Package ‘dynr’

February 11, 2020

Date 2020-02-11
Title Dynamic Modeling in R
Maintainer Michael D. Hunter <mhunter.ou@gmail.com>
Depends R (>= 3.0.0), ggplot2
Imports MASS, Matrix, numDeriv, xtable, latex2exp, grid, reshape2,
   plyr, mice, magrittr, Rdpack, methods, fda, car, stringi,
   tibble, deSolve
Suggests testthat, roxygen2 (>= 3.1), knitr, rmarkdown
VignetteBuilder knitr

Description Intensive longitudinal data have become increasingly prevalent in
   various scientific disciplines. Many such data sets are noisy, multivariate,
   and multi-subject in nature. The change functions may also be continuous, or
   continuous but interspersed with periods of discontinuities (i.e., showing
   regime switches). The package ‘dynr’ (Dynamic Modeling in R) is an R package
   that implements a set of computationally efficient algorithms for handling a
   broad class of linear and nonlinear discrete- and continuous-time models with
   regime-switching properties under the constraint of linear Gaussian measurement
   functions. The discrete-time models can generally take on the form of a state-
   space or difference equation model. The continuous-time models are generally
   expressed as a set of ordinary or stochastic differential equations. All
   estimation and computations are performed in C, but users are provided with the
   option to specify the model of interest via a set of simple and easy-to-learn
   model specification functions in R. Model fitting can be performed using single-
   subject time series data or multiple-subject longitudinal data.

SystemRequirements GNU make
NeedsCompilation yes
License GPL-3
LazyLoad yes
LazyData yes

Collate 'dynrData.R' 'dynrRecipe.R' 'dynrModelInternal.R'
   'dynrModel.R' 'dynrCook.R' 'dynrPlot.R' 'dynrFuncAddress.R'
   'dynrMi.R' 'dynrTaste.R' 'dynrVersion.R' 'dataDoc.R'
   'dynrGetDerivs.R'
R topics documented:

RdMacros Rdpack
Biarch true
Version 0.1.15-25
RoxygenNote 5.0.1
Author Lu Ou [aut],
Michael D. Hunter [aut, cre] (<https://orcid.org/0000-0002-3651-6709>),
Sy-Miin Chow [aut] (<https://orcid.org/0000-0003-1938-027X>),
Lining Ji [aut],
Meng Chen [aut],
Hui-Ju Hung [aut],
Jungmin Lee [aut],
Yanling Li [aut],
Jonathan Park [aut]
Repository CRAN
Date/Publication 2020-02-11 19:10:05 UTC

R topics documented:

dynr-package .............................................................. 3
autoplot.dynrTaste ......................................................... 9
coef.dynrModel ............................................................ 9
confint.dynrCook .......................................................... 10
diag.character-method .................................................... 12
dynr.cook ................................................................. 12
dynr.data ................................................................. 14
dynr.flowField ............................................................ 15
dynr.ggplot ............................................................... 17
dynr.ldl ................................................................. 19
dynr.mi ................................................................. 20
dynr.model .............................................................. 21
dynr.plotFreq ............................................................. 22
dynr.taste ............................................................... 23
dynr.taste2 .............................................................. 25
dynr.trajecotry ........................................................... 26
dynr.version ............................................................ 28
dynrCook-class .......................................................... 29
dynrDynamics-class ....................................................... 29
dynrInitial-class ........................................................ 29
dynrMeasurement-class .................................................. 29
dynrModel-class .......................................................... 30
dynrNoise-class .......................................................... 30
dynrRecipe-class ........................................................ 30
dynrRegimes-class ......................................................... 31
dynrTrans-class .......................................................... 31
EMG ................................................................. 31
EMGSim ................................................................. 32
### dynr-package

**Description**

Intensive longitudinal data have become increasingly prevalent in various scientific disciplines. Many such data sets are noisy, multivariate, and multi-subject in nature. The change functions may also be continuous, or continuous but interspersed with periods of discontinuities (i.e., showing regime switches). The package `dynr` (Dynamic Modeling in R) is an R package that implements a set of computationally efficient algorithms for handling a broad class of linear and nonlinear models.

