Package ‘ctsem’

July 31, 2019

Type Package

Title Continuous Time Structural Equation Modelling

Version 3.0.0

Date 2019-7-28

Description Hierarchical continuous 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), measurement models are typically multivariate normal factor models.
Using the original ctsem formulation based on OpenMx, described in the JSS paper
``Continuous Time Structural Equation Modeling with R Package ctsem'', with updated version
as CRAN vignette <https://cran.r-project.org/web/packages/ctsem/vignettes/ctsem.pdf>,
linear mixed effects SDE’s estimated via maximum likelihood and optimization are possible.
Using the Stan based formulation, described in <https://github.com/cdriveraus/ctsem/raw/master/vignettes/hierarchicalmanual.pdf>,
nonlinearity (state dependent parameters) and random effects on all parameters
are possible, using either optimization (with optional importance sampling) or
Stan’s Hamiltonian Monte Carlo sampling. 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.

License GPL-3

Depends R (>= 3.5.0), Rcpp (>= 0.12.16), OpenMx (>= 2.9.0)

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

Imports rstan (>= 2.19.0), rstantools (>= 1.5.0), plyr, tools, data.table, Matrix, datasets, stats, graphics, grDevices, parallel, MASS, methods, utils, ggplot2, mvtnorm, mize, pkgbuild, cOde, Deriv

Encoding UTF-8

LazyData true

ByteCompile true
R topics documented:

LinkingTo StanHeaders (&ge; 2.17.0), rstan (&ge; 2.17.1), BH (&ge; 1.66.0-1), Rcpp (&ge; 0.12.16), RcppEigen (&ge; 0.3.3.4.0)

SystemRequirements GNU make

NeedsCompilation yes

Suggests knitr, testthat, devtools, DEoptim, GGally, shiny

VignetteBuilder knitr

RoxygenNote 6.1.1

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

Maintainer Charles Driver <driver@mpib-berlin.mpg.de>

Repository CRAN

Date/Publication 2019-07-31 12:40:07 UTC

R topics documented:

  AnomAuth .......................... 2
  ctCheckFit ................................ 3
  ctCI .................................... 4
  ctCollapse ................................ 5
  ctCompareExpected ..................... 5
  ctDeintervalise ...................... 6
  ctDensity ............................. 7
  ctDiscretePars ....................... 7
  ctDiscretiseData ..................... 8
  ctExample1 ............................ 9
  ctExample1ITpred ...................... 9
  ctExample2 ............................ 9
  ctExample2level ...................... 10
  ctExample3 ............................ 10
  ctExample4 ............................ 10
  ctFit .................................. 11
  ctGenerate ............................ 14
  ctGenerateFromFit ..................... 16
  ctIndplot ............................. 17
  ctIntervalise .......................... 18
  ctKalman .............................. 19
  ctKalmanPlot .......................... 20
  ctLongToWide ........................... 22
  ctModel .............................. 23
  ctModelFromFit ....................... 27
  ctModelLatex .......................... 28
  ctMultigroupFit ...................... 29
  ctPlot ................................ 30
R topics documented:

- ctPlotArray .................................................. 32
- ctPoly .......................................................... 33
- ctPostPredict .................................................. 34
- ctRefineTo ..................................................... 35
- ctsen ........................................................... 35
- ctStanContinuousPars ....................................... 36
- ctStanDiscretePars ........................................... 37
- ctStanDiscreteParsPlot ..................................... 38
- ctStanFit ....................................................... 39
- ctStanGenerateFromFit ....................................... 47
- ctStanKalman ................................................... 48
- ctStanModel .................................................... 49
- ctStanParMatrices ........................................... 50
- ctStanParnames ............................................... 51
- ctStanPlotPost ................................................ 51
- ctStanPostPredict ........................................... 52
- ctstantestdat .................................................. 53
- ctstantestfit ................................................... 53
- ctStanTIpredEffects ......................................... 54
- ctStanTIpredMarginal ....................................... 55
- ctStanUpdModel ............................................... 56
- ctWideNames ................................................... 57
- ctWideToLong .................................................. 58
- datastructure ................................................... 59
- extract ......................................................... 59
- inv_logit ....................................................... 60
- isdiag .......................................................... 60
- Kalman .......................................................... 61
- longexample ................................................... 63
- msquare ........................................................ 63
- Oscillating ..................................................... 64
- plot.ctsemFit .................................................. 64
- plot.ctsemFitMeasure ....................................... 66
- plot.ctsemMultigroupFit ................................... 67
- plot.ctStanFit .................................................. 67
- plot.ctStanModel ............................................. 68
- sdpcor2cov ..................................................... 70
- stanoptimis ..................................................... 70
- stanWplot ....................................................... 72
- stan_checkdivergences ..................................... 73
- stan_confidenceRegion ..................................... 74
- stan_postcalc .................................................. 75
- stan_unconstrainsamples .................................. 76
- summary.ctsemFit ............................................ 76
- summary.ctsemMultigroupFit ............................... 78
- summary.ctStanFit .......................................... 78
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

ctCheckFit

Check absolute fit of ctFit or ctStanFit object.

Usage
ctCheckFit(fit, niter = 500, probs = c(0.025, 0.5, 0.975))

Arguments
fit ctFit or ctStanFit object.
niter number of data generation iterations to use to calculate quantiles.
probs 3 digit vector of quantiles to return and to test significance.

Details
for plotting help see plot.ctsemFitMeasure

Value
List containing a means and cov object, computed by sorting data into discrete time points. cov is a numeric matrix containing measures of the covariance matrices for observed and simulated data. The MisspecRatio column shows Z score difference for each lower triangular index of the covariance matrix of data – observed covariance minus mean of generated, weighted by sd of generated covariance. means contains the empirical and generated data means.
Examples

data(ctExample1)
traitmodel <- ctModel(n.manifest=2, n.latent=2, Tpoints=6, LAMBDA=diag(2),
  manifestNames=c('LeisureTime', 'Happiness'),
  latentNames=c('LeisureTime', 'Happiness'), TRAITVAR="auto")
traitfit <- ctFit(dat=ctExample1, ctmodelobj=traitmodel)
check <- ctCheckFit(traitfit,niter=5)
plot(check)

ctCI

ctCI Computes confidence intervals on specified parameters / matrices for already fitted ctsem fit object.

Description

ctCI Computes confidence intervals on specified parameters / matrices for already fitted ctsem fit object.

Usage

ctCI(ctfitobj, confidenceintervals, optimizer = "NPSOL", verbose = 0)

Arguments

ctfitobj Already fit ctsem fit object (class: ctsemFit) to estimate confidence intervals for.
confidenceintervals character vector of matrices and or parameters for which to estimate 95% confidence intervals for.
optimizer character vector. Defaults to NPSOL (recommended), but other optimizers available within OpenMx (e.g. 'CSOLNP') may be specified.
verbose Integer between 0 and 3 reflecting amount of output while calculating.

Details

Confidence intervals typically estimate more reliably using the proprietary NPSOL optimizer available within OpenMx only when installing directly from OpenMx website. Use command "source('http://openmx.psyc.virginia.edu/getOpenMx.R')" to install OpenMx with NPSOL. If estimating for a multigroup model, specify confidence intervals as normal, e.g. confidenceintervals = c('DRIFT', 'diffusion_Y1_Y1'). The necessary group prefixes are added internally.

Value

ctfitobj, with confidence intervals included.
Examples

```r
## Examples set to 'donttest' because they take longer than 5s.
data("ctExample3")
model <- ctModel(n.latent = 1, n.manifest = 3, Tpoints = 100,
LAMBDA = matrix(c(1, "lambda2", "lambda3"), nrow = 3, ncol = 1),
MANIFESTMEANS = matrix(c(0, "manifestmean2", "manifestmean3"), nrow = 3,
col = 1))
fit <- ctFit(dat = ctExample3, ctmodelobj = model, objective = "Kalman",
stationary = c("TOVAR"))
fit <- ctCI(fit, confidenceintervals = 'DRIFT')
summary(fit)$omxsummary$CI
```

collapse an array margin using a specified function.

```r
collapsearray <- array(rnorm(900,2,1),dim=c(100,3,3))
collapse(collapsearray,1,mean)
```

### Description

collapse Easily collapse an array margin using a specified function.

### Usage

```r
collapse(inarray, collapsemargin, collapsefunc, ...)
```

### Arguments

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

### Examples

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

ctCompareExpected Compares model implied to observed means and covariances for panel data fit with ctsem.

Description

ctCompareExpected Compares model implied to observed means and covariances for panel data fit with ctsem.

Usage

ctCompareExpected(fitobj, cov = TRUE, outputmatrices = FALSE, pause = TRUE, varlist = "all", ylim = c(-1, 1), ...)

Arguments

fitobj Fitted model object from OpenMx or ctsem.
cov Logical. If TRUE, show covariance plots, if FALSE show correlations.
outputmatrices if TRUE, output expected, observed, and residual correlation matrices as well as plots.
pause if TRUE (default) output plots interactively, one at a time. If FALSE, output without stopping.
varlist if "all" include all variables in dataset. Otherwise, specify numeric vector of variables to include.
ylim vector of min and max Y axis limits for plot.
... additional arguments passed to plot.

tDeintervalise

tDeintervalise

Description

Converts intervals in ctsem long format data to absolute time

Usage

tDeintervalise(datalong, id = "id", dT = "dT", startoffset = 0)
Arguments

datalog: 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

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))
### ctDiscretePars

**Description**

Generate discrete time parameters for a sequence of times based on a list containing continuous time parameter matrices as used in ctsem.

**Usage**

```r
ctDiscretePars(ctpars, times = seq(0, 10, 0.1), type = "all")
```

**Arguments**

- **ctpars** List of continuous time parameter matrices.
- **times** Numeric vector of positive values, discrete time parameters will be calculated for each.
- **type** String. 'all' returns all possible outputs in a list. 'discreteDRIFT' returns only discrete time auto and cross regression effects. 'latentMeans' returns only the expected latent means, given initial (T0MEANS) level, latent intercept (CINT) and temporal effects (DRIFT).

