Package 'ctsem'

October 30, 2023

Type Package

Title Continuous Time Structural Equation Modelling

Version 3.9.1

Date 2023-10-30

Description Hierarchical continuous (and discrete) time state space modelling, for linear and nonlinear systems measured by continuous variables, with limited support for binary data. The subject specific dynamic system is modelled as a stochastic differential equation (SDE) or difference equation, measurement models are typically multivariate normal factor models.
Linear mixed effects SDE's estimated via maximum likelihood and optimization are the default. Nonlinearities, (state dependent parameters) and random effects on all parameters are possible, using either max likelihood / max a posteriori optimization (with optional importance sampling) or Stan's Hamiltonian Monte Carlo sampling.
See <https://github.com/cdriveraus/ctsem/raw/master/vignettes/hierarchicalmanual.pdf> for details. Priors may be used. For the conceptual overview of the hierarchical Bayesian linear SDE approach, see <https://www.researchgate.net/publication/324093594_Hierarchical_Bayesian_Continuous_Time_Dynamic_Modeling>.
Exogenous inputs may also be included, for an overview of such possibilities see <https://www.researchgate.net/publication/328221807_Understanding_the_Time_Course_of_Interventions_with_Continuous_Time_Dynamic_Models> .
Stan based functions are not available on 32 bit Windows systems at present.
<https://cdriver.netlify.app/> contains some tutorial blog posts.

License GPL-3

Depends R (>= 4.2.0), Rcpp (>= 0.12.16)

URL https://github.com/cdriveraus/ctsem

Imports cOde, data.table (>= 1.12.8), datasets, Deriv, expm, ggplot2, graphics, grDevices, MASS, Matrix, methods, mize, mvtnorm, parallel, plyr, RcppParallel (>= 5.0.1), rstan (>= 2.26.0), rstantools (>= 2.3.0), stats, tibble, tools, utils

Encoding UTF-8

LazyData true

ByteCompile true

LinkingTo BH (>= 1.66.0-1), Rcpp (>= 0.12.16), RcppEigen (>= 0.3.3.4.0), RcppParallel (>= 5.0.1), rstan (>= 2.26), StanHeaders (>= 2.26.0), RcppParallel (>= 5.0.1)

Suggests knitr, testthat, devtools, DEoptim, tinytex, lme4, shiny, gridExtra, arules

VignetteBuilder knitr

RoxygenNote 7.2.3

SystemRequirements GNU make

NeedsCompilation yes

Biarch true

Author Charles Driver [aut, cre, cph], Manuel Voelkle [aut, cph], Han Oud [aut, cph], Trustees of Columbia University [cph]

Maintainer Charles Driver <charles.driver@ife.uzh.ch>

Repository CRAN

Date/Publication 2023-10-30 14:20:02 UTC

R topics documented:

`ctsem-package` *ctsem*

Description

ctsem is an R package for continuous time structural equation modelling of panel (N > 1) and time series (N = 1) data, using either a frequentist or Bayesian approach, or middle ground forms like maximum a posteriori.

The general workflow begins by specifying a model using the `ctModel` function, in which the `type` of model is also specified. Then the model is fit to data using `ctStanFit`. The ctFit function which allows for fitting using the OpenMx / SEM form, as described in the original JSS ctsem paper, can now be found in the ctsemOMX package. The omx forms are no longer in development and for most purposes, the newer stan based forms are more robust and flexible. For examples, see `ctStanFit`. For citation info, please run `citation('ctsem')` .

Author(s)

Maintainer: Charles Driver <charles.driver@ife.uzh.ch> [copyright holder]

Authors:

- Manuel Voelkle [copyright holder]
- Han Oud [copyright holder]

Other contributors:

- Trustees of Columbia University [copyright holder]

References

https://www.jstatsoft.org/article/view/v077i05

Driver, C. C., & Voelkle, M. C. (2018). Hierarchical Bayesian continuous time dynamic modeling. Psychological Methods. Advance online publication.http://dx.doi.org/10.1037/met0000168

Stan Development Team (2018). RStan: the R interface to Stan. R package version 2.17.3. http://mc-stan.org

#' @keywords internal

See Also

Useful links:

- https://github.com/cdriveraus/ctsem

AnomAuth *AnomAuth*

Description

A dataset containing panel data assessments of individuals Anomia and Authoritarianism.

Format

data frame with 2722 rows, 14 columns. Column Y1 represents anomia, Y2 Authoritarianism, dTx the time interval for measurement occasion x.

Source

See https://psycnet.apa.org/record/2012-09124-001 for details.

ctACF *Continuous Time Autocorrelation Function (ctACF)*

Description

This function computes an approximate continuous time autocorrelation function (ACF) for data containing multiple subjects and/or variables.

Usage

```
ctACF(
  dat,
  varnames = "auto",
  ccfnames = "all",
  idcol = "id",
  timecol = "time",
  plot = TRUE,
  timestep = "auto",
  time.max = "auto",
  nboot = 100,
  ...
)
```

Arguments

dat	The input data in data frame or data table format.
varnames	Character vector of variable names in the data to compute the ACF for. 'auto' uses all columns that are not time / id.

ccfnames	Character vector of variable names in the data to compute cross correlation for. 'all' uses all variables in varnames.
idcol	The name of the column containing subject IDs (default is 'id').
timecol	The name of the column containing time values (default is 'time').
plot	A logical value indicating whether to create a plot (default is TRUE).
timestep	The time step for discretizing data. 'auto' to automatically determine the timestep based on data distribution (default is 'auto'). In this case the timestep is computed as 1/10th of the 10th percentile for time intervals in the data.
time.max	The maximum time lag to compute the ACF (default is 10). If 'auto', is set to 10 times the 90th percentile interval in the data.
nboot	The number of bootstrap samples for confidence interval estimation (default is 100).
...	additional arguments (such as demean=FALSE) to pass to the stats::acf function.

Details

This function computes the continuous time ACF by discretizing the data and then performing bootstrapped ACF calculations to estimate the confidence intervals. It can create ACF plots with confidence intervals if 'plot' is set to TRUE.

Value

If 'plot' is TRUE, the function returns a ggplot object of the ACF plot. If 'plot' is FALSE, it returns a data table with ACF estimates and confidence intervals.

See Also

ctDiscretiseData

Examples

```
data.table::setDTthreads(1) #ignore this line
# Example usage:
head(ctstantestdat)
ctACF(ctstantestdat,varnames=c('Y1'),idcol='id',timecol='time',nboot=5)
```

---

ctACFresiduals *Calculate Continuous Time Autocorrelation Function (ACF) for Standardized Residuals of ctsem fit.*

Description

This function takes a fit object from ctsem and computes the continuous time autocorrelation function (ACF) on the standardized residuals.

Usage

```
ctACFresiduals(fit, ...)
```

Arguments

fit	A fitted model object generated by the ctsem package.
...	Additional arguments to be passed to the ctACF function.

Details

This function first extracts the standardized residuals from the fit object using the ctStanKalman function. Then, it calculates the continuous time ACF for these residuals and returns the results as a data table.

Value

A data table containing the continuous time ACF estimates for standardized residuals.

See Also

ctStanKalman

Examples

```
data.table::setDTthreads(1) #ignore this line
# Example usage:
ctACFresiduals(ctstantestfit, varnames='Y1',nboot=5)
```

---

ctAddSamples *Sample more values from an optimized ctstanfit object*

Description

Sample more values from an optimized ctstanfit object

Usage

```
ctAddSamples(fit, nsamples, cores = 2)
```

Arguments

fit	fit object
nsamples	number of samples desired
cores	number of cores to use

Value

fit object with extra samples

Examples

```
## Not run:
newfit <- ctAddSamples(ctstantestfit, 10, 1)

## End(Not run)
```

---

ctCheckFit *Visual model fit diagnostics for ctsem fit objects.*

Description

Visual model fit diagnostics for ctsem fit objects.

Usage

```
ctCheckFit(
  fit,
  data = TRUE,
  postpred = TRUE,
  priorpred = FALSE,
  statepred = FALSE,
  residuals = FALSE,
  by = fit$ctstanmodelbase$timeName,
  TIpredNames = fit$ctstanmodelbase$TIpredNames,
  nsamples = 30,
  covplot = FALSE,
  corr = TRUE,
  combinevars = NA,
  fastcov = FALSE,
  lagcovplot = FALSE,
  aggfunc = mean,
  aggregate = FALSE,
  groupbysplit = FALSE,
  byNA = TRUE,
  lag = 0,
  smooth = TRUE,
  k = 4,
  breaks = 4,
  entropy = FALSE,
  reg = FALSE,
  verbose = 0,
  indlines = 30
)
```

Arguments

fit	ctStanFit object.
data	Include empirical data in plots?
postpred	Include post predictive (conditional on estimated parameters and covariates) distribution data in plots?
priorpred	Include prior predictive (conditional on priors) distribution data in plots?
statepred	Include one step ahead (conditional on estimated parameters, covariates, and earlier data points) distribution data in plots?
residuals	Include one step ahead error (conditional on estimated parameters, covariates, and earlier data points) in plots?
by	Variable name to split or plot by. 'time', 'LogLik', and 'WhichObs' are also possibilities.
TIpredNames	Since time independent predictors do not change with time, by default observations after the first are ignored. For observing attrition it can be helpful to set this to NULL, or when the combinevars argument is used, specifying different names may be useful.
nsamples	Number of samples (when applicable) to include in plots.
covplot	Splits variables in the model by the 'by' argument, according to the number of breaks (breaks argument), and shows the covariance (or correlation) for the different data sources selected, as well as the differences between each pair.
corr	Turns the covplot into a correlation plot. Usually easier to make sense of visually.
combinevars	Can be a list of (possibly new) variable names, where each named element of the list contains a character vector of one or more variable names in the fit object, to combine into the one variable. By default, the mean is used, but see the aggfunc argument. The combinevars argument can also be used to ensure that only certain variables are plotted.
fastcov	Uses base R cov function for computing covariances. Not recommended with missing data.
lagcovplot	Logical. Output lagged covariance type plots?
aggfunc	Function to use for aggregation, if needed.
aggregate	If TRUE, duplicate observation types are aggregated over using aggfunc. For example, if by = 'time' and there are 8 time points per subject, but breaks = 2, there will be 4 duplicate observation types per 'row' that will be collapsed. In most cases it is helpful to not collapse.
groupbysplit	Logical. Affects variable ordering in covariance plots. Defaults to FALSE, grouping by variable, and within variable by split.
byNA	Logical. Create an extra break for when the split variable is missing?
lag	Integer vector. lag = 1 creates additional variables for plotting, prefixed by 'lag1_', containing the prior row of observations for that subject.
smooth	For bivariate plots, use a smoother for estimation?
k	Integer denoting number of knots to use in the smoothing spline.

breaks	Integer denoting number of discrete breaks to split variables by (when covariance plotting).
entropy	Still in development.
reg	Logical. Use regularisation when estimating covariance matrices? Can be necessary / faster for some problems.
verbose	Logical. If TRUE, shows optimization output when estimating covariances.
indlines	Integer number of individual subject lines to draw per data type.

Value

Nothing. Just plots.

Examples

```
ctCheckFit(ctstantestfit)
```

---

ctChisqTest *Chi Square test wrapper for ctStanFit objects.*

Description

Chi Square test wrapper for ctStanFit objects.

Usage

```
ctChisqTest(fit1, fit2)
```

Arguments

fit1	One of the fits to be compared (better fit is assumed as base for comparison)
fit2	Second fit to be compared

Value

Numeric probability

Examples

```
df <- data.frame(id=1, time=1:length(sunspot.year), Y1=sunspot.year)

m1 <- ctModel(type='standt', LAMBDA=diag(1),MANIFESTVAR=0)
m2 <- ctModel(type='standt', LAMBDA=diag(1),MANIFESTVAR=0,DRIFT = .9)

f1 <- ctStanFit(df,m1,cores=1)
```

```
f2 <- ctStanFit(df,m2,cores=1)

ctChisqTest(f1,f2)
```

---

ctCollapse *ctCollapse Easily collapse an array margin using a specified function.*

Description

ctCollapse Easily collapse an array margin using a specified function.

Usage

```
ctCollapse(inarray, collapsemargin, collapsefunc, plyr = TRUE, ...)
```

Arguments

inarray	Input array of more than one dimension.
collapsemargin	Integers denoting which margins to collapse.
collapsefunc	function to use over the collapsing margin.
plyr	Whether to use plyr.
...	additional parameters to pass to collapsefunc.

Examples

```
testarray <- array(rnorm(900,2,1),dim=c(100,3,3))
ctCollapse(testarray,1,mean)
```

---

ctDeintervalise *ctDeintervalise*

Description

Converts intervals in ctsem long format data to absolute time

Usage

```
ctDeintervalise(datalong, id = "id", dT = "dT", startoffset = 0)
```

Arguments

datalong	data to use, in ctsem long format (attained via function ctWideToLong)
id	character string denoting column of data containing numeric identifier for each subject.
dT	character string denoting column of data containing time interval preceding observations in that row.
startoffset	Number of units of time to offset by when converting.

ctDensity *ctDensity*

Description

Wrapper for base R density function that removes outliers and computes 'reasonable' bandwidth and x and y limits. Used for ctsem density plots.

Usage

```
ctDensity(x, bw = "auto", plot = FALSE, ...)
```

Arguments

x	numeric vector on which to compute density.
bw	either 'auto' or a numeric indicating bandwidth.
plot	logical to indicate whether or not to plot the output.
...	Further args to density.

Examples

```
y <- ctDensity(exp(rnorm(80)))
plot(y$density,xlim=y$xlim,ylim=y$ylim)

#### Compare to base defaults:
par(mfrow=c(1,2))
y=exp(rnorm(10000))
ctdens<-ctDensity(y)
plot(ctdens$density, ylim=ctdens$ylim,xlim=ctdens$xlim)
plot(density(y))
```

ctDiscretiseData *Discretise long format continuous time (ctsem) data to specific timestep.*

Description

Extends and rounds timing information so equal intervals, according to specified timestep, are achieved. NA's are inserted in other columns as necessary, any columns specified by TDpredNames or TIpredNames have zeroes rather than NA's inserted (because some estimation routines do not tolerate NA's in covariates).

