Package 'layeranalyzer'

August 28, 2026

Type Package

Title Time Series Analysis Tool using Linear Layered SDEs

Version 0.4.1

Date 2026-08-16

Encoding UTF-8

Maintainer Trond Reitan <trond.reitan@geo.uio.no>

Description Time series analysis tool using linear layered stochastic differential equations. The package allows for multiple time series with correlative and/or causal links between them. Unmeasured causal processes are allowed to affect the measured processes in a layered structure, hence the name of the package. In case of causal feedback loops, the matrix operations (including eigenvalue decompositions) allows for complex numbers. In this case, cyclic behavior can be expected. Details can be found in Reitan and Liow (2019)<doi:10.1111/2041-210X.13299>.

License LGPL-3

Suggests knitr, rmarkdown, markdown

Depends R (>= 2.10)

Imports methods, stats, Rcpp, coda (>= 0.19.1)

LinkingTo Rcpp

URL https://folk.universitetetioslo.no/trondr/R/

LazyData true

ByteCompile yes

Biarch no

NeedsCompilation yes

Author Trond Reitan [aut, cre] (ORCID: <https://orcid.org/0000-0003-0793-8731>), Lee Hsiang Liow [ctb] (ORCID: <https://orcid.org/0000-0002-3732-6069>), Adam T. Kocsis [ctb] (ORCID: <https://orcid.org/0000-0002-9028-665X>)

Repository CRAN

Date/Publication 2026-08-28 09:50:08 UTC

Contents

anova.layered *Analysis of Deviance for return object from 'layer.analyzer'.*

Description

Compute an analysis of deviance table for one or more return object from "layer.analyzer".

Usage

```
## S3 method for class 'layered'
anova(object, ...)
```

Arguments

object, ...	Objects of class 'layered', as returned from the 'layer.analyzer' method.

Details

Performs analysis of deviance for a set of return objects from 'layer.analyzer'. These objects should be classically (ML) estimated, rather than Bayesian analysis (option 'do.maximum.likelihood=TRUE' in 'layer.analyzer'). These objects should represent nested models, with the simplest one first and the most complicated one last.

As long as the difference in degrees of freedom is larger than 0, a p-value from the chi-squared-test, (D_0-D_1)~chisq(res.df.0-res.df.1), will also be shown.

Value

An object of class "anova" inheriting from class "data.frame".

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer.

Examples

```
library(layeranalyzer)

# Read a simulated dataset with 2 layers:
test.ex<-read.table("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt")
names(test.ex)<-c("time","value")
ts<-layer.data.series(time=test.ex$time, value=test.ex$value,name="X")

# Calculate classic maximum likelihoods for an OU model and a 2 layered model.
# Test the number of layers. The dataset itself is simulated using 2 layers:
res1<-layer.analyzer(layer.series.structure(ts,numlayers=1),do.maximum.likelihood=TRUE)
res2<-layer.analyzer(layer.series.structure(ts,numlayers=2),do.maximum.likelihood=TRUE)
anova(res1,res2)
```

---

bi.lext *A dataset of reed warbler body mass over several years.*

Description

This example dataset was collected on bivalve/brachipod diversification rates in Liow et al. (2015) and re-analyzed by Reitan & Liow (2017). The specific dataset contains log-transformed bivalve extinction rates.

Details

The data is a list of class layer.data.series having five elements: time (millions of years (MY) since fozzilisation, negative value so that events are ordered chronologically). value (log-transformed extinction rates) name ("bivalve_lext") std.dev (standard error estimates of log-rates). is.datetime (0=false)

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

Source

https://datadryad.org/resource/doi:10.5061/dryad.hj30r

References

Liow, L. H., Reitan, T. and P. G. Harnik (2015). Ecological interactions on macroevolutionary time scales: clams and brachiopods are more than ships passing in the night. Ecological Letters 18(10): 1030-1039

Reitan, T. and L. H. Liow (2017). An unknown Phanerozoic driver of brachiopod extinction rates unveiled by multivariate linear stochastic differential equations. Paleobiology 43(4): 1-13.

See Also

See also other datasets: hare.norm, lynx.norm, br.lorig, br.lext.

br.lext *A dataset of reed warbler body mass over several years.*

Description

This example dataset was collected on bivalve/brachipod diversification rates in Liow et al. (2015) and re-analyzed by Reitan & Liow (2017). The specific dataset contains log-transformed brachiopod extinction rates.

Details

The data is a list of class layer.data.series having five elements: time (millions of years (MY) since fozzilisation, negative value so that events are ordered chronologically). value (log-transformed extinction rates) name ("brach_lext") std.dev (standard error estimates of log-rates). is.datetime (0=false)

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

Source

https://datadryad.org/resource/doi:10.5061/dryad.hj30r

References

Liow, L. H., Reitan, T. and P. G. Harnik (2015). Ecological interactions on macroevolutionary time scales: clams and brachiopods are more than ships passing in the night. Ecological Letters 18(10): 1030-1039

Reitan, T. and L. H. Liow (2017). An unknown Phanerozoic driver of brachiopod extinction rates unveiled by multivariate linear stochastic differential equations. Paleobiology 43(4): 1-13.

See Also

See also other datasets: hare.norm, lynx.norm, br.lorig, bi.lext.

br.lorig *A dataset of reed warbler body mass over several years.*

Description

This example dataset was collected on bivalve/brachipod diversification rates in Liow et al. (2015) and re-analyzed by Reitan & Liow (2017). The specific dataset contains log-transformed brachiopod origination rates.

Details

The data is a list of class layer.data.series having five elements: time (millions of years (MY) since fozzilisation, negative value so that events are ordered chronologically). value (log-transformed extinction rates) name ("brach_lorig") std.dev (standard error estimates of log-rates). is.datetime (0=false)

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

Source

https://datadryad.org/resource/doi:10.5061/dryad.hj30r

References

Liow, L. H., Reitan, T. and P. G. Harnik (2015). Ecological interactions on macroevolutionary time scales: clams and brachiopods are more than ships passing in the night. Ecological Letters 18(10): 1030-1039

Reitan, T. and L. H. Liow (2017). An unknown Phanerozoic driver of brachiopod extinction rates unveiled by multivariate linear stochastic differential equations. Paleobiology 43(4): 1-13.

See Also

See also other datasets: hare.norm, lynx.norm, br.lext, bi.lext.

---

compare.layered *Perform comparison of several models for the same dataset(s).*

Description

The function compares a set of models for the same dataset(s). The comparison is default done with Bayesian model probabilities. Prior model probabilities can be given, but if not, each model is given the same prior probability. If the model analysis is classic, AIC is default used for model comparison, but BIC or AICc can also be specified.

Usage

```
compare.layered(..., p0=NULL, first.is.nullhypothesis=FALSE, ML.IC="AIC")
```

Arguments

...	A set of 'layered' objects returned from the layer.analyzer function. This set represents the model analyses performed and should contain information used for model comparison (marginal likelihoods for Bayesian analyses, AIC/BIC/AICc for classic analyses).

`p0` A numeric array describing the prior probabilities of each models sent to the function. The size of the array needs to match the number of models sent to the function. If prior probability array, 'p0', is not given each model is default given the same prior probabiliuty, but see item 'first.is.nullhypothesis'.

`first.is.nullhypothesis`
If prior probabilities (item 'p0') is not given, the prior probabilities are default equal for all models. However, if first.is.nullhypothesis is set to 'TRUE', the first model is given 50% of the prior probability, while the rest of the models shares the remaining 50% equally.

`ML.IC` If classic ML analysis is performed instead of Bayesian analysis, "AIC","BIC" or "AICc" will be used for model comparison instead of the Bayesian marginal likelihood. The item 'ML.IC' determines which of these three model comparison criteria are to be used.

Details

The first arguments to this function should be a set of models represented by 'layered' objects returned from the layer.analyzer function.

When Bayesian analyses have been performed (default), Bayesian marginal likelihoods are used. (This means the option 'do.model.likelihood' in the 'layer.analyzer' function must not be set to false in any of the analyses). The model prior probabilities can be given, but if not, equal prior probability will be assumed.

The item 'ML.IC' is used when classic analyses have been performed, in order to determined which model comparison criteria will be used for that purpose.

The same dataset or set of datasets should be analysed for all models. This is something that can go wrong when comparing a hypothesis of no causal connection from process B to A versus a hypothesis of there being such a causal connection. Both time series A and B needs to be given, not only for the analysis of the connection hypothesis but also for the no connection hypothesis.

Value

Returns a matrix containing the names of the model objects and the posterior probabilties (or weights if classic analyses has been used).

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

Liow, L.H., Reitan, T., Harnik, P.G. (2015) Ecological interactions on macroevolutionary time scales; clams and brachiopods are more than ships that pass in the night, Ecological Letters, Volume 18(10): 1030-1039.

See Also

See also layer.analyzer.

Examples

```
library(layeranalyzer)

# Read a simulated dataset with 2 layers:
test.ex<-read.table("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt")
names(test.ex)<-c("time","value")
ts<-layer.data.series(time=test.ex$time, value=test.ex$value,name="X")

# ML estimation for an OU model, low computer resources (including low pre-optimization MCMC)
# to speed things up

res1<-layer.analyzer(layer.series.structure(ts,numlayers=1),
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20,
  num.MCMC=40,burnin=400)

# ML estimation for a two-layered model:
res2<-layer.analyzer(layer.series.structure(ts,numlayers=2),
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20,
  num.MCMC=40,burnin=400)

# ML estimation for a three-layered model:
res3<-layer.analyzer(layer.series.structure(ts,numlayers=3),
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20,
  num.MCMC=40,burnin=400)

# Compare using AIC (Akaike's Information Criterion):
compare.layered(res1,res2,res3, ML.IC="AIC")
```

---

hare.norm *A dataset of hare catchment data from the Hudson Bay Company.*

Description

This example dataset was collected on Canadian hare cathcment data from the Hudson Bay Company and used in Maclulich (1937). Log-transformed, normalized and made into a layer.data.series.

Details

The data is a list of class layer.data.series and which has four elements: 'time' (years), 'value' (normalized log-catchments), name ("hare") and is.datatime (0, which means 'false').

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Maclulich, D. A. (1937). Fluctuations in the number of the varying hare (Lepus Americanus), University of Toronto Studies, Biological series 43.

See Also

See also other datasets: malta, lynx, bi.lext, br.lorig, br.lext.

`hare.orig` *A dataset of hare catchment data from the Hudson Bay Company.*

Description

This example dataset was collected on Canadian hare cathcment data from the Hudson Bay Company and used in Maclulich (1937).

Details

The data has two columns containing year (time) and annual catchment value (value).

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Maclulich, D. A. (1937). Fluctuations in the number of the varying hare (Lepus Americanus), University of Toronto Studies, Biological series 43.

See Also

See also other datasets: malta, hare.norm, lynx.norm, bi.lext, br.lorig, br.lext.

`layer.analyzer` *Time series analysis tool using linear layered SDEs.*

Description

Linear SDE analysis allowing for hidden layers and multiple data series with correlative or causal links between them. Returns parameter estimates and Bayesian model likelihoods or information criteria, depending on whether the analysis is Bayesian (default) or maximum likelihood (ML) based. In case of causal feedback loops, the matrix operations (including eigenvalue decompositions) allows for complex numbers. In this case, cyclic behavior can be expected.

Usage

```
layer.analyzer(... ,
  num.MCMC=1000,spacing=10,burnin=10000,num.temp=1,
  do.model.likelihood=TRUE,
  do.maximum.likelihood=FALSE,maximum.likelihood.numstart=10,
  maximum.likelihood.strategy="MCMC-from-model",
  silent.mode=TRUE,talkative.burnin=FALSE,talkative.likelihood=FALSE,
  id.strategy=2,use.stationary.stdev=TRUE,T.ground=1.5,
  use.half.lives=FALSE, mcmc=FALSE,causal=NULL,causal.symmetric=NULL,corr=NULL,
  smoothing.specs=
     list(do.smoothing=FALSE,smoothing.time.diff=0,
          smoothing.start=NULL,smoothing.end=NULL,
          num.smooth.per.mcmc=10, do.return.smoothing.samples=FALSE),
  realization.specs=
     list(do.realizations=FALSE,num.realizations=1000,strategy="N",
          realization.time.diff=0,realization.start=NULL,realization.end=NULL),
  return.residuals=FALSE,
  loglik.laxness="high",
  external.series=NULL,
  external.series.connection=integer(0),
  external.layer.connection=integer(0),
  site.distance.matrix=NULL,
  conn=NULL)
```

Arguments

...	A set of 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least one such object must be given.
num.MCMC	num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the analysis will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).
spacing	Specifies the number of MCMC iteration between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.
burnin	Markov chain Monte Carlo sampling takes some iterations to converge to the posterior parameter distribution. The burn-in phase is the set of iterations performed where the samples are then discarded, so as to avoid including samples before convergence. Convergence can be check by for instance 'gelman.diag' in the 'coda' package, and the MCMC samples themselves can be gotten by using the option 'mcmc'. Still, stability of the results can be quite a good enough measure for most purposes. Default is 10000 iterations. PS: The burn-in algorithm here includes two steps where the random walk MCMC variance is adjusted in order to optimize the efficiency of the algorithm.
num.temp	The number of tempering chains (default 1, meaning it is just the MCMC chain itself that is beeing run). Parallel tempering (Geyer 1991) is a method for over-

coming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`do.model.likelihood`

Logical variable. Only makes sense for Bayesian analysis (which is default). If set to 'TRUE' (default), the Bayesian model (marginal) likelihood is estimated using an importance sampler. If set to 'FALSE', this will not happen, which saves time but means that Bayesian model comparison will not be possible.

`do.maximum.likelihood`

Logical variable. If set to 'FALSE' (default), Bayesian analysis is performed. If set to 'TRUE', a classic maximum likelihood (ML) is performed, where the starting points of a set of optimizations is drawns from the MCMC samples. (This means that Bayesian MCMC sampling is performed anyway, but that the results from this is not focused on in the summary). The number of optimizations is determined by option 'maximum.likelihood.numstart', which is default set to 10. Estimation of the Bayesian model (marginal) likelihood is switched off when this option is used.

`maximum.likelihood.numstart`

The number of ML optimizations performed, it applicable. (If 'do.maximum.likelihood' is set to 'TRUE'.) Default:10.

`maximum.likelihood.strategy`

In layeranalyzer, ML optimizations are per default (maximum.likelihood.strategy="MCMC-from-model") started from MCMC samples,a s it was found that completely randomly drawn starting points worked poorly for large models. However, MCMC convergence is no guarantee either, especially for models with multiple datasets and connections between them. 'maximum.likelihood.strategy' allows for using alternative startegies for optimization in connection models by MCMC sampling from a model that is the same in structure but without connections and using these samples as starting points for the optimization instead, which is specified by setting maximum.likelihood.strategy="MCMC-from-nonconnected". The third possibility is to set maximum.likelihood.strategy="MCMC-from-hybrid", which alternates betwen using MCMC samples from the connection model and from MCMC samples for the non-connected model.

`silent.mode` If set to 'FALSE', the routine will show a lot debug information while running. This can be useful for tempering purposes, since the number of tempering swaps are shown in this debug information.

`talkative.burnin`

If set to 'TRUE', this triggers the printing of more specific debug information concerning the burn-in phase.

`talkative.likelihood`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the likelihood calculation phase. (PS: This means a lot of printing.)

`id.strategy`
: Determines how to handle the identifiability problem in Bayesian multi-layer analysis. As mentioned in the supplementary of Reitan et al. (2012), one can switch the characteristic times of neighbouring layers and reorganize the stochastic contributions in such a way that the same top layer process takes place. This can be solved by requiring that the lower the layer, the larger the characteristic time. However, simply requiring this means the prior distribution of the characteristic times get shortened for multi-layered models, as compared to one-layered model. As long as the data is within the range of these shortened prior distributions, this puts the one-layer model at a disadvantage when it comes to model comparison. There are various solutions to this problem, and which solution is used is determined by the variable 'id.strategy'.

  The variable 'id.strategy' should be an integer between 0 and 4, default 2. 0 - No identification treatment. (Default) PS: Can and even should yield multimodal characteristic times 1 - Keep upper characteristic time prior. Add lognormally to beneath-lying characteristic times. 2 - Keep lower characteristic time prior. Substract lognormally to above-lying characteristic times. 3 - Keep lower characteristic time prior. Cut depending on that on the above-lying characteristic times. 4 - Keep upper characteristic time prior. Cut depending on that on the below-lying characteristic times.

`use.stationary.stdev`
: If set to 'TRUE', instead of reporting the stochastic contribution size in the stochastic differential equations, the stationary standard deviation is reported. The stationary standard deviation is the standard deviation of the process state at any given time after convergence, unconditional on previous states. In a multi-layered model, this is to be interpreted as the stationary standard deviation if this layer when let alone (with no other process affecting it). This is calculated as s*sqrt(characteristic time/2), where s is the stochastic contribution size. PS: This option does not make any sense if the bottom layer is a Wiener process, as that process is not stationary.

`T.ground`
: This determines how the parallel tempering chains are to be defined. Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

  As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`use.half.lives`
: If set to 'TRUE', reports half-lives rather than characteristic times. Half-lives are the time the auto-correlation of an OU process drops to 1/2, or the time

it takes for a perturbation from the expected value to drop to 1/2 the original size in expected value. (As opposed to dropping to exp(-1) for characteristic times.) Half-life=log(2)*characteristic time. The auto-correlation for an OU process (and thus for the inner dynamics of any linear layer) is exp(-diff.time/characteristic.time)=exp(-log(2)*diff.time/half.life)= (1/2)^(diff.time/half.life), where diff.time is the time difference between two process states.

`mcmc` If set to 'TRUE', returns the MCMC samples as an 'mcmc' object within the return object. The name of the list object will also be 'mcmc'.

`causal` If multiple time series are given, one can specify causal connections from underlying processes belonging to such time series to the underlying processes belonging to some other time series. This specification should come in the form of an 4 x n matrix, where n is the number of causal connections. For each column (i.e. for each connection), row 1 specifies the number of the cause time series (same as the ordering of the input 'layer.series.structure' input to the routine), row 2 specifies the layer of the cause process, row 3 specifies the number of the effect time series and row 4 specifies the layer of the effect process. So if one has two time series, X and Y, both being modelled as having 3 layers, one can specify a causal connection from layer 3 in X to layer 1 in Y by the matrix "causal<-matrix(c(1,3,2,1),ncol=1)".

Note that it is possible to specify a causal connection even with only one time series, if a multi-layered process is considered. So if X is two-layered, "causal<-matrix(c(1,1,1,2),ncol=1)" will work. As layers are defined so that there is always a causal connection from layer 'k' to layer 'k-1' (with strength=1 for identifiability purposes), the reverse specification ("causal<-matrix(c(1,2,1,1),ncol=1)") will not make sense.