**Examples**

```r
pars <- ctStanContinuousPars(ctstantestfit)
ctDiscretePars(pars, times=c(.5,1))
```

---

### ctDiscretiseData

**Description**

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

**Usage**

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

**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**

```r
cDDiscretiseData(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

```r
long <- ctWideToLong(datawide=ctExample2,Tpoints=8,n.manifest=2,n.TDpred=1,
                      manifestNames=c('LeisureTime','Happiness'),
                      TDpredNames=c('MoneyInt'))
long <- ctDeintervalise(long)
long <- ctDiscretiseData(dlong=long, timestep = 1.1,TDpredNames=c('MoneyInt'))
```

---

**ctExample1**

**Description**

Simulated example dataset for the ctsem package

**Format**

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

---

**ctExample1TIpred**

**Description**

Simulated example dataset for the ctsem package

**Format**

100 by 18 matrix 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.
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.

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.

Description
Simulated example dataset for the ctsem package

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

ctFit

Fit a ctsem object

Description

This function fits continuous time SEM models specified via ctModel to a dataset containing one or more subjects.

Usage

ctFit(dat, ctmodelobj, dataform = "wide", objective = "auto", stationary = c("T0TRAITEFFECT", "T0TIPREDEFFECT"), optimizer = "CSOLNP", retryattempts = 5, iterationSummary = FALSE, carefulFit = TRUE, carefulFitWeight = 100, showInits = FALSE, asymptotes = FALSE, meanIntervals = FALSE, plotOptimization = FALSE, crossEffectNegStarts = TRUE, fit = TRUE, nofit = FALSE, discreteTime = FALSE, verbose = 0, useOptimizer = TRUE, omxStartValues = NULL, transformedParams = TRUE, datawide = NA)

Arguments

dat the data you wish to fit a ctsem model to, in either wide format (one individual per row), or long format (one time point of one individual per row). See details.

cemodelobj the ctsem model object you wish to use, specified via the ctModel function.

dataform either "wide" or "long" depending on which input format you wish to use for the data. See details and or vignette.

objective 'auto' selects either 'Kalman', if fitting to single subject data, or 'mxRAM' for multiple subjects. For single subject data, 'Kalman' uses the mxExpectationStateSpace function from OpenMx to implement the Kalman filter. For more than one subject, 'mxRAM' specifies a wide format SEM with a row of data per subject. 'cov' may be specified, in which case the 'meanIntervals' argument is set to
ctFit

TRUE, and the covariance matrix of the supplied data is calculated and fit instead of the raw data. This is much faster but only a rough approximation, unless there are no individual differences in time interval and no missing data. ‘Kalman’ may be specified for multiple subjects, however as no trait matrices are used by the Kalman filter one must consider how average level differences between subjects are accounted for. See ctMultigroupFit for the possibility to apply the Kalman filter over multiple subjects.

stationary Character vector of T0 matrix names in which to constrain any free parameters to stationarity. Defaults to c('T0TRAITEFFECT','T0TIPREDEFFECT'), constraining only between person effects to stationarity. Use NULL for no constraints, or ‘all’ to constrain all T0 matrices.

optimizer character string, defaults to the open-source 'CSOLNP' optimizer that is distributed in all versions of OpenMx. However, 'NPSOL' may sometimes perform better for these problems, though requires that you have installed OpenMx via the OpenMx web site, by running: source('http://openmx.psyc.virginia.edu/getOpenMx.R')

retryattempts Number of times to retry the start value randomisation and fit procedure, if non-convergance or uncertain fits occur.

iterationSummary if TRUE, outputs limited fit details after every fit attempt.

carefulFit if TRUE, first fits the specified model with a penalised likelihood function to force MANIFESTVAR, DRIFT, TRAITVAR, MANIFESTTRAITVAR parameters to remain close to 0, then fits the specified model normally, using these estimates as starting values. Can help to ensure optimization begins at sensible, non-extreme values, though results in any user specified start values being ignored for the final fit (though they are still used for initial fit).

carefulFitWeight Positive numeric. Sets the weight for the penalisation (or prior) applied by the carefulFit algorithm. Generally unnecessary to adjust, may be helpful to try a selection of values (perhaps between 0 and 1000) when optimization is problematic.

showInits if TRUE, prints the list of starting values for free parameters. These are the ‘raw’ values used by OpenMx, and reflect the log (var / cov matrices) or -log(DRIFT matrices) transformations used in ctsem. These are saved in the fit object under fitobject$omxStartValues.

asymptotes when TRUE, optimizes over asymptotic parameter matrices instead of continuous time parameter matrices. Can be faster for optimization and in some cases makes reliable convergance easier. Will result in equivalent models when continuous time input matrices (DRIFT, DIFFUSION, CINT) are free, but fixing the values of any such matrices will result in large differences - a value of 0 in a cell of the normal continuous time DIFFUSION matrix does not necessarily result in a value of 0 for the asymptotic DIFFUSION matrix, for instance.

meanIntervals Use average time intervals for each column for calculation (both faster and inaccurate to the extent that intervals vary across individuals).
plotOptimization
   If TRUE, uses checkpointing for OpenMx function mxRun, set to checkpoint every iteration, output checkpoint file to working directory, then creates a plot for each parameter’s values over iterations.

crossEffectNegStarts
   Logical. If TRUE (default) free DRIFT matrix cross effect parameters have starting values set to small negative values (e.g. -.05), if FALSE, the start values are 0. The TRUE setting is useful for easy initialisation of higher order models, while the FALSE setting is useful when one has already estimated a model without cross effects, and wishes to begin optimization from those values by using the omxStartValues switch. are re-transformed into regular continuous time parameter matrices, and may be interpreted as normal.

fit
   if FALSE, output only openmx model without fitting

nofit
   Deprecated. If TRUE, output only openmx model without fitting

discreteTime
   Estimate a discrete time model - ignores timing information, parameter estimates will correspond to those of classical vector autoregression models, OpenMx fit object will be directly output, thus ctsm summary and plot functionality will be unavailable. Time dependent predictor type also becomes irrelevant.

verbose
   Integer between 0 and 3. Sets mxComputeGradientDescent messaging level, defaults to 0.

useOptimizer
   Logical. Defaults to TRUE. Passes argument to mxRun, useful for using custom optimizers or fitting to specified parameters.

omxStartValues
   A named vector containing the raw (potentially log transformed) OpenMx starting values for free parameters, as captured by OpenMx function omxGetParameters(ctmodelobj$mxobj). These values will take precedence over any starting values already specified using ctModel.

transformedParams
   Logical indicating whether or not to log transform certain parameters internally to allow unconstrained estimation over entire ‘sensible’ range for parameters. When TRUE (default) raw OpenMx parameters (only reported if verbose=True argument used for summary function) will reflect these transformations and may be harder to interpret, but summary matrices are reported as normal.

datawide
   included for compatibility with scripts written for earlier versions of ctsem. Do not use this argument, instead use the dat argument, and the dataform argument now specifies whether the data is in wide or long format.

Details
   For full discussion of how to structure the data and use this function, see the vignette using: vignette('ctsem'), or the data examples data("longexample"); longexample for long and data("datastructure"); datastructure for wide. If using long format, the subject id column must be numeric and grouped by ascending time within subject, and named 'id'. The time column must also be numeric, and representing absolute time (e.g., since beginning of study, *not* time intervals), and called 'time'. Models are specified using the ctModel function. For help regarding the summary function, see summary.ctsemFit, and for the plot function, plot.ctsemFit. Multigroup models may be specified using ctMultigroupFit.
Confidence intervals for any matrices and or parameters may be estimated using \texttt{ctCI}. Difficulties during estimation can sometimes be alleviated using \texttt{ctRefineTo} instead of \texttt{ctFit} – this uses a multistep fit procedure.

Examples

```r
## Examples set to 'donttest' because they take longer than 5s.

mfrowOld<-par()$mfrow
par(mfrow=c(2, 3))

### example from Driver, Oud, Voelkle (2017),
### simulated happiness and leisure time with unobserved heterogeneity.
data(ctExample1)
traitmodel <- ctModel(n.manifest=2, n.latent=2, Tpoints=6, LAMBDA=diag(2),
                       manifestNames=c('LeisureTime', 'Happiness'),
                       latentNames=c('LeisureTime', 'Happiness'), TRAITVAR="auto")
traitfit <- ctFit(dat=ctExample1, ctmodelobj=traitmodel)
summary(traitfit)
plot(traitfit, wait=FALSE)

###Example from Voelkle, Oud, Davidov, and Schmidt (2012) - anomia and authoritarianism.
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)
summary(AnomAuthfit)

### Single subject time series - using Kalman filter (OpenMx statespace expectation)
data('ctExample3')
model <- ctModel(n.latent = 1, n.manifest = 3, Tpoints = 100,
                  LAMBDA = matrix(c(1, 'lambda2', 'lambda3'), nrow = 3, ncol = 1),
                  CINT= matrix('cint'),
                  MANIFESTMEANS = matrix(c(0, 'manifestmean2', 'manifestmean3'), nrow = 3,
                                        ncol = 1))
fit <- ctFit(dat = ctExample3, ctmodelobj = model, objective = 'Kalman',
             stationary = c('T0VAR'))