Usage

```
ctDiscretiseData(
  dlong,
  timestep,
  timecol = "time",
  idcol = "id",
  TDpredNames = NULL,
  TIpredNames = NULL
)
```

Arguments

dlong	Long format data
timestep	Positive real value to discretise
timecol	Name of column containing absolute (not intervals) time information.
idcol	Name of column containing subject id variable.
TDpredNames	Vector of column names of any time dependent predictors
TIpredNames	Vector of column names of any time independent predictors

Value

long format ctsem data.

Examples

```
long <- ctDiscretiseData(dlong=ctstantestdat, timestep = .1,
TDpredNames=c('TD1'),TIpredNames=c('TI1','TI2','TI3'))
```

---

ctDocs *Get documentation pdf for ctsem*

Description

Get documentation pdf for ctsem

Usage

```
ctDocs()
```

Value

Nothing. Opens a pdf.

Examples

```
ctDocs()
```

ctExample1 *ctExample1*

Description

Simulated example dataset for the ctsem package

Format

100 by 17 matrix containing containing ctsem wide format data. 6 measurement occasions of leisure time and happiness and 5 measurement intervals for each of 100 individuals.

ctExample1TIpred *ctExample1TIpred*

Description

Simulated example dataset for the ctsem package

Format

100 by 18 matrix containing containing ctsem wide format data. 6 measurement occasions of leisure time and happiness, 1 measurement of number of friends, and 5 measurement intervals for each of 100 individuals.

ctExample2 *ctExample2*

Description

Simulated example dataset for the ctsem package

Format

100 by 18 matrix containing containing ctsem wide format data. 8 measurement occasions of leisure time and happiness, 7 measurement occasions of a money intervention dummy, and 7 measurement intervals for each of 50 individuals.

ctExample2level *ctExample2level*

Description

Simulated example dataset for the ctsem package

Format

100 by 18 matrix containing ctsem wide format data. 8 measurement occasions of leisure time and happiness, 7 measurement occasions of a money intervention dummy, and 7 measurement intervals for each of 50 individuals.

ctExample3 *ctExample3*

Description

Simulated example dataset for the ctsem package

Format

1 by 399 matrix containing containing ctsem wide format data. 100 observations of variables Y1 and Y2 and 199 measurement intervals, for 1 subject.

ctExample4 *ctExample4*

Description

Simulated example dataset for the ctsem package

Format

20 by 79 matrix containing 20 observations of variables Y1, Y2, Y3, and 19 measurement intervals dTx, for each of 20 individuals.

`ctExtract` *Extract samples from a ctStanFit object*

Description

Extract samples from a ctStanFit object

Usage

```
ctExtract(
  object,
  subjectMatrices = FALSE,
  cores = 2,
  nsamples = "all",
  subjects = "all"
)
```

Arguments

`object`	ctStanFit object, samples may be from Stan's HMC, or the importance sampling approach of ctsem.
`subjectMatrices`	Calculate subject specific system matrices?
`cores`	Only used if subjectMatrices = TRUE . For faster computation use more cores.
`nsamples`	either 'all' or an integer denoting number of random samples to extract.
`subjects`	either 'all', or an integer vector denoting subjects to extract.

Value

Array of posterior samples.

Examples

```
e = ctExtract(ctstantestfit)
```

ctFit *ctFit function placeholder*

Description

For the original ctsem OpenMx functionality, the package ctsemOMX should be loaded.

Usage

```
ctFit(...)
```

Arguments

...	arguments to pass to ctFit, if ctsemOMX is loaded.

Value

message or fit object.

Examples

```
data(AnomAuth)
AnomAuthmodel <- ctModel(LAMBDA = matrix(c(1, 0, 0, 1), nrow = 2, ncol = 2),
  Tpoints = 5, n.latent = 2, n.manifest = 2, MANIFESTVAR=diag(0, 2), TRAITVAR = NULL)
AnomAuthfit <- ctFit(AnomAuth, AnomAuthmodel)
```

ctFitMultiModel *Fit and summarise a list of ctsem models*

Description

Fit and summarise a list of ctsem models

Usage

```
ctFitMultiModel(
  mlist,
  datalong,
  prefix = "",
  type = "stanct",
  cores = 2,
  summaryOutput = TRUE,
  saveFits = TRUE,
  summaryArgs = list(),
  ...
)
```

Arguments

mlist	Named list of models
datalong	ctsem long format data
prefix	prefix for output files.
type	'stanct' for continuous time or 'standt' for discrete time
cores	number of cpu cores to use
summaryOutput	Generate summary output into ctSummary folder? Large datasets can take some time.
saveFits	Save fit objects to working directory?
summaryArgs	Additional arguments for ctSummarise.
...	Additional arguments for ctStanFit.

Value

List containing a named list of model fits ($fits), and a compare object ($compare)

Examples

```
## Not run:
sunspots<-data.frame(id=1,
  time=do.call(seq,(lapply(attributes(sunspot.year)$tsp,function(x) x))),
  sunspots=sunspot.year)

 ssmodel1 <- ctModel(type='omx', manifestNames='sunspots', Tpoints=3,
  latentNames=c('ss_level', 'ss_velocity'),
   LAMBDA=matrix(c( 1, 'ma1| log(1+(exp(param)))' ), nrow=1, ncol=2),
   DRIFT=matrix(c(0, 'a21 | -log(1+exp(param))', 1, 'a22'), nrow=2, ncol=2),
   MANIFESTMEANS=matrix(c('m1|param * 10 + 44'), nrow=1, ncol=1),
   MANIFESTVAR=diag(0,1), #As per original spec
   CINT=matrix(c(0, 0), nrow=2, ncol=1),
   DIFFUSION=matrix(c(0, 0, 0, "diffusion"), ncol=2, nrow=2))

 ssmodel2 <- ssmodel1
 ssmodel2$LAMBDA[2] <- 0

 fits<-ctFitMultiModel(list(m1=ssmodel1,m2=ssmodel2),datalong = sunspots,
   summaryOutput = FALSE,saveFits = FALSE,cores=1)
 print(fits$compare)

## End(Not run)
```

ctGenerate *ctGenerate*

Description

This function generates data according to the specified ctsem model object.

Usage

```
ctGenerate(
  ctmodelobj,
  n.subjects = 100,
  burnin = 0,
  dtmean = 1,
  logdtsd = 0,
  dtmat = NA,
  wide = FALSE
)
```

Arguments

ctmodelobj	ctsem model object from ctModel.
n.subjects	Number of subjects to output.
burnin	Number of initial time points to discard (to simulate stationary data)
dtmean	Positive numeric. Average time interval (delta T) to use.
logdtsd	Numeric. Standard deviation for variability of the time interval.
dtmat	Either NA, or numeric matrix of n.subjects rows and Tpoints-1 columns, containing positive numeric values for all time intervals between measurements. If not NA, dtmean and logdtsd are ignored.
wide	Logical. Output in wide format?

Details

Covariance related matrices are treated as Cholesky factors. TRAITTDPREDCOV and TIPREDCOV matrices are not accounted for, at present. The first 1:n.TDpred rows and columns of TDPREDVAR are used for generating tdpreds at each time point.

Examples

```
#generate data for 2 process model, each process measured by noisy indicator,
#stable individual differences in process levels.

generatingModel<-ctModel(Tpoints=8,n.latent=2,n.TDpred=0,n.TIpred=0,n.manifest=2,
 MANIFESTVAR=diag(.1,2),
 LAMBDA=diag(1,2),
 DRIFT=matrix(c(-.2,-.05,-.1,-.1),nrow=2),
```

```
 TRAITVAR=matrix(c(.5,.2,0,.8),nrow=2),
 DIFFUSION=matrix(c(1,.2,0,4),2),
 CINT=matrix(c(1,0),nrow=2),
 T0MEANS=matrix(0,ncol=1,nrow=2),
 T0VAR=diag(1,2))

data<-ctGenerate(generatingModel,n.subjects=15,burnin=10)
```

---

ctIndplot *ctIndplot*

Description

Convenience function to simply plot individuals trajectories from ctsem wide format data

Usage

```
ctIndplot(
  datawide,
  n.manifest,
  Tpoints,
  n.subjects = "all",
  colourby = "variable",
  vars = "all",
  opacity = 1,
  varnames = NULL,
  xlab = "Time",
  ylab = "Value",
  type = "b",
  start = 0,
  legend = TRUE,
  legendposition = "topright",
  new = TRUE,
  jittersd = 0.05,
  ...
)
```

Arguments

datawide	ctsem wide format data
n.manifest	Number of manifest variables in data structure
Tpoints	Number of discrete time points per case in data structure
n.subjects	Number of subjects to randomly select for plotting, or character vector 'all'.
colourby	set plot colours by "subject" or "variable"
vars	either 'all' or a numeric vector specifying which manifest variables to plot.
opacity	Opacity of plot lines

varnames	vector of variable names for legend (defaults to NULL)
xlab	X axis label.
ylab	Y axis label.
type	character specifying plot type, as per usual base R plot commands. Defaults to 'b', both points and lines.
start	Measurement occasion to start plotting from - defaults to T0.
legend	Logical. Plot a legend?
legendposition	Where to position the legend.
new	logical. If TRUE, creates a new plot, otherwise overlays on current plot.
jittersd	positive numeric indicating standard deviation of noise to add to observed data for plotting purposes.
...	additional plotting parameters.

Examples

```
data(ctExample1)
ctIndplot(ctExample1,n.subjects=1, n.manifest=2,Tpoints=6, colourby='variable')
```

`ctIntervalise` *Converts absolute times to intervals for wide format ctsem panel data*

Description

Converts absolute times to intervals for wide format ctsem panel data

Usage

```
ctIntervalise(
  datawide,
  Tpoints,
  n.manifest,
  n.TDpred = 0,
  n.TIpred = 0,
  imputedefs = F,
  manifestNames = "auto",
  TDpredNames = "auto",
  TIpredNames = "auto",
  digits = 5,
  mininterval = 0.001,
  individualRelativeTime = TRUE,
  startoffset = 0
)
```

Arguments

datawide	Wide format data, containing absolute time measurements, to convert to interval time scale. See ctLongToWide to easily convert long format data.
Tpoints	Maximum number of discrete time points (waves of data, or measurement occasions) for an individual in the input data structure.
n.manifest	number of manifest variables per time point in the data.
n.TDpred	number of time dependent predictors in the data structure.
n.TIpred	number of time independent predictors in the data structure.
imputedefs	if TRUE, impute time intervals based on the measurement occasion (i.e. column) they are in, if FALSE (default), set related observations to NA. FALSE is recommended unless you are certain that the imputed value (mean of the relevant time column) is appropriate. Noise and bias in estimates will result if wrongly set to TRUE.
manifestNames	vector of character strings giving variable names of manifest indicator variables (without _Tx suffix for measurement occasion).
TDpredNames	vector of character strings giving variable names of time dependent predictor variables (without _Tx suffix for measurement occasion).
TIpredNames	vector of character strings giving variable names of time independent predictor variables.
digits	How many digits to round to for interval calculations.
mininterval	set to lower than any possible observed measurement interval, but above 0 - this is used for filling NA values where necessary and has no impact on estimates when set in the correct range. (If all observed intervals are greater than 1, mininterval=1 may be a good choice)
individualRelativeTime	if TRUE (default), the first measurement for each individual is assumed to be taken at time 0, and all other times are adjusted accordingly. If FALSE, new columns for an initial wave are created, consisting only of observations which occurred at the earliest observation time of the entire sample.
startoffset	if 0 (default) uses earliest observation as start time. If greater than 0, all first observations are NA, with distance of startoffset to first recorded observation.

Details

Time column must be numeric!

Examples

```
wideexample <- ctLongToWide(datalong = ctstantestdat, id = "id",
time = "time", manifestNames = c("Y1", "Y2"),
TDpredNames = "TD1", TIpredNames = c("TI1", "TI2","TI3"))

#Then convert the absolute times to intervals, using the Tpoints reported from the prior step.
wide <- ctIntervalise(datawide = wideexample, Tpoints = 10, n.manifest = 2,
n.TDpred = 1, n.TIpred = 3, manifestNames = c("Y1", "Y2"),
```

```
TDpredNames = "TD1", TIpredNames = c("TI1", "TI2","TI3") )

print(wide)
```

---

ctKalman *ctKalman*

Description

Outputs predicted, updated, and smoothed estimates of manifest indicators and latent states, with covariances, for specific subjects from data fit with ctStanFit, based on either the mode (if optimized) or mean (if sampled) of parameter distribution.

Usage

```
ctKalman(
  fit,
  timerange = "asdata",
  timestep = "auto",
  subjects = fit$setup$idmap[1, 1],
  removeObs = FALSE,
  plot = FALSE,
  standardisederrors = TRUE,
  realid = TRUE,
  ...
)
```

Arguments

fit	fit object as generated by ctStanFit.
timerange	Either 'asdata' to just use the observed data range, or a numeric vector of length 2 denoting start and end of time range, allowing for estimates outside the range of observed data. Ranges smaller than the observed data are ignored.
timestep	Either 'asdata' to just use the observed data (which also requires 'asdata' for timerange) or a positive numeric value indicating the time step to use for interpolating values. Lower values give a more accurate / smooth representation, but take a little more time to calculate.
subjects	vector of integers denoting which subjects (from 1 to N) to plot predictions for.
removeObs	Logical or integer. If TRUE, observations (but not covariates) are set to NA, so only expectations based on parameters and covariates are returned. If a positive integer N, every N observations are retained while others are set NA for computing model expectations – useful for observing prediction performance forward further in time than one observation.
plot	Logical. If TRUE, plots output instead of returning it. See plot.ctKalmanDF (Stan based fit) for the possible arguments.

standardisederrors
: if TRUE, also include standardised error output (based on covariance per time point).

realid
: use original (not necessarily integer sequence) subject id's? Otherwise use integers 1:N.