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getdx</td>
<td>33</td>
</tr>
<tr>
<td>internalModelPrep</td>
<td>34</td>
</tr>
<tr>
<td>LogisticSetPointSDE</td>
<td>35</td>
</tr>
<tr>
<td>logLik.dynrCook</td>
<td>36</td>
</tr>
<tr>
<td>names.dynrCook-method</td>
<td>37</td>
</tr>
<tr>
<td>names.dynrModel-method</td>
<td>38</td>
</tr>
<tr>
<td>nobs.dynrCook</td>
<td>39</td>
</tr>
<tr>
<td>nobs.dynrModel</td>
<td>39</td>
</tr>
<tr>
<td>NonlinearDFAsim</td>
<td>40</td>
</tr>
<tr>
<td>oscData</td>
<td>41</td>
</tr>
<tr>
<td>Oscillator</td>
<td>41</td>
</tr>
<tr>
<td>Outliers</td>
<td>43</td>
</tr>
<tr>
<td>PFAsim</td>
<td>45</td>
</tr>
<tr>
<td>plot.dynrCook</td>
<td>47</td>
</tr>
<tr>
<td>plotFormula</td>
<td>48</td>
</tr>
<tr>
<td>plotGCV</td>
<td>49</td>
</tr>
<tr>
<td>PPsim</td>
<td>50</td>
</tr>
<tr>
<td>prep.formulaDynamics</td>
<td>51</td>
</tr>
<tr>
<td>prep.initial</td>
<td>53</td>
</tr>
<tr>
<td>prep.loadings</td>
<td>56</td>
</tr>
<tr>
<td>prep.matrixDynamics</td>
<td>58</td>
</tr>
<tr>
<td>prep.measurement</td>
<td>59</td>
</tr>
<tr>
<td>prep.noise</td>
<td>60</td>
</tr>
<tr>
<td>prep.regimes</td>
<td>62</td>
</tr>
<tr>
<td>prep.tfun</td>
<td>64</td>
</tr>
<tr>
<td>printex</td>
<td>65</td>
</tr>
<tr>
<td>RSPPsim</td>
<td>66</td>
</tr>
<tr>
<td>substituteFormula</td>
<td>67</td>
</tr>
<tr>
<td>summary.dynrCook</td>
<td>68</td>
</tr>
<tr>
<td>theta_plot</td>
<td>68</td>
</tr>
<tr>
<td>TrueInit_Y14</td>
<td>69</td>
</tr>
<tr>
<td>VARsim</td>
<td>70</td>
</tr>
<tr>
<td>vcov.dynrCook</td>
<td>71</td>
</tr>
<tr>
<td>vdpData</td>
<td>72</td>
</tr>
</tbody>
</table>

**Index**

73
discrete- and continuous-time models with regime-switching properties under the constraint of linear Gaussian measurement functions. The discrete-time models can generally take on the form of a state-space or difference equation model. The continuous-time models are generally expressed as a set of ordinary or stochastic differential equations. All estimation and computations are performed in C, but users are provided with the option to specify the model of interest via a set of simple and easy-to-learn model specification functions in R. Model fitting can be performed using single-subject time series data or multiple-subject longitudinal data.