###Oscillating model from Voelkle & Oud (2013).
data("Oscillating")
inits <- c(-39, -.3, 1.01, 10.01, .1, 10.01, 0.05, .9, 0)
names(inits) <- c("crosseffect","autoeffect", "diffusion",
                 "T0var11", "T0var21", "T0var22","m1", "m2", 'manifestmean')
oscillatingm <- ctModel(n.latent = 2, n.manifest = 1, Tpoints = 11,
                         MANIFESTVAR = matrix(c(0), nrow = 1, ncol = 1),
                         LAMBDA = matrix(c(1), nrow = 1, ncol = 2),
                         T0MEANS = matrix(c('m1', 'm2'), nrow = 2, ncol = 1),
                         T0VAR = matrix(c("T0var11", "T0var21", 0, "T0var22"), nrow = 2, ncol = 2),
                         TRAITVAR = NULL)
```
DRIFT = matrix(c(0, "crosseffect", 1, "autoeffect"), nrow = 2, ncol = 2),
CINT = matrix(0, ncol = 1, nrow = 2),
MANIFESTMEANS = matrix('manifestmean', nrow = 1, ncol = 1),
DIFFUSION = matrix(c(0, 0, 0, "diffusion"), nrow = 2, ncol = 2),
startValues=inits)

oscillatingf <- ctFit(Oscillating, oscillatingm, carefulFit = FALSE)

---

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 = TRUE)

Arguments

cemodelobj  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 burnin+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

TRAITVAR and MANIFESTRAITVAR are treated as Cholesky factor covariances of CINT and
MANIFESTMEANS, respectively. TRAITTDPREDCOV and TIPREDCOV matrices are not ac-
counted 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,.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)

cTGenerateFromFit

Generates data according to the model estimated in a ctsemFit object.

Description

Generates data according to the model estimated in a ctsemFit object.

Usage

ctGenerateFromFit(fit, timestep = "asdata", n.subjects = 100,
timerange = "asdata", predictorSubjects = "all", ...)

Arguments

fit object of class ctsemFit as returned from ctFit
timestep positive numeric value indicating the time interval to use for data generation.
n.subjects integer. Number of subjects worth of data to generate
timerange either 'asdata' to calculate range based on data in fit object, or vector of length 2 specifying min and max times for generation.
predictorSubjects vector of integers, or string 'all', defining which subjects to sample time dependent and independent predictors from.
... parameters to pass to ctGenerate function, such as wide=FALSE.

Value

matrix of generated data
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))
AnomAuthfit <- ctFit(AnomAuth, AnomAuthmodel)
dwide <- ctGenerateFromFit(AnomAuthfit, timestep = 1, n.subjects = 5)
par(mfrow=c(1,2))
ctIndplot(datawide = dwide, n.subjects = 5, n.manifest = 2, Tpoints = 4)
ctIndplot(datawide = AnomAuth + rnorm(length(AnomAuth)), vars = 1, n.subjects = 5,
n.manifest = 2, Tpoints = 4)

cIndplot

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

Usage

cIndplot(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

cIntervalise(datawide, Tpoints, n.manifest, n.TDpred = 0, n.TIpred = 0, imputedefs = F, manifestNames = "auto", TIpredNames = "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. Otherwise as used in ctFit. 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

#First load the long format data with absolute times
data('longexample')

#Then convert to wide format
wideexample <- ctLongToWide(datalong = longexample, id = "id", time = "time", manifestNames = c("Y1", "Y2", "Y3"), TDpredNames = "TD1", TIpredNames = c("TI1", "TI2"))

#Then convert the absolute times to intervals, using the Tpoints reported from the prior step.
wide <- ctIntervalise(datawide = wideexample, Tpoints = 4, n.manifest = 3, n.TDpred = 1, n.TIpred = 2, manifestNames = c("Y1", "Y2", "Y3"), TDpredNames = "TD1", TIpredNames = c("TI1", "TI2"))

print(wide)

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 medians of parameter distribution.
Usage

cKalman(fit, datalong = NULL, timerange = "asdata",
timestep = "asdata", subjects = 1, plot = FALSE, ...)

Arguments

fit fit object as generated by ctStanFit.
datalog Optional long format data object as used by ctStanFit. If not included, data from fit will used.
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. Currently unused for ctStan fits.
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. Currently unavailable for ctStan fits.
subjects vector of integers denoting which subjects (from 1 to N) to plot predictions for.
plot Logical. If TRUE, plots output instead of returning it. See ctKalmanPlot for the possible arguments.
... additional arguments to pass to ctKalmanPlot.

Value

Returns a list containing matrix objects etaprior, etauupd, 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, etauupdov, etasmoothcov, ypriorcov, yupdov, 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
cKalman(ctstantestfit, timerange=c(0,60), timestep=.5, plot=TRUE)

#Multiple subjects, y and yprior, showing plot arguments
cKalman(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
ltyvec="auto",
colvec='auto',
lwdvec='auto',
subsetindices=2, #Only plotting 2nd dimension of y and yprior
pchvec='auto', typevec='auto',grid=TRUE,legend=TRUE,
plotcontrol=list(xlim=c(0,55),main='Observations and priors'),
polygoncontrol=list(steps=5))
ctKalmanPlot

Description

Plots Kalman filter output from ctKalman.

Usage

ctKalmanPlot(x, subjects, kalmanvec = c("y", "etaprior"),
errorvec = "auto", errormultiply = 1.96, ltyvec = "auto",
colvec = "auto", lwdvec = "auto", subsetindices = NULL,
pchvec = "auto", typevec = "auto", grid = FALSE, add = FALSE,
plotcontrol = list(ylab = "Value", xlab = "Time", xaxs = "i"),
polygoncontrol = list(steps = 20), polygonalpha = 0.3,
legend = TRUE, legendcontrol = list(x = "topright", bg = "white"))

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 ‘yprior’ plot the original data, ‘y’, and the prior from the Kalman
filter for y. Replacing ‘y’ by ‘eta’ will plot latent variables instead (though ‘eta’
alone does not exist) and replacing ‘prior’ with ‘upd’ or ‘smooth’ respectively
plotting updated (conditional on all data up to current time point) or smoothed
(conditional on all data) estimates.

errorvec

vector of names of covariance elements to use for uncertainty indication around
the kalmanvec items. ‘auto’ uses the latent covariance when plotting latent
states, and total covariance when plotting expectations of observed states. Use
NA to skip uncertainty plotting.

errormultiply

Numeric denoting the multiplication factor of the std deviation of errorvec ob-
jects. Defaults to 1.96, for 95% intervals.

ltyvec

vector of line types, varying over dimensions of the kalmanvec object.

colvec

color vector, varying either over subject if multiple subjects, or otherwise over
the dimensions of the kalmanvec object.

lwdvec

vector of line widths, varying over the kalmanvec objects.

subsetindices

Either NULL, or vector of integers to use for subsetting the (columns) of kalman-
vec objects.

pchvec

vector of symbol types, varying over the dimensions of the kalmanvec object.

typevec

vector of plot types, varying over the kalmanvec objects. ‘auto’ plots lines for
any ‘prior’, ‘upd’, or ‘smooth’ objects, and points otherwise.
ctLongToWide

Logical. Plot a grid?

Logical. Create a new plot or update existing plot?

Logical. List of graphical arguments (see par), though lty,col,lwd,x,y, will all be ignored.

Logical. List of arguments to the ctPoly function for filling the uncertainty region.

Numeric for the opacity of the uncertainty region.

Logical, whether to include a legend if plotting.

Logical, whether to include a legend if plotting.

Value

Nothing. Generates plots.

Examples

### Get output from ctKalman
x<-ctKalman(ctstantestfit,subjects=2)

### Plot with ctKalmanPlot
ctKalmanPlot(x, subjects=2)

### Single step procedure:
ctKalman(ctstantestfit,subjects=2,plot=TRUE)

tctLongToWide

tctLongToWide Reststructures time series / panel data from long format to wide format for ctsem analysis

Description

tctLongToWide Reststructures time series / panel data from long format to wide format for ctsem analysis

Usage

cTcLongToWide(datalong, id, time, manifestNames, TDpredNames = NULL, TIpredNames = NULL)
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 ctFit, or ctStanFit for Bayesian models.

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 column

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

#First load the long format data with absolute times
data('longexample')

#Then convert to wide format
wideexample <- ctLongToWide(datalong = longexample, id = "id",
  time = "time", manifestNames = c("Y1", "Y2", "Y3"),
  TDpredNames = "TD1", TIpredNames = c("TI1", "TI2"))

#Then convert the absolute times to intervals, using the Tpoints reported from the prior step.
wide <- ctIntervalise(datawide = wideexample, Tpoints = 4, n.manifest = 3,
  n.TDpred = 1, n.TIpred = 2, manifestNames = c("Y1", "Y2", "Y3"),
  TDpredNames = "TD1", TIpredNames = c("TI1", "TI2"))
ctModel

Usage

ctModel(LAMBDA, n.manifest = nrow(LAMBDA), n.latent = ncol(LAMBDA),
        type = "omx", Tpoints = NULL, manifestNames = "auto",
        latentNames = "auto", TOVAR = "auto", TMEANS = "auto",
        MANIFESTMEANS = "auto", MANIFESTVAR = "auto", DRIFT = "auto",
        CINT = "auto", DIFFUSION = "auto", n.TDpred = 0,
        TDpredNames = "auto", n.TIpred = 0, TIpredNames = "auto",
        TRAITVAR = NULL, TOTRAITEFFECT = NULL, MANIFESTTRAITVAR = NULL,
        TDPREDMEANS = "auto", TDPREDVAR = "auto", T0TDPREDCOV = "auto",
        TDPREDVAR = "auto", TRAITTDPREDCOV = "auto", TDTIPREDCOV = "auto",
        TIPREDMEANS = "auto", TIPREDEFFECT = "auto",
        TOTIPREDEFFECT = "auto", TIPREDVAR = "auto", PARS = NULL,
        startValues = NULL, timeVarying = 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.

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.

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.

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).

TOVAR lower triangular n.latent*n.latent cholesky matrix of latent process initial variance / covariance. "auto" freely estimates all parameters.

TMEANS 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.

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 parameters, 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 predictor 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).

TDTpredCov

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.

timeVarying

For type='omx' only. character vector of matrices to allow to vary over measurement occasions. Currently only accepts 'MANIFESTMEANS'.

Examples

```r
### 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$TOVAR[3, ] <- 0
tdpredmodel$TOVAR[, 3] <- 0
tdpredmodel$SCINT[3] <- 0
tdpredmodel$SCMEANS[3] <- 0
tdpredmodel$TMEANS[3] <- 0
tdpredmodel$TDPREDMEAN[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))

---

ctModelFromFit

*Extract a ctsem model structure with parameter values from a ctsem fit object.*

**Description**

Extract a ctsem model structure with parameter values from a ctsem fit object.

**Usage**

```r
ctModelFromFit(fit)
```

**Arguments**

- `fit` object output by `ctFit`

**Value**

object of class 'ctsemInit' (as generated by `ctModel`), which can be used with `ctFit` and `Kalman` functions.
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))
AnomAuthfit <- ctFit(AnomAuth, AnomAuthmodel)

fitmodel <- ctModelFromFit(AnomAuthfit)

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(ctmodel, textsize = "normalsize", folder = tempdir(),
  filename = "ctsemTex", tex = TRUE, equationonly = FALSE,
  compile = TRUE, open = TRUE)

Arguments

cmodel: ctsem model object

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 com-
  pileable document.

compile: Compile to .pdf? (Depends on tex = TRUE)

open: Open after compiling? (Depends on compile = TRUE)

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)

ctMultigroupFit fits a multiple group continuous time model.

