detfun0_sin)
plot(datout$true, type = "l") # plot timeseries
Y = datout$true # extract true values

# run s-mapping
sout = S_map_Sugihara1994(Y = Y, E = 2, theta = 0.5)
s_coef = process_scof(sout$C) # process coefficients from the S-mapping output

# find best E/theta
fitout = data.frame(E = 1:5, theta = NA, RMSE = NA)

for(i in 1:nrow(fitout)) {
   E = fitout$E[i]
   Ytmp = Y[-c(1:E)]
  optout = optimize(f = function(x) {S_map_Sugihara1994(Ytmp, E, x)$RMSE}, interval = c(0,10))

  fitout$theta[i] = optout$minimum # get best theta for given E
  fitout$RMSE[i] = optout$objective # get error
}
ps = which.min(fitout$RMSE)

E = fitout$E[ps] # get best E
theta = fitout$theta[ps] # get best theta
X = makeblock(Y, E) # get X for analysis
Y = Y[-c(1:E)] # trim NA values (corresponding to positions in X)
X = X[(E+1):nrow(X),] # trim NA values
sout = S_map_Sugihara1994(Y = Y, E = E,
  theta = theta, X = X) # run S-mapping for best paramter combination
sout$R2 # look at R-squared

# check fit
plot(sout$Y_hat, Y)
abline(a=0, b=1, lty=2)
```

Index