`causal.symmetric`

Specifies a set of causal connections going both ways with equal strength. For more on the usage, see option 'causal', but note that symmetric causal connection between one layer and the next does not make sense, as the connection strength from layer 'k' to layer 'k-1' is always set to "1" for identifiability purposes. Note that which series is first and which is second does not matter, so the matrix "causal.symmetric<-matrix(c(2,1,1,3),ncol=1)" would specify the same model as "causal.symmetric<-matrix(c(1,3,2,1),ncol=1)".

`corr` A 4 x n matrix which specifies the correlative connections between the processes, where n is the number of such connections. For each column (i.e. for each connection), row 1 specifies the number of the first time series (same as the ordering of the input 'layer.series.structure' input to the routine), row 2 specifies the layer of the first process, row 3 specifies the number of the second time series and row 4 specifies the layer of the second process. So if one has two time series, X and Y, both being modelled as having 3 layers, one can specify a correlative connection between layer 3 in X and layer 1 in Y by the matrix "corr<-matrix(c(1,3,2,1),ncol=1)". Note that which series is first and which is second does not matter, so the matrix "corr<-matrix(c(2,1,1,3),ncol=1)" would specify the exact same model.

`smoothing.specs`

A list that specifies whether smoothing samples are to be returned and if so how. Smoothing samples are a posteriori samples of the process states at measurement

times and possibly also other times. Smoothing samples can be turned on by setting smoothing.specs$do.smoothing=TRUE. If so, the smoothing.specs$smoothing.time.diff and smoothing.specs$num.smooth.per.mcmc must also be given. If smoothing.specs$smoothing.time.diff is set to "0", only the states at the measurement times will be inferred. If smoothing.specs$smoothing.time.diff is set to a positive real value, the state at the regularly spaced time points (smoothing.specs$smoothing.start, smoothing.specs$smoothing.start+smoothing.specs$smoothing.time.diff, smoothing.specs$smoothing.start+2*smoothing.specs$smoothing.time.diff, ... smoothing.specs$smoothing.end) will also be reported. Note that this means that smoothing.specs$smoothing.start and smoothing.specs$smoothing.end must also be given. Also, if smoothing samples are to be given, smoothing.specs$num.smooth.per.mcmc must be given. This is the number of smoothing samples fetched per MCMC sample. Lastly, smoothing.specs$do.return.smoothing.sample determines if the smoothing samples are to be returned or not. If smoothing.specs$do.smoothing=TRUE, it rarely makes sense to set smoothing.specs$do.return.smoothing.sample<-FALSE, so if this is not given, the algorithm sets smoothing.specs$do.return.smoothing.sample<-smoothing.specs$do.smoothing.

`realization.specs`

A list that specifies whether realization samples are to be returned and if so how. Realization samples are a posteriori samples of the process itself (including the correlations between time points) at measurement times and possibly also other times. Realization samples can be turned on by setting realization.specs$do.realizations=TRUE. If so, the realization.specs$realization.time.diff and realization.specs$num.realizations must also be given. If realization.specs$realization.time.diff is set to "0", only the states at the measurement times will be inferred. If realization.specs$realization.time.diff is set to a positive real value, the state at the regularly spaced time points (realization.specs$realization.start, realization.specs$realization.start+realization.specs$realization.time.diff, realization.specs$realization.start+2*r ... realization.specs$realization.end) will also be reported. Note that this means that realization.specs$realization.start and realization.specs$realization.end must also be given. Also, if realization samples are to be given, realization.specs$num.realizations must be given. This is the number of realization samples fetched in total. (This cannot be more than the number of MCMC samples. the reason it is not equal to that, is that when censoring is in use, not all MCMC samples will result in valid realizations.)

Lastly, realization.specs$strategy determines whether censoring of realizations is to take place. Censoring has been used when the process in question represented sedimentation, in order to avoid realizations such that the process would have erased one or more of the measurements. Thus if X is sedimentation depth and X_measurement < X_later_time_point for any combination of measurement or later time point, a censoring would take place. realization.specs$strategy="N" means no censoring. realization.specs$strategy="A" means ascending censoring (censors realizations that anywhere ascends beyond the value of the previous measurement point and realization.specs$strategy="D" means descending censoring (censors realizations that anywhere descends beyond the value of the previous measurement point).

`return.residuals`

If set to 'TRUE', the standardized residuals for all observed series as a matrix

called 'standardized.residuals' (one column for each measured series) and the returns the time points for the residuals ('residuals.time'). Also returns the prior expected values from the Kalman filter ('prior.expected.values'), which should be independent of the residuals if the model assumptions are correct. Note that the residuals are calculated for the Bayesian median parameters if Bayesian analysis is used and maximum likelihood estimated parameters if classic (ML) analysis is used. Also note that the residuals for each observed series is the measurement minus the expected value divided by the uncertainty from the Kalman *filter* not the Kalman smoother (which is used for process inference), as it is the deviations in the Kalman filter that are assumed to be independent from measurement to measurement.

`loglik.laxness` There are sanity checks for each measurement point handled during the log-likelihood calculation, in order to avoid errors in the calculation due to numerical errors. However, some numerical inaccuracies will always exist, so some laxness in these checks must be allowed. Varianles that should be equal might not be exactly equal. And covariance matrices, which should be symmetric, might not be exactly symmetric or be so nearly singular that numerical errors destroys positive definiteness (only checked for levels "low" and "moderate"). Too much laxness and erroneous log-likelihoods may be used. Too much rigidity, i.e. too little laxness and parameter values may needlessly be excluded and the optimal estimation is not achieved. Default laxness is "high", but options are also "low" (very rigid), "moderate" and "complete" (sanity checks are not performed).

PS: The test covariance matrix test for positive definiteness (positive eigenvalues) do take quite a lot of computational resources. If you run with loglik.laxness="complete" or "high" (default), you might run things as much as 3 times faster! But unless you run some fairly simple models, be sure to check both parameter estimates and the stability of the log-likelihood. (Perhaps compare it to a simpler model). loglik.laxness="high" (default), doesn't cost much more than loglik.laxness="complete" (nothing is checked), but doesn't check for positive definiteness.

`external.series`

External series are time series that are treated as explanatory variables for the time series that are analyzed. They are not themselves analyzed statistically. Instead they are taken as is, with the assumption that what happenes between time points can be exppromately attained through linear interpolation. Since the stochastic nature of these series are not analyzed, only causal links are possible. Note that if external time series are given, one must also specify which series and layers they go to (see the next two entries.) Note that external data series are assumed global, and this connect equally to all sites (if that is a thing in your datasets).

`external.series.connection`

An vector of integers that should specify which external time series are causally connected to which observed series, numbered from 1 to the number of series. The length of this vector should match the number of external time series.

`external.layer.connection`

An vector of integers that should specify which external time series are causally connected to which layers of the observed series, numbered from 1 to the number of layers in the series specified in 'external.series.conneciton'. The length

of this vector should match the number of external time series.

`site.distance.matrix`

A matrix that specifies the distance between each pair of sites. Only relevant when on series uses the 'distance.correlated.sigma' option to specify between-site correlation based on distance and a distance correlation falloff parameter. But in such a case, the site distance matrix is very necessary! (Keep in mind, sites are numbered from 0 to number of sites-1, so element i,j in the matrix should tell the distance between site i-1 and site j-1. Also keep in mind that the size of the matrix must match the number of sites in the dataset. Also, diagonal elements are the distance from a specific site to itself and so shoulød be 0. Also, the matrix must be square and symmetric, since the distance from site i-1 to site j-1 is the distance from site j-1 to site i-1.)

`conn`

Alternative connection specification through an array of integers between 0 and 4, one for each possible pairing of two processes (series+layer) belonging to different series (see 'layer.pairs'). For each pair, 0 means no connection between the two, 1 means correlative connection between the two, 2 means causal connection from the first process in the pair to the second, 3 means causal connection from the second process in the pair to the first and 4 means causal connection going both ways between the pair of processes.

Used instead of 'causal' and 'corr' if one wants a compact (but less user-friendly) way to specify connections. This alternative way of specifying a connection was introduced in order to make it easy to produce different connection models by simply traversing the possible connection between each pair of processes. If one use a script to traverse all possible connections models, reads a script and corrects the connection specification, one has a tool for outputing one script for each connection models. These scripts can then possibly be run in parallel on a computer with many cores or a computing cluster.

See the function 'layer.pairs' and 'layer.connections' for ways to make sure that you use this specification correctly.

Details

The function takes a set of time series along with their process structure (represented by layer.series.structure objects), together with specifications for causal and correlative links, thus specifying data and the model. Run conditions for the numerics can also be specified. Performs parameter inference, either Bayesian using MCMC or classic (ML) using optimization from MCMC samples. Bayesian model likelihoods are calculated using an importance sampling method, while ML optimization gives what is needed for information criterion based model comparison. Process inference can also be performed, if specified. The numerical work is performed by a compiled C++ routine, in order to minimize computation time.

Each data set+process structure can also contain a prior hyperparameter specification for the Bayesian analysis, see 'layer.prior'.

Analyses of different models can be compared using the 'compare.layered' routine.

Value

Returns an object of type 'layered', a list containing parameter estimates and uncertainties, plus anything of extra output specified by the user. Extra output includes log(BML) for bayesian analy-

sis, AIC, AICc, BIC for classic analysis, MCMC samples, process smoothing samples and process realization samples.

An object of the class 'layered' will typically have the following components:

mu_<series name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the expected value parameter of each series. (PS: Not given for Wiener processes or layered processes having the Wiener process as the bottom layer).

dt_<series_name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the expected characteristic of each series. (Alternatively, the half-life will be given if use.half.life=TRUE).

sigma_<series_name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the stochastic contribution parameter of each series. (Alternatively the stationary standard deviation will be given, if use.stationary.stdev=TRUE).

model.log.lik: Bayesian marginal model likelihood (unless do.model.likelihood=FALSE or ML analysis).

obs_sd_<series_name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the observational standard deviation (if not all measurements are assigned and observational standard deviation). Thus this parameter describes the uncertainty in the measurements, not the stochastic nature of the process.

computing.time: Computing time in seconds.

parameter.names: Array of parameter names.

data.structure: The data strucucture(s) sent to the analysis (including data, prior, number of layes etc.)

causal: Causal connections specified.

corr: Correlative connections specified.

causal.symmetric: Symmetric causal connections specified.

mcmc: Only returned if "mcmc=TRUE" in the input. Contains MCMC samples for each parameter.

mcmc.origpar: Only returned if "mcmc=TRUE" in the input. Contains MCMC samples for each parameter using the original parametrization, i.e. the parametrization used internally. (So log characteristic time instead of characteristic time, for instance.)

est.origpar: Estimates for original parametrization (see 'mcmc.origpar').

input.options: A full list of the options used in this analysis.

process.time.points, process.mean, process.lower95, process.upper95: Process time points, means, upper and lower credibility bands for each process. (PS: There can be multiple prcoesses per serie, since number of layers can be greater than one). Only returned if option "smoothing.specs=list(do.smoothing=TRUE,...)" is used.

smoothing.samples: Smoothing samples for each MCMC sample for each process. Only returned if option "smoothing.specs=list(do.smoothing=TRUE,...,do.return.smoothing.samples = TRUE)" is used.

standardized.residuals: Standardized residuals are measurements minus expected values from the Kalman filter, divided by the standard deviation also from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

prior.expected.values: Expected values from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

residuals.time: Time points for each residual. Only returned if "return.residuals=TRUE" is set.

realizations: Process realizations given measurements. Only returned if "realization.specs=list(do.realizations=TRUE,...)" is set.

realization.time.points: Time point for each realization. Only returned if "realization.specs=list(do.realizations=TRUE,...)" is set.

description: A description of the model can be found in the field 'description',containing structure descriptions and causal and correlative connections.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

Geyer CJ (1991), Comp Sci Stat Proc 23rd Symp Inter: 156. Am Stat Ass, New York

See Also

See also layer.series.structure, layer.data.series, layer.prior, summary.layered and compare.layered.

Examples

```
library(layeranalyzer)
# Simulated a sample from the Ornstein-Uhlenbeck (OU) process with mu=0,
# characteristic time=50 and sigma=0.2:
n=1000
x=rep(rnorm(1),n)
for(i in 2:n)
 x[i]=exp(-1/50)*x[i-1]+0.2*rnorm(1)

# Create measurement set as a subsample of the process, plus
# noise with standard deviation 0.1
t=sort(sample(1:n,300))
y=(x+rnorm(length(x),0,0.1))[t]

# Create the data series object from the time points and "measurements":
X=layer.data.series(time.points=t, value.points=y,name="X")

# Run with few resources (since this is a test example).
# PS: ML-based estimates (default is Bayesian)
res=layer.analyzer(layer.series.structure(X,numlayers=1),
  num.MCMC=40,burnin=400,
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20)

# Look at the results:
summary(res)
```

```
# PS: Uncertainty estimates comes from Bayesian analysis and will
# in this case not be very stable (since few resources).
```

`layer.analyzer.timeseries.list` *Time series analysis tool using linear layered SDEs.*

Description

This is a version of the analysis function, "layer.analyzer" that takes a list of "layer.series.structure" objects as input, instead of a set of specified "layer.series.structure" objects, as "layer.analyzer" do. The "layer.analyzer" function calls this function, which again calls a C++ library to get the work done.

Linear SDE analysis allowing for hidden layers and multiple data series with correlative or causal links between them. Returns parameter estimates and Bayesian model likelihoods or information criteria, depending on whether the analysis is Bayesian (default) or maximum likelihood (ML) based. In case of causal feedback loops, the matrix operations (including eigenvalue decompositions) allows for complex numbers. In this case, cyclic behavior can be expected.

Usage

```
layer.analyzer.timeseries.list(data.structure ,
  num.MCMC=1000,spacing=10,burnin=10000,num.temp=1,
  do.model.likelihood=TRUE,
  do.maximum.likelihood=FALSE,maximum.likelihood.numstart=10,
  silent.mode=TRUE,talkative.burnin=FALSE,talkative.likelihood=FALSE,
  id.strategy=2,use.stationary.stdev=TRUE,T.ground=1.5,
  use.half.lives=FALSE, mcmc=FALSE,causal=NULL,causal.symmetric=NULL,corr=NULL,
  smoothing.specs=
     list(do.smoothing=FALSE,smoothing.time.diff=0,
          smoothing.start=NULL,smoothing.end=NULL,
          num.smooth.per.mcmc=10, do.return.smoothing.samples=FALSE),
  realization.specs=
     list(do.realizations=FALSE,num.realizations=1000,strategy="N",
          realization.time.diff=0,realization.start=NULL,realization.end=NULL),
  return.residuals=FALSE, smooth.previous.run=FALSE, previous.run=NULL,
  layer.analyzer.mode="Bayes",
  loglik.laxness="high", external.series=NULL,
  external.series.connection=integer(0),
  external.layer.connection=integer(0),
  site.distance.matrix=NULL)
```

Arguments

`data.structure`	A list containing 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such

series is to be modelles. (See 'layer.series.structure'). At least one such object must be given.

`num.MCMC` num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the analysis will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).

`spacing` Specifies the number of MCMC iteration between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.

`burnin` Markov chain Monte Carlo sampling takes some iterations to converge to the posterior parameter distribution. The burn-in phase is the set of iterations performed where the samples are then discarded, so as to avoid including samples before convergence. Convergence can be check by for instance 'gelman.diag' in the 'coda' package, and the MCMC samples themselves can be gotten by using the option 'mcmc'. Still, stability of the results can be quite a good enough measure for most purposes. Default is 10000 iterations. PS: The burn-in algorithm here includes two steps where the random walk MCMC variance is adjusted in order to optimize the efficiency of the algorithm.

`num.temp` The number of tempering chains (default 1, meaning it is just the MCMC chain itself that is beeing run). Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`do.model.likelihood`

Logical variable. Only makes sense for Bayesian analysis (which is default). If set to 'TRUE' (default), the Bayesian model (marginal) likelihood is estimated using an importance sampler. If set to 'FALSE', this will not happen, which saves time but means that Bayesian model comparison will not be possible.

`do.maximum.likelihood`

Logical variable. If set to 'FALSE' (default), Bayesian analysis is performed. If set to 'TRUE', a classic maximum likelihood (ML) is performed, where the starting points of a set of optimizations is drawns from the MCMC samples. (This means that Bayesian MCMC sampling is performed anyway, but that the results from this is not focused on in the summary). The number of optimizations is determined by option 'maximum.likelihood.numstart', which is default set to 10. Estimation of the Bayesian model (marginal) likelihood is switched off when this option is used.

`maximum.likelihood.numstart`
: The number of ML optimizations performed, it applicable. (If 'do.maximum.likelihood' is set to 'TRUE'.) Default:10.

`silent.mode`
: If set to 'FALSE', the routine will show a lot debug information while running. This can be useful for tempering purposes, since the number of tempering swaps are shown in this debug information.

`talkative.burnin`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the burn-in phase.

`talkative.likelihood`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the likelihood calculation phase. (PS: This means a lot of printing.)

`id.strategy`
: Determines how to handle the identifiability problem in Bayesian multi-layer analysis. As mentioned in the supplementary of Reitan et al. (2012), one can switch the characteristic times of neighbouring layers and reorganize the stochastic contributions in such a way that the same top layer process takes place. This can be solved by requiring that the lower the layer, the larger the characteristic time. However, simply requiring this means the prior distribution of the characteristic times get shortened for multi-layered models, as compared to one-layered model. As long as the data is within the range of these shortened prior distributions, this puts the one-layer model at a disadvantage when it comes to model comparison. There are various solutions to this problem, and which solution is used is determined by the variable 'id.strategy'.

  The variable 'id.strategy' should be an integer between 0 and 4, default 2. 0 - No identification treatment. (Default) PS: Can and even should yield multimodal characteristic times 1 - Keep upper characteristic time prior. Add lognormally to beneath-lying characteristic times. 2 - Keep lower characteristic time prior. Substract lognormally to above-lying characteristic times. 3 - Keep lower characteristic time prior. Cut depending on that on the above-lying characteristic times. 4 - Keep upper characteristic time prior. Cut depending on that on the below-lying characteristic times.

`use.stationary.stdev`
: If set to 'TRUE', instead of reporting the stochastic contribution size in the stochastic differential equations, the stationary standard deviation is reported. The stationary standard deviation is the standard deviation of the process state at any given time after convergence, unconditional on previous states. In a multi-layered model, this is to be interpreted as the stationary standard deviation if this layer when let alone (with no other process affecting it). This is calculated as s*sqrt(characteristic time/2), where s is the stochastic contribution size. PS: This option does not make any sense if the bottom layer is a Wiener process, as that process is not stationary.

`T.ground`
: This determines how the parallel tempering chains are to be defined. Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior),

is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`use.half.lives` If set to 'TRUE', reports half-lives rather than characteristic times. Half-lives are the time the auto-correlation of an OU process drops to 1/2, or the time it takes for a perturbation from the expected value to drop to 1/2 the original size in expected value. (As opposed to dropping to exp(-1) for characteristic times.) Half-life=log(2)*characteristic time. The auto-correlation for an OU process (and thus for the inner dynamics of any linear layer) is exp(-diff.time/characteristic.time)=exp(-log(2)*diff.time/half.life)= (1/2)^(diff.time/half.life), where diff.time is the time difference between two process states.