Description

Fits a single continuous time structural equation models to multiple groups (where each group contains 1 or more subjects), by default, all parameters are free across groups. Can also be used to easily estimate separate models for each group.

Usage

ctMultigroupFit(dat, groupings, ctmodelobj, dataform = "wide",
fixedmodel = NA, freemodel = NA, carefulFit = TRUE,
omxStartValues = NULL, retryattempts = 5, showInits = FALSE, ...)

Arguments

dat Wide format data, as used in ctFit. See ctLongToWide to easily convert long format data.

groupings For wide format: Vector of character labels designating group membership for each row of dat. For long format: Named list of groups, with each list element containing a vector of subject id's for the group. In both cases, group names will be prefixed on relevant parameter estimates in the summary.

tcmodelobj Continuous time model to fit, specified via ctModel function.

dataform either "wide" or "long" depending on which input format you wish to use for the data. See details of ctFit and or vignette.

fixedmodel Modified version of ctmodelobj, wherein any parameters you wish to keep fixed over groups should be given the value 'groupfixed'. If specified, all other parameters will be free across groups.
freemodel Modified version of ctmodelobj, wherein any parameters you wish to free across groups should be given the label 'groupfree'. If specified, all other parameters will be fixed across groups. If left NULL, the default, all parameters are free across groups.
carefulFit if TRUE, first fits the specified model with a penalised likelihood function to discourage parameters from boundary conditions, then fits the specified model normally, using these estimates as starting values. Can help / speed optimization, though results in user specified inits being ignored for the final fit.
omxStartValues A named vector containing the raw (potentially log transformed) OpenMx starting values for free parameters, as captured by OpenMx function omxGetParameters(ctmodelobj$mxobj). These values will take precedence over any starting values already specified using ctModel.
retryattempts Number of fit retries to make.
showInits Displays start values prior to optimization
... additional arguments to pass to ctFit.

Details

Additional ctFit parameters may be specified as required. Confidence intervals for any matrices and or parameters may be estimated after fitting using ctCI.

Value

Returns an OpenMx fit object.

See Also

ctFit and ctModel

Examples

#Two group model, all parameters except LAMBDA[3,1] constrained across groups.
data(ctExample4)
basemodel<-ctModel(n.latent=1, n.manifest=3, Tpoints=20,
    LAMBDA=matrix(c(1, 'lambda2', 'lambda3'), nrow=3, ncol=1),
    MANIFESTMEANS=matrix(c(0, 'manifestmean2', 'manifestmean3'),
        nrow=3, ncol=1), TRAITVAR = 'auto')

freemodel<-basemodel
freemodel$LAMBDA[3,1]<-'groupfree'
groups<-paste0('g',rep(1:2, each=10),'_')
multif<-ctMultigroupFit(dat=ctExample4, groupings=groups,
    ctmodelobj=basemodel, freemodel=freemodel)
summary(multif, group=1)
# fixed model approach
fixedmodel <- basemodel
fixedmodel$LAMBDA[2, 1] <- 'groupfixed'
groups <- paste0('g', rep(1:2, each=10), '_')

multif <- ctMultigroupFit(dat=ctExample4, groupings=groups, 
                        ctmodelobj=basemodel, fixedmodel=fixedmodel)
summary(multif, group=2)

---

## Description

Plots mean trajectories, autoregression, and crossregression plots, for ctsemFit objects. More customizable than basic plot.ctsemFit function.

## Usage

```r
cfPlot(x, plotType, xlim, resolution = 50, impulseIndex = NULL, 
       subject = 1, typeVector = "auto", colVector = "auto", 
       ltyVector = "auto", ...)
```

## Arguments

- `x` ctsemFit object as generated by `ctFit`.
- `plotType` string. "mean" for expectation independent of any data, "AR" for autoregressions, "CR" for cross regressions, "standardiseCR" for standardised cross regressions (standardised based on estimated within subject variance), "withinVar" for within variance and covariance, "randomImpulse" for expected change in processes given a random fluctuation of +1 for each process (so a mixture of DIFFUSION and DRIFT characteristics), "experimentalImpulse" for expected change in processes given an exogenous input of +1 for each process, provides alternate characterisation of autoregressive and cross regressive plots.
- `xlim` vector. As per usual for plot(), but xlim may not be negative.
- `resolution` Numeric. Plot points between each unit of time. Default of ’auto’ adapts to xlim and results in 500 points in total.
- `impulseIndex` Numeric. Only required for impulse plot types, specifies which column of the DRIFT matrix the impulse relates to.
- `subject` numeric. Specifies the subject (row of data from the mxobj) to plot for factorScores type plot.
- `typeVector` Vector of plot types to use for plotting.
colVector  vector of colours to use for plotting.
ltyVector  Vector of line types to use for plotting.
...  Other options passed to plot(). ylim is required.

Value

Character vector of labels from the DRIFT matrix in order plotted - useful for legends. Side-effect: plots graphs.

Examples

## Examples set to 'donttest' because they take longer than 5s.
### example from Driver, Oud, Voelkle (2016),
### simulated happiness and leisure time with unobserved heterogeneity.
data(ctExample1)
traitmodel <- ctModel(n.manifest=2, n.latent=2, Tpoints=6, LAMBDA=diag(2),
  manifestNames=c('LeisureTime', 'Happiness'),
  latentNames=c('LeisureTime', 'Happiness'), TRAITVAR="auto")
traitfit <- ctFit(dat=ctExample1, ctmodelobj=traitmodel)
ctPlot(traitfit, plotType='CR', xlim=c(0,5), ylim=c(-1,1))

ctPlotArray  

Plots three dimensional y values for quantile plots

Description

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

Usage

cPlotArray(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.
ctPoly

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()
**ctPostPredict**

Value

Nothing. Adds a polygon to existing plot.

Examples

```r
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))
```

---

**ctPostPredict**  
*Posterior predictive type check for ctsemFit.*

Description

Samples data according to the ctsemFit object, computes quantiles over time based on model fit, plots these against original data.

Usage

```r
cPostPredict(fit, timestep = 0.1, n.subjects = 100, probs = c(0.025, 0.5, 0.975), plot = TRUE, ctPlotArrayArgs = list(grid = FALSE, legend = FALSE), indPlotArgs = list(colourby = "subject", lwd = 2, new = FALSE, type = "p", opacity = 0.3), mfrow = "auto")
```

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit</td>
<td>object of class ctsemFit as returned from ctFit</td>
</tr>
<tr>
<td>timestep</td>
<td>positive value denoting the time interval to use for sampling.</td>
</tr>
<tr>
<td>n.subjects</td>
<td>Number of subjects worth of data to sample.</td>
</tr>
<tr>
<td>probs</td>
<td>Vector of values between 0 and 1 denoting quantiles to generate. For plotting, vector should be of length 3 and values should be rising.</td>
</tr>
<tr>
<td>plot</td>
<td>Whether to plot or return the generated data.</td>
</tr>
<tr>
<td>ctPlotArrayArgs</td>
<td>additional arguments to pass to ctPlotArray function, for plotting generated distributions.</td>
</tr>
<tr>
<td>indPlotArgs</td>
<td>list of parameters to pass to ctIndplot, for plotting original data. Only used if plot=TRUE.</td>
</tr>
<tr>
<td>mfrow</td>
<td>2 dimensional integer vector defining number of rows and columns of plots, as per the mfrow argument to par. 'auto' determines automatically, to a maximum of 4 by 4, while NULL uses the current system setting.</td>
</tr>
</tbody>
</table>

Value

Either nothing (if plot=TRUE) or an array containing generated data over quantiles.
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 = 'auto')
AnomAuthFit <- ctFit(AnomAuth, AnomAuthmodel)
ctPostPredict(AnomAuthFit,timestep=.5,n.subjects=100)

ctRefineTo

Description

Fits a ctsem m in a stepwise fashion to help with difficult optimization.

Usage

crctRefineTo(datawide, ctmodelobj, modfunc = NULL, ...)

Arguments

datawide          Data in ctsem wide format
ctmodelobj       A continuous time m specified via the ctModel function.
modfunc           function to run prior to each optimization step, that takes ctsem fit object, modifies it as desired, and returns the fit object.
...               additional parameters to pass to ctFit.

Details

This function fits a sequence of ctsem models increasing in complexity, starting with a m involving fixed and relatively strong auto effects, no cross effects, no predictors, and no off-diagonal covariances. For many models this can improve the speed and robustness of fitting

Value

Returns a fitted ctsem object in the same manner as ctFit.
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. The frequentist approach is faster but can only estimate random-effects on the intercepts, while the Bayesian approach allows for random-effects across all model parameters.

Details

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 either `ctFit` if an 'omx' (OpenMx, frequentist) model is specified or `ctStanFit` if a 'stanct' or 'standt' (Stan, continuous / discrete time, Bayesian) model is specified. For examples, see either `ctFit` or `ctStanFit`. For more detailed information, see the frequentist vignette by running: `vignette('ctsem')` For citation info, please run `citation('ctsem')`.

References

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


ctStanContinuousPars

Description

Returns the continuous time parameter matrices for specified subjects of a ctStanFit fit object

Usage

```r
ctStanContinuousPars(ctstanfitobj, subjects = "all", iter = "all", calcfunc = quantile, calcfuncargs = list(probs = 0.5))
```
Arguments

ctstanfitobj fit object from ctStanFit

subjects Either ‘all’, or integers denoting which subjects to perform the calculation over. When multiple subjects are specified, the returned matrices will be a mean over subjects.

iter Either character string ‘all’ which will then use all post-warmup iterations, or an integer specifying which iteration/s to use.

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.

Examples

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

# posterior 97.5% quantiles for subject 2
cTStanContinuousPars(ctstantestfit, subjects=2, calcfunc=quantile,
calcfuncargs=list(probs=0.975))

cTStanDiscretePars ctTStanDiscretePars

description

Calculate model implied regressions for a sequence of time intervals based on a continuous time model fit from ctStanFit, for specified subjects.

Usage

cTStanDiscretePars(ctstanfitobj, subjects = "all", times = seq(from = 0, to = 10, by = 0.1), quantiles = c(0.025, 0.5, 0.975), nsamples = 500, observational = FALSE, standardise = FALSE, plot = FALSE, ...)