...
: additional arguments to pass to plot.ctKalmanDF.

Value

Returns a list containing matrix objects etaprior, etaupd, etasmooth, y, yprior, yupd, ysmooth, prederror, time, loglik, with values for each time point in each row. eta refers to latent states and y to manifest indicators - y itself is thus just the input data. Covariance matrices etapriorcov, etaupdcov, etasmoothcov, ypriorcov, yupdcov, ysmoothcov, are returned in a row * column * time array. Some outputs are unavailable for ctStan fits at present. If plot=TRUE, nothing is returned but a plot is generated.

Examples

```
#Basic
ctKalman(ctstantestfit, timerange=c(0,60), plot=TRUE)

#Multiple subjects, y and yprior, showing plot arguments
plot1<-ctKalman(ctstantestfit, timerange=c(0,60), timestep=.1, plot=TRUE,
  subjects=2:3,
  kalmanvec=c('y','yprior'),
  errorvec=c(NA,'ypriorcov')) #'auto' would also have achieved this

 #modify plot as per normal with ggplot
 print(plot1+ggplot2::coord_cartesian(xlim=c(0,10)))

 #or generate custom plot from scratch:#'
 k=ctKalman(ctstantestfit, timerange=c(0,60), timestep=.1, subjects=2:3)
 library(ggplot2)
 ggplot(k[k$Element %in% 'yprior',],
   aes(x=Time, y=value,colour=Subject,linetype=Row)) +
   geom_line() +
   theme_bw()
```

---

ctLongToWide *ctLongToWide Restructures time series / panel data from long format to wide format for ctsem analysis*

Description

ctLongToWide Restructures time series / panel data from long format to wide format for ctsem analysis

Usage

```
ctLongToWide(
  datalong,
  id,
  time,
  manifestNames,
  TDpredNames = NULL,
  TIpredNames = NULL
)
```

Arguments

datalong	dataset in long format, including subject/id column, observation time (or change in observation time, with 0 for first observation) column, indicator (manifest / observed) variables, any time dependent predictors, and any time independent predictors.
id	character string giving column name of the subject/id column
time	character string giving column name of the time columnn
manifestNames	vector of character strings giving column names of manifest indicator variables
TDpredNames	vector of character strings giving column names of time dependent predictor variables
TIpredNames	vector of character strings giving column names of time independent predictor variables

Details

Time column must be numeric

See Also

ctIntervalise

Examples

```
wideexample <- ctLongToWide(datalong = ctstantestdat, id = "id",
time = "time", manifestNames = c("Y1", "Y2"),
TDpredNames = "TD1", TIpredNames = c("TI1", "TI2","TI3"))

#Then convert the absolute times to intervals, using the Tpoints reported from the prior step.
wide <- ctIntervalise(datawide = wideexample, Tpoints = 10, n.manifest = 2,
n.TDpred = 1, n.TIpred = 3, manifestNames = c("Y1", "Y2"),
TDpredNames = "TD1", TIpredNames = c("TI1", "TI2","TI3") )

print(wide)
```

ctLOO *K fold cross validation for ctStanFit objects*

Description

K fold cross validation for ctStanFit objects

Usage

```
ctLOO(
  fit,
  folds = 10,
  cores = 2,
  parallelFolds = FALSE,
 subjectwise = ifelse(length(unique(fit$standata$subject)) > folds, TRUE, FALSE),
  keepfirstobs = FALSE,
  leaveOutN = NA,
  refit = TRUE
)
```

Arguments

fit	ctStanfit object
folds	Number of cross validation splits to use – 10 folds implies that the model is re-fit 10 times, each time to a data set with 1/10 of the observations randomly removed.
cores	Number of processor cores to use.
parallelFolds	compute folds in parallel or use cores to finish single folds faster. parallelFolds will use folds times as much memory.
subjectwise	drop random subjects instead of data rows?
keepfirstobs	do not drop first observation (more stable estimates)
leaveOutN	if a positive integer is given, the folds argument is ignored and instead the folds are calculated by leaving out every Nth row from the data when fitting. Leaving 2 out would result in 3 folds (starting at rows 1,2,3), each containing one third of the data.
refit	if FALSE, do not optimise parameters for the new data set, just compute the likelihoods etc from the original parameters

Value

list

Examples

```
ctLOO(ctstantestfit)
```

ctModel *Define a ctsem model*

Description

This function is used to specify a continuous time structural equation model, which can then be fit to data with function ctStanFit.

Usage

```
ctModel(
  LAMBDA,
  type = "omx",
  n.manifest = "auto",
  n.latent = "auto",
  Tpoints = NULL,
  manifestNames = "auto",
  latentNames = "auto",
  id = "id",
  time = "time",
  silent = FALSE,
  T0VAR = "auto",
  T0MEANS = "auto",
  MANIFESTMEANS = "auto",
  MANIFESTVAR = "auto",
  DRIFT = "auto",
  CINT = "auto",
  DIFFUSION = "auto",
  n.TDpred = "auto",
  TDpredNames = "auto",
  n.TIpred = "auto",
  TIpredNames = "auto",
  tipredDefault = TRUE,
  TRAITVAR = NULL,
  T0TRAITEFFECT = NULL,
  MANIFESTTRAITVAR = NULL,
  TDPREDMEANS = "auto",
  TDPREDEFFECT = "auto",
  T0TDPREDCOV = "auto",
  TDPREDVAR = "auto",
  TRAITTDPREDCOV = "auto",
  TDTIPREDCOV = "auto",
  TIPREDMEANS = "auto",
  TIPREDEFFECT = "auto",
  T0TIPREDEFFECT = "auto",
  TIPREDVAR = "auto",
  PARS = NULL,
```

```
  startValues = NULL
)
```

Arguments

LAMBDA	n.manifest*n.latent loading matrix relating latent to manifest variables, with latent processes 1:n.latent along the columns, and manifest variables 1:n.manifest in the rows.
type	character string. If 'omx' (default) configures model for maximum likelihood fitting with ctFit, using OpenMx. If 'stanct' or 'standt' configures either continuous ('stanct') or discrete ('standt') time model for Bayesian fitting with ctStanFit, using Stan.
n.manifest	Number of manifest indicators per individual at each measurement occasion / time point. Manifest variables are included as the first element of the wide data matrix, with all the 1:n.manifest manifest variables at time 1 followed by those of time 2, and so on.
n.latent	Number of latent processes.
Tpoints	For type='omx' only. Number of time points, or measurement occasions, in the data. This will generally be the maximum number of time points for a single individual, but may be one extra if sample relative time intervals are used, see ctIntervalise.
manifestNames	n.manifest length vector of manifest variable names as they appear in the data structure, without any _Tx time point suffix that may be present in wide data. Defaults to Y1, Y2, etc.
latentNames	n.latent length vector of latent variable names (used for naming parameters, defaults to eta1, eta2, etc).
id	character string denoting column name containing subject identification variables. id data may be of any form, though will be coerced internally to an integer sequence rising from 1.
time	character string denoting column name containing timing data. Timing data must be numeric.
silent	Suppress all output to console.
T0VAR	lower triangular n.latent*n.latent cholesky matrix of latent process initial variance / covariance. "auto" freely estimates all parameters.
T0MEANS	n.latent*1 matrix of latent process means at first time point, T0. "auto" freely estimates all parameters.
MANIFESTMEANS	n.manifest*1 matrix of manifest intercept parameters. "auto" frees all parameters.
MANIFESTVAR	lower triangular n.manifest*n.manifest cholesky matrix of variance / covariance between manifests at each measurement occasion (i.e. measurement error / residual). "auto" freely estimates variance parameters, and fixes covariances between manifests to 0. "free" frees all values, including covariances.
DRIFT	n.latent*n.latent DRIFT matrix of continuous auto and cross effects, relating the processes over time. "auto" freely estimates all parameters.

CINT — n.latent * 1 matrix of latent process intercepts, allowing for non 0 asymptotic levels of the latent processes. Generally only necessary for additional trends and more complex dynamics. "auto" fixes all parameters to 0.

DIFFUSION — lower triangular n.latent*n.latent cholesky matrix of diffusion process variance and covariance (latent error / dynamic innovation). "auto" freely estimates all parameters.

n.TDpred — Number of time dependent predictor variables in the dataset.

TDpredNames — n.TDpred length vector of time dependent predictor variable names, as they appear in the data structure, without any _Tx time point suffix that may appear in wide data. Default names are TD1, TD2, etc.

n.TIpred — Number of time independent predictors. Each TIpredictor is inserted at the right of the data matrix, after the time intervals.

TIpredNames — n.TIpred length vector of time independent predictor variable names, as they appear in the data structure. Default names are TI1, TI2, etc.

tipredDefault — Logical. TRUE sets any parameters with unspecified time independent predictor effects to have effects estimated, FALSE fixes the effect to zero unless individually specified.

TRAITVAR — For type='omx' only. Either NULL, if no trait / unobserved heterogeneity effect, or lower triangular n.latent*n.latent cholesky matrix of trait variance / covariance across subjects. "auto" freely estimates all parameters.

T0TRAITEFFECT — For type='omx' only. Either NULL, if no trait / individual heterogeneity effect, or lower triangular n.latent*n.latent cholesky matrix of initial trait variance / covariance. "auto" freely estimates all parametrers, if the TRAITVAR matrix is specified.

MANIFESTTRAITVAR — For type='omx' only. Either NULL (default) if no trait variance / individual heterogeneity in the level of the manifest indicators, otherwise a lower triangular n.manifest * n.manifest variance / covariance matrix. Set to "auto" to include and free all parameters - but identification problems will arise if TRAITVAR is also set.

TDPREDMEANS — For type='omx' only. (n.TDpred * (Tpoints - 1)) rows * 1 column matrix of time dependent predictor means. If 'auto', the means are freely estimated. Otherwise, the means for the Tpoints observations of your first time dependent predictor are followed by those of TDpred 2, and so on.

TDPREDEFFECT — n.latent*n.TDpred matrix of effects from time dependent predictors to latent processes. Effects from 1:n.TDpred columns TDpredictors go to 1:n.latent rows of latent processes. "auto" freely estimates all parameters.

T0TDPREDCOV — For type='omx' only. n.latent rows * (Tpoints * n.TDpred) columns covariance matrix between latents at T0 and time dependent predictors. Default of "auto" restricts covariance to 0, which is consistent with covariance to other time points. To freely estimate parameters, specify either 'free', or the desired matrix.

TDPREDVAR — For type='omx' only. lower triangular (n.TDpred * Tpoints) rows * (n.TDpred * Tpoints) columns variance / covariance cholesky matrix for time dependent predictors. "auto" (default) freely estimates all parameters.

TRAITTDPREDCOV For type='omx' only. n.latent rows * (n.TDpred*Tpoints) columns covariance matrix of latent traits and time dependent predictors. Defaults to zeroes, assuming predictors are independent of subjects baseline levels. When predictors depend on the subjects, this should instead be set to 'free' or manually specified. The Tpoints columns of the first preditor are followed by those of the second and so on. Covariances with the trait variance of latent process 1 are specified in row 1, process 2 in row 2, etc. "auto" (default) sets this matrix to zeroes, (if both traits and time dependent predictors exist, otherwise this matrix is set to NULL, and ignored in any case).

TDTIPREDCOV For type='omx' only. (n.TDpred * Tpoints) rows * n.TIpred columns covariance matrix between time dependent and time independent predictors. "auto" (default) freely estimates all parameters.

TIPREDMEANS For type='omx' only. n.TIpred * 1 matrix of time independent predictor means. If 'auto', the means are freely estimated.

TIPREDEFFECT For type='omx' only. n.latent*n.TIpred effect matrix of time independent predictors on latent processes. "auto" freely estimates all parameters and generates starting values. TIPREDEFFECT parameters for type='stan' are estimated by default on all subject level parameters, to restrict this, manually edit the model object after creation.

T0TIPREDEFFECT For type='omx' only.n.latent*n.TIpred effect matrix of time independent predictors on latents at T0. "auto" freely estimates all parameters, though note that under the default setting of `stationary` for ctFit, this matrix is ignored as the effects are determined based on the overall process parameters.

TIPREDVAR For type='omx' only.lower triangular n.TIpred * n.TIpred Cholesky decomposed covariance matrix for all time independent predictors. "auto" (default) freely estimates all parameters.

PARS for types 'stanct' and 'standt' only. May be of any structure, only needed to contain extra parameters for certain non-linear models.

startValues For type='omx' only. A named vector, where the names of each value must match a parameter in the specified model, and the value sets the starting value for that parameter during optimization. If not set, random starting values representing relatively stable processes with small effects and covariances are generated by ctFit. Better starting values may improve model fit speed and the chance of an appropriate model fit.

Examples

```
### Frequentist example:
### impulse and level change time dependent predictor
### example from Driver, Oud, Voelkle (2015)
data('ctExample2')
tdpredmodel <- ctModel(n.manifest = 2, n.latent = 3, n.TDpred = 1,
Tpoints = 8, manifestNames = c('LeisureTime', 'Happiness'),
TDpredNames = 'MoneyInt',
latentNames = c('LeisureTime', 'Happiness', 'MoneyIntLatent'),
LAMBDA = matrix(c(1,0, 0,1, 0,0), ncol = 3), TRAITVAR = "auto")

tdpredmodel$TRAITVAR[3, ] <- 0
```

```
 tdpredmodel$TRAITVAR[, 3] <- 0
 tdpredmodel$DIFFUSION[, 3] <- 0
 tdpredmodel$DIFFUSION[3, ] <- 0
 tdpredmodel$T0VAR[3, ] <- 0
 tdpredmodel$T0VAR[, 3] <- 0
 tdpredmodel$CINT[3] <- 0
 tdpredmodel$T0MEANS[3] <- 0
 tdpredmodel$TDPREDEFFECT[3, ] <- 1
 tdpredmodel$DRIFT[3, ] <- 0


###Bayesian example:
model<-ctModel(type='stanct',
n.latent=2, latentNames=c('eta1','eta2'),
n.manifest=2, manifestNames=c('Y1','Y2'),
n.TDpred=1, TDpredNames='TD1',
n.TIpred=3, TIpredNames=c('TI1','TI2','TI3'),
LAMBDA=diag(2))
```

`ctModelHigherOrder` *Raise the order of a ctsem model object of type 'omx'.*

Description

Raise the order of a ctsem model object of type 'omx'.