`mcmc` If set to 'TRUE', returns the MCMC samples as an 'mcmc' object within the return object. The name of the list object will also be 'mcmc'.

`causal` If multiple time series are given, one can specify causal connections from underlying processes belonging to such time series to the underlying processes belonging to some other time series. This specification should come in the form of an 4 x n matrix, where n is the number of causal connections. For each column (i.e. for each connection), row 1 specifies the number of the cause time series (same as the ordering of the input 'layer.series.structure' input to the routine), row 2 specifies the layer of the cause process, row 3 specifies the number of the effect time series and row 4 specifies the layer of the effect process. So if one has two time series, X and Y, both being modelled as having 3 layers, one can specify a causal connection from layer 3 in X to layer 1 in Y by the matrix "causal<-matrix(c(1,3,2,1),ncol=1)".

Note that it is possible to specify a causal connection even with only one time series, if a multi-layered process is considered. So if X is two-layered, "causal<-matrix(c(1,1,1,2),ncol=1)" will work. As layers are defined so that there is always a causal connection from layer 'k' to layer 'k-1' (with strength=1 for identifiability purposes), the reverse specification ("causal<-matrix(c(1,2,1,1),ncol=1)") will not make sense.

`causal.symmetric`

Specifies a set of causal connections going both ways with equal strength. For more on the usage, see option 'causal', but note that symmetric causal connection between one layer and the next does not make sense, as the connection strength from layer 'k' to layer 'k-1' is always set to "1" for identifiability purposes. Note that which series is first and which is second does not matter, so the matrix "causal.symmetric<-matrix(c(2,1,1,3),ncol=1)" would specify the same model as "causal.symmetric<-matrix(c(1,3,2,1),ncol=1)".

`corr` A 4 x n matrix which specifies the correlative connections between the processes, where n is the number of such connections. For each column (i.e. for each connection), row 1 specifies the number of the first time series (same as the ordering of the input 'layer.series.structure' input to the routine), row 2 specifies the layer of the first process, row 3 specifies the number of the second time

series and row 4 specifies the layer of the second process. So if one has two time series, X and Y, both being modelled as having 3 layers, one can specify a correlative connection between layer 3 in X and layer 1 in Y by the matrix "corr<-matrix(c(1,3,2,1),ncol=1)". Note that which series is first and which is second does not matter, so the matrix "corr<-matrix(c(2,1,1,3),ncol=1)" would specify the exact same model.

`smoothing.specs`

A list that specifies whether smoothing samples are to be returned and if so how. Smoothing samples are a posteriori samples of the process states at measurement times and possibly also other times. Smoothing samples can be turned on by setting smoothing.specs$do.smoothing=TRUE. If so, the smoothing.specs$smoothing.time.diff and smoothing.specs$num.smooth.per.mcmc must also be given. If smoothing.specs$smoothing.time.diff is set to "0", only the states at the measurement times will be inferred. If smoothing.specs$smoothing.time.diff is set to a positive real value, the state at the regularly spaced time points (smoothing.specs$smoothing.start, smoothing.specs$smoothing.start+smoothing.specs$smoothing.time.diff, smoothing.specs$smoothing.start+2*smoothing.specs$smoothing.time.diff, ... smoothing.specs$smoothing.end) will also be reported. Note that this means that smoothing.specs$smoothing.start and smoothing.specs$smoothing.end must also be given. Also, if smoothing samples are to be given, smoothing.specs$num.smooth.per.mcmc must be given. This is the number of smoothing samples fetched per MCMC sample. Lastly, smoothing.specs$do.return.smoothing.sample determines if the smoothing samples are to be returned or not. If smoothing.specs$do.smoothing=TRUE, it rarely makes sense to set smoothing.specs$do.return.smoothing.sample<-FALSE, so if this is not given, the algorithm sets smoothing.specs$do.return.smoothing.sample<-smoothing.specs$do.smoothing.

`realization.specs`

A list that specifies whether realization samples are to be returned and if so how. Realization samples are a posteriori samples of the process itself (including the correlations between time points) at measurement times and possibly also other times. Realization samples can be turned on by setting realization.specs$do.realizations=TRUE. If so, the realization.specs$realization.time.diff and realization.specs$num.realizations must also be given. If realization.specs$realization.time.diff is set to "0", only the states at the measurement times will be inferred. If realization.specs$realization.time.diff is set to a positive real value, the state at the regularly spaced time points (realization.specs$realization.start, realization.specs$realization.start+realization.specs$realization.time.diff, realization.specs$realization.start+2*r ... realization.specs$realization.end) will also be reported. Note that this means that realization.specs$realization.start and realization.specs$realization.end must also be given. Also, if realization samples are to be given, realization.specs$num.realizations must be given. This is the number of realization samples fetched in total. (This cannot be more than the number of MCMC samples. the reason it is not equal to that, is that when censoring is in use, not all MCMC samples will result in valid realizations.)

Lastly, realization.specs$strategy determines whether censoring of realizations is to take place. Censoring has been used when the process in question represented sedimentation, in order to avoid realizations such that the process would have erased one or more of the measurements. Thus if X is sedimentation depth

and X_measurement < X_later_time_point for any combination of measurement or later time point, a censoring would take place. realization.specs$strategy="N" means no censoring. realization.specs$strategy="A" means ascending censoring (censors realizations that anywhere ascends beyond the value of the previous measurement point and realization.specs$strategy="D" means descending censoring (censors realizations that anywhere descends beyond the value of the previous measurement point).

`return.residuals`

If set to 'TRUE', the standardized residuals for all observed series as a matrix called 'standardized.residuals' (one column for each measured series) and the returns the time points for the residuals ('residuals.time'). Also returns the prior expected values from the Kalman filter ('prior.expected.values'), which should be independent of the residuals if the model assumptions are correct. Note that the residuals are calculated for the Bayesian median parameters if Bayesian analysis is used and maximum likelihood estimated parameters if classic (ML) analysis is used. Also note that the residuals for each observed series is the measurement minus the expected value divided by the uncertainty from the Kalman *filter* not the Kalman smoother (which is used for process inference), as it is the deviations in the Kalman filter that are assumed to be independent from measurement to measurement.

`smooth.previous.run`

If set true, performs smoothing (process inference) on a previously performed MCMC run, defined in the subsecquent item, 'previous.run'. Does not perform an new MCMC analysis. Note that the new data can still be given, so that the inference is for the process parameters as revealed by the previously run MCMC sampling, but with data equal to the new input. Note that the 'smoothing.specs' option also has to be set, in order for this to make sense.

`previous.run` A 'layered' object representing a previous run analysis, where MCMC samples have been saved (using the 'mcmc=True' option). Only makes sense if the 'smooth.previous.run' has been set to TRUE, otherwise the object is irrelevant.

`layer.analyzer.mode`

Mode literally set the mode for the call to the C++ code. Options are "Bayes","ML-from-MCMC","ML-from-input","ML-from-hybrid", "Loglik-from-input","Smooth-from-input" and "Numpar". "Bayes" is Bayesian analysis, "ML-from-MCMC" runs ML optimizations starting from an MCMC run. "ML-from-input" instead starts the ML optimizations from the MCMC samples in a previous run (this also allows for explicitely stating the starting points, as one can set the "MCMC" samples from the previous run), while "ML-from-hybrid" starts alternating between MCMC samples fromt he current analysis and the previous run. "Loglik-from-input" is used by the layer.param.loglik(s) methods. "Smooth-from-input" is used by the layer.predict.mcmc methods. "Numpar" is used for extracting just the number of parameters (and the parameter names), which is useful if you need to know the number of parameters before running an anlysis (used in 'layer.analyzer' in case the user wants to start ML optimizations from a previous run without connections). Note that mode needs to match input. So for instance, "Bayes" and "ML-from-MCMC" have no need or want for a previous run, while "ML-from-input","ML-from-hybrid","Loglik-from-input" and "Smooth-from-input" do.

`loglik.laxness` There are sanity checks for each measurement point handled during the log-likelihood calculation, in order to avoid errors in the calculation due to numerical errors. However, some numerical inaccuracies will always exist, so some laxness in these checks must be allowed. Varianles that should be equal might not be exactly equal. And covariance matrices, which should be symmetric, might not be exactly symmetric or be so nearly singular that numerical errors destroys positive definiteness (only checked for levels "low" and "moderate"). Too much laxness and erroneous log-likelihoods may be used. Too much rigidity, i.e. too little laxness and parameter values may needlessly be excluded and the optimal estimation is not achieved. Default laxness is "high", but options are also "low" (very rigid), "moderate" and "complete" (sanity checks are not performed).

PS: The test covariance matrix test for positive definiteness (positive eigenvalues) do take quite a lot of computational resources. If you run with loglik.laxness="complete" or "high" (default), you might run things as much as 3 times faster! But unless you run some fairly simple models, be sure to check both parameter estimates and the stability of the log-likelihood. (Perhaps compare it to a simpler model). loglik.laxness="high" (default), doesn't cost much more than loglik.laxness="complete" (nothing is checked), but doesn't check for positive definiteness.

`external.series`

External series are time series that are treated as explanatory variables for the time series that are analyzed. They are not themselves analyzed statistically. Instead they are taken as is, with the assumption that what happenes between time points can be exppromately attained through linear interpolation. Since the stochastic nature of these series are not analyzed, only causal links are possible. Note that if external time series are given, one must also specify which series and layers they go to (see the next two entries.) Note that external data series are assumed global, and this connect equally to all sites (if that is a thing in your datasets).

`external.series.connection`

An vector of integers that should specify which external time series are causally connected to which observed series, numbered from 1 to the number of series. The length of this vector should match the number of external time series.

`external.layer.connection`

An vector of integers that should specify which external time series are causally connected to which layers of the observed series, numbered from 1 to the number of layers in the series specified in 'external.series.conneciton'. The length of this vector should match the number of external time series.

`site.distance.matrix`

A matrix that specifies the distance between each pair of sites. Only relevant when on series uses the 'distance.correlated.sigma' option to specify between-site correlation based on distance and a distance correlation falloff parameter. But in such a case, the site distance matrix is very necessary! (Keep in mind, sites are numbered from 0 to number of sites-1, so element i,j in the matrix should tell the distance between site i-1 and site j-1. Also keep in mind that the size of the matrix must match the number of sites in the dataset. Also, diagonal elements are the distance from a specific site to itself and so shoulød be 0. Also, the matrix must be square and symmetric, since the distance from site i-1 to site

j-1 is the distance from site j-1 to site i-1.)

Details

The function takes a set of time series along with their process structure (represented by layer.series.structure objects), together with specifications for causal and correlative links, thus specifying data and the model. Run conditions for the numerics can also be specified. Performs parameter inference, either Bayesian using MCMC or classic (ML) using optimization from MCMC samples. Bayesian model likelihoods are calculated using an importance sampling method, while ML optimization gives what is needed for information criterion based model comparison. Process inference can also be performed, if specified. The numerical work is performed by a compiled C++ routine, in order to minimize computation time.

Each data set+process structure can also contain a prior hyperparameter specification for the Bayesian analysis, see 'layer.prior'.

Analyses of different models can be compared using the 'compare.layered' routine.

Value

Returns an object of type 'layered', a list containing parameter estimates and uncertainties, plus anything of extra output specified by the user. Extra output includes log(BML) for bayesian analysis, AIC, AICc, BIC for classic analysis, MCMC samples, process smoothing samples and process realization samples.

An object of the class 'layered' will typically have the following components:

mu_<series name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the expected value parameter of each series. (PS: Not given for Wiener processes or layered processes having the Wiener process as the bottom layer).

dt_<series_name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the expected characteristic of each series. (Alternatively, the half-life will be given if use.half.life=TRUE).

sigma_<series_name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the stochastic contribution parameter of each series. (Alternatively the stationary standard deviation will be given, if use.stationary.stdev=TRUE).

model.log.lik: Bayesian marginal model likelihood (unless do.model.likelihood=FALSE or ML analysis).

obs_sd_<series_name>: A list containing "mean", "median", "lower02_5" (2.5% quantile) and "upper97_5" (97.5% quantile) for the observational standard deviation (if not individual standard deviations are given in the data). Thus, this parameter describe the uncertainty of the measurements, not the stochasticity of the process.

computing.time: Computing time in seconds.

parameter.names: Array of parameter names.

data.structure: The data strucucture(s) sent to the analysis (including data, prior, number of layes etc.)

causal: Causal connections specified.

corr: Correlative connections specified.

causal.symmetric: Symmetric causal connections specified.

mcmc: Only returned if "mcmc=TRUE" in the input. Contains MCMC samples for each parameter.

process.time.points, process.mean, process.lower95, process.upper95: Process time points, means, upper and lower credibility bands for each process. (PS: There can be multiple prcoesses per serie, since number of layers can be greater than one). Only returned if option "smoothing.specs=list(do.smoothing=TRUE,...)" is used.

smoothing.samples: Smoothing samples for each MCMC sample for each process. Only returned if option "smoothing.specs=list(do.smoothing=TRUE,...,do.return.smoothing.samples = TRUE)" is used.

standardized.residuals: Standardized residuals are measurements minus expected values from the Kalman filter, divided by the standard deviation also from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

prior.expected.values: Expected values from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

residuals.time: Time points for each residual. Only returned if "return.residuals=TRUE" is set.

realizations: Process realizations given measurements. Only returned if "realization.specs=list(do.realizations=TRUE,...)" is set.

realization.time.points: Time point for each realization. Only returned if "realization.specs=list(do.realizations=TRUE,...)" is set.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

Geyer CJ (1991), Comp Sci Stat Proc 23rd Symp Inter: 156. Am Stat Ass, New York

See Also

See also layer.series.structure, layer.data.series, layer.prior, summary.layered and compare.layered.

Examples

```
library(layeranalyzer)
# Simulated a sample from the Ornstein-Uhlenbeck (OU) process with mu=0,
# characteristic time=50 and sigma=0.2:
n=1000
x=rep(rnorm(1),n)
for(i in 2:n)
 x[i]=exp(-1/50)*x[i-1]+0.2*rnorm(1)

# Create measurement set as a subsample of the process, plus
# noise with standard deviation 0.1
t=sort(sample(1:n,300))
y=(x+rnorm(length(x),0,0.1))[t]
```

```
# Create the data series object from the time points and "measurements":
X=layer.data.series(time.points=t, value.points=y,name="X")

# Run with few resources (since this is a test example).
# PS: ML-based estimates (default is Bayesian)
res=layer.analyzer.timeseries.list(list(x1=layer.series.structure(X,numlayers=1)),
  num.MCMC=40,burnin=400,layer.analyzer.mode="ML-from-MCMC",
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20)

# Look at the results:
summary(res)
# PS: Uncertainty estimates comes from Bayesian analysis and will
# in this case not be very stable (since few resources).
```

`layer.connections` *Alternative connection specification, taking a structure list.*

Description

The function takes a set of of "layer.series.structure" objects as input together with an alternative connecton specification and translates that into the default connection specification.

Usage

```
layer.connections(... , conn)
```

Arguments

`...`	A set of 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least two such objects must be given.
`conn`	An array of integers between 0 and 4, one for each possible pairing of two processes (series+layer) belonging to different series (see 'layer.pairs'). For each pair, 0 means no connection between the two, 1 means correlative connection between the two, 2 means causal connection from the first process in the pair to the second, 3 means causal connection from the second process in the pair to the first and 4 means causal connection going both ways between the pair of processes.

Details

The function takes a set of time series along with their process structure (represented by layer.series.structure objects), and an array of integers specifying the nature of the connection between each possible pair of processes. It then outputs a specification of what the corresponding specification for the 'causal' and 'corr' input for layer.analyzer should be.

This function is used in 'layer.analzyer' but is also provided separately for a user that wants to see how this alternative connection specification works.

The alternative way of specifying a connection was introduced in order to make it easy to produce different connection models by simply traversing the possible connection between each pair of processes. If one use a script to traverse all possible connections models, reads a script and corrects the connection specification, one has a tool for outputing one script for each connection models. These scripts can then possibly be run in parallel on a computer with many cores or a computing cluster.

Value

Returns an object of type 'layer.connections', a list two specifications called 'causal' and 'corr'. In 'layer.analyzer', these specifications are then sent further into the analysis system. But the output can also be used for checking that the user understands the alternative connection specification.

Author(s)

Trond Reitan, trondr@uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.connections.list, layer.pairs, layer.pairs.list and layer.analyzer.

Examples

```
library(layeranalyzer)

# Creates two 2-layered structures based on the two hare/lynx datasets.
# outputs the connection structure specified by the
# alternative conenction specification array c(0,1,2,4)

hare.st=layer.series.structure(hare.norm,numlayers=2)
lynx.st=layer.series.structure(lynx.norm,numlayers=2)

layer.connections(hare.st, lynx.st, conn=c(0,1,2,4))
```

---

layer.connections.list

Alternative connection specification, taking a structure list.

Description

This is a version of "layer.connections" that takes a list of "layer.series.structure" objects as input together with an alternative connecton specification and translates that into the default connection specification.

Usage

```
layer.connections.list(data.structure , conn)
```

Arguments

data.structure	A list containing 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least two such objects must be given.
conn	An array of integers between 0 and 4, one for each possible pairing of two processes (series+layer) belonging to different series (see 'layer.pairs'). For each pair, 0 means no connection between the two, 1 means correlative connection between the two, 2 means causal connection from the first process in the pair to the second, 3 means causal connection from the second process in the pair to the first and 4 means causal connection going both ways between the pair of processes.

Details

The function takes a set of time series along with their process structure (represented by layer.series.structure objects), and an array of integers specifying the nature of the connection between each possible pair of processes. It then outputs a specification of what the corresponding specification for the 'causal' and 'corr' input for layer.analyzer should be.

This function is used in 'layer.analzyer' but is also provided separately for a user that wants to see how this alternative connection specification works.

The alternative way of specifying a connection was introduced in order to make it easy to produce different connection models by simply traversing the possible connection ebtween each pair of processes. If one use a script to traverse all possible connections models, reads a script and corrects the connection specification, one has a tool for outputing one script for each connection models. These scripts can then possibly be run in parallel on a computer with many cores or a computing cluster.

Value

Returns an object of type 'layer.connections', a list two specifications called 'causal' and 'corr'. In 'layer.analyzer', these specifications are then sent further into the analysis system. But the output can also be used for checking that the user understands the alternative connection specification.

Author(s)

Trond Reitan, trondr@uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.connections, layer.pairs, layer.pairs.list and layer.analyzer.

Examples

```
library(layeranalyzer)

# Creates two 2-layered structures based on the two hare/lynx datasets.
# outputs the connection structure specified by the
# alternative conenction specification array c(0,1,2,4)

hare.st=layer.series.structure(hare.norm,numlayers=2)
lynx.st=layer.series.structure(lynx.norm,numlayers=2)

layer.connections.list(list(hare.st, lynx.st), conn=c(0,1,2,4))
```

---

layer.data.series *Represents a measurement time series.*

Description

This class function represents a measurement time series. Can include multiple sites. Individual measurement error can be added.