Arguments

ctstanfitobj Continuous time model fit from ctStanFit

subjects Either ‘all’, to take the average over all subjects, or a vector of integers denoting which subjects.
times Numeric vector of positive values, discrete time parameters will be calculated for each.

quantiles Which quantiles to return. If plotting, specify 3 quantiles, the 2nd will be plotted as a line with 1 and 3 as uncertainty bounds.

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 asymDIFFUSION matrix.

plot Logical. If TRUE, plots output using ctStanDiscreteParsPlot instead of returning output.

... additional plotting arguments to control ctStanDiscreteParsPlot

Examples

ctStanDiscretePars(ctstantestfit,times=seq(.5,4,.1), plot=TRUE,indices='all')

Description

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

Usage

ctStanDiscreteParsPlot(x, indices = "all", add = FALSE, legend = TRUE, polygon = TRUE, quantiles = c(0.025, 0.5, 0.975), times = seq(0, 10, 0.1), latentNames = "auto", lwdvec = "auto", colvec = "auto", ltyvec = "auto", plotcontrol = list(ylab = "Value", xlab = "Time interval", main = "Regression coefficients", type = "l", xaxs = "i"), grid = FALSE, legendcontrol = list(x = "topright", bg = "white"), polygonalpha = 0.1, polygoncontrol = list(steps = 20))
Arguments

\textbf{x} \hspace{1cm} \text{list object returned from \texttt{ctStanDiscretePars}.}

\textbf{indices} \hspace{1cm} \text{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.}

\textbf{add} \hspace{1cm} \text{Logical. If FALSE, a new plot is generated, if TRUE, specified plot/s are overlayed on existing plot.}

\textbf{legend} \hspace{1cm} \text{Logical. If TRUE, generates a legend.}

\textbf{polygon} \hspace{1cm} \text{Logical. If TRUE, fills a polygon between the first and last specified quantiles.}

\textbf{quantiles} \hspace{1cm} \text{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\%.}

\textbf{times} \hspace{1cm} \text{Numeric vector of positive values, discrete time parameters will be calculated for each.}

\textbf{latentNames} \hspace{1cm} \text{Vector of character strings denoting names for the latent variables. ‘auto’ just uses eta1 eta2 etc.}

\textbf{lwdvec} \hspace{1cm} \text{Either ‘auto’, or a vector of positive integers denoting line widths for each quantile. ‘auto’ specifies c(1,3,1) if there are 3 quantiles to be plotted (default), otherwise simply 3.}

\textbf{colvec} \hspace{1cm} \text{Either ‘auto’, or a vector of color values denoting colors for each index to be plotted. ‘auto’ generates colors using the \texttt{grDevices::rainbow} function.}

\textbf{ltyvec} \hspace{1cm} \text{Either ‘auto’, or a vector of line type integers (as for the lty parameter normally) denoting line types for each quantile. ‘auto’ specifies c(3, 1, 3) if there are 3 quantiles to be plotted (default), otherwise simply 1.}

\textbf{plotcontrol} \hspace{1cm} \text{list of arguments to pass to plot function. The following arguments are ignored: ylim, lwd, lty, col, x, y.}

\textbf{grid} \hspace{1cm} \text{Logical. Plot with a grid?}

\textbf{legendcontrol} \hspace{1cm} \text{list of arguments to pass to legend function. ‘legend=’ and ‘text.col=’ arguments will be ignored.}

\textbf{polygonalpha} \hspace{1cm} \text{Numeric between 0 and 1 to multiply the alpha (transparency) of colvec by for the fill polygon.}

\textbf{polygoncontrol} \hspace{1cm} \text{list of arguments to pass to \texttt{ctPoly} function (if polygon=TRUE). x,y, and col arguments will be ignored. Steps specifies the number of polygons to overlay to create a graduated transparency. Set to 1 for a flat looking plot.}

Examples

\begin{verbatim}
x <- ctStanDiscretePars(ctstantestfit)

ctStanDiscreteParsPlot(x, 'CR')
\end{verbatim}
ctStanFit

Description

Fits a ctsem model specified via ctModel with type either 'stanct' or 'standt', using Bayesian inference software Stan.

Usage

ctStanFit(datalong, ctstanmodel, stanmodeltext = NA, iter = 1000, intoverstates = TRUE, binomial = FALSE, fit = TRUE, intoverpop = FALSE, stationary = FALSE, plot = FALSE, derrind = "all", optimize = FALSE, optimcontrol = list(), nlcontrol = list(), nopriors = FALSE, chains = 2, cores = "maxneeded", inits = NULL, forcerecompile = FALSE, savescores = FALSE, gendata = FALSE, control = list(), verbose = 0, ...)

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 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 TRUE, integrates over population distribution of parameters rather than full sampling. Allows for optimization of non-linearities and random effects.
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, a Shiny program is launched upon fitting to interactively plot samples. May struggle with many (e.g., > 5000) parameters, and may leave sample files in working directory if sampling is terminated.

derrind | vector of integers denoting which latent variables are involved in dynamic error calculations. Latents involved only in deterministic trends or input effects can be removed from matrices (i.e., that obtain no additional stochastic inputs after first observation), speeding up calculations. If unsure, leave default of 'all'! Ignored if intoverstates=FALSE.

optimize | if TRUE, use stanoptimis function for maximum a posteriori estimates.

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

nlcontrol | List of non-linear control parameters. nldynamics defaults to "auto", but may also be a logical. Set to FALSE to use estimator that assumes linear dynamics, TRUE to use non-linear estimator. "auto" selects linear when the model is obviously linear, otherwise nonlinear – nonlinear is slower. nlmeasurement defaults to "auto", but may also be a logical. Set to TRUE to use non-linear measurement model estimator, FALSE to use linear model. "auto" selects linear if appropriate, otherwise nonlinear. Non-linear methods are slower but applicable to both linear and non-linear cases. ukffull may be TRUE or FALSE. If FALSE, nonlinear filtering via the unscented filter uses a minimal number of sigma points, that does not capture skew in the resulting distribution. 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. ukfspread should be a small positive numeric value, indicating what fraction of a standard deviation to use for unscented sigma points. Values between 1e-4 and 2 have tended to be reasonable, in our experience. In general, larger values may not make sense when using the default of ukffull=FALSE.

nopriors | logical. If TRUE, any priors are disabled – sometimes desirable for optimization.

chains | 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.

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.

inits | vector of parameter start values, as returned by the rstan function rstan::unconstrain_pars for instance.
forcerecompile  
logical. For development purposes. If TRUE, stan model is recompiled, regardless of apparent need for compilation.

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.

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  
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.

additional arguments to pass to stan function.

Examples

#test data with 2 manifest indicators measuring 1 latent process each,  
# 1 time dependent predictor, 3 time independent predictors
head(ctstantestdat)

#generate a ctStanModel
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))

#set all parameters except manifest means to be fixed across subjects
model$pars$indvarying[-c(19,20)] <- FALSE

#fit model to data (takes a few minutes - but insufficient  
# iterations and max_treedepth for inference!)
fit<-ctStanFit(ctstantestdat, model, iter=200, chains=2,  
control=list(max_treedepth=6))

#output functions
summary(fit)
plot(fit,wait=FALSE)

## Not run:

#### EXTENDED EXAMPLES ####
library(ctsem)
set.seed(3)

#recommended to adjust these to appropriate number of cores on machine /
# chains desired. (min 3 chains recommended, but not necessary here)
setcores <- 3
setchains <- 3

### Data generation (this section needs to be run, but not necessary 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, .1, 0, -.5),nlatent,nlatent),
    TDPREDEFFECT=matrix(c(tdpredeffect[i],0),nrow=nlatent),
    TDPREDEFFECTS=matrix(c(rep(0,Tpoints-10),1,rep(0,9)),ncol=1),
    DIFFUSION = matrix(c(.5, 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))),
    wide = FALSE)
  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)
}

### Model fitting (from here it is good to understand!)

#Specify univariate linear growth curve
#page 5 of https://cran.r-project.org/web/packages/ctsem/vignettes/hierarchical.pdf
# documents these arguments (or use ?ctModel )

m1 <- ctModel(n.manifest = 1,n.latent = 1,n.TIpred = 1, type = 'stanct',
               manifestNames = c('Y1'), latentNames=c('L1'),TIpredNames = 'age',
               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('merror1'),nrow=1,ncol=1))

#modify between subject aspects -- alternatively, run: edit(ml$pars)
ml$pars$indvarying <- TRUE
ml$pars$indvarying[[-which(ml$pars$matrix %in% c('T0MEANS','CINT'))]] <- FALSE
ml$pars$age_effect[[-which(ml$pars$matrix %in% c('T0MEANS','CINT'))]] <- FALSE

plot(ml,wait=FALSE) #plot prior distributions

#fit
f1 <- ctStanFit(datalong = dat, ctstanmodel = m1, optimize=TRUE,
                 cores = setcores,chains = setchains,plot=TRUE,
                 control=list(max_treedepth=7),iter=150)

summary(f1)

#plots of individual subject models v data
ctKalman(f1,timestep=.01,plot=TRUE,subjects=1:2,kalmanvec=c('y','yprior'))
ctStanPlotPost(f1, wait=FALSE) #compare prior to posterior distributions
ctStanPlotPost(f1, priorwidth = FALSE, wait=FALSE) #rescale to width of posterior

ctStanPostPredict(f1, wait=FALSE) #compare randomly generated data from posterior to observed data
cfc<-ctCheckFit(f1) #compare covariance of randomly generated data to observed cov plot(cfc,wait=FALSE)

#Specify model including dynamics
m2 <- ctModel(n.manifest = 1,n.latent = 1,n.TIpred = 1, type = 'stanct', manifestNames = c('Y1'), latentNames=c('L1'),TIpredNames = 'age', DRIFT=matrix('drift11',nrow=1,ncol=1), DIFFUSION=matrix('diffusion11',nrow=1,ncol=1), CINT=matrix(c('cint1'),ncol=1), TOMEANS=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))

m2$pars$indvarying <- TRUE
m2$pars$indvarying[which(m2$pars$matrix %in% c('TOMEANS','CINT'))] <- FALSE
m2$pars$age_effect[which(m2$pars$matrix %in% c('TOMEANS','CINT'))] <- FALSE

f2 <- ctStanFit(datalong = dat, ctstanmodel = m2, optimize=TRUE, cores = setcores, #chains = setchains,plot=TRUE, control=list(max_treedepth=7),iter=150)

summary(f2,parmatrices=TRUE,timeinterval=1)

ctKalman(f2,timestep=.01,plot=TRUE,subjects=1,kalmanvec=c('y','etaprior'))

ctStanPlotPost(f2)

ctStanPostPredict(f2,wait=FALSE)