Usage

```
ctModelHigherOrder(
  ctm,
  indices,
  diffusion = TRUE,
  crosseffects = FALSE,
  cint = FALSE,
  explosive = FALSE
)
```

Arguments

`ctm`	ctModel
`indices`	Vector of integers, which latents to raise the order of.
`diffusion`	Shift the diffusion parameters / values to the higher order?
`crosseffects`	Shift cross coupling parameters of the DRIFT matrix to the higher order?
`cint`	shift continuous intercepts to higher order?
`explosive`	Allow explosive (non equilibrium returning) processes?

Value

extended ctModel

Examples

```
om <- ctModel(LAMBDA=diag(1,2),DRIFT=0,
  MANIFESTMEANS=0,type='omx',Tpoints=4)

om <- ctModelHigherOrder(om,1:2)
print(om$DRIFT)

m <- ctStanModel(om)
print(m$pars)
```

`ctModelLatex` *Generate and optionally compile latex equation of subject level ctsem model.*

Description

Generate and optionally compile latex equation of subject level ctsem model.

Usage

```
ctModelLatex(
  x,
  matrixnames = TRUE,
  digits = 3,
  linearise = class(x) %in% "ctStanFit",
  textsize = "normalsize",
  folder = tempdir(),
  filename = paste0("ctsemTex", as.numeric(Sys.time())),
  tex = TRUE,
  equationonly = FALSE,
  compile = TRUE,
  open = TRUE,
  includeNote = TRUE,
  minimal = FALSE
)
```

Arguments

`x`	ctsem model object or ctStanFit object.
`matrixnames`	Logical. If TRUE, includes ctsem matrix names such as DRIFT and DIFFUSION under the matrices.
`digits`	Precision of decimals for numeric values.

linearise	Logical. Show the linearised normal approximation for subject parameters and covariate effects, or the raw parameters?
textsize	Standard latex text sizes – tiny scriptsize footnotesize small normalsize large Large LARGE huge Huge. Useful if output overflows page.
folder	Character string specifying folder to save to, defaults to temporary directory, use "./" for working directory.
filename	filename, without suffix, to output .tex and .pdf files too.
tex	Save .tex file? Otherwise latex is simply returned within R as a string.
equationonly	Logical. If TRUE, output is only the latex relevant to the equation, not a compileable document.
compile	Compile to .pdf? (Depends on tex = TRUE)
open	Open after compiling? (Depends on compile = TRUE)
includeNote	Include text describing matrix transformations and subject notation? triangular matrices (which results in a covariance or Cholesky matrix) is shown – the latter is a more direct representation of the model, while the former is often simpler to convey.
minimal	if TRUE, outputs reduced form version displaying matrix dimensions and equation structure only.

Value

character string of latex code. Side effects include saving a .tex, .pdf, and displaying the pdf.

Examples

```
ctmodel <- ctModel(type='stanct',
n.latent=2, n.manifest=1,
manifestNames='sunspots',
latentNames=c('ss_level', 'ss_velocity'),
LAMBDA=matrix(c( 1, 'ma1' ), nrow=1, ncol=2),
DRIFT=matrix(c(0, 1,   'a21', 'a22'), nrow=2, ncol=2, byrow=TRUE),
MANIFESTMEANS=matrix(c('m1'), nrow=1, ncol=1),
CINT=matrix(c(0, 0), nrow=2, ncol=1),
DIFFUSION=matrix(c(
  0, 0,
  0, "diffusion"), ncol=2, nrow=2, byrow=TRUE))

l=ctModelLatex(ctmodel,compile=FALSE, open=FALSE)
cat(l)
```

ctPlotArray *Plots three dimensional y values for quantile plots*

Description

1st margin of $Y sets line values, 2nd sets variables, 3rd quantiles.

Usage

```
ctPlotArray(
  input,
  grid = FALSE,
  add = FALSE,
  colvec = "auto",
  lwdvec = "auto",
  ltyvec = "auto",
  typevec = "auto",
  plotcontrol = list(ylab = "Array values", xaxs = "i"),
  legend = TRUE,
  legendcontrol = list(),
  polygon = TRUE,
  polygonalpha = 0.1,
  polygoncontrol = list(steps = 25)
)
```

Arguments

input	list containing 3 dimensional array to use for Y values, $y and vector of corresponding x values $x.
grid	Logical. Plot with a grid?
add	Logical. If TRUE, plotting is overlayed on current plot, without creating new plot.
colvec	color vector of same length as 2nd margin.
lwdvec	lwd vector of same length as 2nd margin.
ltyvec	lty vector of same length as 2nd margin.
typevec	type vector of same length as 2nd margin.
plotcontrol	list of arguments to pass to plot.
legend	Logical. Draw a legend?
legendcontrol	list of arguments to pass to legend.
polygon	Logical. Draw the uncertainty polygon?
polygonalpha	Numeric, multiplier for alpha (transparency) of the uncertainty polygon.
polygoncontrol	list of arguments to pass to ctPoly

Value

Nothing. Generates plots.

Examples

```
#'
input<-ctStanTIpredeffects(ctstantestfit, plot=FALSE, whichpars='CINT',
 nsamples=10,nsubjects=10)

ctPlotArray(input=input)
```

---

ctPoly *Plots uncertainty bands with shading*

Description

Plots uncertainty bands with shading

Usage

```
ctPoly(x, y, ylow, yhigh, steps = 20, ...)
```

Arguments

x	x values
y	y values
ylow	lower limits of y
yhigh	upper limits of y
steps	number of polygons to overlay - higher integers lead to smoother changes in transparency between y and yhigh / ylow.
...	arguments to pass to polygon()

Value

Nothing. Adds a polygon to existing plot.

Examples

```
plot(0:100,sqrt(0:100),type='l')
ctPoly(x=0:100, y=sqrt(0:100),
yhigh=sqrt(0:100) - runif(101),
ylow=sqrt(0:100) + runif(101),
col=adjustcolor('red',alpha.f=.1))
```

ctPostPredData *Create a data.table to compare data generated from a ctsem fit with the original data.*

Description

This function allows for easy comparison of data generated from a fitted ctsem model with the original data used to fit the model. It provides options to include residuals in the comparison.

Usage

```
ctPostPredData(fit, residuals = F)
```

Arguments

fit	A fitted ctsem model.
residuals	If set to TRUE, includes residuals in the comparison.

Value

A data table containing the comparison between generated and original data.

See Also

Other ctsem functions for model fitting and analysis.

Examples

```
data_comparison <- ctPostPredData(ctstantestfit)
```

ctPostPredPlots *Create diagnostic plots to assess the goodness-of-fit for a ctsem model.*

Description

This function generates a set of diagnostic plots to assess the goodness-of-fit for a fitted ctsem model.

Usage

```
ctPostPredPlots(fit)
```

Arguments

fit	A fitted ctsem model.

Details

The function calculates various statistics and creates visualizations to evaluate how well the generated data matches the original data used to fit the model. The plots included are as follows: - A scatter plot comparing observed values and the median of generated data. - A plot showing the proportion of observed data outside the 95 - A density plot of the proportion of observed data greater than the generated data. - A time series plot of the proportion of observed data greater than generated data.

See Also

Other ctsem functions for model fitting and analysis.

Examples

```
ctPostPredPlots(ctstantestfit)
```

`ctResiduals` *Extract Standardized Residuals from a ctsem Fit*

Description

This function takes a fit object from the ctsem package and extracts the standardized residuals.

Usage

```
ctResiduals(fit)
```

Arguments

`fit`	A fitted model object generated by the ctsem package.

Details

This function uses the `ctStanKalman` function to calculate the standardized residuals and then extracts and formats them as a data table. The standardized residuals represent the differences between the observed and predicted values, divided by the standard errors of the observations.

Value

A data table containing the standardized residuals for each subject and time point.

See Also

`ctStanKalman`

Examples

```
data.table::setDTthreads(1) #ignore this line
# Example usage:
residuals <- ctResiduals(ctstantestfit)
```

---

ctStanContinuousPars *ctStanContinuousPars*

Description

Returns the continuous time parameter matrices of a ctStanFit fit object

Usage

```
ctStanContinuousPars(
  fit,
  calcfunc = quantile,
  calcfuncargs = list(probs = 0.5),
  timeinterval = 1
)
```

Arguments

fit	fit object from ctStanFit
calcfunc	Function to apply over samples, must return a single value. By default the median over all samples is returned using the quantile function, but one might also be interested in the mean or sd, for instance.
calcfuncargs	A list of additional parameters to pass to calcfunc. For instance, with the default of calcfunc = quantile, the probs argument is needed to ensure only a single value is returned.
timeinterval	time interval for discrete time parameter matrix computation.

Examples

```
#posterior median over all subjects (also reflects mean of unconstrained pars)
ctStanContinuousPars(ctstantestfit)
```

ctStanDiscretePars *ctStanDiscretePars*

Description

Calculate model implied regressions for a sequence of time intervals (if ct) or steps (if dt) based on a ctStanFit object, for specified subjects.

Usage

```
ctStanDiscretePars(
  ctstanfitobj,
  subjects = "popmean",
  times = seq(from = 0, to = 10, by = 0.1),
  nsamples = 100,
  observational = FALSE,
  standardise = FALSE,
  cov = FALSE,
  plot = FALSE,
  cores = 2,
  ...
)
```

Arguments

ctstanfitobj	model fit from ctStanFit
subjects	Either 'popmean', to use the population mean parameter, or a vector of integers denoting which subjects.
times	Numeric vector of positive values, discrete time parameters will be calculated for each. If the fit object is a discrete time model, these should be positive integers.
nsamples	Number of samples from the stanfit to use for plotting. Higher values will increase smoothness / accuracy, at cost of plotting speed. Values greater than the total number of samples will be set to total samples.
observational	Logical. If TRUE, outputs expected change in processes *conditional on observing* a 1 unit change in each – this change is correlated according to the DIFFUSION matrix. If FALSE, outputs expected regression values – also interpretable as an independent 1 unit change on each process, giving the expected response under a 1 unit experimental impulse.
standardise	Logical. If TRUE, output is standardised according to expected total within subject variance, given by the asymDIFFUSIONcov matrix.
cov	Logical. If TRUE, covariances are returned instead of regression coefficients.
plot	Logical. If TRUE, plots output using ctStanDiscreteParsPlot instead of returning output.

cores	Number of cpu cores to use for computing subject matrices. If subject matrices were saved during fiting, not used.
...	additional plotting arguments to control ctStanDiscreteParsPlot

Examples

```
data.table::setDTthreads(1) #ignore this line
ctStanDiscretePars(ctstantestfit,times=seq(.5,4,.1),
 plot=TRUE,indices='CR')

#modify plot
require(ggplot2)
g=ctStanDiscretePars(ctstantestfit,times=seq(.5,4,.1),
 plot=TRUE,indices='CR')
g= g+ labs(title='Cross effects')
print(g)
```

---

ctStanDiscreteParsPlot *ctStanDiscreteParsPlot*

Description

Plots model implied regression strengths at specified times for continuous time models fit with ctStanFit.

Usage

```
ctStanDiscreteParsPlot(
  x,
  indices = "all",
  quantiles = c(0.025, 0.5, 0.975),
  latentNames = "auto",
  ylab = "Coefficient",
  xlab = "Time interval",
  ylim = NA,
  facets = NA,
  splitSubjects = TRUE,
  colour = "Effect",
  title = "Temporal regressions | independent shock of 1.0",
  polygonalpha = 0.1,
  ggcode = NA
)
```

Arguments

x	list object returned from ctStanDiscretePars.
indices	Either a string specifying type of plot to create, or an n by 2 matrix specifying which indices of the output matrix to plot. 'AR' specifies all diagonals, for discrete time autoregression parameters. 'CR' specifies all off-diagonals,for discrete time cross regression parameters. 'all' plots all AR and CR effects at once.
quantiles	numeric vector of length 3, with values between 0 and 1, specifying which quantiles to plot. The default of c(.05,.5,.95) plots 95% credible intervals and the posterior median at 50%.
latentNames	Vector of character strings denoting names for the latent variables. 'auto' just uses eta1 eta2 etc.
ylab	y label.
xlab	x label.
ylim	Custom ylim.
facets	May be 'Subject' or 'Effect'.
splitSubjects	if TRUE, subjects are plotted separately, if FALSE they are combined.
colour	Character string denoting how colour varies. 'Effect' or 'Subject'.
title	Character string.
polygonalpha	Numeric between 0 and 1 to multiply the alpha of the fill.
ggcode	if TRUE, returns a list containing the data.table to plot, and a character string that can be evaluated (with the necessary arguments such as ylab etc filled in). For modifying plots.

Value

A ggplot2 object. This can be modified by the various ggplot2 functions, or displayed using print(x).

Examples

```
data.table::setDTthreads(1) #ignore this line
x <- ctStanDiscretePars(ctstantestfit)
ctStanDiscreteParsPlot(x, indices='CR')

#to modify plot:
g <- ctStanDiscreteParsPlot(x, indices='CR') +
 ggplot2::labs(title='My ggplot modification')
print(g)
```

ctStanFit *ctStanFit*

Description

Fits a ctsem model specified via ctModel with type either 'stanct' or 'standt'.