Usage

```
layer.data.series(time.points, value.points, name, std.dev=NULL,
                  num.meas.per.value=NULL, site=NULL)
```

Arguments

time.points	The time points of the time series in chronological (ascending) order. Does not need
value.points	The corresponding measurements of the time series (or the average measurement if more than one goes into the value, if 'num.meas.per.value' is given and larger than 1). Must necessarily be the same size as 'time.points'.
name	The name of the time series. Will be used in assigning parameter names for parameters associated with this series.
std.dev	The individual measurement standard deviations of each measurement (if applicable). If this does not exist, the model will estimate an overall measurement standard deviation belonging to all measurements. If 'num.meas.per.value', these are the standard deviations of of the individual measurements going into each value. Must necessarily be the same size as 'time.points', if given.
num.meas.per.value	The number of each individual measurement that goes into each value in the time series. Must necessarily be the same size as 'time.points', if given.

`site` The site number of each measurement, if applicable. Should be labelled from 0 to <number of sites>-1. Must necessarily be the same size as 'time.points', if given.

Details

This class function is used as input to the analysis and represents a single time series. It should contain time points and values. The values are the measurements or measurement averages, if many individual at a given time point goes into each value. Standard deviation of the individual values can also be given and the number of individual measurements per value, if applicable. Multiple sites are also possible. PS: There are extra options for the process structure if sites are involved.

Value

Returns a time series data set as a 'layer.data.series' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also read.layer.data.series and layer.series.structure.

Examples

```
library(layeranalyzer)

# Create a sample from the Ornstein-Uhlenbeck (OU) process with mu=0,
# characteristic time=50 and sigma=0.2:
n=1000
x=rep(rnorm(1),n)
for(i in 2:n)
 x[i]=exp(-1/50)*x[i-1]+0.2*rnorm(1)

# Create measurement set as a subsample of the process, plus
# noise with standard deviation 0.1
t=sort(sample(1:n,300))
y=(x+rnorm(length(x),0,0.1))[t]

# Create the data series object from the time points and "measurements":
X=layer.data.series(time.points=t, value.points=y, name="X")
```

`layer.load.prior` *Reading prior specification for the layer.analyzer from a file.*

Description

The function reads a prior specification by the way of hyperparameters from a csv file with semi-colon-separated instances.

Usage

```
layer.load.prior(filename)
```

Arguments

`filename`	The name of the semi-colon-separated csv file that contains the prior specification.

Details

This csv file should have a header row and one value row. It should have the elements "is_log;mu1;mu2;dt1;dt2;s1;s2;lin1;lin2
where is_log is an indicator of whether the layer.analyzer should log-transform the data or not (0=no, 1=yes, 2=already log-transformed). mu1;mu2 specifies a 95% credibility interval for the expected value. Similarly, dt1;dt2, s1;s2, lin1;lin2, beta1;beta2, init1;init2 specifies a 95% credbility interval for the characteristic time, size of stochastic contributions, linear time trend, causal connections and initial value, respectively. Only "is_log;mu1;mu2;dt1;dt2;s1;s2" is mandatory for the simplest analysis. More options might make the rest necessary also, see the documentation for 'layer.analyzer'.

Value

Returns prior as a 'layer.prior' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.prior.

Examples

```
library(layeranalyzer)
pr=layer.load.prior("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_prior.txt")
```

`layer.pairs` *Alternative connection specification, taking a structure list.*

Description

This function takes a set of of "layer.series.structure" objects as input and then shows the possible pairings in the order that an alternative connection specification (using 'conn' in 'layer.analzyer') will refer to them.

Usage

```
layer.pairs(...)
```

Arguments

...	A set of 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least two such objects must be given.

Details

The function takes a set of time series along with their process structure (represented by layer.series.structure objects), and outputs an array describing the pairings and also a set of strings containing a verbal description.

Used by 'layer.analyzer' via the 'layer.connections' function, for assigning and referring to pairs of processes when using alternative connection specification (the 'conn' option in 'layer.analyzer').

This function is also provided separately for a user that wants to see what the different paris are and how they are ordered.

The alternative way of specifying a connection was introduced in order to make it easy to produce different connection models by simply traversing the possible connection ebtween each pair of processes. If one use a script to traverse all possible connections models, reads a script and corrects the connection specification, one has a tool for outputing one script for each connection models. These scripts can then possibly be run in parallel on a computer with many cores or a computing cluster.

Value

Returns an object of type 'layer.pairs', a list with two specifications called 'causal' and 'corr'. In 'layer.analyzer', these specifications are then sent further into the analysis system. But the output can also be used for checking that the user understands the alternative connection specification.

Author(s)

Trond Reitan, trondr@uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.connections, layer.connections.list, layer.pairs.list and layer.analyzer.

Examples

```
library(layeranalyzer)

# Creates two 2-layered structures based on the two hare/lynx datasets.
# outputs the pairs and a description

hare.st=layer.series.structure(hare.norm,numlayers=2)
lynx.st=layer.series.structure(lynx.norm,numlayers=2)

layer.pairs(hare.st, lynx.st)
```

---

layer.pairs.list *Alternative connection specification, taking a structure list.*

Description

This is a version of "layer.pairs" that takes a list of "layer.series.structure" objects as input together and then shows the possible pairings in the order that an alternative connection specification (using 'conn' in 'layer.analzyer') will refer to them.

Usage

```
layer.pairs.list(data.structure)
```

Arguments

`data.structure`	A list containing 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least two such object must be given.

Details

The function takes a set of time series along with their process structure (represented by layer.series.structure objects), and outputs an array describing the pairings and also a set of strings containing a verbal description.

Used by 'layer.analyzer' via the 'layer.connections' function, for assigning and referring to pairs of processes when using alternative connection specification (the 'conn' option in 'layer.analyzer').

This function is also provided separately for a user that wants to see what the different paris are and how they are ordered.

The alternative way of specifying a connection was introduced in order to make it easy to produce different connection models by simply traversing the possible connection ebtween each pair of processes. If one use a script to traverse all possible connections models, reads a script and corrects the connection specification, one has a tool for outputing one script for each connection models. These scripts can then possibly be run in parallel on a computer with many cores or a computing cluster.

Value

Returns an object of type 'layer.pairs', a list with two specifications called 'causal' and 'corr'. In 'layer.analyzer', these specifications are then sent further into the analysis system. But the output can also be used for checking that the user understands the alternative connection specification.

Author(s)

Trond Reitan, trondr@uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.connections, layer.connections.list, layer.pairs and layer.analyzer.

Examples

```
library(layeranalyzer)

# Creates two 2-layered structures based on the two hare/lynx datasets.
# outputs the pairs and a description

hare.st=layer.series.structure(hare.norm,numlayers=2)
lynx.st=layer.series.structure(lynx.norm,numlayers=2)

layer.pairs.list(list(hare.st, lynx.st))
```

---

layer.param.loglik *Returning log-likelihood for a set of parameter values.*

Description

Calculates the log-likelihood for a set of parameter values (or estimates if not given) for a given model specified by a previous analysis result. If silent.mode is set off, extra debug info is given. Meant primarily for debug and for exploration of the likelihood surface, post-analysis.

Usage

```
layer.param.loglik(analysis, new.param.values=NULL, silent.mode=TRUE, num.optim=100,
           do.preanalysis.mcmc=FALSE,
    num.MCMC=1000,spacing=10,burnin=10000,num.temp=1)
```

Arguments

analysis	The 'analysis' variable must be a 'layered' object. I.e. a 'layer.analyzer' analysis needs to be performed first. This variable is not an option. If it is NULL, the function will complain. it is from here the function gets the model specification (including conneciton elements in the case of multiple time series). Thus a small analysis should have been performed alread. Alternatively, one can try to fill in a 'layered' object by hand, but that takes effort.
new.param.values	Specifies the parameter values for the log-likelihood calculation. If this is not specified (NULL), then the parameter estimates from the analysis is used instead. For the order of the parameters, see for instance analysis$parameter.names. Note that the scale is the original scale, not the reparametrized scale used internally (where for instance stochastic contributions, characteristic times and observational noise level is log-trnasformed, and correlation is logit-transformed.) It is possible to set some of the parameter values to NA, in which case these will be optimized over.
silent.mode	If set to FALSE, shows debug info for each iteration in the Kalman filter.
num.optim	If some parameter values are set to NA, optimization is performed on those parameters. 'num.optim' specifies the number of optimizations performed on each such parameter (since optimization is performed by hill-climbing and can thus converge to different optima from different starting positions).
do.preanalysis.mcmc	Specifies that an MCMC analysis should take palce before doing the log-likleihood calculation(s). Only a point if some of the parameters are unspecified. If so, and do.preanalysis.mcmc is set to TRUE, then the first optimization will use the parameter values from closest MCMC samples closest to the specified parts of the parameter set.
num.MCMC	num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the pre-analysis (see do.preanalysis.mcmc) will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).
spacing	Specifies the number of MCMC iteration (see do.preanalysis.mcmc) between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.
burnin	Markov chain Monte Carlo sampling (see do.preanalysis.mcmc) takes some iterations to converge to the posterior parameter distribution.
num.temp	The number of tempering chains (default 1, meaning it is just the MCMC chain (see do.preanalysis.mcmc)itself that is beeing run). Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results.

Details

This function calculated log-likelihood for a specified (or under-specified) parameter set. If under-specified (having NA values) these are optimized over. Meant mostly as a debug tool (if silent.mode=FALSE) but can also be used for exploring the likelihood surface. Note that the specified parameter set is on the original scale, not the reparametrized version used internally.

Value

Returns an numeric value for the log-likelihood.

Author(s)

Trond Reitan, trond.reitan@geo.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer.

Examples

```
# Define time series (time points, values and in this case also
# sample standard deviations and sample size, used for calculating
# standard errors for each measurement).
# install.packages("../www_docs/R/layeranalyzer_0.2.1.tar.gz",type="source")
library(layeranalyzer)
X=layer.data.series(time.points=malta$Time.Year,
  value.points=malta$Mean..log.body.mass.,
  std.dev=sqrt(malta$Variance.calculated.from.the.data),
  num.meas.per.value=malta$Sample.size,name="log.body.size")

# Define process structure for analysis purposes:
ser=layer.series.structure(X,numlayers=1)

# Perform Bayesian analysis:
res=layer.analyzer(ser,num.MCMC=100,burnin=1000)

# Run with debug info:
layer.param.loglik(res, silent.mode=FALSE)
```

layer.param.logliks *Returning log-likelihoods for a set of parameter value sets.*

Description

Calculates the log-likelihoods for a set of parameter value sets for a given model specified by a previous analysis result. If silent.mode is set off, some extra debug info is given. Meant for exploration of the likelihood surface, post-analysis.

Usage

```
layer.param.logliks(analysis, new.param.value.sets, silent.mode=TRUE,
           num.optim=100, do.preanalysis.mcmc=FALSE,
    num.MCMC=1000,spacing=10,burnin=10000,num.temp=1)
```

Arguments

analysis	The 'analysis' variable must be a 'layered' object. I.e. a 'layer.analyzer' analysis needs to be performed first. This variable is not an option. If it is NULL, the function will complain. it is from here the function gets the model specification (including conneciton elements in the case of multiple time series). Thus a small analysis should have been performed alread. Alternatively, one can try to fill in a 'layered' object by hand, but that takes effort.
new.param.value.sets	Specifies the set of parameter value sets for the log-likelihood calculations. For the order of the parameters, see for instance analysis$parameter.names. Note that the scale is the original scale, not the reparametrized scale used internally (where for instance stochastic contributions, characteristic times and observational noise level is log-trnasformed, and correlation is logit-transformed.) It is possible to set some of the parameter values to NA, in which case these will be optimized over.
silent.mode	If set to FALSE, shows some debug info.
num.optim	If some parameter values are set to NA, optimization is performed on those parameters. 'num.optim' specifies the number of optimizations performed on each such parameter (since optimization is performed by hill-climbing and can thus converge to different optima from different starting positions).
do.preanalysis.mcmc	Specifies that an MCMC analysis should take palce before doing the log-likleihood calculation(s). Only a point if some of the parameters are unspecified. If so, and do.preanalysis.mcmc is set to TRUE, then the first optimization will use the parameter values from closest MCMC samples closest to the specified parts of the parameter set.
num.MCMC	num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the pre-analysis (see do.preanalysis.mcmc) will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).

spacing	Specifies the number of MCMC iteration (see do.preanalysis.mcmc) between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.
burnin	Markov chain Monte Carlo sampling (see do.preanalysis.mcmc) takes some iterations to converge to the posterior parameter distribution.
num.temp	The number of tempering chains (default 1, meaning it is just the MCMC chain (see do.preanalysis.mcmc)itself that is beeing run). Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results.

Details

This function calculated log-likelihoods for a specified (or under-specified) set of parameter sets. If under-specified (having NA values) these are optimized over. Meant mostly for exploring the likelihood surface. Note that the specified parameter set is on the original scale, not the reparametrized version used internally.

Value

Returns an set of numeric values for the log-likelihoods.

Author(s)

Trond Reitan, trond.reitan@geo.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer.

Examples

```
# Define time series (time points, values and in this case also
# sample standard deviations and sample size, used for calculating
# standard errors for each measurement).
library(layeranalyzer)
X=layer.data.series(time.points=malta$Time.Year,
  value.points=malta$Mean..log.body.mass.,
  std.dev=sqrt(malta$Variance.calculated.from.the.data),
  num.meas.per.value=malta$Sample.size,name="log.body.size")

# Define process structure for analysis purposes:
ser=layer.series.structure(X,numlayers=1)

# Perform Bayesian analysis, short run:
res=layer.analyzer(ser,num.MCMC=100,burnin=1000)
```

```
# Check one specific parameter set, then the same without
# mu (expected value), dt (characteristic time) and
# stat-sdev (stationary standaard deviation) specified. (Unspecified
# parameters are optimized over.)
sets=rbind(c(2.4, 300, 0.04), c(NA, 300, 0.04), c(2.4, NA, 0.04), c(2.4,300,NA))
lls=layer.param.logliks(res, sets, silent.mode=FALSE)
```

`layer.predict.estimate` *Prediction based on previously performed analysis using single point estimate for the parameter set*

Description

When a linear SDE analysis has been performed, predictions for the estimated parameter set can be made afterwards, using 'layer.predict.estimate'. The data sent does not need to be the same as for what the parameter estimates in 'layer.analyzer' was trained on. It could be a new dataset or the old with some new data added or a few data points removed. The predictions are the mean and upper/lower 95% credibility interval for the process state using the Kalman smoother. The individual covariance matrices for each time point are also returned.

Usage

```
layer.predict.estimate(... ,
   analysis,
   smoothing.time.diff=0,
   smoothing.start=NULL,smoothing.end=NULL,
  return.residuals=FALSE)
```

Arguments

`...`	A set of 'layer.data.series' objects that represents the set of time series that are to be predicted on. Note that these must match the ones used for running 'layer.analyzer' when it comes to number of time series and order. (See 'layer.data.series' for how they are defined).
`analysis`	The 'analysis' variable must be a 'layer.analyzer' object. I.e. a 'layer.analyzer' analysis needs to be performed first. This variable is not an option. If it is NULL, the function will complain.
`smoothing.time.diff`	If 'smoothing.time.diff' is set to "0", only the states at the measurement times will be inferred. (PS: you can get the function to perform process state inference for unmeasured time points by injecting them into the time series with' missing values). If 'smoothing.time.diff' is set to a positive real value, the state at the regularly spaced time points (smoothing.start, smoothing.start+smoothing.time.diff, smoothing.start+smoothing.time.diff, ... smoothing.end) will also be reported.

Note that this means that 'smoothing.start' and 'smoothing.end' must also be given.

`smoothing.start`
: Start time for extra smoothing points.

`smoothing.end`
: End time for extra smoothing points.

`return.residuals`
: The difference between predicted and actual measurements are returned. If set to 'TRUE', the standardized residuals for all observed series as a matrix called 'standardized.residuals' (one column for each measured series) and the returns the time points for the residuals ('residuals.time'). Also returns the prior expected values from the Kalman filter ('prior.expected.values'), which should be independent of the residuals if the model assumptions are correct. Note that the residuals are calculated for the Bayesian median parameters if Bayesian analysis is used and maximum likelihood estimated parameters if classic (ML) analysis is used. Also note that the residuals for each observed series is the measurement minus the expected value divided by the uncertainty from the Kalman *filter* not the Kalman smoother (which is used for process inference), as it is the deviations in the Kalman filter that are assumed to be independent from measurement to measurement.

Details

This function generates Kalman smoother-based process state samples for the estimated parameters in the previously performed analysis (returned in the 'est.origpar' variable. This is performed in the same way as when the 'smoothing.specs' option is actived in 'layer.analyzer' and the same underlying C++ routines is utilized (though there is a fork in the code for when MCMC samples are predefined rather than in need of sampling).

PS: This uses the 'layer.predict.mcmc.list' function using 'num.smooth.per.mcmc=1' and only one MCMC sample which is set ot the 'est.origpar. variable. This represents the parameter set in the original parametrization, i.e. the parametrization used internally.

Value

Returns an object of type 'layered', just as 'layer.analyzer' does. However, note that no new parameter sampling or estimation is performed. The variables of note are those that have to do with the process state inference, so only those will be described here:

process.time.points, process.mean, process.lower95, process.upper95: Process time points, means, upper and lower credibility bands for each process. (PS: There can be multiple prcoesses per serie, since number of layers can be greater than one).

P.k: A list of process state covariance matrices returned from the Kalman smoother, one for each time point (see 'process.time.points').

standardized.residuals: Standardized residuals are measurements minus expected values from the Kalman filter, divided by the standard deviation also from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

prior.expected.values: Expected values from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

residuals.time: Time points for each residual. Only returned if "return.residuals=TRUE" is set.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer, layer.predict.mcmc.list and layer.data.series.