#Include intervention
m3 <- ctModel(n.manifest = 1,n.latent = 1,n.TIpred = 1, type = 'stanct', manifestNames = c('Y1'), latentNames=c('L1'),TIpredNames = 'age', 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('diffusion11',nrow=1,ncol=1), CINT=matrix(c('cint1'),ncol=1), TOMEANS=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))

m3$pars$indvarying <- TRUE
m3$pars$indvarying[!which(m3$pars$matrix %in% c('T0MEANS','CINT','TDPREDEFFECT'))] <- FALSE
m3$pars$age_effect[!which(m3$pars$matrix %in% c('T0MEANS','CINT','TDPREDEFFECT'))] <- FALSE

f3 <- ctStanFit(datalong = dat2, ctstanmodel = m3, optimize=TRUE, 
cores = setcores, 
chains = setchains,plot=TRUE, 
control=list(max_treedepth=7),iter=150)

summary(f3,parmatrices=TRUE)

ctkalman(f3,timestep=.01,plot=TRUE,subjects=1,kalmanvec=c('y','etaprior'))

cTStanPlotPost(f3,wait=FALSE)
cTStanPostPredict(f3, datarows=1:100, wait=FALSE)

#include 2nd latent process

# use either full explicit specification
m4 <- ctModel(n.manifest = 2,n.latent = 2,n.TIpred = 1, type = 'stanct', # no of vars updated
manifestNames = c('Y1','Y2'), latentNames=c('L1','L2'),TIpredNames = 'age',
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','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('t0var1','t0var2','t0var1','t0var2'),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))

# restrict between subjects variation / covariate effects
m4$pars$indvarying <- TRUE
m4$pars$indvarying[!which(m4$pars$matrix %in% c('T0MEANS','CINT','TDPREDEFFECT'))] <- FALSE
m4$pars$age_effect[!which(m4$pars$matrix %in% c('T0MEANS','CINT','TDPREDEFFECT'))] <- FALSE

# or rely on defaults (MANIFESTMEANS now free instead of CINT --
# no substantive difference for one indicator factors)

m4 <- ctModel(n.manifest = 2,n.latent = 2,n.TIpred = 1, type = 'stanct',
manifestNames = c('Y1','Y2'), latentNames=c('L1','L2'),TIpredNames = 'age',
n.TDpred=1,TDpredNames = 'TD1',
MANIFESTMEANS=matrix(c(0,0),nrow=2,ncol=1),
MANIFESTVAR=matrix(c('merror1',0,0,'merror2'),nrow=2,ncol=2))
LAMBDA = matrix(c(1,0,0,1),nrow=2,ncol=2)

# restrict between subjects variation / covariate effects
m4$pars$indvarying <- TRUE
m4$pars$indvarying[-which(m4$pars$matrix %in% c('T0MEANS','MANIFESTMEANS','TDPREDEFFECT'))] <- FALSE
m4$pars$age_effect[-which(m4$pars$matrix %in% c('T0MEANS','MANIFESTMEANS','TDPREDEFFECT'))] <- FALSE

f4 <- ctStanFit(datalong = dat2, ctstanmodel = m4, cores = setcores,
chains = setchains,plot=TRUE,
optimize=TRUE,verbose=0,
control=list(max_treedepth=7),iter=150)
summary(f4)

cTStanDiscretePars(f4,plot=TRUE) # auto and cross regressive plots over time
cTKalman(f4,timestep=.01,plot=TRUE,subjects=1,kalmanvec=c('y','etaprior'))
cTKalman(f4,timestep=.01,plot=TRUE,subjects=1:2,kalmanvec=c('y','etaprior'))
cTStanPlotPost(f4, wait=FALSE)
cTStanPostPredict(f4, wait=FALSE)

# non-linear dependencies - based on m3 model (including intervention)
# specify intervention as dependent on extra parameter in PARS matrix, and latent process 1

m3nl <- ctModel(n.manifest = 1,n.latent = 1,n.TIpred = 1, type = 'stanct',
manifestNames = c('Y1'), latentNames=c('L1'),TIpredNames = 'age',
n.TDpred=1,TDpredNames = 'TD1',
TDPREDEFFECT=matrix(c('PARS[1,1] * state[1]'),nrow=1,ncol=1),
PARS=matrix(c('tdpredeffect'),1,1),
DRIFT=matrix('drift11',nrow=1,ncol=1),
DIFFUSION=matrix('diffusion11',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))

m3nl$pars$indvarying <- TRUE
m3nl$pars$indvarying[-which(m3nl$pars$matrix %in% c('T0MEANS','CINT','TDPREDEFFECT'))] <- FALSE
m3nl$pars$age_effect[-which(m3nl$pars$matrix %in% c('T0MEANS','CINT','TDPREDEFFECT'))] <- FALSE

# here fit using optimization instead of sampling -- not appropriate in all cases!
f3nl <- ctStanFit(datalong = dat2, ctstanmodel = m3nl,
cores = setcores, chains = setchains,
optimize=TRUE)
summary(f3nl)

#generate data from fitted model and add to $generated subobject
f3nl <- ctStanGenerateFromFit(f3nl, nsamples=100, fullposterior=TRUE)

#plot functions need updating for non-linearities! (as of ctsem v 2.7.3)
#extract can be used to extract samples and create own plots.
#ctStanKalman can be used to obtain predictions, errors, updated states etc
ctStanPostPredict(f3nl, datarows=1:100, wait=FALSE)
ctKalman(f3nl, plot=TRUE)

## End(Not run)

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**

```r
cr StanGenerateFromFit(fit, nsamples = 1, fullposterior = FALSE)
```

**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.

**Value**

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

**Examples**

```r
gen <- ctStanGenerateFromFit(ctstantestfit, nsamples=3, fullposterior=TRUE)
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

cfStanKalman(fit, nsamples = NA, collapsefunc = NA, cores = 2, ...)

Arguments

fit    fit object from ctStanFit.
nsamples either NA (to extract all) or a positive integer from 1 to maximum samples in
the fit.
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.
... additional arguments to collapsefunc.

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)

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

Description

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

Usage

cfStanModel(ctmodelobj, type = "stanct")
**ctStanParMatrices**

**Arguments**

- `ctmodelobj`: ctsem model object of type 'omx' (default)
- `type`: either 'stanct' for continuous time, or 'standt' for discrete time.

**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**

```r
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)
```

**ctStanParMatrices**  
Returns population system matrices from a ctStanFit object, and vector of values for free parameters.

**Description**

Returns population system matrices from a ctStanFit object, and vector of values for free parameters.

**Usage**

```r
ctStanParMatrices(fit, parvalues, timeinterval = 1, sf = NA)
```
Arguments

- **fit**: ctStanFit object.
- **parvalues**: vector of parameter values to assign to free parameters in the model.
- **timeinterval**: time interval to use for discrete time (dt) matrix calculations.
- **sf**: stanfit object. Generally not necessary, but for repeated calls to this function, can speed things up.

Value

A list containing various matrices related to a continuous time dynamic model. Matrices with "dt" in front refers to discrete time, "asym" refers to asymptotic (time interval = infinity), and "cor" refers to correlations.

Examples

```r
cTStanParMatrices(ctstantestfit,rnorm(17,0,.1))
```

---

**Description**

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

**Usage**

```r
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**

```r
cTStanParnames(ctstantestfit,substrings=c('pop_','popsd'))
```
ctStanPlotPost

Description

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

Usage

ctStanPlotPost(obj, rows = "all", priorwidth = TRUE, mfrow = "auto", lwd = 2, smoothness = 1, parcontrol = list(mgp = c(1.3, 0.5, 0), mar = c(3, 2, 2, 1) + 0.2), 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.
- **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.
- **mfrow**: 2 dimensional integer vector defining number of rows and columns of plots, as per the mfrow argument to `par`. 'auto' determines automatically, to a maximum of 4 by 4, while NULL uses the current system setting.
- **lwd**: line width for plotting.
- **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.
- **parcontrol**: parameters to pass to `par` which temporarily change plot settings.
- **wait**: If true, user is prompted to continue before plotting next graph. If false, graphs are plotted one after another without waiting.

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, legend = TRUE, diffsize = 1, jitter = 0.02, wait = TRUE, probs = c(0.025, 0.5, 0.975), shading = TRUE, datarows = "all", nsamples = 500, ...)

Arguments

- **fit**: ctStanFit object.
- **legend**: Logical, whether to plot a legend.
- **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, 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.
- **shading**: Logical – show smoothed shading over generated data points? Otherwise, plot shaded polygon based on quantile estimate. Shading is better for non-linearities.
- **datarows**: integer vector specifying rows of data to plot. Otherwise 'all' uses all data.
- **nsamples**: Number of datasets to generate for comparisons.
- **...**: extra arguments to pass to plot function.

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.

Examples

cTStanPostPredict(ctstantestfit, wait=FALSE, shading=FALSE, datarows=1:25, diffsize=2)
ctstantestdat

**Description**

Generated dataset for testing `ctStanFit` from ctsem package.

**Format**

matrix

ctstantestfit

**Description**

Minimal output from `ctStanFit` from ctsem package.