Details

The DESCRIPTION file:

Package: dynr
Date: 2020-02-11
Title: Dynamic Modeling in R
Authors@R: c(person("Lu", "Ou", role="aut"), person(c("Michael", "D."), "Hunter", role=c("aut", "cre"), email="mhunter.ou@gmail.com"), person("Yanling", "Li", role="aut", email=""), person("Jonathan", "Park", role="aut", email=""))
Maintainer: Michael D. Hunter <mhunter.ou@gmail.com>
Depends: R (>= 3.0.0), ggplot2
Imports: MASS, Matrix, numDeriv, xtable, grid, reshape2, plyr, mice, magrittr, Rdpack, methods, fda, car, stringi, tibble, deSolve
Suggests: testthat, roxygen2 (>= 3.1), knitr, rmarkdown
VignetteBuilder: knitr
Description: Intensive longitudinal data have become increasingly prevalent in various scientific disciplines. Many such data sets have the form of single-subject time series or multiple-subject longitudinal data. Dynr is a purpose-built R package that offers many options for fitting state-space or time series regression models to such data. Model fitting can be performed using single-subject time series data or multiple-subject longitudinal data.
SystemRequirements: GNU make
NeedsCompilation: yes
License: GPL-3
LazyLoad: yes
LazyData: yes
RdMacros: Rdpack
Biarch: true
Version: 0.1.15-25
RoxygenNote: 5.0.1
Author: Lu Ou [aut], Michael D. Hunter [aut, cre] (<https://orcid.org/0000-0002-3651-6709>), Sy-Miin Chow [aut], Ji [aut], Meng Chen [aut], Hui-Ju Hung [aut], Jungmin Lee [aut], Yanling Li [aut], Jonathan Park [aut]

Index of help topics:

EMG Single-subject time series of facial electromyography data
EMGsim Simulated single-subject time series to capture features of facial electromyography data
LinearOsc Simulated time series data for a deterministic linear damped oscillator model
LogisticSetPointSDE Simulated time series data for a stochastic linear damped oscillator model with logistic time-varying setpoints
NonlinearDFAsim Simulated multi-subject time series based on a dynamic factor analysis model with nonlinear relations at the latent level
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oscillator</td>
<td>Simulated time series data of a damped linear oscillator</td>
</tr>
<tr>
<td>Outliers</td>
<td>Simulated time series data for detecting outliers.</td>
</tr>
<tr>
<td>PFAsim</td>
<td>Simulated time series data of a multisubject process factor analysis</td>
</tr>
<tr>
<td>PPsim</td>
<td>Simulated time series data for multiple eco-systems based on a predator-and-prey model</td>
</tr>
<tr>
<td>RSPPsim</td>
<td>Simulated time series data for multiple eco-systems based on a regime-switching predator-and-prey model</td>
</tr>
<tr>
<td>TrueInit_Y14</td>
<td>Simulated multilevel multi-subject time series of a Van der Pol Oscillator</td>
</tr>
<tr>
<td>VARsim</td>
<td>Simulated time series data for multiple imputation in dynamic modeling.</td>
</tr>
<tr>
<td>autoplot.dynrTaste</td>
<td>The ggplot of the outliers estimates.</td>
</tr>
<tr>
<td>coef.dynrModel</td>
<td>Extract fitted parameters from a dynrCook Object</td>
</tr>
<tr>
<td>confint.dynrCook</td>
<td>Confidence Intervals for Model Parameters</td>
</tr>
<tr>
<td>diag,character-method</td>
<td>Create a diagonal matrix from a character vector</td>
</tr>
<tr>
<td>dynr-package</td>
<td>Dynamic Modeling in R</td>
</tr>
<tr>
<td>dynr.cook</td>
<td>Cook a dynr model to estimate its free parameters</td>
</tr>
<tr>
<td>dynr.data</td>
<td>Create a list of data for parameter estimation (cooking dynr) using 'dynr.cook'</td>
</tr>
<tr>
<td>dynr.flowField</td>
<td>A Function to plot the flow or velocity field for a one or two dimensional autonomous ODE system from the phaseR package written by Michael J. Grayling.</td>
</tr>
<tr>
<td>dynr.ggplot</td>
<td>The ggplot of the smoothed state estimates and the most likely regimes</td>
</tr>
<tr>
<td>dynr.ldl</td>
<td>LDL Decomposition for Matrices</td>
</tr>
<tr>
<td>dynr.mi</td>
<td>Multiple Imputation of dynrModel objects</td>
</tr>
<tr>
<td>dynr.model</td>
<td>Create a dynrModel object for parameter estimation (cooking dynr) using 'dynr.cook'</td>
</tr>
<tr>
<td>dynr.plotFreq</td>
<td>Plot of the estimated frequencies of the regimes across all individuals and time points based on their smoothed regime probabilities</td>
</tr>
<tr>
<td>dynr.taste</td>
<td>Detect outliers in state space models.</td>
</tr>
<tr>
<td>dynr.taste2</td>
<td>Re-fit state-space model using the estimated outliers.</td>
</tr>
<tr>
<td>dynr.trajectory</td>
<td>A Function to perform numerical integration of the chosen ODE system, for a user-specified set of initial conditions. Plots the resulting solution(s) in the phase plane. This function from the phaseR package written by Michael J. Grayling.</td>
</tr>
</tbody>
</table>
dynr.version