Examples

```
library(layeranalyzer)

# Define time series (time points, values and in this case also
# sample standard deviations and sample size, used for calculating
# standard errors for each measurement).
X=layer.data.series(time.points=malta$Time.Year,
  value.points=malta$Mean..log.body.mass.,
  std.dev=sqrt(malta$Variance.calculated.from.the.data),
  num.meas.per.value=malta$Sample.size,name="log.body.size")

# Define process structure for analysis purposes:
ser=layer.series.structure(X,numlayers=1)

# Perform Bayesian analysis:
res=layer.analyzer(ser,mcmc=TRUE,num.MCMC=200,burnin=800)

# Now make up some new values much later:
t2=2100:2140
v2=rnorm(length(t2),2+0.1*sin(2*pi*t2/50),0.02)
X2=layer.data.series(time.points=t2,
   value.points=v2, std.dev=rep(0.02,length(t2)),
   num.meas.per.value=rep(1,length(t2)),
   name="log.body.size")

# Perform predictions (old data not conditioned on):
p3=layer.predict.estimate(X2, analysis=res,  smoothing.time.diff=0.25,
   smoothing.start=1900,smoothing.end=2300)

# Fetch the results
t3=p3$process.time.points
m3=p3$process.mean[1,]
l3=p3$process.lower95[1,]
u3=p3$process.upper95[1,]

# Plot the results
plot(t3,m3,type="l",ylim=c(min(l3),max(u3)))
points(t2,v2)
lines(t3,l3,col="red")
```

```
lines(t3,u3,col="red")


# Perform ML analysis:
# Do the same for ML estimate
res.ml=layer.analyzer(ser,do.maximum.likelihood=TRUE,
  maximum.likelihood.numstart=100,do.model.likelihood=FALSE)

# Now make up some new values much later:
t2=2100:2199
v2=rnorm(length(t2),2+0.1*sin(2*pi*t2/50),0.02)
X2=layer.data.series(time.points=t2,
   value.points=v2, std.dev=rep(0.02,length(t2)),
   num.meas.per.value=rep(1,length(t2)),
   name="log.body.size")

# Perform the prediction:
p4=layer.predict.estimate(X2, analysis=res.ml,  smoothing.time.diff=0.025,
   smoothing.start=1900,smoothing.end=2400)

# Fetch the results:
t4=p4$process.time.points
m4=p4$process.mean[1,]
l4=p4$process.lower95[1,]
u4=p4$process.upper95[1,]

# Plot the results:
plot(t4,m4,type="l",ylim=c(min(c(l4,m4)),max(c(m4,u4))))
 points(t2,v2)
 lines(t4,l4,col="red")
 lines(t4,u4,col="red")
```

`layer.predict.mcmc` *Prediction based on previously performed MCMC analysis*

Description

When a linear SDE analysis has been performed using 'layer.analyzer' with the MCMC samples saved (the option 'mcmc=TRUE'), predictions can be made afterwards, using 'layer.predict.mcmc'. The data sent does not need to be the same as for what the parameter estimates in 'layer.analyzer' was trained on. It could be a new dataset or the old with some new data added or a few data points removed. The predictiong runs through the MCMC samples and samples process state predictions using the Kalman smoother.

Usage

```
layer.predict.mcmc(... ,
   analysis,
```

```
smoothing.time.diff=0,
smoothing.start=NULL,smoothing.end=NULL,
num.smooth.per.mcmc=10, do.return.smoothing.samples=FALSE)
```

Arguments

`...`	A set of 'layer.data.series' objects that represents the set of time series that are to be predicted on. Note that these must match the ones used for running 'layer.analyzer' when it comes to number of time series and order. (See 'layer.data.series' for how they are defined).
`analysis`	The 'analysis' variable must be a 'layer.analyzer' object. I.e. a 'layer.analyzer' analysis needs to be performed first. This variable is not an option. If it is NULL, the function will complain.
`smoothing.time.diff`	If 'smoothing.time.diff' is set to "0", only the states at the measurement times will be inferred. (PS: you can get the function to perform process state inference for unmeasured time points by injecting them into the time series with' missing values). If 'smoothing.time.diff' is set to a positive real value, the state at the regularly spaced time points (smoothing.start, smoothing.start+smoothing.time.diff, smoothing.start+smoothing.time.diff, ... smoothing.end) will also be reported. Note that this means that 'smoothing.start' and 'smoothing.end' must also be given.
`smoothing.start`	Start time for extra smoothing points.
`smoothing.end`	End time for extra smoothing points.
`num.smooth.per.mcmc`	This is the number of smoothing process state samples fetched per MCMC sample.
`do.return.smoothing.samples`	'do.return.smoothing.sample' determines if the raw smoothing samples are to be returned or not, or only the mean and lower/upper 95% credibility band of the processes. These represents samples of the individual process states given the new data. Note that these are not process realizations, as the autocorrelation between one time point and the next is not taken into account when sampling. Process realizations may perhaps be added later.

Details

This function generates Kalman smoother-based process state samples for each MCMC sample in the previously performed analysis. This is performed in the same way as when the 'smoothing.specs' option is actived in 'layer.analyzer' and the same underlying c++ routines is utilized (though there is a fork in the code for when MCMC samples are predefined rather than in need of sampling).

PS: If 'num.smooth.per.mcmc=1' and there is only one MCMC sample, the routine will assume that a direct Kalman smoother result is instead required, based on the estimated parameter set.

Value

Returns an object of type 'layered', just as 'layer.analyzer' does. However, note that no new parameter sampling or estimation is performed. The variables of note are those that have to do with the process state inference, so only those will be described here:

process.time.points, process.mean, process.lower95, process.upper95: Process time points, means, upper and lower credibility bands for each process. (PS: There can be multiple prcoesses per serie, since number of layers can be greater than one).

smoothing.samples: Smoothing samples for each MCMC sample for each process. Only returned if option "do.return.smoothing.samples = TRUE)" is used.

standardized.residuals: Standardized residuals are measurements minus expected values from the Kalman filter, divided by the standard deviation also from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

prior.expected.values: Expected values from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

residuals.time: Time points for each residual. Only returned if "return.residuals=TRUE" is set.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer, layer.predict.mcmc.list and layer.data.series.

Examples

```
library(layeranalyzer)

# Define time series (time points, values and in this case also
# sample standard deviations and sample size, used for calculating
# standard errors for each measurement).
X<-layer.data.series(time.points=malta$Time.Year,
  value.points=malta$Mean..log.body.mass.,
  std.dev=sqrt(malta$Variance.calculated.from.the.data),
  num.meas.per.value=malta$Sample.size,name="log.body.size")

# Define process structure for analysis purposes:
ser<-layer.series.structure(X,numlayers=1)

# Perform analysis and make sure MCMC samples are returned:

res<-layer.analyzer(ser,mcmc=TRUE)

# Now make up some new values much later:
```

```
t2<-2100:2199
v2<-rnorm(100,2+0.1*sin(2*pi*t2/50),0.02)

X2<-layer.data.series(time.points=t2,
   value.points=v2, std.dev=rep(0.02,100), num.meas.per.value=rep(1,100),
   name="log.body.size")

# Perform predictions:
p2<-layer.predict.mcmc(X2, analysis=res,  smoothing.time.diff=0.25,
   smoothing.start=1500,smoothing.end=2800,
   num.smooth.per.mcmc=10, do.return.smoothing.samples=TRUE)

# Plot the results:
plot(p2$process.time.points,p2$process.mean[1,],type="l")
points(t2,v2)
```

`layer.predict.mcmc.list` *Prediction based on previously performed MCMC analysis*

Description

When a linear SDE analysis has been performed using 'layer.analyzer' with the MCMC samples saved (the option 'mcmc=TRUE') or based on estimates, predictions can be made afterwards, using 'layer.predict.mcmc.list'. The data sent does not need to be the same as for what the parameter estimates in 'layer.analyzer' was trained on. It could be a new dataset or the old with some new data added or a few data points removed. The predictiong runs through the MCMC samples or just a single estimate. This is a way for the functions 'layer.predict.mcmc' and 'layer.predict.estimate' to access the prediction module, and is made for those rather than direct use.

Usage

```
layer.predict.mcmc.list(new.data.list ,
   analysis=NULL,
   smoothing.time.diff=0,
   smoothing.start=NULL,smoothing.end=NULL,
   num.smooth.per.mcmc=10, do.return.smoothing.samples=FALSE,
  return.residuals=FALSE)
```

Arguments

`new.data.list`	A list item of 'layer.data.series' objects that represents the set of time series that are to be predicted on. Note that these must match the ones used for running 'layer.analyzer' when it comes to number of time series and order. (See 'layer.data.series' for how they are defined).

`analysis` The 'analysis' variable must be a 'layer.analyzer' object. I.e. a 'layer.analyzer' analysis needs to be performed first. This variable is not an option. If it is NULL, the function will complain.

`smoothing.time.diff`
If 'smoothing.time.diff' is set to "0", only the states at the measurement times will be inferred. (PS: you can get the function to perform process state inference for unmeasured time points by injecting them into the time series with' missing values). If 'smoothing.time.diff' is set to a positive real value, the state at the regularly spaced time points (smoothing.start, smoothing.start+smoothing.time.diff, smoothing.start+smoothing.time.diff, ... smoothing.end) will also be reported. Note that this means that 'smoothing.start' and 'smoothing.end' must also be given.

`smoothing.start`
Start time for extra smoothing points.

`smoothing.end` End time for extra smoothing points.

`num.smooth.per.mcmc`
This is the number of smoothing process state samples fetched per MCMC sample.

`do.return.smoothing.samples`
'do.return.smoothing.sample' determines if the raw smoothing samples are to be returned or not, or only the mean and lower/upper 95% credibility band of the processes. These represents samples of the individual process states given the new data. Note that these are not process realizations, as the autocorrelation between one time point and the next is not taken into account when sampling. Process realizations may perhaps be added later.

`return.residuals`
The difference between predicted and actual measurements are returned. If set to 'TRUE', the standardized residuals for all observed series as a matrix called 'standardized.residuals' (one column for each measured series) and the returns the time points for the residuals ('residuals.time'). Also returns the prior expected values from the Kalman filter ('prior.expected.values'), which should be independent of the residuals if the model assumptions are correct. Note that the residuals are calculated for the Bayesian median parameters if Bayesian analysis is used and maximum likelihood estimated parameters if classic (ML) analysis is used. Also note that the residuals for each observed series is the measurement minus the expected value divided by the uncertainty from the Kalman *filter* not the Kalman smoother (which is used for process inference), as it is the deviations in the Kalman filter that are assumed to be independent from measurement to measurement.

Details

This function generates Kalman smoother-based process state samples for each MCMC sample in the previously performed analysis. This is performed in the same way as when the 'smoothing.specs' option is actived in 'layer.analyzer' and the same underlying c++ routines is utilized (though there is a fork in the code for when MCMC samples are predefined rather than in need of sampling).

PS: If 'num.smooth.per.mcmc=1' and there is only one MCMC sample, the routine will assume that a direct Kalman smoother result is instead required, based on the estimated parameter set.

Value

Returns an object of type 'layered', just as 'layer.analyzer' does. However, note that no new parameter sampling or estimation is performed. The variables of note are those that have to do with the process state inference, so only those will be described here:

process.time.points, process.mean, process.lower95, process.upper95: Process time points, means, upper and lower credibility bands for each process. (PS: There can be multiple prcoesses per serie, since number of layers can be greater than one).

smoothing.samples: Smoothing samples for each MCMC sample for each process. Only returned if option "do.return.smoothing.samples = TRUE)" is used.

standardized.residuals: Standardized residuals are measurements minus expected values from the Kalman filter, divided by the standard deviation also from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

prior.expected.values: Expected values from the Kalman filter. Only returned if "return.residuals=TRUE" is set.

residuals.time: Time points for each residual. Only returned if "return.residuals=TRUE" is set.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer and layer.data.series.

Examples

```
library(layeranalyzer)

# Define time series (time points, values and in this case also
# sample standard deviations and sample size, used for calculating
# standard errors for each measurement).
X<-layer.data.series(time.points=malta$Time.Year,
  value.points=malta$Mean..log.body.mass.,
  std.dev=sqrt(malta$Variance.calculated.from.the.data),
  num.meas.per.value=malta$Sample.size,name="log.body.size")

# Define process structure for analysis purposes:
ser<-layer.series.structure(X,numlayers=1)

# Perform analysis and make sure MCMC samples are returned:
```

```
res<-layer.analyzer(ser,mcmc=TRUE)

# Now make up some new values much later:
t2<-2100:2199
v2<-rnorm(100,2+0.1*sin(2*pi*t2/50),0.02)

X2<-layer.data.series(time.points=t2,
   value.points=v2, std.dev=rep(0.02,100), num.meas.per.value=rep(1,100),
   name="log.body.size")

# Perform predictions:
p2<-layer.predict.mcmc.list(list(X2=X2), analysis=res,
   smoothing.time.diff=0.25,
   smoothing.start=1500,smoothing.end=2800,
   num.smooth.per.mcmc=10, do.return.smoothing.samples=TRUE)

# Plot the results:
plot(p2$process.time.points,p2$process.mean[1,],type="l")
points(t2,v2)
```

`layer.prior` *Specifying a prior distribution for the layer.analyzer.*

Description

This class function represents a Bayesian prior distribution specified by a couple of hyper-parameters.

Usage

```
layer.prior (mu, dt, sigma, stat.sdev = NULL, init = NULL, lin = NULL,
    beta = NULL, obs = NULL, dist.corr.falloff=NULL, islog = 0)
```

Arguments

`mu`	An array with 2 elements that specifies the low and high end of the 95% credibility band of mu, the expected value for the process.
`dt`	An array with 2 elements that specifies the low and high end of the 95% credibility band of dt, the characteristic time (or the half-life if that is specified).
`sigma`	An array with 2 elements that specifies the low and high end of the 95% credibility band of sigma, the stochastic contribution.
`stat.sdev`	An array with 2 elements that specifies the low and high end of the 95% credibility band of the stationary standard deviation. If not specified, the specifications for the prior of sigma is copied.
`init`	An array with 2 elements that specifies the low and high end of the 95% credibility band of init, the initical value of the process.

`lin` An array with 2 elements that specifies the low and high end of the 95% credibility band of lin, the linear trend (if applicable).

`beta` An array with 2 elements that specifies the low and high end of the 95% credibility band of beta, the strength of causal links (if applicable).

`obs` An array with 2 elements that specifies the low and high end of the 95% credibility band of obs, the standard deviation of the observational errors.

`dist.corr.falloff`
If distance-based spatial correlation on sites is wanted, there will be a parameter describing how the correlation decreases (exponentially) with distance, the distance correlation falloff parameter. The prior distribution for this is log-normal with 95 if not specified is set extremely wide, going from 1e-6 to 1e+6.

`islog` If set to 1, the analysis will log-transform the data before analysis. If set to 2, the analysis will assume the data has already been log-transformed.

Details

Specifies the Bayesian prior distribution by specifying the boundries of the 95% credibiluity band of each class of parameters. Only 'mu', 'dt' and 'sigma' are mandatory. The objects 'layer.standard.prior', 'layer.standard.log.prior', 'layer.wide.prior' are pre-specified layer.prior objects.

Value

Returns prior as a 'layer.prior' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.load.prior and layer.series.structure.

Examples

```
library(layeranalyzer)

pr=layer.prior(mu=c(-10,10),dt=c(0.001,1000),sigma=c(0.01,10),
                          lin=c(-1,1),beta=c(-1,1),init=c(-10,10),
                                 obs=c(0.01,1),islog=0)
```

`layer.series.structure` *Represents a measurement time series.*

Description

This class function represents a measurement time series plus the way the underlying process is modelled, not counting connections to other processes. Primarly, that means the number of layers used for modelling that process and whether the bottom layer is an Ornstein-Uhlenbeck (OU) process, a Wiener process or an OU with linear trend, but other options do also exists.

Usage

```
layer.series.structure(timeseries, numlayers=1, lin.time=FALSE,
                       time.integral=NULL, no.pull=FALSE, no.sigma=NULL,
                       regional.mu=FALSE, regional.lin.time=FALSE,
                       regional.pull=NULL, regional.sigma=NULL,
                       correlated.sigma=NULL, pairwise.correlated.sigma=NULL,
                       distance.correlated.sigma=NULL,
          one.dim.sigma=NULL, grouping.sigma=NULL,
          remove.sigma=NULL, differentiate.sigma=NULL,
          differentiate.pull=NULL, differentiate.mu=FALSE,
                       differentiate.lin.time=FALSE,
                       init.0=FALSE, init.time=NULL, init.same.sites=FALSE,
          init.same.layers=FALSE, init.specified=NULL,
          allow.pos.pull=FALSE, period=NULL,
                       prior=layer.standard.prior)
```

Arguments

`timeseries`	A 'layer.data.series' object representing a measurement time series.
`numlayers`	The number of layers in the model (hidden layers + measured layer). Setting numlayers=1 and nothing more means modellings the process behind the time series as an OU process. Note that increasing the number of layers can dramatically increase the computer resources needed (by as much as the cube of the number of layers), since matrix operations are needed in order to update knowledge of all layered processes.
`lin.time`	If set to 'TRUE', this indicates that a linear time trend is to be included. The bottom layer must be an OU process if so (i.e. 'no.pull' must be set to "False").
`time.integral`	An array of integers specifying which layers are the time integral of the layer below it. In differential equation terms, that means dX_n=X_(n+1) where 'n' is the layer number in question. This option was introduced in order to model sedimention height by letting it be the time integral of sedimentation rate and letting sedimentation rate be an OU process. Usually, only one time integral layer is needed, but the analysis allows for multiple such layers. Note that the bottom layer cannot be a time integral layer, as it is not tracking anything, so 'time.integral' must be less that 'numlayers'.

`no.pull` If set to 'TRUE', it specifies that the bottom layers has no pull term, i.e. that it is a Wiener process (random walk) rather than an OU process. When such models are included, it pays to use the initial value options, see the 'init' options in this routine. If not, a very wide generic prior for the initial value is used, giving the Wiener process model an unfair disadvantage.

`no.sigma` An array of integers indicating which layers are modelled to have no stochastic contributions. That means each layer specified will be a deterministic tracking of the layer below and will in practise be a smoothed version of the underlying process. because of this, it does not make sense setting 'no.sigma' to 'numlayers', as the bottom layers needs to have stoachasticity in order to have any dynamics at all.

`prior` Should be a 'layer.prior' object that specifies the prior distribution of the parameters, see the 'layer.prior' documentation. If not given, a standard prior is used. (Warning: This will not be appropriate for all circumstances).

`period` If given, should specify an array of period lengths, in order to include in the model periodic sine/cosine fluctuations (such as seasonality). Regression parameters for the sine/cosine terms are estimated.

`init.0` If set to 'TRUE', the model includes a parameter for the initial values of each process (one for each combination of site and layer). The initial time is that of the first measurement (unless 'init.time' is specified).

`init.time` If set, the model includes a parameter for the initial values of each process (one for each combination of site and layer). The initial time is that of 'init.time'.

`init.same.sites`
Only makes sense for time series with multiple sites. If set, an initial value treatment will assume the same initial value for all sites (if applicable).

`init.same.layers`
If set, an initial value treatment will assume the same initial value for all layers (if applicable, i.e. if 'numlayers'>1).

`init.specified` If given, should be an array of two elements, initial time and initial value.

`regional.mu` Only makes sense for time series with multiple sites. If set to 'TRUE', specifies that the process of each site should have a separate expected value (mu).

`regional.lin.time`
Only makes sense for time series with multiple sites. If set to 'TRUE', specifies that the process of each site should have a separate linear time trend.

`regional.pull` Only makes sense for time series with multiple sites. Should be an integer array specifying which layers have separate pulls (and thus characteristic times/halflives and stationary standard deviations) for each site.