**Format**

stanfit class.

ctStanTIpred.effects

*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**

```r
ctStanTIpred.effects(fit, returndifference = FALSE, probs = c(0.025, 0.5, 0.975), includeMeanUncertainty = FALSE, whichTIpreds = 1, parmatrices = TRUE, whichpars = "all", nsamples = 100, timeinterval = 1, nsubjects = 50, 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**
umerical 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 ttipreds 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), the ctStanParMatrices function is used to return an expanded range of possible matrices of interest.

**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. Character string ‘all’ can also be used. Time taken 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.

... arguments to pass to ctPlotArray for plotting.
ctStanTIpredMarginal

Value

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

Examples

#samples reduced here for speed
cTStanTIpredEffects(ctstantestfit, plot=TRUE, whichpars='CINT', nsamples=10, nsubjects=10)

cTStanTIpredMarginal

Plot marginal relationships between covariates and parameters for a ctStanFit object.

Description

Plot marginal relationships between covariates and parameters for a ctStanFit object.

Usage

cTStanTIpredMarginal(fit, tipred, pars, probs = c(0.025, 0.5, 0.975), useimputed = TRUE, plot = TRUE)

Arguments

- **fit**: ctStanFit object.
- **tipred**: Integer representing which tipred to use – integer corresponds to TIpredNames specification.
- **pars**: Subject level matrices from the ctStanFit output – e.g, 'DRIFT' or 'DIFFUSION'.
- **probs**: vector of 3 quantile probabilities, the 2nd will be plotted as a line, the outer two as shaded regions.
- **useimputed**: Logical, include imputed tipreds or only observed?
- **plot**: Logical, whether to plot.

Value

If plot=TRUE, nothing, otherwise an array that can be used with ctPlotArray.

Examples

cTStanTIpredMarginal(ctstantestfit, pars='CINT', tipred=3)
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**

```r
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

**Examples**

```r
newm <- ctModel(type = 'stanct',
    n.latent = ctstantestfit$ctstanmodel$n.latent,
    n.TDpred = ctstantestfit$ctstanmodel$n.TDpred,
    n.TIpred = ctstantestfit$ctstanmodel$n.TIpred,
    MANIFESTVAR = matrix(c('merror', 0, 0, 'merror'), 2, 2),
    MANIFESTMEANS = matrix(0, nrow = ctstantestfit$ctstanmodel$n.manifest),
    CINT = matrix(c(0, 'cint2'), ncol = 1),
    n.manifest = ctstantestfit$ctstanmodel$n.manifest,
    LAMBDA = diag(2))
newdat <- ctstantestdat
newdat <- newdat[newdat[, 'id'] != 1,]
newfit <- ctStanUpdModel(ctstantestfit, newdat, newm)
```

---

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

```r
ctwWideToLong = ctwWideToLong
```

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.

---

**Description**

`ctwWideToLong` Convert ctssem wide to long format

Usage

```r
ctwWideToLong = ctwWideToLong
```

Arguments

- `datawide`: ctssem 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

```r
# First load the example ctsem wide format data with absolute times
data('datastructure')
datastructure # contains two time intervals (dTx), therefore 3 time points.
# Then convert to long format
longexample <- ctWideToLong(datawide = datastructure, Tpoints=3,
n.manifest=3, manifestNames = c("Y1", "Y2", "Y3"),
n.TDpred=1, TDpredNames = "TD1",
n.TIpred=2, TIpredNames = c("TI1", "TI2"))

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

**datastructure**

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.

**extract**

*Extract samples from a ctStanFit object*

**Description**

Extract samples from a ctStanFit object

**Usage**

```r
extract(object, ...)
```
Arguments

object ctStanFit object, samples may be from Stan’s HMC, or the importance sampling approach of ctsem.

... additional arguments to pass to rstan::extract.

Value

Array of posterior samples.

Examples

e = extract(ctstantestfit)

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
Kalman

Value

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

Examples

```r
# 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', n.latent=2, n.manifest=1,
  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),
  DIFFUSION=matrix(c(0.0001, 0, 0, "diffusion"), ncol=2, nrow=2))

# fit and plot importance sampling diagnostic
fit <- ctStanFit(datalong, model, chains=1,
  optimcontrol=list(isloops=5,finishsamples=500),optimize=TRUE)
isdiag(fit)
```

Description

Takes list containing ctsem subject matrices, as well as long form data object, and calculates predicted and updated latent states, likelihoods, and predicted observations using the Kalman filter.

Usage

```r
Kalman(kpars, datalong, manifestNames, latentNames,
  imputeMissings = FALSE, TDpredNames = NULL, continuoustime = TRUE,
  idcol = "id", timecol = "time", derrind = "all",
  optimize = FALSE, ukf = FALSE, plotoptim = FALSE)
```
Arguments

kpars  list object containing `DRIFT, T0VAR, DIFFUSION, CINT, T0MEANS, TDPREDEFFECT, MANIFESTMEANS, LAMBDA,` and `MANIFESTVAR` matrices, with list elements named accordingly. Such a list is returned by `ctStanContinuousPars`.

datalong  long format data object as used by `ctStanFit`, but must contain only a single subjects’ data and does not need an id column.

manifestNames  String vector of names of manifest variables to use from datalong.

latentNames  String vector of names of latent variables.

imputeMissings  Logical. If TRUE, randomly generate any missing observations of manifest variables according to model.

TDpredNames  If model contains time dependent predictors, string vector of their names in the data.

continuousTime  Logical, whether to use a continuous time Kalman filter or discrete time. Refers only to latent states, observations are always at discrete time points.

idcol  Character string giving name of subject identification column in data.

timecol  name of time column in datalong. Note that time column must be an ascending sequence of numeric values from row 1 to row n. Ignored if continuousTime=FALSE.

derrind  vector of integers denoting which latent variables are involved in covariance calcs.

optimize  Set to TRUE when using for optimization.

ukf  set to TRUE to use the unscented Kalman filter, only necessary for fitting non-linear models, currently only for optimizing.

plotoptim  set to TRUE to plot / print optimization steps.

Value

When optimize=TRUE, returns log likelihood. Else, returns a list containing matrix objects etaprior, etaupd, etasmooth, y, yprior, yupd, ysmooth, preerror, 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.

Examples

```r
### ctstantestfit is a dummy ctStanFit object with 2 manifest indicators, 4 latents, and 1 time dependent predictor.
### get parameter matrices
kpars <- ctStanContinuousPars(ctstantestfit)

#construct dummy data
data long <- cbind(0:9, 1, matrix(rnorm(20,2,1),ncol=2))
```
datalong[c(1:3,9:10),3:4]<-NA #missing data to pre/fore cast
colnames(datalog) <- c('time', 'id', paste0('Y',1:2))
print(datalog)

#obtain Kalman filtered estimates
kout <- Kalman(kpars=kpars, datalong=datalog,
manifestNames=paste0('Y',1:nrow(kpars$MANIFESTMEANS)),
latentNames=paste0('eta',1:nrow(kpars$DRIFT)))

#print and plot smoothed estimates (conditional on all states) of indicators.
print(kout$ysmooth)
matplot(kout$time,kout$ysmooth,type='l')
matplot(kout$time,datalog[,3:4],type='p',add=TRUE,pch=1)

---

**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.

---

**msquare**

Right multiply a matrix by its transpose.

**Description**

Right multiply a matrix by its transpose.

**Usage**

msquare(x)

**Arguments**

x matrix.

**Value**

matrix.

**Examples**

msquare(t(chol(diag(3,4)+1)))
Description

Simulated example dataset for the ctsem package.

Format

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

Source


plot.ctsemFit  Plotting function for object class ctsemFit

Description

Outputs mean trajectories, autoregression, and crossregression plots. For more customization possibilities, see ctPlot.

Usage

```r
## S3 method for class 'ctsemFit'
plot(x, resolution = 50, wait = TRUE,
     max.time = "auto", mean = TRUE, withinVariance = TRUE, AR = TRUE,
     CR = TRUE, standardiseCR = FALSE, randomImpulse = FALSE,
     experimentalImpulse = FALSE, xlab = "Time", meansylim = "auto",
     ARylim = "auto", CRylim = "auto", ylab = "Value", ...)
```

Arguments

- **x**: ctsemFit object as generated by ctFit.
- **resolution**: Numeric. Plot points between each unit of time. Default of ‘auto’ adapts to max.time and results in 500 in total.
- **wait**: If true, user is prompted to continue before plotting next graph. If false, graphs are plotted one after another without waiting.
- **max.time**: Time scale on which to plot parameters. If auto, parameters are plotted for full range of observed variables.
- **mean**: If TRUE, plot of means from 0 to max.time included in output.
withinVariance
if TRUE, plot within subject variance / covariance.

AR
if TRUE, plot of autoregressive values from 0 to max.time included in output.

CR
if TRUE, plot of cross regressive values from 0 to max.time included in output.

standardiseCR
if TRUE, cross regression values are standardised based on estimated within subject variance.

randomImpulse
if TRUE (default), plots expected change in processes given a random fluctuation of +1 for each process – plot is then a mixture of DIFFUSION and DRIFT characteristics.

experimentalImpulse
if TRUE (default), plots expected change in processes given an exogenous input of +1 for each process – alternate characterisation of autoregressive and cross regressive plots.

xlab
X axis label.

meansylim
Vector of min and max limits for mean trajectory plot. 'auto' calculates automatically.

ARylim
Vector of min and max limits for autoregression plot. 'auto' is c(0,1), and expands if necessary.

CRylim
Vector of min and max limits for cross regression plot. 'auto' is c(-1,1), and expands if necessary.

ylab
Y axis label.

... Other options passed to plot().

Value

Examples

## Examples set to 'donttest' because they take longer than 5s.

### example from Driver, Oud, Voelkle (2015),
### simulated happiness and leisure time with unobserved heterogeneity.

data(ctExample1)
traitmodel <- ctModel(n.manifest=2, n.latent=2, Tpoints=6, LAMBDA=diag(2),
manifestNames=c('LeisureTime', 'Happiness'),
latentNames=c('LeisureTime', 'Happiness'), TRAITVAR="auto")
traitfit <- ctFit(dat=ctExample1, cmodelobj=traitmodel)
plot(traitfit, wait=FALSE)
**Description**

Misspecification plot using ctCheckFit output

**Usage**

```r
## S3 method for class 'ctsemFitMeasure'
plot(x, indices = "all", means = TRUE,
     separatemeans = TRUE, cov = TRUE, covtype = "MisspecRatio",
     cov2cor = FALSE, wait = TRUE, ggcorrArgs = list(data = NULL,
     cor_matrix = get(covtype), limits = limits, geom = "circle", max_size =
     10, name = covtype), ...)
```

**Arguments**

- `x` Object output from ctCheckFit function.
- `indices` Either 'all' or a vector of integers denoting which observations to include (from 1 to n.manifest * maximum number of obs for a subject, blocked by manifest).
- `means` Logical – plot simulated means vs observed?
- `separatemeans` Logical – means from different variables on same or different plots?
- `cov` Logical – plot simulated cov vs observed?
- `covtype` Column name of $cov$ sub object
- `cov2cor` Logical – convert covariances to correlations?
- `wait` Logical – wait for input before new plot?
- `ggcorrArgs` List of arguments to GGally::ggcorr .

**Value**

Nothing, just plots.

**Examples**