dynrCook-class

dynrDynamics-class

dynrInitial-class

dynrMeasurement-class

dynrModel-class

dynrNoise-class

dynrRecipe-class

dynrRegimes-class

dynrTrans-class

getdx

internalModelPrep

logLik.dynrCook

names,dynrCook-method

names,dynrModel-method

nobs.dynrCook

nobs.dynrModel

oscData

plot.dynrCook

plotFormula

plotGCV

prep.formulaDynamics

prep.initial

prep.loadings

prep.matrixDynamics

prep.measurement

prep.noise

prep.regimes

prep.tfun

A wrapper function to call functions in the fda
package to obtain smoothed estimated
derivatives at a specified order

Do internal model preparation for dynr

Extract the log likelihood from a dynrCook
Object

Extract the free parameter names of a dynrCook
object

Extract the free parameter names of a dynrModel
object

Extract the number of observations for a
dynrCook object

Extract the number of observations for a
dynrModel object

Another simulated multilevel multi-subject time
series of a damped oscillator model

Plot method for dynrCook objects

Plot the formula from a model

A function to evaluate the generalized
cross-validation (GCV) values associated with
derivative estimates via Bsplines at a range of
specified smoothing parameter (lambda) values

Recipe function for specifying dynamic
functions using formulas

Recipe function for preparing the initial
conditions for the model.

Recipe function to quickly create factor
loadings

Recipe function for creating Linear Dynamcis
using matrices

Prepare the measurement recipe

Recipe function for specifying the measurement
error and process noise covariance structures

Recipe function for creating regime switching
(Markov transition) functions

Create a dynrTrans object to handle the
transformations and inverse transformations of
model parameters

printex  The printex Method
substituteFormula  A internal-use only function for substituting
                     formula. If the RHS of 'formula' has terms in
                     the LHS of 'term.formula', this function
                     replaces any appearance with the RHS of
                     'term.formula'

summary.dynrCook  Get the summary of a dynrCook object
theta_plot  A function to plot simple slopes and region of
            significance.
v cov.dynrCook  Extract the Variance-Covariance Matrix of a
                dynrCook object
vdpData  Another simulated multilevel multi-subject time
          series of a Van der Pol Oscillator

Because the dynr package compiles C code in response to user input, more setup is required for the dynr package than for many others. We acknowledge that this additional setup can be bothersome, but we believe the ease of use for the rest of the package and the wide variety of models it is possible to fit with it will compensate for this initial burden. Hopefully you will agree!

See the installation vignette referenced in the Examples section below for installation instructions.

The naming convention for dynr exploits the pronunciation of the package name, dynr, pronounced the same as “dinner”. That is, the names of functions and methods are specifically designed to relate to things done surrounding dinner, such as gathering ingredients (e.g., the data), preparing recipes, cooking, and serving the finished product. The general procedure for using the dynr package can be summarized in five steps as below.

1. Data are prepared using with the dynr.data() function.

2. Recipes are prepared. To each part of a model there is a corresponding prep.*() recipe function. Examples of such prep.*() functions include: prep.measurement(), prep.matrixDynamics(), prep.formulaDynamics(), prep.initial(), prep.noise(), and prep.regimes().