`regional.sigma` Only makes sense for time series with multiple sites. Should be an integer array specifying which layers have separate stochastic noise size (and thus stationary standard deviations) for each site.

`correlated.sigma`
Only makes sense for time series with multiple sites. Should be an array of integers specifying which layers have correlated stochastic contributions between the sites. The same correlation is assumed between any two sites.

`pairwise.correlated.sigma`
: Only makes sense for time series with multiple sites. Should be an array of integers specifying which layers have correlated stochastic contributions between the sites. Correlation between a pair of sites depend on the distance between them, through a distance falloff parameter. A distance matrix needs to be supplied!

`distance.correlated.sigma`
: Only makes sense for time series with multiple sites. Should be an array of integers specifying which layers have correlated stochastic contributions between the sites. Different correlations are allowed between different pairs of sites.

`one.dim.sigma`
: Only makes sense for time series with multiple sites. Should be an array of integers specifying which layers have perfect correlated stochastic contributions (meaning correlation=1, i.e. the same stochastic contribution for all sites) between the sites. Can introduce numerical difficulties.

`grouping.sigma`
: Only makes sense for time series with multiple sites. Should be an array of integers specifying which layers have correlated stochastic contributions between the sites. However, the sites will be put into two groups, with no correlation between sites beloning to one group and the other. The grouping will be inferred. Not well tested.

`remove.sigma`
: Only makes sense for time series with multiple sites. Should be an array of integers specifying which layers have a grouping between sites with and without correlation to the stochastic contributions of other sites. Not well tested.

`differentiate.sigma`
: Only makes sense for time series with multiple sites. Attempts to find two groups of sites for the stochastic contribution sizes for the given set of layers, as specified by an array of integers.

`differentiate.pull`
: Only makes sense for time series with multiple sites. Attempts to find two groups of sites for the pull (characteristic times/halftimes) for the given set of layers, as specified by an array of integers.

`differentiate.mu`
: Only makes sense for time series with multiple sites. If set to 'True', attempts to find two groups of sites for the expected value (mu) of the processes.

`differentiate.lin.time`
: Only makes sense for time series with multiple sites. If set to 'True', attempts to find two groups of sites for the linear time trend.

`allow.pos.pull`
: If set to 'TRUE', allows for positive pull parameters in the linear stochastic equations. Positive pulls means the process is prone to "explode", but for a short time (typically not much more than the characteristic time), such a process could none the less make sense.

Details

This class function is used as input to the analysis and represents how a single time series (which can include several sites) is to be modelled. This includes the number of layers, the nature of the bottom layer and trends (linear or period in time). Stochasticity can be switched off for given layers. if multiple sites are included, there are options for controlling whether the parameters of the different

sites are different or the same. Also, correlation between the processes of different sites can be correlated with certain restrictions, or not correlated. The objects of this class is used as input in the 'layer.analyzer' routine.

Value

Returns the modelling specifications for a time series data set as a 'layer.series.structure' object. Note the field 'description' which contains a description of the object. Best outputted with 'cat(obj$description)

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer and layer.data.series.

Examples

```
library(layeranalyzer)

# Will look at the structure of a single simulated
# time series (simulated using a 2-layered model)

# Read a simulated dataset with 2 layers:
test.ex=read.table("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt")
names(test.ex)<-c("time","value")
ts<-layer.data.series(time=test.ex$time, value=test.ex$value,name="X")

# Look at 2 models, where the only difference is the structure
# specified in 'layer.series.structure':

# ML estimates for an OU model:
summary(layer.analyzer(layer.series.structure(ts,numlayers=1),
  num.MCMC=40,burnin=400,
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20))

# ML estimates for an 2-layered model model:
summary(layer.analyzer(layer.series.structure(ts,numlayers=2),
  num.MCMC=40,burnin=400,
  do.maximum.likelihood=TRUE,maximum.likelihood.numstart=20))

# PS: The models can then be compared, see 'compare.layered'.
```

`layer.standard.log.prior` *Specifying a suggestion for prior distribution, for data that will be log-transformed.*

Description

This 'layer.prior' object defines a Bayesian prior distribution suggested for use when the user wants readily defined prior for log-transformed data.

Usage

```
layer.standard.log.prior
```

Details

Specifies a "standard" Bayesian prior distribution suitable for log-transformed data. Warning: Will not be suitable in every context.

Value

A 'layer.prior' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.prior, layer.standard.prior and layer.series.structure.

Examples

```
library(layeranalyzer)

# Output the hyperparameters of the standard prior for
# log-transformed measurements
layer.standard.log.prior
```

`layer.standard.prior` *Specifying a default prior distribution for the layer.analyzer.*

Description

This 'layer.prior' object defines a Bayesian prior distribution used as the default prior when the user does not give a prior specification.

Usage

```
layer.standard.prior
```

Details

Specifies the standard Bayesian prior distribution used for default purposes. Warning: Will not be suitable in every context.

Value

A 'layer.prior' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.prior and layer.series.structure.

Examples

```
library(layeranalyzer)

# Output the hyperparameters of the standard (default) prior
layer.standard.prior
```

`layer.struct.name` *Generate a description of the process structure of a time series.*

Description

This function generates a string describing the structure specified for a time series.

Usage

```
layer.struct.name(struct)
```

Arguments

`struct`	A 'layer.series.structure' object that should be described.

Details

Describes as a string the number of layers and the properties of each layer, as specified in the input 'layer.series.structure' object. For series with sites (regional data), the regional settings for each layer is also described. Used when generating a 'layer.series.structure', which stores the result in a field named 'description'. This is again used when generating a description for an analysis in 'layer.analyzer'.

Value

Returns a descriptive string.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.series.structure.

Examples

```
library(layeranalyzer)
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt",
 name="X")

# Insert the data intro a structure specificatiion
# with the default structure (OU):
```

```
struct1=layer.series.structure(X)
layer.struct.name(struct1)
# Output should be "OU"

# Insert the data into a structure with 2 layers, where the
# lowest layer has no pull (i.e. Brownian motion, here also called
# "Random Walk" which is abbreviated to "RW"). Note that when the
# lowest layer is Brownian, the processes are nonstationary and
# need initial value treatment (init.0=TRUE).
# The top layer is specified to have no stochasticity
# (i.e. deterministic given the layer below it).
struct2=layer.series.structure(X,numlayers=2, no.pull=TRUE, no.sigma=1,
   init.0=TRUE)
layer.struct.name(struct2)
# Output should be "2-layered: Layer 2: RW , Layer 1: deterministic tracking"
```

`layer.wide.prior` *Specifying a suggestion for a wide (little information) prior distribution.*

Description

This 'layer.prior' object defines a Bayesian prior distribution suggested for use when the user wants a 'wide' prior, i.e. one specifying little prior knowledge.

Usage

```
layer.wide.prior
```

Details

Specifies a "standard" Bayesian prior distribution suitable for when very little prior knowledge exists. Suitable when the user knows very little about the nature of the data and the underlying processes.

Value

A 'layer.prior' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.prior, layer.standard.prior and layer.series.structure.

Examples

```
library(layeranalyzer)

# Output the hyperparameters of a prior distribution that is assumes
# very little (little information):
layer.wide.prior
```

`layeranalyzer` *Time Series Analysis Tool using Linear Layered SDEs*

Description

Time series analysis tool using linear layered stochastic differential equations. The package allows for multiple time series with correlative and/or causal links between them. Unmeasured causal processes are allowed to affect the measured processes in a layered structure, hence the name of the package. In case of causal feedback loops, the matrix operations (including eigenvalue decompositions) allows for complex numbers. In this case, cyclic behavior can be expected. Details can be found in Reitan and Liow (2019)<doi:10.1111/2041-210X.13299>.

Details

The DESCRIPTION file:

Index of help topics:

```
anova.layered           Analysis of Deviance for return object from
                        'layer.analyzer'.
bi.lext                 A dataset of reed warbler body mass over
                        several years.
br.lext                 A dataset of reed warbler body mass over
                        several years.
br.lorig                A dataset of reed warbler body mass over
                        several years.
compare.layered         Perform comparison of several models for the
                        same dataset(s).
hare.norm               A dataset of hare catchment data from the
                        Hudson Bay Company.
hare.orig               A dataset of hare catchment data from the
                        Hudson Bay Company.
layer.analyzer          Time series analysis tool using linear layered
                        SDEs.
layer.analyzer.timeseries.list
                        Time series analysis tool using linear layered
                        SDEs.
```

layer.connections	Alternative connection specification, taking a structure list.
layer.connections.list	Alternative connection specification, taking a structure list.
layer.data.series	Represents a measurement time series.
layer.load.prior	Reading prior specification for the layer.analyzer from a file.
layer.pairs	Alternative connection specification, taking a structure list.
layer.pairs.list	Alternative connection specification, taking a structure list.
layer.param.loglik	Returning log-likelihood for a set of parameter values.
layer.param.logliks	Returning log-likelihoods for a set of parameter value sets.
layer.predict.estimate	Prediction based on previously performed analysis using single point estimate for the parameter set
layer.predict.mcmc	Prediction based on previously performed MCMC analysis
layer.predict.mcmc.list	Prediction based on previously performed MCMC analysis
layer.prior	Specifying a prior distribution for the layer.analyzer.
layer.series.structure	Represents a measurement time series.
layer.standard.log.prior	Specifying a suggestion for prior distribution, for data that will be log-transformed.
layer.standard.prior	Specifying a default prior distribution for the layer.analyzer.
layer.struct.name	Generate a description of the process structure of a time series.
layer.wide.prior	Specifying a suggestion for a wide (little information) prior distribution.
layeranalyzer	Time Series Analysis Tool using Linear Layered SDEs
logLik.layered	Return maximum likelihood on the logarithmic scale for return object from 'layer.analyzer'.
lrate.pr	A "layer.prior" object used for diversifications rate analysis.
lynx.norm	A dataset of lynx catchment data from the Hudson Bay Company.
lynx.orig	A dataset of lynx catchment data from the Hudson Bay Company.

```
malta                       A dataset of reed warbler body mass over
                            several years.
n.combinations              Returns all ways of combining 'n' objects going
                            form 0 to <base>.
nobs.layered                Return number of observations for return object
                            from 'layer.analyzer'.
print.layer.series.structure
                            Outputs the description of a
                            'layer'series'structure' object.
print.summary.layered       Summarizes the return object from
                            'layer.analyzer'.
read.layer.data.series
                            Read a series representing a measurement time
                            series.
stepwise.connections.layered
                            Time series analysis tool using linear layered
                            SDEs.
summary.layered             Summarizes the return object from
                            'layer.analyzer'.
traverse.connections.layered
                            Time series analysis tool using linear layered
                            SDEs.
traverse.standalone.layered
                            Model traversal tool for standalone time
                            series.
```

Time series analysis tool using linear layered SDEs. Allows for multiple time series with correlative or causal links between them. In case of causal feedback loops, the matrix operations (including eigenvalue decompositions) allows for complex numbers. In this case, cyclic behavior can be expected.

Author(s)

Trond Reitan [aut, cre] (ORCID: <https://orcid.org/0000-0003-0793-8731>), Lee Hsiang Liow [ctb] (ORCID: <https://orcid.org/0000-0002-3732-6069>), Adam T. Kocsis [ctb] (ORCID: <https://orcid.org/0000-0002-9028-665X>)

Maintainer: Trond Reitan <trond.reitan@geo.uio.no>

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

Examples

```
library(layeranalyzer)

# Create a sample from the Ornstein-Uhlenbeck (OU) process with mu=0,
# characteristic time=50 and sigma=0.2:
```

```
n=1000
x=rep(rnorm(1),n)
for(i in 2:n)
 x[i]=exp(-1/50)*x[i-1]+0.2*rnorm(1)

# Create measurement set as a subsample of the process, plus
# noise with standard deviation 0.1
t=sort(sample(1:n,300))
y=(x+rnorm(length(x),0,0.1))[t]

# Create the data series object from the time points and "measurements":
X=layer.data.series(time.points=t, value.points=y,name="X")

# Specify layered structure. (In this case default structure, one layered=OU):
X.struct=layer.series.structure(X)

# Perform Bayesian analysis (low number of MCMCs):
res<-layer.analyzer(X.struct, num.MCMC=40, burnin=400)

# Show the results:
summary(res)
```

`logLik.layered` *Return maximum likelihood on the logarithmic scale for return object from 'layer.analyzer'.*

Description

Retrieve maximum log-likelihood for one or more return object from 'layer.analyzer'. (Requires classic analysis.)

Usage

```
## S3 method for class 'layered'
logLik(object, ...)
```

Arguments

`object, ...`	Objects of class 'layered', as returned from the 'layer.analyzer' function.

Details

Used for extracting the maximum log-likelihood in order to do a model comparison, using "anova.layered" (likelihood-ratio-testing).

Value

An object of class "logLik".

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer and anova.layered.

Examples

```
library(layeranalyzer)
pr=layer.load.prior("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_prior.txt")
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt",
 name="X")

# Calculate classic maximum likelihoods for an OU model, a 2 layered model
# and a 3 layered model. Look at log-likelihoods:
res1=layer.analyzer(layer.series.structure(X,numlayers=1,prior=pr),do.maximum.likelihood=TRUE)
res2=layer.analyzer(layer.series.structure(X,numlayers=2,prior=pr),do.maximum.likelihood=TRUE)
res3=layer.analyzer(layer.series.structure(X,numlayers=2,prior=pr),do.maximum.likelihood=TRUE)
c(logLik(res1), logLik(res2), logLik(res3))
```

---

lrate.pr *A "layer.prior" object used for diversifications rate analysis.*

Description

This 'layer.prior' object can be used for analysing diversification rates.

Usage

```
lrate.pr
```

Details

Specified in order to use for analysing diversification rates in Liow et al. (2015) and Reitan & Liow (2017).

Value

A 'layer.prior' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Liow, L.H., Reitan, T., Harnik, P.G. (2015) Ecological interactions on macroevolutionary time scales; clams and brachiopods are more than ships that pass in the night Ecology Letters, Volume 18(10): 1030-1039. DOI: 10.1111/ele.12485

Reitan, T., Liow, L. H. (2017) An unknown Phanerozoic driver of brachiopod extinction rates unveiled by multivariate linear stochastic differential equations. Paleobiology, Published online. DOI: 10.1017/pab.2017.11

See Also

See also layer.prior, layer.standard.prior and layer.series.structure.

Examples

```
library(layeranalyzer)

# Output the hyperparameters of a prior distribution we have used
# for diversification rate datasets:
lrate.pr
```

---

lynx.norm *A dataset of lynx catchment data from the Hudson Bay Company.*

Description

This example dataset was collected on Canadian lynx cathcment data from the Hudson Bay Company and used in Elton & Nicholson (1942). Log-transformed, normalized and made into a layer.data.series.

Details

The data is a list of class layer.data.series and which has four elements: 'time' (years), 'value' (normalized log-catchments), name ("lynx") and is.datatime (0, which means 'false').

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Elton, C. and M. Nicholson (1942). The ten-year cycle in numbers of lynx in Canada. J. Animal Ecology 11:215-244.

See Also

See also other datasets: malta, hare, bi.lext, br.lorig, br.lext.

lynx.orig *A dataset of lynx catchment data from the Hudson Bay Company.*

Description

This example dataset was collected on Canadian lynx cathcment data from the Hudson Bay Company and used in Elton & Nicholson (1942).

Details

The data has two columns containing year (time) and annual catchment value (value).

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Elton, C. and M. Nicholson (1942). The ten-year cycle in numbers of lynx in Canada. J. Animal Ecology 11:215-244.

See Also

See also other datasets: malta, hare.norm, lynx.norm, bi.lext, br.lorig, br.lext.

malta *A dataset of reed warbler body mass over several years.*

Description

This example dataset was collected on reed warblers in Malta in order to study short time span evolution and analyzed originally by Camilla Lo Cascio Sætre et al. (2017).

Details

The data has four columns with hopefully self-explanatory names: "Mean..log.body.mass." "Sample.size" "Time.Year" "Variance.calculated.from.the.data"

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

Source

https://datadryad.org/resource/doi:10.5061/dryad.hj30r

References

Sætre, C. L. C. Coleiro, C., Austad, M., Gauci, M., Sætre, G. and F. Eroukhmanoff (2017). Rappid adaptive phenotypic change following colonization of a newly restored habitat. Nature Communication, DOI: 10:1038/ncomms14159.

See Also

See also other datasets: hare.norm, lynx.norm, bi.lext, br.lorig, br.lext.

`n.combinations` *Returns all ways of combining 'n' objects going form 0 to <base>.*

Description

Gives all ways of combining 'n' objects going form 0 to <base>. Used for traversing all model combinations in "traverse.standalone.layered" and "traverse.connections.layered".

Usage

```
n.combinations(length, base.number=2)
```

Arguments

`length`	The number of objects to be combined.
`base.number`	The span of the possible object values, going from 0,1,...,base.number. Thus, base.number=2 (default) only allows "0" and "1".

Details

Used for going through all combination of options in "traverse.standalone.layered" and "traverse.connections.layered".

Value

A matrix where each row represents a unique sequence of {0,1,...,base.number}.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also traverse.standalone.layered and traverse.connections.layered.

Examples

```
library(layeranalyzer)

# List the set of combinations up to (but not including) 2
# of 3 elements (used by the model traversal algorithms):
n.combinations(3,2)
# This should be the output:
#    [,1] [,2] [,3]
#[1,]    0    0    0
#[2,]    0    0    1
#[3,]    0    1    0
#[4,]    0    1    1
#[5,]    1    0    0
#[6,]    1    0    1
#[7,]    1    1    0
#[8,]    1    1    1
```

`nobs.layered` *Return number of observations for return object from 'layer.analyzer'.*

Description

Retrieve number of observations for one or more return object from 'layer.analyzer'.

Usage

```
## S3 method for class 'layered'
nobs(object, ...)
```

Arguments

`object, ...`	Objects of class 'layered', as returned from the 'layer.analyzer' function.

Details

Used for extracting the number of observations in order to do a model comparison, using "anova.layered" (likelihood-ratio-testing).

Value

An integer representing the number of observations.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer, logLik.layered and anova.layered.

Examples

```
library(layeranalyzer)
# Read prior and data:
pr=layer.load.prior("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_prior.txt")
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt",
 name="X")

# Do a very short analysis, jsut to get a "layered" object we can run
# "nobs" on.
res1<-layer.analyzer(layer.series.structure(X,numlayers=1,prior=pr),
  num.MCMC=40,burnin=400,do.model.likelihood=FALSE)
nobs(res1)
# Output should be: 400.
```

---

print.layer.series.structure

Outputs the description of a 'layer'series'structure' object.

Description

This function (which should override 'print') outputs the description of the process structure of a series.

Usage

```
## S3 method for class 'layer.series.structure'
print(x,...)
```

Arguments

x	A 'layer.series.structure' object for which the description should be printed.
...	Not used, only there to keep complicance with print statements.

Details

Shows the assigned description of a 'layer.series.structure' object (typically generated by 'layer.struct.name').