```r
data(ctExample1)
traitmodel <- ctModel(n.manifest=2, n.latent=2, Tpoints=6, LAMBDA=diag(2),
                      manifestNames=c('LeisureTime', 'Happiness'),
                      latentNames=c('LeisureTime', 'Happiness'), TRAITVAR="auto")
traitfit <- ctFit(dat=ctExample1, ctmodelobj=traitmodel)
```
check <- ctCheckFit(traitfit,niter=50)
plot(check)

scheck <- ctCheckFit(ctstantestfit,niter=500)
plot(scheck,wait=FALSE)

---

plot.ctsemMultigroupFit

*Plot function for ctsemMultigroupFit object*

**Description**

Plots *ctMultigroupFit* objects.

**Usage**

```r
## S3 method for class 'ctsemMultigroupFit'
plot(x, group = "show chooser", ...)
```

**Arguments**

- `x` : *ctsemMultigroupFit* object as generated by *ctMultigroupFit*
- `group` : character string of subgroup to plot. Default of 'show chooser' displays list and lets you select.
- `...` : additional parameters to pass to *plot.ctsemFit* function.

**Value**


---

plot.ctStanFit

*plot.ctStanFit*

**Description**

Plots for *ctStanFit* objects.

**Usage**

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

**Arguments**

- `x` : *ctStanFit* object
- `types` : character vector of plot types to plot. Default of 'all' plots all types.
- `wait` : logical; if TRUE, waits for input before continuing.
- `...` : additional parameters to pass to *plot.ctsemFit* function.
plot.ctStanModel

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)

### example plot using rstan functions
rstan::stan_trace(ctstantestfit$stanfit,
pars=ctStanParnames(ctstantestfit, 'pop_DRIFT'))

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),
 ...)
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.
- **...**: 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

```r
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)
plot(stanmodel,rows=8)
```
**sdpcor2cov**

**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, cholesky = FALSE)
```

**Arguments**

- `mat`: input square matrix with std dev on diagonal and lower tri of partial correlations.
- `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
```

---

**stanoptimis**

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

**Description**

Optimize / importance sample a stan or ctStan model.

**Usage**

```
stanoptimis(standata, sm, init = "random", initstd = 0.01,
sampleinit = NA, deoptim = FALSE, estonly = FALSE, tol = 1e-14,
decontrol = list(), stochastic = FALSE, nopriors = FALSE,
carefulfit = TRUE, plotsgd = FALSE, is = FALSE,
isloopsiz = 1000, finishsamples = 500, tdf = 10,
chancethreshold = 100, finishmultiply = 5, verbose = 0,
cores = 2)
```
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.

**nopriors**
logical. If TRUE, a nopriors 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 nopriors=TRUE.

**plotsgd**
Logical. If TRUE, plot iteration details when using stochastic optimizer.

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

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

**finishesamples**
Number of samples to draw for final results of importance sampling.

**tdf**
degrees of freedom of multivariate t distribution. Higher (more normal) generally gives more efficient 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 finishesamples * 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.

Value

list containing fit elements
Examples

```r
library(rstan)
scode <- 
  "parameters {
    real y[2];
  }
  model {
    y[1] ~ normal(0, 1);
    y[2] ~ double_exponential(0, 2);
  }"

sm <- stan_model(model_code=scode)
fit <- sampling(sm, iter = 10000)
summary(fit)$summary

## extract samples as a list of arrays
e <- extract(fit, permuted = TRUE)

# for ml or map estimates
optimis <- stanoptimis(standata = list(), sm = sm,finishsamples = 3000, cores=2)
optimis$optimfit

# for posterior distributions
optimis <- stanoptimis(standata = list(), sm = sm,finishsamples = 3000, cores=2, tdf=5)
apply(optimis$rawposterior,2,mean)
apply(optimis$rawposterior,2,sd)
isdiag(optimis)
plot(density(optimis$rawposterior[,2], bw=.05))
points(density(e$y[,2], bw=.05), type='l', col=2)
```

---

### Description

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

### Usage

```r
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

```r
### example 1
scode <- "
parameters {
  real y[2];
}
model {
  y[1] ~ normal(0, .5);
  y[2] ~ double_exponential(0, 2);
}
"
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

```r
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 contain 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

```r
library(rstan)
scode <- "
parameters {
  real y[2];
}
model {
  y[1] ~ normal(0, 1);
  y[2] ~ double_exponential(0, y[1]);
}
"
fit1 <- stan(model_code = scode, iter = 10)
stan_checkdivergences(fit1)
```

---

**stan_confidenceRegion**

*Extract functions of multiple variables from a stanfit object*

**Description**

Can be useful for determining quantiles or plotting multidimensional regions – for instance in case of colinearity of predictors.

**Usage**

```r
stan_confidenceRegion(stanfit, parstrings, prefuncstring = "",
                      joinfuncstring = " + ", postfuncstring = ")")
```

**Arguments**

- **stanfit**: object of class stanfit.
- **parstrings**: vector of strings containing partial (or full) matches of parameter names. When more than one string is passed, functions are computed based on the combination of the first match for each string, then the second match for each string, etc. The first match of the first string is only ever combined with the first match of the second, similarly for the 2nd match, etc.
- **prefuncstring**: string containing front element of function. E.g., ’exp(‘ for an exponential region.
joinfuncstring

string used to join the (possibly) multiple parameters involved.

postfuncstring

string containing end element of function. E.g., ')*2' to multiply the result by 2.

Value

matrix of values of the specified interactions at each iteration.

Examples

temp<-stan_confidenceRegion(stanfit=ctstantestfit$stanfit,
parstrings=c('pop_DRIFT[1,2]', 'pop_DRIFT[2,1]'))
t(apply(temp,2,quantile))

stan_postcalc  Compute functions of matrices from samples of a stanfit object

Description

Compute functions of matrices from samples of a stanfit object

Usage

stan_postcalc(stanfit, object, calc = "object", objectindices = "all",
summary = TRUE)

Arguments

stanfit object of class stanfit.
object name of stan sub object from stanfit to use for calculations.
calc string containing R calculation to evaluate, with the string 'object' in place of
the actual object name.
objectindices matrix of indices, with the number of columns matching the number of dimen-
sions of the object. 'all' computes which(array(1, objdims)==1, arr.ind=TRUE),
where objdims is what would be returned by dim(object) if the object existed in
the R environment.
summary if FALSE, a iterations * parameters matrix is returned, if TRUE, rstan::monitor
is first run on the output.

Value

matrix of values of the specified interactions at each iteration.
stan_unconstrainsamples

Convert samples from a stanfit object to the unconstrained scale

Examples

```r
temp<-stan_postcalc(stanfit=ctstantestfit$stanfit,
object='DRIFT', objectindices='all', calc='exp(object)')
```

Description

Convert samples from a stanfit object to the unconstrained scale

Usage

```r
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

```r
umat <- stan_unconstrainsamples(ctstantestfit$stanfit, ctstantestfit$standata)
```

summary.ctsemFit

Summary function for ctsemFit object

Description

Provides summary details for ctsemFit objects.

Usage

```r
## S3 method for class 'ctsemFit'
summary(object, ridging = FALSE, timeInterval = 1, verbose = FALSE, ...)
```
Arguments

- **object**: ctsemFit object as generated by ctFit.
- **ridging**: if TRUE, adds a small amount of variance to diagonals when calculating standardised (correlation) matrices, should only be used if standardised matrices return NAN.
- **timeInterval**: positive numeric value specifying time interval to use for discrete parameter matrices, defaults to 1.
- **verbose**: Logical. If TRUE, displays the raw, internally transformed (when fitting with default arguments) OpenMx parameters and corresponding standard errors, as well as additional summary matrices. Parameter transforms are described in the vignette, vignette('ctsem'). Additional summary matrices include: 'discrete' matrices – matrices representing the effect for the given time interval (default of 1); 'asymptotic' matrices – represents the effect as time interval approaches infinity (therefore asymCINT describes mean level of processes at the asymptote, asymDIFFUSION describes total within-subject variance at the asymptote, etc); 'standardised' matrices – transforms covariance matrices to correlation matrices, and transforms discreteDRIFT based on DIFFUSION, to give effect sizes.
- **...** additional parameters to pass.

Details

Important: Although ctModel takes cholesky decomposed variance-covariance matrices as input, the summary function displays the full variance-covariance matrices. These can be cholesky decomposed for comparison purposes using `t(chol(summary(ctfitobject)$covariancematrix))`. Standard errors are displayed in the $ctparameters section, however if ctFit was used with transformedParams=TRUE (the default, and recommended) covariance matrix standard errors will have been approximated using the delta method. For inferential purposes, maximum likelihood confidence intervals may be estimated using the ctCI function.

Value

Summary of ctsemFit object

Examples

```r
## Examples set to 'donttest' because they take longer than 5s.

### example from Driver, Oud, Voelkle (2015),
### simulated happiness and leisure time with unobserved heterogeneity.
data(ctExample1)
traitmodel <- ctModel(n.manifest=2, n.latent=2, Tpoints=6, LAMBDA=diag(2),
                        manifestNames=c('LeisureTime', 'Happiness'),
                        latentNames=c('LeisureTime', 'Happiness'), TRAITVAR="auto")
traitfit <- ctFit(dat=ctExample1, ctmodelobj=traitmodel)
summary(traitfit,timeInterval=1)
```
Summary function for ctsemMultigroupFit object

Description

Provides summary details for objects fitted with ctMultigroupFit.

Usage

## S3 method for class 'ctsemMultigroupFit'
summary(object, group = "show chooser", ...)

Arguments

object  ctsemMultigroupFit object as generated by ctMultigroupFit

group   character string of subgroup to display summary parameters for. Default of
         'show chooser' displays list and lets you select.

...   additional parameters to pass to summary.ctsemFit.

Value

Summary of ctsemMultigroupFit object

Summary function for ctStanFit object

Description

Summarise a ctStanFit object that was fit using ctStanFit.

Usage

## S3 method for class 'ctStanFit'
summary(object, timeinterval = 1, digits = 3, parmatrices = TRUE, priorcheck = 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.

...  Additional arguments to pass to ctsem:::priorcheckreport, such as meanlim, or sdlim.

Value

List containing summary items.

Examples

summary(ctstantestfit)