3. The function dynr.model() mixes the data and recipes together into a model object of class dynrModel.

4. The model is cooked with dynr.cook().

5. Results from model fitting and related estimation are served using functions such as summary(), plot(), dynr.ggplot() (or its alias autoplot()), plotFormula(), and printex().

Note

State-space modeling, dynamic model, differential equation, regime switching, nonlinear

Author(s)

NA

Maintainer: Michael D. Hunter <mhunter.ou@gmail.com>
References


See Also

For other annotated tutorials using the dynr package see https://quantdev.ssri.psu.edu/resources/what%E2%80%99s-dynr-package-linear-and-nonlinear-dynamic-modeling-r

Examples

```r
# For installation instructions see the package vignette below
# Not run:
# vignette(package='dynr', 'InstallationForUsers')

# End(Not run)
# This should open a pdf/html file to guide you through proper
# installation and configuration.

# For illustrations of the functions in dynr, check out some of the demo examples in:
# Not run:
# demo(package='dynr')

# End(Not run)

# For example, to run the demo 'LinearSDE' type
# the following without the comment character (#) in front of it.
# Not run:
# demo('LinearSDE', package='dynr')

# End(Not run)
```
autplot.dynrTaste

The ggplot of the outliers estimates.

Description

The ggplot of the outliers estimates.

Usage

```r
## S3 method for class 'dynrTaste'
autplot(object, numSubjDemo = 2, idtoPlot = NULL,
          names.state = NULL, names.observed = NULL, ...)
```

Arguments

- **object**: A dynrTaste object.
- **numSubjDemo**: The number of subjects, who have largest joint chi-square statistic, to be selected for plotting.
- **idtoPlot**: Values of the ID variable to plot.
- **names.state**: (optional) The names of the states to be plotted, which should be a subset of the state.names slot of the measurement slot of dynrModel. If NULL, the t statistic plots for all state variables will be included.
- **names.observed**: (optional) The names of the observed variables to be plotted, which should be a subset of the obs.names slot of the measurement slot of dynrModel. If NULL, the t statistic plots for all observed variables will be included.
- **...**: Place holder for other arguments. Please do not use.

Value

A list of ggplot objects for each ID. The plots of chi-square statistics (joint and independent), and the plots of t statistic for names.state and names.observed will be included. Users can modify the ggplot objects using ggplot grammar. If a filename is provided, a pdf of plots will be saved additionally.

---

coeff.dynrModel

Extract fitted parameters from a dynrCook Object

Description

aliases coef.dynrModel coef<- coef<-.dynrModel
Usage

```r
## S3 method for class 'dynrModel'
coef(object, ...)

coef(object) <- value

## S3 replacement method for class 'dynrModel'
coef(object) <- value

## S3 method for class 'dynrCook'
coef(object, ...)
```

Arguments

- `object`: The `dynrCook` object for which the coefficients are desired
- `...`: further named arguments, ignored for this method
- `value`: values for setting

Value

A numeric vector of the fitted parameters.

See Also

Other S3 methods `logLik.dynrCook`

Examples

```r
# Let cookedModel be the output from dynr.cook
# coef(cookedModel)
```

---

**confint.dynrCook**

Confidence Intervals for Model Parameters

Description

Confidence Intervals for Model Parameters

Usage

```r
## S3 method for class 'dynrCook'
confint(object, parm, level = 0.95,
        type = c("delta.method", "endpoint.transformation"),
        transformation = NULL, ...)
```
Arguments

object a fitted model object
parm which parameters are to be given confidence intervals
level the confidence level
type The type of confidence interval to compute. See details. Partial name matching is used.
transformation For type='endpoint.transformation' the transformation function used.
... further named arguments. Ignored.

Details

The parm argument can be a numeric vector or a vector of names. If it is missing then it defaults to using all the parameters.