Value

Prints a descriptive string.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.series.structure and layer.struct.name.

Examples

```
library(layeranalyzer)
# Read an example dataset:
X<-read.layer.data.series(
  "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt",
  "X")

# Make a structure specification that happens to coincide with the way
# the data was simulated, namely as 2 layers with OU as the lowest layer:
struct=layer.series.structure(X,numlayers=2)

# Show a description of the structure:
print.layer.series.structure(struct)
# Should output:
# Timeseries: X, structure:
# 2-layered: Layer 2: OU , Layer 1: OU-like tracking
```

---

print.summary.layered *Summarizes the return object from 'layer.analyzer'.*

Description

The function prints a summary, in the form of a 'summary.layered' object, of a model analysis, as performed by 'layer.analyzer'. This comes in the form of a table with parameter estimates and uncertainties, plus indicators of model fit.

Usage

```
## S3 method for class 'layered'
print.summary(x, ...)
```

Arguments

x	An 'summary.layered' object, as returned from the 'summary.layered' method.
...	Not used. Only included for compatibility with other other 'print.summary' methods.

Details

Prints a 'summary.layered' object, which in turn performs a summary of a 'layered' object, as returned from 'layer.analyzer'. For each parameter, mean- and median-based estimates and 95% credibility intervals are given, when Bayesian analysis is performed. In addition, the Bayesian model (marginal) likelihood is given. For classic maximum-likelihood (ML) analysis, the parameter estimates (ML) and various model selection criteria are given. Bayesian credibility intervals are still given in order to inform about parameter uncertainty.

Value

No return value. The method only prints to screen.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer and summary.layered.

Examples

```
library(layeranalyzer)

# Read prior and data:
pr<-layer.load.prior("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_prior.txt")
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_1layer.txt",
 "X")

# Calculate Bayesian model likelihood for an OU model.
# (Devotes a large amount of computer resources in the form of MCMC
# samples, in order to get a stable results).

res1<-layer.analyzer(layer.series.structure(X,numlayers=1,prior=pr),
    num.MCMC=1200,burnin=12000)

# Call for a print out of a summary:
summary(res1)
```

`read.layer.data.series` *Read a series representing a measurement time series.*

Description

This class function reads a file representing a measurement time series. Can include multiple sites. Individual measurement error can be added.

Usage

```
read.layer.data.series(filename,name,column.type=c("time","value"),
                header=FALSE,dec=".",sep="",
               quote="\"'", numerals = c("allow.loss", "warn.loss", "no.loss"),
                row.names=NULL, col.names=column.type,
                na.strings = "NA", colClasses = NA, nrows = -1,
                skip = 0, check.names = TRUE, fill = !blank.lines.skip,
                strip.white = FALSE, blank.lines.skip = TRUE,
                comment.char = "#",
                allowEscapes = FALSE, flush = FALSE,
                stringsAsFactors = FALSE, as.is = !stringsAsFactors,
                fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Arguments

`filename`	Name of the input file.
`name`	The name given to the series. Will be used in assigning parameter names for parameters associated with this series.
`column.type`	This should specify the content of each column in the file. The file must contain a "time" and a "value" column, as that is the bare minimum in order to specify a time series. In addition, the a measurement standard deviation column can be specified by "std.dev". If in addition a "num.meas.per.value" column exists, the standard error of each mueasurement will be taken to be the content of the "std.dev" column divied by the square root of the content of the "num.meas.per.value" column. A "site" column can also be added, if the time series contains the same typ of measurements from different sites (note that regional options then becomes a possibility). The site index must run from 0 to the number of sites minus one.
`header`	This and the rest of the options are just sent to "read.table", so the help text has just been lifted from the manual of that function. You should check that manual for an updated version of this help text. A logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: 'header'

is set to 'TRUE' if and only if the first row contains one fewer field than the number of columns.

`dec` the character used in the file for decimal points.

`sep` the field separator character. Values on each line of the file are separated by this character. If 'sep = ""' (the default for 'read.table') the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.

`quote` the set of quoting characters. To disable quoting altogether, use 'quote = ""'. See 'scan' for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless 'colClasses' is specified.

`numerals` string indicating how to convert numbers whose conversion to double precision would lose accuracy, see 'type.convert'. Can be abbreviated. (Applies also to complex-number inputs.)

`row.names` a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names.

If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if 'row.names' is missing, the rows are numbered.

Using 'row.names = NULL' forces row numbering. Missing or 'NULL' 'row.names' generate row names that are considered to be 'automatic' (and not preserved by 'as.matrix').

`col.names` a vector of optional names for the variables. The default is to use '"V"' followed by the column number.

`na.strings` a character vector of strings which are to be interpreted as 'NA' values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens _after_ white space is stripped from the input, so 'na.strings' values may need their own white space stripped in advance.

`colClasses` character. A vector of classes to be assumed for the columns. If unnamed, recycled as necessary. If named, names are matched with unspecified values being taken to be 'NA'.

Possible values are 'NA' (the default, when 'type.convert' is used), '"NULL"' (when the column is skipped), one of the atomic vector classes (logical, integer, numeric, complex, character, raw), or '"factor"', '"Date"' or '"POSIXct"'. Otherwise there needs to be an 'as' method (from package 'methods') for conversion from '"character"' to the specified formal class.

Note that 'colClasses' is specified per column (not per variable) and so includes the column of row names (if any).

`nrows` integer: the maximum number of rows to read in. Negative and other invalid values are ignored.

`skip` integer: the number of lines of the data file to skip before beginning to read data.

`check.names` logical. If 'TRUE' then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by 'make.names') so that they are, and also to ensure that there are no duplicates.

`fill` logical. If 'TRUE' then in case the rows have unequal length, blank fields are implicitly added. See 'Details'.

`strip.white` logical. Used only when 'sep' has been specified, and allows the stripping of leading and trailing white space from unquoted 'character' fields ('numeric' fields are always stripped). See 'scan' for further details (including the exact meaning of 'white space'), remembering that the columns may include the row names.

`blank.lines.skip` logical: if 'TRUE' blank lines in the input are ignored.

`comment.char` character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.

`allowEscapes` logical. Should C-style escapes such as "\n" be processed or read verbatim (the default)? Note that if not within quotes these could be interpreted as a delimiter (but not as a comment character). For more details see 'scan'.

`flush` logical: if "TRUE", "scan" will flush to the end of the line after reading the last of the fields requested. This allows putting comments after the last field.

`stringsAsFactors` logical: should character vectors be converted to factors? Note that this is overridden by 'as.is' and 'colClasses', both of which allow finer control.

`as.is` the default behavior of 'read.table' is to convert character variables (which are not converted to logical, numeric or complex) to factors. The variable 'as.is' controls the conversion of columns not otherwise specified by 'colClasses'. Its value is either a vector of logicals (values are recycled if necessary), or a vector of numeric or character indices which specify which columns should not be converted to factors.

Note: to suppress all conversions including those of numeric columns, set 'colClasses = "character"'.

Note that 'as.is' is specified per column (not per variable) and so includes the column of row names (if any) and any columns to be skipped.

`fileEncoding` character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the 'Encoding' section of the help for 'file', the 'R Data Import/Export Manual' and 'Note'.

`encoding` encoding to be assumed for input strings. It is used to mark character strings as known to be in Latin-1 or UTF-8 (see 'Encoding'): it is not used to re-encode the input, but allows R to handle encoded strings in their native encoding (if one of those two). See 'Value' and 'Note'.

`text` character string: if 'file' is not supplied and this is, then data are read from the value of 'text' via a text connection. Notice that a literal string can be used to include (small) data sets within R code.

`skipNul` logical: should nuls be skipped?

Details

This routine reads from a file representing a time series and creates a 'layer.data.series' object which can be used in the analysis. The file should at minimum contain two columns containing

the time points and the corresponding measurements. Which column is value and which is time should be specified in the input. (If not, it is assumed that time points comes first, then values). The file can also have additional columns descibing the standard deviation, the number of individual measurements per value and site indexes.

Value

Returns a time series data set as a 'layer.data.series' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.data.series and layer.series.structure.

Examples

```
library(layeranalyzer)

# Read a simulated 2 layer data set from an external web site, call it "Xseries":
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_2layer.txt",
 name="X")
```

---

stepwise.connections.layered

Time series analysis tool using linear layered SDEs.

Description

Traverse locally around a given connection model in a step-wise fashion, starting locally around the null model (no connections). Can perform larger model selections, but may not be robust enough still. Parallel runs using the standalone program 'layeranalyzer' should be preferred for larger model selection tasks. PS: No support for external series or spatial correlation so far (unlike traverse.connections.layered).

Usage

```
stepwise.connections.layered(..., num.MCMC = 1000, spacing = 10, burnin = 10000,
                  num.temp = 1, do.maximum.likelihood = FALSE,
                  maximum.likelihood.numstart = 10, silent.mode = TRUE,
                  talkative.burnin = FALSE, talkative.likelihood =
                  FALSE, talkative.traversal = TRUE, test.mode = FALSE,
                  id.strategy = 2, use.stationary.stdev = TRUE,
                  T.ground = 1.5, use.half.lives = FALSE, mcmc = FALSE,
                  allow.causal = TRUE, allow.correlation = TRUE,
                  allow.direct.feedback = TRUE, first.is.nullhypothesis
                  = FALSE, ML.IC = "AIC")
```

Arguments

`...`	A set of 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least one such object must be given. 'traverse.standalone.layered' can be used for settling the structure of each time series.
`num.MCMC`	num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the analysis will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).
`spacing`	Specifies the number of MCMC iteration between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.
`burnin`	Markov chain Monte Carlo sampling takes some iterations to converge to the posterior parameter distribution. The burn-in phase is the set of iterations performed where the samples are then discarded, so as to avoid including samples before convergence. Convergence can be check by for instance 'gelman.diag' in the 'coda' package, and the MCMC samples themselves can be gotten by using the option 'mcmc'. Still, stability of the results can be quite a good enough measure for most purposes. Default is 10000 iterations. PS: The burn-in algorithm here includes two steps where the random walk MCMC variance is adjusted in order to optimize the efficiency of the algorithm.
`num.temp`	The number of tempering chains (default 1, meaning it is just the MCMC chain itself that is beeing run). Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima. As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5.

If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`do.maximum.likelihood`
: Logical variable. If set to 'FALSE' (default), Bayesian analysis is performed. If set to 'TRUE', a classic maximum likelihood (ML) is performed, where the starting points of a set of optimizations is drawns from the MCMC samples. (This means that Bayesian MCMC sampling is performed anyway, but that the results from this is not focused on in the summary). The number of optimizations is determined by option 'maximum.likelihood.numstart', which is default set to 10. Estimation of the Bayesian model (marginal) likelihood is switched off when this option is used.

`maximum.likelihood.numstart`
: The number of ML optimizations performed, it applicable. (If 'do.maximum.likelihood' is set to 'TRUE'.) Default:10.

`silent.mode`
: If set to 'FALSE', the routine will show a lot debug information while running. This can be useful for tempering purposes, since the number of tempering swaps are shown in this debug information.

`talkative.burnin`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the burn-in phase.

`talkative.likelihood`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the likelihood calculation phase. (PS: This means a lot of printing.)

`talkative.traversal`
: If set to 'TRUE'(default), the connection pairs and the causal and correlative connections of each model.

`test.mode`
: If set to 'TRUE', the analyses themselves are not performed. Only the connection pairs and the causal and correlative connections of each model are shown. This can be useful if one wants to know how many models are are to be analyzed.

`id.strategy`
: Determines how to handle the identifiability problem in Bayesian multi-layer analysis. As mentioned in the supplementary of Reitan et al. (2012), one can switch the characteristic times of neighbouring layers and reorganize the stochastic contributions in such a way that the same top layer process takes place. This can be solved by requiring that the lower the layer, the larger the characteristic time. However, simply requiring this means the prior distribution of the characteristic times get shortened for multi-layered models, as compared to one-layered model. As long as the data is within the range of these shortened prior distributions, this puts the one-layer model at a disadvantage when it comes to model comparison. There are various solutions to this problem, and which solution is used is determined by the variable 'id.strategy'.

  The variable 'id.strategy' should be an integer between 0 and 4, default 2. 0 - No identification treatment. (Default) PS: Can and even should yield multimodal characteristic times 1 - Keep upper characteristic time prior. Add lognormally to beneath-lying characteristic times. 2 - Keep lower characteristic time prior. Substract lognormally to above-lying characteristic times. 3 - Keep lower characteristic time prior. Cut depending on that on the above-lying characteristic

times. 4 - Keep upper characteristic time prior. Cut depending on that on the below-lying characteristic times.

`use.stationary.stdev`
: If set to 'TRUE', instead of reporting the stochastic contribution size in the stochastic differential equations, the stationary standard deviation is reported. The stationary standard deviation is the standard deviation of the process state at any given time after convergence, unconditional on previous states. In a multi-layered model, this is to be interpreted as the stationary standard deviation if this layer when let alone (with no other process affecting it). This is calculated as s*sqrt(characteristic time/2), where s is the stochastic contribution size. PS: This option does not make any sense if the bottom layer is a Wiener process, as that process is not stationary.

`T.ground`
: This determines how the parallel tempering chains are to be defined. Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

  As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`use.half.lives`
: If set to 'TRUE', reports half-lives rather than characteristic times. Half-lives are the time the auto-correlation of an OU process drops to 1/2, or the time it takes for a perturbation from the expected value to drop to 1/2 the original size in expected value. (As opposed to dropping to exp(-1) for characteristic times.) Half-life=log(2)*characteristic time. The auto-correlation for an OU process (and thus for the inner dynamics of any linear layer) is exp(-diff.time/characteristic.time)=exp(-log(2)*diff.time/half.life)= (1/2)^(diff.time/half.life), where diff.time is the time difference between two process states.

`mcmc`
: If set to 'TRUE', returns the MCMC samples as an 'mcmc' object within the return object. The name of the list object will also be 'mcmc'.

`allow.causal`
: If set to 'FALSE', no causal connection models will be analyzed, only correlative ones.

`allow.correlation`
: If set to 'FALSE', no correlative conneciton models will be analyzed, only causal ones. (Incompatible with 'allow.causal=FALSE').

`allow.direct.feedback`
: If set to 'FALSE', no direct causal loops will be examined, i.e. causal links from series 1, layer 1 to series 2, layer 2 and another liunk back again. (Does not rule out more elaborate causal loops.)

`first.is.nullhypothesis`
: If prior probabilities (item 'p0') is not given, the prior probabilities are default

equal for all models. However, if first.is.nullhypothesis is set to 'TRUE', the first model is given 50% of the prior probability, while the rest of the models shares the remaining 50% equally. This variable is sent to 'compared.layered' for model comparison.

`ML.IC` If classic ML analysis is performed instead of Bayesian analysis, "AIC","BIC" or "AICc" will be used for model comparison instead of the Bayesian marginal likelihood. The item 'ML.IC' determines which of these three model comparison criteria are to be used. This variable is sent to 'compared.layered' for model comparison.

Details

Traverses all causal and correlative link combinations between a set of time series. This can mean a lot of models. With even just 3 one layered time series, that's 125 models, while 4 one layered time series means 15625 models!

Value

Returns an array of objects of type 'layered', each of which is a list containing parameter estimates and uncertainties, plus anything of extra output specified by the user.

This array can be inserted into 'compare.layered' or 'anova.layered' (if ML analysis).

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer, layer.analyzer.timeseries.list, compare.layered and anova.layered.

Examples

```
library(layeranalyzer)

# Read data:
X1<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_cause.txt",
 name="cause")
X2<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_effect.txt",
 name="effect")

# Set structure:
X1struct=layer.series.structure(X1, numlayers=1)
X2struct=layer.series.structure(X2, numlayers=1)
```

```
# Perform traversal of analyses. (PS: this will take some time!)
res=stepwise.connections.layered(X1struct,X2struct, num.MCMC=100, burnin=1000)

# PS: Since only the model of the last iteration is available,
# model comparison is not a priority and the method simply returns
# the best model according to the model selection criterion.

# Show the estiamtes of the best model:
summary(res)
# Highlight of summary:
# Causal links:
#   cause->effect
# As it should be.
```

`summary.layered` *Summarizes the return object from 'layer.analyzer'.*

Description

The function summarizes a model analysis, as performed by 'layer.analyzer'. This comes in the form of a table with parameter estimates and uncertainties, plus indicators of model fit.

Usage

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

Arguments

`object`	An 'layered' object, as returned from 'layer.analyzer'.
`...`	Not used. Only included for compatibility with other other 'summary' methods.

Details

Summary function for objects returned from 'layer.analyzer'. For each parameter, mean- and median-based estimates and 95% credibility intervals are given, when Bayesian analysis is performed. In addition, the Bayesian model (marginal) likelihood is given. For classic maximum-likelihood (ML) analysis, the parameter estimates (ML) and various model selection criteria are given. Bayesian credibility intervals are still given in order to inform about parameter uncertainty.

Value

Returns a list containing a matrix with parameter summaries and model fit measures (Bayesian model likelihood for Bayesian analysis and AIC/BIC/AICc for classic ML analysis), as a 'summary.layered' object.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer.

Examples

```
library(layeranalyzer)

# Read prior and dataset:
pr=layer.load.prior("https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_prior.txt")
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_1layer.txt",
 "X")

# Calculate Bayesian model likelihood for an OU model.
# (PS: The MCMC is restricted, to save time in the example).
res1<-layer.analyzer(layer.series.structure(X,numlayers=1,prior=pr),
  num.MCMC=60,burnin=600)
summary(res1)

# PS: There will be moderately-sized numeric variation in the estimates
# due to the randomness of the MCMC algorithm. A longer MCMC (or
# multiple of them) would yield more stable results.
```

---

traverse.connections.layered

Time series analysis tool using linear layered SDEs.

Description

Traverse all possible connections between time series. Return an array of analyses in the form of 'layered' objects, which can be used as input in the function 'compare.layered' (or 'anova.layered' of ML estimated analyses).

Usage

```
traverse.connections.layered(... ,
  num.MCMC=1000,spacing=10,burnin=10000,num.temp=1,
  do.maximum.likelihood=FALSE,maximum.likelihood.numstart=10,
  silent.mode=TRUE,talkative.burnin=FALSE,talkative.likelihood=FALSE,
```