Usage

```
ctStanFit(
  datalong,
  ctstanmodel,
  stanmodeltext = NA,
  iter = 1000,
  intoverstates = TRUE,
  binomial = FALSE,
  fit = TRUE,
  intoverpop = "auto",
  sameInitialTimes = FALSE,
  stationary = FALSE,
  plot = FALSE,
  derrind = NA,
  optimize = TRUE,
  optimcontrol = list(),
  nlcontrol = list(),
  nopriors = NA,
  priors = FALSE,
  chains = 2,
  cores = ifelse(optimize, getOption("mc.cores", 2L), "maxneeded"),
  inits = NULL,
  forcerecompile = FALSE,
  saveCompile = TRUE,
  savescores = FALSE,
  savesubjectmatrices = FALSE,
  saveComplexPars = FALSE,
  gendata = FALSE,
  control = list(),
  verbose = 0,
  vb = FALSE,
  ...
)
```

Arguments

datalong	long format data containing columns for subject id (numeric values, 1 to max subjects), manifest variables, any time dependent (i.e. varying within subject) predictors, and any time independent (not varying within subject) predictors.

`ctstanmodel` model object as generated by `ctModel` with type='stanct' or 'standt', for continuous or discrete time models respectively.

`stanmodeltext` already specified Stan model character string, generally leave NA unless modifying Stan model directly. (Possible after modification of output from fit=FALSE)

`iter` used when `optimize=FALSE`. number of iterations, half of which will be devoted to warmup by default when sampling. When optimizing, this is the maximum number of iterations to allow – convergence hopefully occurs before this!

`intoverstates` logical indicating whether or not to integrate over latent states using a Kalman filter. Generally recommended to set TRUE unless using non-gaussian measurement model.

`binomial` Deprecated. Logical indicating the use of binary rather than Gaussian data, as with IRT analyses. This now sets `intoverstates = FALSE` and the `manifesttype` of every indicator to 1, for binary.

`fit` If TRUE, fit specified model using Stan, if FALSE, return stan model object without fitting.

`intoverpop` if 'auto', set to TRUE if optimizing and FALSE if using hmc. if TRUE, integrates over population distribution of parameters rather than full sampling. Allows for optimization of non-linearities and random effects.

`sameInitialTimes` if TRUE, include an empty observation for every subject that has no observation at the earliest observation time of the dataset. This ensures that the T0MEANS occurs for every subject at the same time, rather than just at the earliest observation for that subject. Important when modelling trends over time, age, etc.

`stationary` Logical. If TRUE, T0VAR and T0MEANS input matrices are ignored, the parameters are instead fixed to long run expectations. More control over this can be achieved by instead setting parameter names of T0MEANS and T0VAR matrices in the input model to 'stationary', for elements that should be fixed to stationarity.

`plot` if TRUE, for sampling, a Shiny program is launched upon fitting to interactively plot samples. May struggle with many (e.g., > 5000) parameters. For optimizing, various optimization details are plotted – in development.

`derrind` deprecated, latents involved in dynamic error calculations are determined automatically now.

`optimize` if TRUE, use `stanoptimis` function for maximum a posteriori / importance sampling estimates, otherwise use the HMC sampler from Stan, which is (much) slower, but generally more robust, accurate, and informative.

`optimcontrol` list of parameters sent to `stanoptimis` governing optimization / importance sampling.

`nlcontrol` List of non-linear control parameters. `maxtimestep` must be a positive numeric, specifying the largest time span covered by the numerical integration. The large default ensures that for each observation time interval, only a single step of exponential integration is used. When `maxtimestep` is smaller than the observation time interval, the integration is nested within an Euler like loop. Smaller values may offer greater accuracy, but are slower and not always necessary. Given the exponential integration, linear model elements are fit exactly with only a single step.

nopriors	deprecated, use priors argument. logical. If TRUE, any priors are disabled – sometimes desirable for optimization.
priors	if TRUE, priors are included in computations, otherwise specified priors are ignored.
chains	used when optimize=FALSE. Number of chains to sample, during HMC or post-optimization importance sampling. Unless the cores argument is also set, the number of chains determines the number of cpu cores used, up to the maximum available minus one. Irrelevant when optimize=TRUE.
cores	number of cpu cores to use. Either 'maxneeded' to use as many as available minus one, up to the number of chains, or a positive integer. If optimize=TRUE, more cores are generally faster.
inits	either character string 'optimize, NULL, or vector of (unconstrained) parameter start values, as returned by the rstan function rstan::unconstrain_pars, or the parameter values found in a ctsem fit object myfit$stanfit$rawest (or $rawposterior) for instance.
forcerecompile	logical. For development purposes. If TRUE, stan model is recompiled, regardless of apparent need for compilation.
saveCompile	if TRUE and compilation is needed / requested, writes the stan model to the parent frame as ctsem.compiled (unless that object already exists and is not from ctsem), to avoid unnecessary recompilation.
savescores	Logical. If TRUE, output from the Kalman filter is saved in output. For datasets with many variables or time points, will increase file size substantially.
savesubjectmatrices	Logical. If TRUE, subject specific matrices are saved – only relevant when either time dependent predictors or individual differences are used. Can increase memory usage dramatically in large models, and can be computed after fitting using ctExtract or ctStanSubjectPars .
saveComplexPars	Logical. If TRUE, also save rowwise output of any complex parameters specified, i.e. combinations of parameters, functions and states.
gendata	Logical – If TRUE, uses provided data for only covariates and a time and missingness structure, and generates random data according to the specified model / priors. Generated data is in the $Ygen subobject after running extract on the fit object. For datasets with many manifest variables or time points, file size may be large. To generate data based on the posterior of a fitted model, see ctStanGenerateFromFit.
control	Used when optimize=FALSE. List of arguments sent to stan control argument, regarding warmup / sampling behaviour. Unless specified, values used are: list(adapt_delta = .8, adapt_window=2, max_treedepth=10, adapt_init_buffer=2, stepsize = .001)
verbose	Integer from 0 to 2. Higher values print more information during model fit – for debugging.
vb	Logical. Use variational Bayes algorithm from stan? Only kind of working, not recommended.
...	additional arguments to pass to stan function.

Examples

```
#generate a ctStanModel relying heavily on defaults
model<-ctModel(type='stanct',
  latentNames=c('eta1','eta2'),
  manifestNames=c('Y1','Y2'),
  MANIFESTVAR=diag(.1,2),
  TDpredNames='TD1',
  TIpredNames=c('TI1','TI2','TI3'),
  LAMBDA=diag(2))

fit<-ctStanFit(ctstantestdat, model,priors=TRUE)

summary(fit)

plot(fit,wait=FALSE)

#### extended examples

library(ctsem)
set.seed(3)

#  Data generation (run this, but no need to understand!) -----------------

Tpoints <- 20
nmanifest <- 4
nlatent <- 2
nsubjects<-20

#random effects
age <- rnorm(nsubjects) #standardised
cint1<-rnorm(nsubjects,2,.3)+age*.5
cint2 <- cint1*.5+rnorm(nsubjects,1,.2)+age*.5
tdpredeffect <- rnorm(nsubjects,5,.3)+age*.5

for(i in 1:nsubjects){
  #generating model
  gm<-ctModel(Tpoints=Tpoints,n.manifest = nmanifest,n.latent = nlatent,n.TDpred = 1,
    LAMBDA = matrix(c(1,0,0,0, 0,1,.8,1.3),nrow=nmanifest,ncol=nlatent),
    DRIFT=matrix(c(-.3, .2, 0, -.5),nlatent,nlatent),
    TDPREDMEANS=matrix(c(rep(0,Tpoints-10),1,rep(0,9)),ncol=1),
    TDPREDEFFECT=matrix(c(tdpredeffect[i],0),nrow=nlatent),
    DIFFUSION = matrix(c(1, 0, 0, .5),2,2),
    CINT = matrix(c(cint1[i],cint2[i]),ncol=1),
    T0VAR=diag(2,nlatent,nlatent),
    MANIFESTVAR = diag(.5, nmanifest))

  #generate data
  newdat <- ctGenerate(ctmodelobj = gm,n.subjects = 1,burnin = 2,
    dtmat<-rbind(c(rep(.5,8),3,rep(.5,Tpoints-9))))
  newdat[,'id'] <- i #set id for each subject
```

```
  newdat <- cbind(newdat,age[i]) #include time independent predictor
  if(i ==1) {
    dat <- newdat[1:(Tpoints-10),] #pre intervention data
    dat2 <- newdat #including post intervention data
  }
  if(i > 1) {
    dat <- rbind(dat, newdat[1:(Tpoints-10),])
    dat2 <- rbind(dat2,newdat)
  }
}
colnames(dat)[ncol(dat)] <- 'age'
colnames(dat2)[ncol(dat)] <- 'age'


#plot generated data for sanity
plot(age)
matplot(dat[,gm$manifestNames],type='l',pch=1)
plotvar <- 'Y1'
plot(dat[dat[,'id']==1,'time'],dat[dat[,'id']==1,plotvar],type='l',
  ylim=range(dat[,plotvar],na.rm=TRUE))
for(i in 2:nsubjects){
  points(dat[dat[,'id']==i,'time'],dat[dat[,'id']==i,plotvar],type='l',col=i)
}


dat2[,gm$manifestNames][sample(1:length(dat2[,gm$manifestNames]),size = 100)] <- NA


#data structure
head(dat2)


# Model fitting ----------------------------------------------------------

##simple univariate default model

m <- ctModel(type = 'stanct', manifestNames = c('Y1'), LAMBDA = diag(1))
ctModelLatex(m)

#Specify univariate linear growth curve

m1 <- ctModel(type = 'stanct',
  manifestNames = c('Y1'), latentNames=c('eta1'),
  DRIFT=matrix(-.0001,nrow=1,ncol=1),
  DIFFUSION=matrix(0,nrow=1,ncol=1),
  T0VAR=matrix(0,nrow=1,ncol=1),
  CINT=matrix(c('cint1'),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  LAMBDA = diag(1),
  MANIFESTMEANS=matrix(0,ncol=1),
  MANIFESTVAR=matrix(c('merror'),nrow=1,ncol=1))

ctModelLatex(m1)
```

```
#fit
f1 <- ctStanFit(datalong = dat2, ctstanmodel = m1, optimize=TRUE, priors=FALSE)

summary(f1)

#plots of individual subject models v data
ctKalman(f1,plot=TRUE,subjects=1,kalmanvec=c('y','yprior'),timestep=.01)
ctKalman(f1,plot=TRUE,subjects=1:3,kalmanvec=c('y','ysmooth'),timestep=.01,errorvec=NA)

ctStanPostPredict(f1, wait=FALSE) #compare randomly generated data from posterior to observed data

cf<-ctCheckFit(f1) #compare mean and covariance of randomly generated data to observed cov
plot(cf,wait=FALSE)

 ### Further example models

#Include intervention
m2 <- ctModel(type = 'stanct',
  manifestNames = c('Y1'), latentNames=c('eta1'),
  n.TDpred=1,TDpredNames = 'TD1', #this line includes the intervention
  TDPREDEFFECT=matrix(c('tdpredeffect'),nrow=1,ncol=1), #intervention effect
  DRIFT=matrix(-1e-5,nrow=1,ncol=1),
  DIFFUSION=matrix(0,nrow=1,ncol=1),
  CINT=matrix(c('cint1'),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  T0VAR=matrix(0,nrow=1,ncol=1),
  LAMBDA = diag(1),
  MANIFESTMEANS=matrix(0,ncol=1),
  MANIFESTVAR=matrix(c('merror'),nrow=1,ncol=1))



#Individual differences in intervention, Bayesian estimation, covariates
m2i <- ctModel(type = 'stanct',
  manifestNames = c('Y1'), latentNames=c('eta1'),
  TIpredNames = 'age',
  TDpredNames = 'TD1', #this line includes the intervention
  TDPREDEFFECT=matrix(c('tdpredeffect||TRUE'),nrow=1,ncol=1), #intervention effect
  DRIFT=matrix(-1e-5,nrow=1,ncol=1),
  DIFFUSION=matrix(0,nrow=1,ncol=1),
  CINT=matrix(c('cint1'),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  T0VAR=matrix(0,nrow=1,ncol=1),
  LAMBDA = diag(1),
  MANIFESTMEANS=matrix(0,ncol=1),
  MANIFESTVAR=matrix(c('merror'),nrow=1,ncol=1))


#Including covariate effects
m2ic <- ctModel(type = 'stanct',
  manifestNames = c('Y1'), latentNames=c('eta1'),
  n.TIpred = 1, TIpredNames = 'age',
```

```
  n.TDpred=1,TDpredNames = 'TD1', #this line includes the intervention
  TDPREDEFFECT=matrix(c('tdpredeffect'),nrow=1,ncol=1), #intervention effect
  DRIFT=matrix(-1e-5,nrow=1,ncol=1),
  DIFFUSION=matrix(0,nrow=1,ncol=1),
  CINT=matrix(c('cint1'),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  T0VAR=matrix(0,nrow=1,ncol=1),
  LAMBDA = diag(1),
  MANIFESTMEANS=matrix(0,ncol=1),
  MANIFESTVAR=matrix(c('merror'),nrow=1,ncol=1))

m2ic$pars$indvarying[m2ic$pars$matrix %in% 'TDPREDEFFECT'] <- TRUE


#Include deterministic dynamics
m3 <- ctModel(type = 'stanct',
  manifestNames = c('Y1'), latentNames=c('eta1'),
  n.TDpred=1,TDpredNames = 'TD1', #this line includes the intervention
  TDPREDEFFECT=matrix(c('tdpredeffect'),nrow=1,ncol=1), #intervention effect
  DRIFT=matrix('drift11',nrow=1,ncol=1),
  DIFFUSION=matrix(0,nrow=1,ncol=1),
  CINT=matrix(c('cint1'),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  T0VAR=matrix('t0var11',nrow=1,ncol=1),
  LAMBDA = diag(1),
  MANIFESTMEANS=matrix(0,ncol=1),
  MANIFESTVAR=matrix(c('merror1'),nrow=1,ncol=1))




#Add system noise to allow for fluctuations that persist in time
m3n <- ctModel(type = 'stanct',
  manifestNames = c('Y1'), latentNames=c('eta1'),
  n.TDpred=1,TDpredNames = 'TD1', #this line includes the intervention
  TDPREDEFFECT=matrix(c('tdpredeffect'),nrow=1,ncol=1), #intervention effect
  DRIFT=matrix('drift11',nrow=1,ncol=1),
  DIFFUSION=matrix('diffusion',nrow=1,ncol=1),
  CINT=matrix(c('cint1'),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  T0VAR=matrix('t0var11',nrow=1,ncol=1),
  LAMBDA = diag(1),
  MANIFESTMEANS=matrix(0,ncol=1),
  MANIFESTVAR=matrix(c(0),nrow=1,ncol=1))



#include 2nd latent process

m4 <- ctModel(n.manifest = 2,n.latent = 2, type = 'stanct',
  manifestNames = c('Y1','Y2'), latentNames=c('L1','L2'),
  n.TDpred=1,TDpredNames = 'TD1',
```

```
  TDPREDEFFECT=matrix(c('tdpredeffect1','tdpredeffect2'),nrow=2,ncol=1),
  DRIFT=matrix(c('drift11','drift21','drift12','drift22'),nrow=2,ncol=2),
  DIFFUSION=matrix(c('diffusion11','diffusion21',0,'diffusion22'),nrow=2,ncol=2),
  CINT=matrix(c('cint1','cint2'),nrow=2,ncol=1),
  T0MEANS=matrix(c('t0m1','t0m2'),nrow=2,ncol=1),
  T0VAR=matrix(c('t0var11','t0var21',0,'t0var22'),nrow=2,ncol=2),
  LAMBDA = matrix(c(1,0,0,1),nrow=2,ncol=2),
  MANIFESTMEANS=matrix(c(0,0),nrow=2,ncol=1),
  MANIFESTVAR=matrix(c('merror1',0,0,'merror2'),nrow=2,ncol=2))

#dynamic factor model -- fixing CINT to 0 and freeing indicator level intercepts

m3df <- ctModel(type = 'stanct',
  manifestNames = c('Y2','Y3'), latentNames=c('eta1'),
  n.TDpred=1,TDpredNames = 'TD1', #this line includes the intervention
  TDPREDEFFECT=matrix(c('tdpredeffect'),nrow=1,ncol=1), #intervention effect
  DRIFT=matrix('drift11',nrow=1,ncol=1),
  DIFFUSION=matrix('diffusion',nrow=1,ncol=1),
  CINT=matrix(c(0),ncol=1),
  T0MEANS=matrix(c('t0m1'),ncol=1),
  T0VAR=matrix('t0var11',nrow=1,ncol=1),
  LAMBDA = matrix(c(1,'Y3loading'),nrow=2,ncol=1),
  MANIFESTMEANS=matrix(c('Y2_int','Y3_int'),nrow=2,ncol=1),
  MANIFESTVAR=matrix(c('Y2residual',0,0,'Y3residual'),nrow=2,ncol=2))
```