These are Wald-type confidence intervals based on the standard errors of the (transformed) parameters. Wald-type confidence intervals are known to be inaccurate for variance parameters, particularly when the variance is near zero (See references for issues with Wald-type confidence intervals).

Value

A matrix with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 as a percentage (e.g. by default 2.5

References


Examples

# Let cookedModel be the output from dynr.cook
#confint(cookedModel)
diag.character-method  
*Create a diagonal matrix from a character vector*

**Description**  
Create a diagonal matrix from a character vector

**Usage**  
```r  
## S4 method for signature 'character'  
diag(x = 1, nrow, ncol)  
```

**Arguments**  
- `x`: Character vector used to create the matrix  
- `nrow`: Numeric. Number of rows for the resulting matrix.  
- `ncol`: Numeric. Number of columns for the resulting matrix.

**Details**  
The default behavior for missing `nrow` and/or `ncol` arguments is the same as for the `diag` function in the base package. Off-diagonal entries are filled with "0".

**Examples**  
```r  
diag(letters[1:3])  
```

**dynr.cook**  
*Cook a dynr model to estimate its free parameters*

**Description**  
Cook a dynr model to estimate its free parameters

**Usage**  
```r  
dynr.cook(dynrModel, conf.level = 0.95, infile, optimization_flag = TRUE,  
hessian_flag = TRUE, verbose = TRUE, weight_flag = FALSE,  
debug_flag = FALSE)  
```
Arguments

dynrModel a dynr model compiled using dynr.model, consisting of recipes for submodels, starting values, parameter names, and C code for each submodel
conf.level a cumulative proportion indicating the level of desired confidence intervals for the final parameter estimates (default is .95)
infile (not required for models specified through the recipe functions) the name of a file that has the C codes for all dynr submodels for those interested in specifying a model directly in C
optimization_flag a flag (TRUE/FALSE) indicating whether optimization is to be done.
hessian_flag a flag (TRUE/FALSE) indicating whether the Hessian matrix is to be calculated.
verbose a flag (TRUE/FALSE) indicating whether more detailed intermediate output during the estimation process should be printed
weight_flag a flag (TRUE/FALSE) indicating whether the negative log likelihood function should be weighted by the length of the time series for each individual
debug_flag a flag (TRUE/FALSE) indicating whether users want additional dynr output that can be used for diagnostic purposes

Details

Free parameter estimation uses the SLSQP routine from NLOPT.

The typical items returned in the cooked model are the filtered and smoothed latent variable estimates. \texttt{eta_smooth_final}, \texttt{error_cov_smooth_final} and \texttt{pr_t_given_T} are respectively time-varying smoothed latent variable mean estimates, smoothed error covariance estimates, and smoothed regime probability. \texttt{eta_filtered}, \texttt{error_cov_filtered} and \texttt{pr_t_given_t} are respectively time-varying filtered latent variable mean estimates, filtered error covariance matrix estimates, and filtered regime probability. Note that if \texttt{theta.formula} is provided in \texttt{dynrModel@dynamics}, this assumes that random effects are present in the dynamic equation. This would call an internal function to insert the random effect components as additional state variables. In this case, the last set of elements (rows) in \texttt{eta_smooth_final} would contain the estimated random effect components.

When \texttt{debug_flag} is \texttt{TRUE}, then additional information is passed into the cooked model. \texttt{eta_predicted}, \texttt{error_cov_predicted}, \texttt{innov_vec}, and \texttt{residual_cov} are respectively time-varying predicted latent variable mean estimates, predicted error covariance matrix estimates, the error/residual estimates (innovation vector), and the error/residual covariance matrix estimates.

The exit flag given after optimization has finished is from the SLSQP optimizer. Generally, error codes have negative values and successful codes have positive values. However, codes 5 and 6 do not indicate the model converged, but rather simply ran out of iterations or time, respectively. A more full description of each code is available at \url{http://ab-initio.mit.edu/wiki/index.php/NLopt_Reference#Return_values} and is also listed in the table below.