```
  talkative.traversal=TRUE, test.mode=FALSE,
  id.strategy=2,use.stationary.stdev=TRUE,T.ground=1.5,
  use.half.lives=FALSE, mcmc=FALSE,
  allow.causal=TRUE, allow.correlation=TRUE, allow.direct.feedback=TRUE,
  external.series=NULL,
  external.series.connection=integer(0),
  external.layer.connection=integer(0),
  site.distance.matrix=NULL)
```

Arguments

...	A set of 'layer.series.structure' objects that represents the set of time series that are to be analyzed and how the underlying processes of each such series is to be modelles. (See 'layer.series.structure'). At least one such object must be given. 'traverse.standalone.layered' can be used for settling the structure of each time series.
num.MCMC	num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the analysis will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).
spacing	Specifies the number of MCMC iteration between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.
burnin	Markov chain Monte Carlo sampling takes some iterations to converge to the posterior parameter distribution. The burn-in phase is the set of iterations performed where the samples are then discarded, so as to avoid including samples before convergence. Convergence can be check by for instance 'gelman.diag' in the 'coda' package, and the MCMC samples themselves can be gotten by using the option 'mcmc'. Still, stability of the results can be quite a good enough measure for most purposes. Default is 10000 iterations. PS: The burn-in algorithm here includes two steps where the random walk MCMC variance is adjusted in order to optimize the efficiency of the algorithm.
num.temp	The number of tempering chains (default 1, meaning it is just the MCMC chain itself that is beeing run). Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima. As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`do.maximum.likelihood`
: Logical variable. If set to 'FALSE' (default), Bayesian analysis is performed. If set to 'TRUE', a classic maximum likelihood (ML) is performed, where the starting points of a set of optimizations is drawns from the MCMC samples. (This means that Bayesian MCMC sampling is performed anyway, but that the results from this is not focused on in the summary). The number of optimizations is determined by option 'maximum.likelihood.numstart', which is default set to 10. Estimation of the Bayesian model (marginal) likelihood is switched off when this option is used.

`maximum.likelihood.numstart`
: The number of ML optimizations performed, it applicable. (If 'do.maximum.likelihood' is set to 'TRUE'.) Default:10.

`silent.mode`
: If set to 'FALSE', the routine will show a lot debug information while running. This can be useful for tempering purposes, since the number of tempering swaps are shown in this debug information.

`talkative.burnin`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the burn-in phase.

`talkative.likelihood`
: If set to 'TRUE', this triggers the printing of more specific debug information concerning the likelihood calculation phase. (PS: This means a lot of printing.)

`talkative.traversal`
: If set to 'TRUE'(default), the connection pairs and the causal and correlative connections of each model.

`test.mode`
: If set to 'TRUE', the analyses themselves are not performed. Only the connection pairs and the causal and correlative connections of each model are shown. This can be useful if one wants to know how many models are are to be analyzed.

`id.strategy`
: Determines how to handle the identifiability problem in Bayesian multi-layer analysis. As mentioned in the supplementary of Reitan et al. (2012), one can switch the characteristic times of neighbouring layers and reorganize the stochastic contributions in such a way that the same top layer process takes place. This can be solved by requiring that the lower the layer, the larger the characteristic time. However, simply requiring this means the prior distribution of the characteristic times get shortened for multi-layered models, as compared to one-layered model. As long as the data is within the range of these shortened prior distributions, this puts the one-layer model at a disadvantage when it comes to model comparison. There are various solutions to this problem, and which solution is used is determined by the variable 'id.strategy'.

  The variable 'id.strategy' should be an integer between 0 and 4, default 2. 0 - No identification treatment. (Default) PS: Can and even should yield multimodal characteristic times 1 - Keep upper characteristic time prior. Add lognormally to beneath-lying characteristic times. 2 - Keep lower characteristic time prior. Substract lognormally to above-lying characteristic times. 3 - Keep lower characteristic time prior. Cut depending on that on the above-lying characteristic times. 4 - Keep upper characteristic time prior. Cut depending on that on the below-lying characteristic times.

`use.stationary.stdev`
: If set to 'TRUE', instead of reporting the stochastic contribution size in the stochastic differential equations, the stationary standard deviation is reported. The stationary standard deviation is the standard deviation of the process state at any given time after convergence, unconditional on previous states. In a multi-layered model, this is to be interpreted as the stationary standard deviation if this layer when let alone (with no other process affecting it). This is calculated as s*sqrt(characteristic time/2), where s is the stochastic contribution size. PS: This option does not make any sense if the bottom layer is a Wiener process, as that process is not stationary.

`T.ground`
: This determines how the parallel tempering chains are to be defined. Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

  As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`use.half.lives`
: If set to 'TRUE', reports half-lives rather than characteristic times. Half-lives are the time the auto-correlation of an OU process drops to 1/2, or the time it takes for a perturbation from the expected value to drop to 1/2 the original size in expected value. (As opposed to dropping to exp(-1) for characteristic times.) Half-life=log(2)*characteristic time. The auto-correlation for an OU process (and thus for the inner dynamics of any linear layer) is exp(-diff.time/characteristic.time)=exp(-log(2)*diff.time/half.life)= (1/2)^(diff.time/half.life), where diff.time is the time difference between two process states.

`mcmc`
: If set to 'TRUE', returns the MCMC samples as an 'mcmc' object within the return object. The name of the list object will also be 'mcmc'.

`allow.causal`
: If set to 'FALSE', no causal connection models will be analyzed, only correlative ones.

`allow.correlation`
: If set to 'FALSE', no correlative conneciton models will be analyzed, only causal ones. (Incompatible with 'allow.causal=FALSE'.

`allow.direct.feedback`
: If set to 'FALSE', no direct causal loops will be examined, i.e. causal links from series 1, layer 1 to series 2, layer 2 and another liunk back again. (Does not rule out more elaborate causal loops.)

`external.series`
: External series are time series that are treated as explanatory variables for the time series that are analyzed. They are not themselves analyzed statistically. Instead they are taken as is, with the assumption that what happenes between

time points can be exppromately attained through linear interpolation. Since the stochastic nature of these series are not analyzed, only causal links are possible. Note that if external time series are given, one must also specify which series and layers they go to (see the next two entries.) Note that external data series are assumed global, and this connect equally to all sites (if that is a thing in your datasets).

`external.series.connection`
: An vector of integers that should specify which external time series are causally connected to which observed series, numbered from 1 to the number of series. The length of this vector should match the number of external time series.

`external.layer.connection`
: An vector of integers that should specify which external time series are causally connected to which layers of the observed series, numbered from 1 to the number of layers in the series specified in 'external.series.conneciton'. The length of this vector should match the number of external time series.

`site.distance.matrix`
: A matrix that specifies the distance between each pair of sites. Only relevant when on series uses the 'distance.correlated.sigma' option to specify between-site correlation based on distance and a distance correlation falloff parameter. But in such a case, the site distance matrix is very necessary! (Keep in mind, sites are numbered from 0 to number of sites-1, so element i,j in the matrix should tell the distance between site i-1 and site j-1. Also keep in mind that the size of the matrix must match the number of sites in the dataset. Also, diagonal elements are the distance from a specific site to itself and so shoulød be 0. Also, the matrix must be square and symmetric, since the distance from site i-1 to site j-1 is the distance from site j-1 to site i-1.)

Details

Traverses all causal and correlative link combinations between a set of time series. This can mean a lot of models. With even just 3 one layered time series, that's 125 models, while 4 one layered time series means 15625 models!

Value

Returns an array of objects of type 'layered', each of which is a list containing parameter estimates and uncertainties, plus anything of extra output specified by the user.

This array can be inserted into 'compare.layered' or 'anova.layered' (if ML analysis).

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

See Also

See also layer.analyzer, layer.analyzer.timeseries.list, compare.layered and anova.layered.

Examples

```
library(layeranalyzer)

# Read data:
X1<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_cause.txt",
 name="cause")
X2<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_effect.txt",
 name="effect")

# Set structure:
X1struct<-layer.series.structure(X1, numlayers=1)
X2struct<-layer.series.structure(X2, numlayers=1)

# Perform traversal of analyses (but note that this will take some time):
res<-traverse.connections.layered(X1struct,X2struct, num.MCMC=100, burnin=1000)

# Compare analyses:
compare.layered(res)
```

---

traverse.standalone.layered

Model traversal tool for standalone time series.

Description

Traverses all models for a single standalone time series, up to a given model complexity, as given by the maximum number of layers.

Usage

```
traverse.standalone.layered(timeseries,
  max.layers=3,
  talkative=FALSE, allow.one.feedback.loop=FALSE,
  just.stationary=FALSE, no.rw=FALSE, regional.options=FALSE,
  time.integrals.possible=FALSE, allow.deterministic.layers=TRUE,
  num.MCMC=1000,spacing=10,burnin=10000,num.temp=1,
  do.maximum.likelihood=FALSE,maximum.likelihood.numstart=10,
  id.strategy=2,use.stationary.stdev=TRUE,T.ground=1.5,
  use.half.lives=FALSE, mcmc=FALSE,
  init.0=FALSE, init.time=NULL, init.same.sites=FALSE,
  init.same.layers=FALSE, init.specified=NULL,
  allow.pos.pull=FALSE, period=NULL,
```

```
  prior=layer.standard.prior,
  site.distance.matrix=NULL)
```

Arguments

timeseries	A single time series in the form of a 'layer.data.series' object. This is the input time series for the analysis.
max.layers	The maximal number of layers (hidden and observed) that is allowed in the model traversal. If any of the switches for how each layer is handled (such as 'regional.options=TRUE', 'time.integrals.possible=TRUE', 'allow.one.feedback.loop=TRUE' and 'allow.deterministic.layers=TRUE') are in use, the number of models to traverse will quickly increase with this maximal number of layers.
talkative	Gives a quick description of each model traversed and shows the progress.
allow.one.feedback.loop	If set to 'TRUE', allows for one feedback loop from a higher to a lower layer (so 'against' the normal flow of causality in these models). This can provide an explanation for quasi-cyclical behaviour.
just.stationary	If set to 'FALSE' allows for other models than OU (Ornstein-Uhlenbeck) at the lowest layer, namely also the Wiener process (also called Random Walk) and linear trend plus OU. If set to 'TRUE*, only OU is allowed at the bottom layer.
no.rw	Only in use when 'just.stationary=FALSE'. Switches off looking at the Wiener process (Random Walk) in the lowest layer if set to 'TRUE'.
regional.options	If set to 'TRUE' switches on traversal of the different regional options. This only makes sense if the number of sites is larger than one. Also, it can massively expland the number of models traversed. Not implemented yet!
time.integrals.possible	If set to 'TRUE' allows for time integral layers, i.e. the existence of one or more layers which are the time integral of the layer below. This of course only makes sense if the number of layers is alrger than one.
allow.deterministic.layers	If set to 'TRUE', allows for traversing models where there is no stochasitcity (i.e. determinism) on some layers. This can easily be the case as long as the number of layers is larger than one. (The lowest layer cannot be deterministic, as that just means a flat line.)
num.MCMC	num.MCMC is an integer specifying the number of Markov chain Monte Carlo (MCMC) samples that the analysis will rest on. Default 1000. More MCMC samples are run internally, depending on the spacing (default 10) and burnin (default 1000).
spacing	Specifies the number of MCMC iteration between each sample that is used (default 10). A higher number means the samples will be less correlated but requires proptionally more computer time.
burnin	Markov chain Monte Carlo sampling takes some iterations to converge to the posterior parameter distribution. The burn-in phase is the set of iterations performed where the samples are then discarded, so as to avoid including samples

before convergence. Convergence can be check by for instance 'gelman.diag' in the 'coda' package, and the MCMC samples themselves can be gotten by using the option 'mcmc'. Still, stability of the results can be quite a good enough measure for most purposes. Default is 10000 iterations. PS: The burn-in algorithm here includes two steps where the random walk MCMC variance is adjusted in order to optimize the efficiency of the algorithm.

`num.temp` The number of tempering chains (default 1, meaning it is just the MCMC chain itself that is beeing run). Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`do.maximum.likelihood`

Logical variable. If set to 'FALSE' (default), Bayesian analysis is performed. If set to 'TRUE', a classic maximum likelihood (ML) is performed, where the starting points of a set of optimizations is drawns from the MCMC samples. (This means that Bayesian MCMC sampling is performed anyway, but that the results from this is not focused on in the summary). The number of optimizations is determined by option 'maximum.likelihood.numstart', which is default set to 10. Estimation of the Bayesian model (marginal) likelihood is switched off when this option is used.

`maximum.likelihood.numstart`

The number of ML optimizations performed, it applicable. (If 'do.maximum.likelihood' is set to 'TRUE'.) Default:10.

`id.strategy` Determines how to handle the identifiability problem in Bayesian multi-layer analysis. As mentioned in the supplementary of Reitan et al. (2012), one can switch the characteristic times of neighbouring layers and reorganize the stochastic contributions in such a way that the same top layer process takes place. This can be solved by requiring that the lower the layer, the larger the characteristic time. However, simply requiring this means the prior distribution of the characteristic times get shortened for multi-layered models, as compared to one-layered model. As long as the data is within the range of these shortened prior distributions, this puts the one-layer model at a disadvantage when it comes to model comparison. There are various solutions to this problem, and which solution is used is determined by the variable 'id.strategy'.

The variable 'id.strategy' should be an integer between 0 and 4, default 2. 0 - No identification treatment. (Default) PS: Can and even should yield multimodal characteristic times 1 - Keep upper characteristic time prior. Add lognormally to beneath-lying characteristic times. 2 - Keep lower characteristic time prior.

Substract lognormally to above-lying characteristic times. 3 - Keep lower characteristic time prior. Cut depending on that on the above-lying characteristic times. 4 - Keep upper characteristic time prior. Cut depending on that on the below-lying characteristic times.

`use.stationary.stdev`

If set to 'TRUE', instead of reporting the stochastic contribution size in the stochastic differential equations, the stationary standard deviation is reported. The stationary standard deviation is the standard deviation of the process state at any given time after convergence, unconditional on previous states. In a multi-layered model, this is to be interpreted as the stationary standard deviation if this layer when let alone (with no other process affecting it). This is calculated as s*sqrt(characteristic time/2), where s is the stochastic contribution size. PS: This option does not make any sense if the bottom layer is a Wiener process, as that process is not stationary.

`T.ground` This determines how the parallel tempering chains are to be defined. Parallel tempering (Geyer 1991) is a method for overcoming local optima and can thus increase the stability of the results. The way it is done is that in addition to the MCMC chain for the posterior distribution, parallel chains sampling from the distribution proportional to exp(-log(likelihood*prior)/T) where T is a "temperature" larger than one (this is in practise a "smoothed" version of the posterior), is also sampled from. Swaps between neighbouring chains are then allowed, making it possible for the MCMC chain for the posterior distribution to find a new optima.

As the code is not parallel, the computer time will increase proportional to the number of tempering chains. The set of "temperatures" is set to (1, T.ground, T.ground^2, ..., T.ground^(numtemp-1)), where 'T.ground' is default set to 1.5. If no swapping is done (this can be seen if 'silent.mode' is set to 'FALSE'), this "ground temperature" must be lowered, in order to facilitate the swaps.

`use.half.lives` If set to 'TRUE', reports half-lives rather than characteristic times. Half-lives are the time the auto-correlation of an OU process drops to 1/2, or the time it takes for a perturbation from the expected value to drop to 1/2 the original size in expected value. (As opposed to dropping to exp(-1) for characteristic times.) Half-life=log(2)*characteristic time. The auto-correlation for an OU process (and thus for the inner dynamics of any linear layer) is exp(-diff.time/characteristic.time)=exp(-log(2)*diff.time/half.life)= (1/2)^(diff.time/half.life), where diff.time is the time difference between two process states.

`mcmc` If set to 'TRUE', returns the MCMC samples as an 'mcmc' object within the return object. The name of the list object will also be 'mcmc'.

`init.0` If set to 'TRUE', the model includes a parameter for the initial values of each process (one for each combination of site and layer). The initial time is that of the first measurement (if not 'init.time' is specified).

`init.time` If set, the model includes a parameter for the initial values of each process (one for each combination of site and layer). The initial time is that of 'init.time'.

`init.same.sites`

Only makes sense for time series with multiple sites. If set, an initial value treatment will assume the same initial value for all sites (if applicable).

`init.same.layers`
: If set, an initial value treatment will assume the same initial value for all layers (if applicable, i.e. if 'numlayers'>1).

`init.specified`
: If given, should be an array of two elements, initial time and initial value.

`allow.pos.pull`
: If set to 'TRUE', allows for positive pull parameters in the linear stochastic equations. Positive pulls means the process is prone to "explode", but for a short time (typically not much more than the characteristic time), such a process could none the less make sense.

`period`
: If given, should specify an array of period lengths, in order to include in the model periodic sine/cosine fluctuations (such as seasonality). Regression parameters for the sine/cosine terms are estimated.

`prior`
: An object specifying the prior distribution, in the form of a 'layer.prior' object.

`site.distance.matrix`
: A matrix that specifies the distance between each pair of sites. Only relevant when on series uses the 'distance.correlated.sigma' option to specify between-site correlation based on distance and a distance correlation falloff parameter. But in such a case, the site distance matrix is very necessary! (Keep in mind, sites are numbered from 0 to number of sites-1, so element i,j in the matrix should tell the distance between site i-1 and site j-1. Also keep in mind that the size of the matrix must match the number of sites in the dataset. Also, diagonal elements are the distance from a specific site to itself and so shoulød be 0. Also, the matrix must be square and symmetric, since the distance from site i-1 to site j-1 is the distance from site j-1 to site i-1.)

Details

This routine traverses all models for a time series up to a certain model complexity, as given by the maximal number of layers. All the traversed model are returned and this array of models can be sent directly to the 'compare.layered' routine. Certain options can be explored, such as stationarity, determinstic layers, time integral layers and the existance of a feedback loop. Using these options means more models are traversed, though.

Value

Returns a list object where each element will be of the type 'layered', representing a set of model analyses. If the turned object is called 'models', then each model analysis can be looked at through with 'models[[i]]' where 'i' is the model number. Using the 'talkative' option means a short description of each model will be shown during traversal. Or one can use 'summary(models[[i]])' to see both the structure alongside the parameter estimates.

Author(s)

Trond Reitan, trond.reitan@ibv.uio.no

References

Reitan, T., Schweder, T., Henderiks, J. (2012), Phenotypic Evolution studied by Layered Stochastic Differential Equations, Annals of Applied Statistics, Volume 6 (4): 1531-1551.

Geyer CJ (1991), Comp Sci Stat Proc 23rd Symp Inter: 156. Am Stat Ass, New York

See Also

See also layer.data.series, layer.prior, summary.layered and compare.layered.

Examples

```
library(layeranalyzer)

# Specify the prior distribution by giving 95% credibility bands
# for the parameter types:
p<-layer.prior(mu=c(-5,5),init=c(-100,100),dt=c(0.5,200),sigma=c(0.01,2),obs=c(0.01,1))

# Read the data, a simulated OU (1-layered) process:
X<-read.layer.data.series(
 "https://github.com/trondreitan/layeranalyzer/raw/master/doc/test_1layer.txt",
 "X")

# Traverse the possible standalone models up to 2 layers:

models<-traverse.standalone.layered(X, max.layers=2,
  talkative=TRUE, allow.one.feedback.loop=FALSE,
  just.stationary=FALSE, no.rw=FALSE,
  time.integrals.possible=FALSE,
  allow.deterministic.layers=TRUE,
  num.MCMC=200,spacing=10,burnin=2000, prior=p)

compare.layered(models)
cat(models[[1]]$description)
```

Index