`ctStanFitUpdate` *Update a ctStanFit object*

Description

Either to include different data, or because you have upgraded ctsem and the internal data structure has changed.

Usage

```
ctStanFitUpdate(oldfit, data = NA, recompile = FALSE, refit = FALSE, ...)
```

Arguments

`oldfit`	fit object to be upgraded
`data`	replacement long format data object
`recompile`	whether to force a recompile – safer but slower and usually unnecessary.
`refit`	if TRUE, refits the model using the old estimates as a starting point. Only applicable for optimized fits, not sampling.
`...`	extra arguments to pass to ctStanFit

Value

updated ctStanFit object.

Examples

```
newfit <- ctStanFitUpdate(ctstantestfit,refit=FALSE)
```

`ctStanGenerate` *Generate data from a ctstanmodel object*

Description

Generate data from a ctstanmodel object

Usage

```
ctStanGenerate(
  cts,
  datastruct = NA,
  is = FALSE,
  fullposterior = TRUE,
  nsamples = 200,
  parsonly = FALSE,
  cores = 2
)
```

Arguments

`cts`	`ctStanModel` , or `ctStanFit`,object.
`datastruct`	long format data structure as used by ctsem. Not used if cts is a ctStanFit object.
`is`	If optimizing, follow up with importance sampling?
`fullposterior`	Generate from the full posterior or just the (unconstrained) mean?
`nsamples`	How many samples to generate?
`parsonly`	If TRUE, only return samples of raw parameters, don't generate data.
`cores`	Number of cpu cores to use.

Value

List contining Y, and array of nsamples by data rows by manifest variables, and llrow, an array of nsamples by data rows log likelihoods.

Examples

```
#generate and plot samples from prior predictive
priorpred <- ctStanGenerate(cts = ctstantestfit,cores=2,nsamples = 50)
```

`ctStanGenerateFromFit` *Add a* `$generated` *object to ctstanfit object, with random data generated from posterior of ctstanfit object*

Description

Add a `$generated` object to ctstanfit object, with random data generated from posterior of ctstanfit object

Usage

```
ctStanGenerateFromFit(
  fit,
  nsamples = 200,
  fullposterior = FALSE,
  verboseErrors = FALSE,
  cores = 2
)
```

Arguments

`fit`	ctstanfit object
`nsamples`	Positive integer specifying number of datasets to generate.
`fullposterior`	Logical indicating whether to sample from the full posterior (original nsamples) or the posterior mean.
`verboseErrors`	if TRUE, print verbose output when errors in generation encountered.
`cores`	Number of cpu cores to use.

Value

Matrix of generated data – one dataset per iteration, according to original time and missingness structure.

Examples

```
gen <- ctStanGenerateFromFit(ctstantestfit, nsamples=3,fullposterior=TRUE,cores=1)
plot(gen$generated$Y[3,,2],type='l') #Third random data sample, 2nd manifest var, all time points.
```

`ctStanKalman` *Get Kalman filter estimates from a ctStanFit object*

Description

Get Kalman filter estimates from a ctStanFit object

Usage

```
ctStanKalman(
  fit,
  nsamples = NA,
  pointest = TRUE,
  collapsefunc = NA,
  cores = 1,
  subjects = 1:max(fit$standata$subject),
  timestep = "asdata",
  timerange = "asdata",
  standardisederrors = FALSE,
  subjectpars = TRUE,
  tformsubjectpars = TRUE,
  indvarstates = FALSE,
  removeObs = F,
  ...
)
```

Arguments

`fit`	fit object from `ctStanFit`.
`nsamples`	either NA (to extract all) or a positive integer from 1 to maximum samples in the fit.
`pointest`	If TRUE, uses the posterior mode as the single sample.
`collapsefunc`	function to apply over samples, such as `mean`
`cores`	Integer number of cpu cores to use. Only needed if savescores was set to FALSE when fitting.
`subjects`	integer vector of subjects to compute for.
`timestep`	Either a positive numeric value, 'asdata' to use the times in the dataset, or 'auto' to select a timestep automatically (resulting in some interpolation but not excessive computation).
`timerange`	only relevant if timestep is not 'asdata'. Positive numeric vector of length 2 denoting time range for computations.
`standardisederrors`	If TRUE, computes standardised errors for prior, upd, smooth conditions.
`subjectpars`	if TRUE, state estimates are not returned, instead, predictions of each subjects parameters are returned, for parameters that had random effects specified.

tformsubjectpars	if FALSE, subject level parameters are returned in raw, pre transformation form.
indvarstates	if TRUE, do not remove indvarying states from output
removeObs	Logical or integer. If TRUE, observations (but not covariates) are set to NA, so only expectations based on parameters and covariates are returned. If a positive integer N, every N observations are retained while others are set NA for computing model expectations – useful for observing prediction performance forward further in time than one observation.
...	additional arguments to collpsefunc.

Value

list containing Kalman filter elements, each element in array of iterations, data row, variables. llrow is the log likelihood for each row of data.

Examples

```
k=ctStanKalman(ctstantestfit,subjectpars=TRUE,collapsefunc=mean)
```

---

ctStanModel *Convert a frequentist (omx) ctsem model specification to Bayesian (Stan).*

Description

Convert a frequentist (omx) ctsem model specification to Bayesian (Stan).

Usage

```
ctStanModel(ctmodelobj, type = "stanct", tipredDefault = TRUE)
```

Arguments

ctmodelobj	ctsem model object of type 'omx' (default)
type	either 'stanct' for continuous time, or 'standt' for discrete time.
tipredDefault	Logical. TRUE sets any parameters with unspecified time independent predictor effects to have effects estimated, FALSE fixes the effect to zero unless individually specified.

Value

List object of class ctStanModel, with random effects specified for any intercept type parameters (T0MEANS, MANIFESTMEANS, and or CINT), and time independent predictor effects for all parameters. Adjust these after initial specification by directly editing the `pars` subobject, so `model$pars` .

Examples

```
model <- ctModel(type='omx', Tpoints=50,
n.latent=2, n.manifest=1,
manifestNames='sunspots',
latentNames=c('ss_level', 'ss_velocity'),
LAMBDA=matrix(c( 1, 'ma1' ), nrow=1, ncol=2),
DRIFT=matrix(c(0, 1,   'a21', 'a22'), nrow=2, ncol=2, byrow=TRUE),
MANIFESTMEANS=matrix(c('m1'), nrow=1, ncol=1),
# MANIFESTVAR=matrix(0, nrow=1, ncol=1),
CINT=matrix(c(0, 0), nrow=2, ncol=1),
DIFFUSION=matrix(c(
  0, 0,
  0, "diffusion"), ncol=2, nrow=2, byrow=TRUE))

stanmodel=ctStanModel(model)
```

`ctStanParnames` *ctStanParnames*

Description

Gets internal stan parameter names of a ctStanFit object sampled via stan based on specified substrings.

Usage

```
ctStanParnames(x, substrings = c("pop_", "popsd"))
```

Arguments

`x`	ctStanFit object
`substrings`	vector of character strings, parameter names of the stan model containing any of these strings will be returned. Useful strings may be 'pop_' for population means, 'popsd' for population standard deviations, or specific combinations such as 'pop_DRIFT' for the population means of temporal dynamics parameters

Value

vector of character strings.

Examples

```
sunspots<-sunspot.year
sunspots<-sunspots[50: (length(sunspots) - (1988-1924))]
id <- 1
time <- 1749:1924
datalong <- cbind(id, time, sunspots)

#setup model
ssmodel <- ctModel(type='stanct', n.latent=2, n.manifest=1,
 manifestNames='sunspots',
 latentNames=c('ss_level', 'ss_velocity'),
 LAMBDA=matrix(c( 1, 'ma1| log(1+(exp(param)))' ), nrow=1, ncol=2),
 DRIFT=matrix(c(0, 'a21 | -log(1+exp(param))', 1, 'a22'), nrow=2, ncol=2),
 MANIFESTMEANS=matrix(c('m1|param * 10 + 44'), nrow=1, ncol=1),
 MANIFESTVAR=diag(0,1), #As per original spec
 CINT=matrix(c(0, 0), nrow=2, ncol=1),
 DIFFUSION=matrix(c(0, 0, 0, "diffusion"), ncol=2, nrow=2))

#fit
ssfit <- ctStanFit(datalong, ssmodel, iter=2,
  optimize=FALSE, chains=1)
ctStanParnames(ssfit,substrings=c('pop_','popsd'))
```

---

ctStanPlotPost *ctStanPlotPost*

Description

Plots prior and posterior distributions of model parameters in a ctStanModel or ctStanFit object.

Usage

```
ctStanPlotPost(
  obj,
  rows = "all",
  npp = 6,
  priorwidth = TRUE,
  smoothness = 1,
  priorsamples = 10000,
  plot = TRUE,
  wait = FALSE,
  ...
)
```

Arguments

obj	fit or model object as generated by ctStanFit, ctModel, or ctStanModel.
rows	vector of integers denoting which rows of obj$setup$popsetup to plot priors for. Character string 'all' plots all rows with parameters to be estimated.
npp	Integer number of parameters to show per page.
priorwidth	if TRUE, plots will be scaled to show bulk of both the prior and posterior distributions. If FALSE, scale is based only on the posterior.
smoothness	Positive numeric – multiplier to modify smoothness of density plots, higher is smoother but can cause plots to exceed natural boundaries, such as standard deviations below zero.
priorsamples	number of samples from prior to use. More is slower.
plot	Logical, if FALSE, ggplot objects are returned in a list instead of plotting.
wait	If true, user is prompted to continue before plotting next graph. If false, graphs are plotted one after another without waiting.
...	Parameters to pass to ctStanFit. cores = x will speed things up, where x is the number of cpu cores to use.

Examples

```
ctStanPlotPost(ctstantestfit, rows=3:4)
```

---

ctStanPostPredict *Compares model implied density and values to observed, for a ctStanFit object.*

Description

Compares model implied density and values to observed, for a ctStanFit object.

Usage

```
ctStanPostPredict(
  fit,
  diffsize = 1,
  jitter = 0.02,
  wait = TRUE,
  probs = c(0.025, 0.5, 0.975),
  datarows = "all",
  nsamples = 500,
  resolution = 100,
  plot = TRUE
)
```

Arguments

fit	ctStanFit object.
diffsize	Integer > 0. Number of discrete time lags to use for data viz.
jitter	Positive numeric between 0 and 1, if TRUE, jitters empirical data by specified proportion of std dev.
wait	Logical, if TRUE and plot=TRUE, waits for input before plotting next plot.
probs	Vector of length 3 containing quantiles to plot – should be rising numeric values between 0 and 1.
datarows	integer vector specifying rows of data to plot. Otherwise 'all' uses all data.
nsamples	Number of datasets to generate for comparisons, if fit object does not contain generated data already.
resolution	Positive integer, the number of rows and columns to split plots into for shading.
plot	logical. If FALSE, a list of ggplot objects is returned.

Details

This function relies on the data generated during each iteration of fitting to approximate the model implied distributions – thus, when limited iterations are available, the approximation will be worse.