<table>
<thead>
<tr>
<th>NLOPT Term</th>
<th>Numeric Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>1</td>
<td>Generic success return value.</td>
</tr>
<tr>
<td>STOPVAL_REACHED</td>
<td>2</td>
<td>Optimization stopped because stopval (above) was reached.</td>
</tr>
<tr>
<td>FTOL_REACHED</td>
<td>3</td>
<td>Optimization stopped because ftol_rel or ftol_abs (above) was reached.</td>
</tr>
<tr>
<td>XTOL_REACHED</td>
<td>4</td>
<td>Optimization stopped because xtol_rel or xtol_abs (above) was reached.</td>
</tr>
</tbody>
</table>
MAXEVAL_REACHED 5 Optimization stopped because maxeval (above) was reached.
MAXTIME_REACHED 6 Optimization stopped because maxtime (above) was reached.
FAILURE -1 Generic failure code.
INVALID_ARGS -2 Invalid arguments (e.g. lower bounds are bigger than upper bounds, an unknown algorithm was specified, etcetera).
OUT_OF_MEMORY -3 Ran out of memory.
ROUNDOFF_LIMITED -4 Halted because roundoff errors limited progress. (In this case, the optimization still typically returns a useful result.)
FORCED_STOP -5 Halted because of a forced termination: the user called nlopt_force_stop(opt) on the optimization's nlopt_opt object opt from the user's objective function or constraints.
NONFINITE_FIT -6 Fit function is not finite (i.e., is NA, NaN, Inf or -Inf).

The last row of this table corresponding to an exit code of -6, is not from NLOPT, but rather is specific to the dynr package.

See Also

autoplot, coef, confint, deviance, initialize, logLik, names, nobs, plot, print, show, summary, vcov.

Examples

#fitted.model <- dynr.cook(model)

dynr.data(dataframe, id = "id", time = "time", observed, covariates)

Description

Create a list of data for parameter estimation (cooking dynr) using dynr.cook

Usage

dynr.data(dataframe, id = "id", time = "time", observed, covariates)

Arguments

dataframe either a "ts" class object of time series data for a single subject or a data frame object of data for potentially multiple subjects that contain a column of subject ID numbers (i.e., an ID variable), a column indicating subject-specific measurement occasions (i.e., a TIME variable), at least one column of observed values, and any number of covariates. If the data are fit to a discrete-time model, the TIME variable should contain subject-specific sequences of (subsets of) consecutively equally spaced numbers (e.g., 1, 2, 3, ...). That is, the program assumes that the input dataframe is equally spaced with potential missingness. If the measurement occasions for a subject are a subset of an arithmetic sequence but are not consecutive, NAs will be inserted automatically to create an equally spaced data set before estimation. If the data are fit to a continuous-time model,
the TIME variables can contain subject-specific increasing sequences of irregularly spaced real numbers. Missing values in the observed variables should be indicated by NA. Missing values in the covariates are not allowed. That is, missing values in the covariates, if there are any, should be imputed first.

**id**
A character string of the name of the ID variable in the data. Optional for a “ts” class object.

**time**
A character string of the name of the TIME variable in the data. Optional for a “ts” class object.

**observed**
A vector of character strings of the names of the observed variables in the data. Optional for a “ts” class object.

**covariates**
(Optional) a vector of character strings of the names of the covariates in the data, which can be missing.

**Examples**

```r
data(EMGsim)
dd <- dynr.data(EMGsim, id = 'id', time = 'time', observed = 'EMG', covariates = 'self')
z <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 1), frequency = 12)
dz <- dynr.data(z)
```

**Description**

A Function to plot the flow or velocity field for a one or two dimensional autonomous ODE system from the phaseR package written by Michael J. Grayling.