Value

If plot=FALSE, an array containing quantiles of generated data. If plot=TRUE, nothing, only plots.

if plot=TRUE, nothing is returned and plots are created. Otherwise, a list containing ggplot objects is returned and may be customized as desired.

Examples

```
#'
ctStanPostPredict(ctstantestfit,wait=FALSE, diffsize=2,resolution=100)
```

---

ctStanSubjectPars *Extract an array of subject specific parameters from a ctStanFit object.*

Description

Extract an array of subject specific parameters from a ctStanFit object.

Usage

```
ctStanSubjectPars(fit, pointest = TRUE, cores = 2, nsamples = "all")
```

Arguments

fit	fit object
pointest	if TRUE, returns only the set of individual difference parameters based on the max a posteriori estimate (or the median if sampling approaches were used).
cores	Number of cores to use.
nsamples	Number of samples to calculate parameters for. Not used if pointest=TRUE.

Details

This function returns the estimates of individual parameters, taking into account any covariates and random effects.

Value

an nsamples by nsubjects by nparams array.

Examples

```
indpars <- ctStanSubjectPars(ctstantestfit)
dimnames(indpars)
plot(indpars[1,,'cint1'],indpars[1,,'cint2'])
```

---

ctstantestdat *ctstantestdat*

Description

Generated dataset for testing ctStanFit from ctsem package.

Format

matrix

---

ctstantestfit *ctstantestfit*

Description

Dummy fit for testing functions from ctsem package.

Format

ctStanFit object

`ctStanTIpredeffects` *Get time independent predictor effect estimates*

Description

Computes and plots combined effects and quantiles for effects of time independent predictors on subject level parameters of a ctStanFit object.

Usage

```
ctStanTIpredeffects(
  fit,
  returndifference = FALSE,
  probs = c(0.025, 0.5, 0.975),
  includeMeanUncertainty = FALSE,
  whichTIpreds = 1,
  parmatrices = TRUE,
  whichpars = "all",
  nsamples = 100,
  timeinterval = 1,
  nsubjects = 20,
  filter = NA,
  plot = FALSE
)
```

Arguments

`fit`	fit object from `ctStanFit`
`returndifference`	logical. If FALSE, absolute parameter values are returned. If TRUE, only the effect of the covariate (i.e. without the average value of the parameter) are returned. The former can be easier to interpret, but the latter are more likely to fit multiple plots together. Not used if `parmatrices=TRUE`.
`probs`	numeric vector of quantile probabilities from 0 to 1. Specify 3 values if plotting, the 2nd will be drawn as a line with uncertainty polygon based on 1st and 3rd.
`includeMeanUncertainty`	if TRUE, output includes sampling variation in the mean parameters. If FALSE, mean parameters are fixed at their median, only uncertainty in time independent predictor effects is included.
`whichTIpreds`	integer vector specifying which of the tipreds in the fit object you want to use to calculate effects. Unless quadratic / higher order versions of predictors have been included, selecting more than one probably doesn't make sense. If for instance a squared predictor has been included, then you can specify both the linear and squared version. The x axis of the plot (if generated) will be based off the first indexed predictor. To check what predictors are in the model, run `fit$ctstanmodel$TIpredNames`.

parmatrices	Logical. If TRUE (default), system matrices rather than specific parameters are referenced – e.g. 'DRIFT' instead of a parameter name like drift12.
whichpars	if parmatrices==TRUE, character vector specifying which matrices, and potentially which indices of the matrices, to plot. c('dtDRIFT[2,1]', 'DRIFT') would output for row 2 and column 1 of the discrete time drift matrix, as well as all indices of the continuous time drift matrix. If parmatrices==FALSE, integer vector specifying which of the subject level parameters to compute effects on. The integers corresponding to certain parameters can be found in the `param` column of the `fit$setup$matsetup` object. In either case 'all' uses all available parameters.
nsamples	Positive integer specifying the maximum number of saved iterations to use. Character string 'all' can also be used.
timeinterval	positive numeric indicating time interval to use for discrete time parameter matrices, if `parmatrices=TRUE`.
nsubjects	Positive integer specifying the number of subjects to compute values for. When only one TIpred is used, this specifies the number of points along the curve. Character string 'all' can also be used. Time taken for plotting is a function of nsubjects*niterations.
filter	either NA, or a length 2 vector, where the first element contains the time independent predictor index to filter by, and the second contains the comparison operator in string form (e.g. "< 3", to only calculate effects for subjects where the tipreds of the denoted index are less than 3).
plot	Logical. If TRUE, nothing is returned but instead `ctPlotArray` is used to plot the output instead.

Value

Either a three dimensional array of predictor effects, or nothing with a plot generated.

Examples

```
ctStanTIpredeffects(ctstantestfit,
 whichpars=c('CINT','dtDIFFUSION[2,2]'), plot=TRUE)
```

`ctStanUpdModel` *Update an already compiled and fit ctStanFit object*

Description

Allows one to change data and or model elements that don't require recompiling, then re fit.

Usage

```
ctStanUpdModel(fit, datalong, ctstanmodel, ...)
```

Arguments

fit	ctStanFit object
datalong	data as normally passed to ctStanFit
ctstanmodel	model as normally passed to ctStanFit
...	extra args for ctStanFit

---

ctWideNames *ctWideNames sets default column names for wide ctsem datasets. Primarily intended for internal ctsem usage.*

Description

ctWideNames sets default column names for wide ctsem datasets. Primarily intended for internal ctsem usage.

Usage

```
ctWideNames(
  n.manifest,
  Tpoints,
  n.TDpred = 0,
  n.TIpred = 0,
  manifestNames = "auto",
  TDpredNames = "auto",
  TIpredNames = "auto"
)
```

Arguments

n.manifest	number of manifest variables per time point in the data.
Tpoints	Maximum number of discrete time points (waves of data, or measurement occasions) for an individual in the input data structure.
n.TDpred	number of time dependent predictors in the data structure.
n.TIpred	number of time independent predictors in the data structure.
manifestNames	vector of character strings giving column names of manifest indicator variables
TDpredNames	vector of character strings giving column names of time dependent predictor variables
TIpredNames	vector of character strings giving column names of time independent predictor variables

ctWideToLong *ctWideToLong Convert ctsem wide to long format*

Description

ctWideToLong Convert ctsem wide to long format

Usage

```
ctWideToLong(
  datawide,
  Tpoints,
  n.manifest,
  n.TDpred = 0,
  n.TIpred = 0,
  manifestNames = "auto",
  TDpredNames = "auto",
  TIpredNames = "auto"
)
```

Arguments

datawide	ctsem wide format data
Tpoints	number of measurement occasions in data
n.manifest	number of manifest variables
n.TDpred	number of time dependent predictors
n.TIpred	number of time independent predictors
manifestNames	Character vector of manifest variable names.
TDpredNames	Character vector of time dependent predictor names.
TIpredNames	Character vector of time independent predictor names.

Details

Names must account for *all* the columns in the data - i.e. do not leave certain variables out just because you do not need them.

Examples

```
#create wide data
wideexample <- ctLongToWide(datalong = ctstantestdat, id = "id",
time = "time", manifestNames = c("Y1", "Y2"),
TDpredNames = "TD1", TIpredNames = c("TI1", "TI2","TI3"))

wide <- ctIntervalise(datawide = wideexample, Tpoints = 10, n.manifest = 2,
n.TDpred = 1, n.TIpred = 3, manifestNames = c("Y1", "Y2"),
TDpredNames = "TD1", TIpredNames = c("TI1", "TI2","TI3") )
```

```
#Then convert to long format
longexample <- ctWideToLong(datawide = wideexample, Tpoints=10,
n.manifest=2, manifestNames = c("Y1", "Y2"),
n.TDpred=1, TDpredNames = "TD1",
n.TIpred=3, TIpredNames = c("TI1", "TI2","TI3"))

#Then convert the time intervals to absolute time
long <- ctDeintervalise(datalong = longexample, id='id', dT='dT')
head(long,22)
```

`datastructure` *datastructure*

Description

Simulated example dataset for the ctsem package

Format

2 by 15 matrix containing containing ctsem wide format data. 3 measurement occasions of manifest variables Y1 and Y2, 2 measurement occasions of time dependent predictor TD1, 2 measurement intervals dTx, and 2 time independent predictors TI1 and TI2, for 2 individuals.

`inv_logit` *Inverse logit*

Description

Maps the stan function so the same code works in R.

Usage

```
inv_logit(x)
```

Arguments

`x`	value to calculate the inverse logit for.

Examples

```
inv_logit(-3)
```

isdiag *Diagnostics for ctsem importance sampling*

Description

Diagnostics for ctsem importance sampling

Usage

```
isdiag(fit)
```

Arguments

fit	Output from ctStanFit when optimize=TRUE and isloops > 0

Value

Nothing. Plots convergence of parameter mean estimates from initial Hessian based distribution to final sampling distribution.

Examples

```
#get data
sunspots<-sunspot.year
sunspots<-sunspots[50: (length(sunspots) - (1988-1924))]
id <- 1
time <- 1749:1924
datalong <- cbind(id, time, sunspots)

#setup model
model <- ctModel(type='stanct',
 manifestNames='sunspots',
 latentNames=c('ss_level', 'ss_velocity'),
  LAMBDA=matrix(c( -1, 'ma1 | log(exp(-param)+1)' ), nrow=1, ncol=2),
  DRIFT=matrix(c(0, 'a21', 1, 'a22'), nrow=2, ncol=2),
  MANIFESTMEANS=matrix(c('m1 | (param)*5+44'), nrow=1, ncol=1),
  CINT=matrix(c(0, 0), nrow=2, ncol=1),
  T0VAR=matrix(c(1,0,0,1), nrow=2, ncol=2), #Because single subject
  DIFFUSION=matrix(c(0.0001, 0, 0, "diffusion"), ncol=2, nrow=2))

#fit and plot importance sampling diagnostic
fit <- ctStanFit(datalong, model,verbose=0,
  optimcontrol=list(is=TRUE, finishsamples=500),priors=TRUE)
isdiag(fit)
```

log1p_exp *log1p_exp*

Description

Maps the stan function so the same code works in R.

Usage

```
log1p_exp(x)
```

Arguments

x	value to use.

Examples

```
log1p_exp(-3)
```

longexample *longexample*

Description

Simulated example dataset for the ctsem package

Format

7 by 8 matrix containing ctsem long format data, for two subjects, with three manifest variables Y1, Y2, Y3, one time dependent predictor TD1, two time independent predictors TI1 and TI2, and absolute timing information Time.

Oscillating *Oscillating*

Description

Simulated example dataset for the ctsem package.

Format

200 by 21 matrix containing containing ctsem wide format data. 11 measurement occasions and 10 measurement intervals for each of 200 individuals

Source

See https://bpspsychub.onlinelibrary.wiley.com/doi/10.1111/j.2044-8317.2012.02043.x

plot.ctKalmanDF *Plots Kalman filter output from ctKalman.*

Description

Plots Kalman filter output from ctKalman.

Usage

```
## S3 method for class 'ctKalmanDF'
plot(
  x,
  subjects = unique(x$Subject),
  kalmanvec = c("y", "yprior"),
  errorvec = "auto",
  errormultiply = 1.96,
  plot = TRUE,
  elementNames = NA,
  polygonsteps = 10,
  polygonalpha = 0.1,
  facets = vars(Variable),
  ...
)
```

Arguments

x	Output from ctKalman. In general it is easier to call ctKalman directly with the plot=TRUE argument, which calls this function.
subjects	vector of integers denoting which subjects (from 1 to N) to plot predictions for.
kalmanvec	string vector of names of any elements of the output you wish to plot, the defaults of 'y' and 'ysmooth' plot the original data, 'y', and the estimates of the 'true' value of y given all data. Replacing 'y' by 'eta' will plot latent states instead (though 'eta' alone does not exist) and replacing 'smooth' with 'upd' or 'prior' respectively plots updated (conditional on all data up to current time point) or prior (conditional on all previous data) estimates.
errorvec	vector of names indicating which kalmanvec elements to plot uncertainty bands for. 'auto' plots all possible.
errormultiply	Numeric denoting the multiplication factor of the std deviation of errorvec objects. Defaults to 1.96, for 95% intervals.
plot	if FALSE, plots are not generated and the ggplot object is simply returned invisibly.
elementNames	if NA, all relevant object elements are included – e.g. if yprior is in the kalmanvec argument, all manifest variables are plotted, and likewise for latent states if etasmooth was specified. Alternatively, a character vector specifying the manifest and latent names to plot explicitly can be specified.
polygonsteps	Number of steps to use for uncertainty band shading.
polygonalpha	Numeric for the opacity of the uncertainty region.
facets	when multiple subjects are included in multivariate plots, the default is to facet plots by variable type. This can be set to NA for no facets, or ggplot2::vars(Subject) for facetting by subject.
...	not used.

Value

A ggplot2 object. Side effect – Generates plots.

Examples

```
### Get output from ctKalman
x<-ctKalman(ctstantestfit,subjects=2,timestep=.01)

### Plot with plot.ctKalmanDF
plot(x, subjects=2)

###Single step procedure:
ctKalman(ctstantestfit,subjects=2,
  kalmanvec=c('y','yprior'),
  elementNames=c('Y1','Y2'),
  plot=TRUE,timestep=.01)
```

plot.ctStanFit *plot.ctStanFit*

Description

Plots for ctStanFit objects

Usage

```
## S3 method for class 'ctStanFit'
plot(x, types = "all", wait = TRUE, ...)
```

Arguments

x	Fit object from ctStanFit.
types	Vector of character strings defining which plots to create. 'all' plots all possible types, including: 'regression', 'kalman', 'priorcheck', 'trace', 'density','intervals'.
wait	Logical. Pause between plots?
...	Arguments to pass through to the specific plot functions. Bewar of clashes may occur if types='all'. For details see the specific functions generating each type of plot.

Details

This function is just a wrapper calling the necessary functions for plotting - it may be simpler in many cases to access those directly. They are: ctStanDiscretePars,ctKalman, ctStanPlotPost,stan_trace, stan_dens,stan_plot rstan offers many plotting possibilities not available here, to use that functionality one must simply call the relevant rstan plotting function. Use x$stanfit as the stan fit object (where x is the name of your ctStanFit object). Because a ctStanFit object has many parameters, the additional argument pars=ctStanParnames(x,'pop_') is recommended. This denotes population means, but see ctStanParnames for other options.

Value

Nothing. Generates plots.

Examples

```
plot(ctstantestfit,types=c('regression','kalman','priorcheck'), wait=FALSE)
```

`plot.ctStanModel` *Prior plotting*

Description

Plots priors for free model parameters in a ctStanModel.

Usage

```
## S3 method for class 'ctStanModel'
plot(
  x,
  rows = "all",
  wait = FALSE,
  nsamples = 1e+06,
  rawpopsd = "marginalise",
  inddifdevs = c(-1, 1),
  inddifsd = 0.1,
  plot = TRUE,
  ...
)
```

Arguments

`x`	ctStanModel object as generated by `ctModel` with type='stanct' or 'standt'.
`rows`	vector of integers denoting which rows of ctstanmodel$pars to plot priors for. Character string 'all' plots all rows with parameters to be estimated.
`wait`	If true, user is prompted to continue before plotting next graph.
`nsamples`	Numeric. Higher values increase fidelity (smoothness / accuracy) of density plots, at cost of speed.
`rawpopsd`	Either 'marginalise' to sample from the specified (in the ctstanmodel) prior distribution for the raw population standard deviation, or a numeric value to use for the raw population standard deviation for all subject level prior plots - the plots in dotted blue or red.
`inddifdevs`	numeric vector of length 2, setting the means for the individual differences distributions.
`inddifsd`	numeric, setting the standard deviation of the population means used to generate individual difference distributions.
`plot`	If FALSE, ouputs list of GGplot objects that can be further modified.
`...`	not used.

Details

Plotted in black is the prior for the population mean. In red and blue are the subject level priors that result given that the population mean is estimated as 1 std deviation above the mean of the prior, or 1 std deviation below. The distributions around these two points are then obtained by marginalising over the prior for the raw population std deviation - so the red and blue distributions do not represent any specific subject level prior, but rather characterise the general amount and shape of possible subject level priors at the specific points of the population mean prior.

Examples

```
model <- ctModel(type='stanct',
manifestNames='sunspots',
latentNames=c('ss_level', 'ss_velocity'),
LAMBDA=matrix(c( 1, 'ma1' ), nrow=1, ncol=2),
DRIFT=matrix(c(0, 1,   'a21', 'a22'), nrow=2, ncol=2, byrow=TRUE),
MANIFESTMEANS=matrix(c('m1'), nrow=1, ncol=1),
# MANIFESTVAR=matrix(0, nrow=1, ncol=1),
CINT=matrix(c(0, 0), nrow=2, ncol=1),
DIFFUSION=matrix(c(
  0, 0,
  0, "diffusion"), ncol=2, nrow=2, byrow=TRUE))

plot(model,rows=8)
```

---

sdpcor2cov *sdcor2cov*

Description

Converts a lower triangular matrix with standard deviations on the diagonal and partial correlations on lower triangle, to a covariance (or cholesky decomposed covariance)

Usage

```
sdpcor2cov(mat, coronly = FALSE, cholesky = FALSE)
```

Arguments

mat	input square matrix with std dev on diagonal and lower tri of partial correlations.
coronly	if TRUE, ignores everything except the lower triangle and outputs correlation.
cholesky	Logical. To return the cholesky decomposition instead of full covariance, set to TRUE.

Examples

```
testmat <- diag(exp(rnorm(5,-3,2)),5) #generate arbitrary std deviations
testmat[row(testmat) > col(testmat)] <- runif((5^2-5)/2, -1, 1)
print(testmat)
covmat <- sdpcor2cov(testmat) #convert to covariance
cov2cor(covmat) #convert covariance to correlation
```

---

standatact_specificsubjects *Adjust standata from ctsem to only use specific subjects*

Description

Adjust standata from ctsem to only use specific subjects

Usage

```
standatact_specificsubjects(standata, subjects, timestep = NA)
```

Arguments

standata	standata
subjects	vector of subjects
timestep	ignored at present

Value

list of updated structure

Examples

```
d <- standatact_specificsubjects(ctstantestfit$standata, 1:2)
```

---

stanoptimis *Optimize / importance sample a stan or ctStan model.*

Description

Optimize / importance sample a stan or ctStan model.

Usage

```
stanoptimis(
  standata,
  sm,
  init = "random",
  initsd = 0.01,
  sampleinit = NA,
  deoptim = FALSE,
  estonly = FALSE,
  tol = 1e-08,
  decontrol = list(),
  stochastic = TRUE,
  priors = TRUE,
  carefulfit = TRUE,
  bootstrapUncertainty = FALSE,
  subsamplesize = 1,
  parsteps = c(),
  plot = FALSE,
  hessianType = "numerical",
  stochasticHessianSamples = 50,
  stochasticHessianEpsilon = 1e-05,
  is = FALSE,
  isloopsize = 1000,
  finishsamples = 1000,
  tdf = 10,
  chancethreshold = 100,
  finishmultiply = 5,
  verbose = 0,
  cores = 2,
  matsetup = NA,
  nsubsets = 100,
  stochasticTolAdjust = 1000
)
```

Arguments

standata	list object conforming to rstan data standards.
sm	compiled stan model object.
init	vector of unconstrained parameter values, or character string 'random' to initialise with random values very close to zero.
initsd	positive numeric specifying sd of normal distribution governing random sample of init parameters, if init='random' .
sampleinit	either NA, or an niterations * nparams matrix of samples to initialise importance sampling.
deoptim	Do first pass optimization using differential evolution? Slower, but better for cases with multiple minima / difficult optimization.
estonly	if TRUE,just return point estimates under $rawest subobject.

`tol` objective tolerance.

`decontrol` List of control parameters for differential evolution step, to pass to `DEoptim.control`.

`stochastic` Logical. Use stochastic gradient descent instead of mize (bfgs) optimizer. Still experimental, worth trying for either robustness checks or problematic, high dimensional, nonlinear, problems.

`priors` logical. If TRUE, a priors integer is set to 1 (TRUE) in the standata object – only has an effect if the stan model uses this value.

`carefulfit` Logical. If TRUE, priors are always used for a rough first pass to obtain starting values when priors=FALSE

`bootstrapUncertainty` Logical. If TRUE, subject wise gradient contributions are resampled to estimate the hessian, for computing standard errors or initializing importance sampling.

`subsamplesize` value between 0 and 1 representing proportion of subjects to include in first pass fit.

`parsteps` ordered list of vectors of integers denoting which parameters should begin fixed at zero, and freed sequentially (by list order). Useful for complex models, e.g. keep all cross couplings fixed to zero as a first step, free them in second step.

`plot` Logical. If TRUE, plot iteration details. Probably slower.

`hessianType` either 'numerical' or 'stochastic', the latter is experimental at present.

`stochasticHessianSamples` number of samples to use for stochastic Hessian, if selected.

`stochasticHessianEpsilon` SD of random samples for stochastic hessian, if selected.

`is` Logical. Use importance sampling, or just return map estimates?

`isloopsize` Number of samples of approximating distribution per iteration of importance sampling.

`finishsamples` Number of samples to draw (either from hessian based covariance or posterior distribution) for final results computation.

`tdf` degrees of freedom of multivariate t distribution. Higher (more normal) generally gives more efficent importance sampling, at risk of truncating tails.

`chancethreshold` drop iterations of importance sampling where any samples are chancethreshold times more likely to be drawn than expected.

`finishmultiply` Importance sampling stops once available samples reach `finishsamples * finishmultiply` , then the final samples are drawn without replacement from this set.

`verbose` Integer from 0 to 2. Higher values print more information during model fit – for debugging.

`cores` Number of cpu cores to use, should be at least 2.

`matsetup` subobject of ctStanFit output. If provided, parameter names instead of numbers are output for any problem indications.

`nsubsets` number of subsets for stochastic optimizer. Subsets are further split across cores, but each subjects data remains whole – processed by one core in one subset.

`stochasticTolAdjust` Multiplier for stochastic optimizer tolerance.

Value

list containing fit elementsF

`stanWplot` *Runs stan, and plots sampling information while sampling.*

Description

Runs stan, and plots sampling information while sampling.

Usage

```
stanWplot(object, iter = 2000, chains = 4, ...)
```

Arguments

`object`	stan model object
`iter`	Number of iterations
`chains`	Number of chains
`...`	All the other regular arguments to stan()

Details

On windows, requires Rtools installed and able to be found by pkgbuild::rtools_path()

Examples

```
library(rstan)
#### example 1
scode <- "
parameters {
  real y[2];
}
model {
  y[1] ~ normal(0, .5);
  y[2] ~ double_exponential(0, 2);
}
"
#Uncomment the following lines -- launches rscript not compatible with cran check.
#sm <- stan_model(model_code = scode)
#fit1 <- stanWplot(object = sm,iter = 100000,chains=2,cores=1)
```

stan_checkdivergences *Analyse divergences in a stanfit object*

Description

Analyse divergences in a stanfit object

Usage

```
stan_checkdivergences(sf, nupars = "all")
```

Arguments

sf	stanfit object.
nupars	either the string 'all', or an integer reflecting how many pars (from first to nupars) to use.

Value

A list of four matrices. $locationsort and $sdsort contian the bivariate interactions of unconstrained parameters, sorted by either the relative location of any divergences, or the relative standard deviation. $locationmeans and $sdmeans collapse across the bivariate interactions to return the means for each parameter.

Examples

```
sunspots<-sunspot.year
sunspots<-sunspots[50: (length(sunspots) - (1988-1924))]
id <- 1
time <- 1749:1924
datalong <- cbind(id, time, sunspots)

#setup model
ssmodel <- ctModel(type='stanct', n.latent=2, n.manifest=1,
 manifestNames='sunspots',
 latentNames=c('ss_level', 'ss_velocity'),
 LAMBDA=matrix(c( 1, 'ma1| log(1+(exp(param)))' ), nrow=1, ncol=2),
 DRIFT=matrix(c(0, 'a21 | -log(1+exp(param))', 1, 'a22'), nrow=2, ncol=2),
 MANIFESTMEANS=matrix(c('m1|param * 10 + 44'), nrow=1, ncol=1),
 MANIFESTVAR=diag(0,1), #As per original spec
 CINT=matrix(c(0, 0), nrow=2, ncol=1),
 DIFFUSION=matrix(c(0, 0, 0, "diffusion"), ncol=2, nrow=2))

#fit
ssfit <- ctStanFit(datalong, ssmodel, iter=2,
  optimize=FALSE, chains=1)

stan_checkdivergences(ssfit$stanfit$stanfit) #stan object
```

stan_reinitsf *Quickly initialise stanfit object from model and data*

Description

Quickly initialise stanfit object from model and data

Usage

```
stan_reinitsf(model, data, fast = FALSE)
```

Arguments

model	stanmodel
data	standata
fast	Use cut down form for speed

Value

stanfit object

Examples

```
sf <- stan_reinitsf(ctstantestfit$stanmodel,ctstantestfit$standata)
```

stan_unconstrainsamples *Convert samples from a stanfit object to the unconstrained scale*

Description

Convert samples from a stanfit object to the unconstrained scale

Usage

```
stan_unconstrainsamples(fit, standata = NA)
```

Arguments

fit	stanfit object.
standata	only necessary if R session has been restarted since fitting model – used to reinitialize stanfit object.

Value

Matrix containing columns of unconstrained parameters for each post-warmup iteration.

Examples

```
#get data
sunspots<-sunspot.year
sunspots<-sunspots[50: (length(sunspots) - (1988-1924))]
id <- 1
time <- 1749:1924
datalong <- cbind(id, time, sunspots)

#setup model
ssmodel <- ctModel(type='stanct', n.latent=2, n.manifest=1,
  manifestNames='sunspots',
  latentNames=c('ss_level', 'ss_velocity'),
  LAMBDA=matrix(c( 1, 'ma1| log(1+(exp(param)))' ), nrow=1, ncol=2),
  DRIFT=matrix(c(0, 'a21 | -log(1+exp(param))', 1, 'a22'), nrow=2, ncol=2),
  MANIFESTMEANS=matrix(c('m1|param * 10 + 44'), nrow=1, ncol=1),
  MANIFESTVAR=diag(0,1), #As per original spec
  CINT=matrix(c(0, 0), nrow=2, ncol=1),
  DIFFUSION=matrix(c(0, 0, 0, "diffusion"), ncol=2, nrow=2))

#fit
ssfit <- ctStanFit(datalong, ssmodel,
  iter=200, chains=2,optimize=FALSE, priors=TRUE,control=list(max_treedepth=4))
umat <- stan_unconstrainsamples(ssfit$stanfit$stanfit)
```

`summary.ctStanFit` *summary.ctStanFit*

Description

Summarise a ctStanFit object that was fit using `ctStanFit`.

Usage

```
## S3 method for class 'ctStanFit'
summary(
  object,
  timeinterval = 1,
  digits = 4,
  parmatrices = TRUE,
  priorcheck = TRUE,
  residualcov = TRUE,
  ...
)
```

Arguments

object	fit object from ctStanFit, of class ctStanFit.
timeinterval	positive numeric indicating time interval to use for discrete time parameter calculations reported in summary.
digits	integer denoting number of digits to report.
parmatrices	if TRUE, also return additional parameter matrices – can be slow to compute for large models with many samples.
priorcheck	Whether or not to use ctsem:::priorchecking to compare posterior mean and sd to prior mean and sd.
residualcov	Whether or not to show standardised residual covariance. Takes a little longer to compute.
...	Additional arguments to pass to ctsem:::priorcheckreport, such as meanlim, or sdlim.

Value

List containing summary items.

Examples

```
summary(ctstantestfit)
```

Index