**Usage**

dynr.flowField(deriv, xlim, ylim, parameters = NULL, system = "two.dim",
points = 21, col = "gray", arrow.type = "equal", arrow.head = 0.05,
frac = 1, add = TRUE, xlab = "x", ylab = "y", state.names = c("x",
"y"), ...)

**Arguments**

**deriv**
A function computing the derivative at a point for the ODE system to be analysed. For examples see the phaseR package guide.

**xlim**
A vector of length two setting the lower and upper limits of the variable to be plotted on the horizontal axis (usually the first variable returned by the function deriv)
dyrn.flowField

ylim A vector of length two setting the lower and upper limits of the variable to be plotted on the vertical axis (usually the second variable returned by the function deriv)

parameters Parameters of the ODE system, to be passed to deriv. Supplied as a vector; the order of the parameters can be found from the deriv file. Defaults to NULL.

system Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".

points Sets the density of the line segments to be plotted. Defaults to 11.

col Sets the color of the plotted line segments. Defaults to "gray". Should be a vector of length one. Will be reset accordingly if it is a vector of the wrong length.

arrow.type Sets the type of line segments plotted. Options include: "proportional" = the length of the line segments reflects the magnitude of the derivative. "equal" the line segments take equal lengths, simply reflecting the gradient of the derivative(s). Defaults to "equal".

arrow.head Sets the length of the arrow heads. Passed to arrows. Defaults to 0.05.

frac Sets the fraction of the theoretical maximum length line segments can take without overlapping, that they can actually attain. In practice, frac can be set to greater than 1 without line segments overlapping.

add Logical. Defaults to TRUE. TRUE = the flow field is added to an existing plot; FALSE = a new plot is created.

xlab Label for the x-axis of the resulting plot. Defaults to "x".

ylab Label for the y-axis of the resulting plot. Defaults to "y".

state.names State names for ode functions that do not use positional states

... Additional arguments to be passed to either plot or arrows.

Value

Returns a list with the following components: add, arrow.head, arrow.type, col, deriv, dx, dy, frac, parameters, points, system, x, xlab, xlim, y, ylab, ylim. Most of these components correspond simply to their original input values.

The only new elements are:

dx = A matrix. In the case of a two dimensional system, the values of the derivative of the first dependent derivative at all evaluated points.

dy = A matrix. In the case of a two dimensional system, the values of the derivative of the second dependent variable at all evaluated points. In the case of a one dimensional system, the values of the derivative of the dependent variable at all evaluated points.

x = A vector. In the case of a two dimensional system, the values of the first dependent variable at which the derivatives were computed. In the case of a one dimensional system, the values of the independent variable at which the derivatives were computed.

y = A vector. In the case of a two dimensional system, the values of the second dependent variable at which the derivatives were computed. In the case of a one dimensional system, the values of the dependent variable at which the derivatives were computed.
**Note**

The phaseR package was taken off cran as of 10/1/2019 so we are exporting some selected functions from phaseR_2.0 published on 8/20/2018. For details of these functions please see original documentations on the phaseR package.

**References**


**Examples**

```r
#Osc <- function(t, y, parameters) {
#  dy <- numeric(2)
#  return(list(dy))
#}
#
#param <- coef(g)
#dynr.flowField(Osc, xlim = c(-3, 3),
#  ylim = c(-3, 3),
#  xlab="x", ylab="dx/dt",
#  main=paste0("Oscillator model"),
#  cex.main=2,
#  parameters = param,
#  points = 15, add = FALSE,
#  col="blue",
#  arrow.type="proportional",
#  arrow.head=.05)
```

**dynr.ggplot**

The ggplot of the smoothed state estimates and the most likely regimes

**Description**

The ggplot of the smoothed state estimates and the most likely regimes

**Usage**

```r
dynr.ggplot(res, dynrModel, style = 1, numSubjDemo = 2, idtoPlot = c(), names.state, names.observed, names.regime, shape.values, title, ylab, is.bw = FALSE, colorPalette = "Set2", fillPalette = "Set2", mancolorPalette, manfillPalette, ...)
```

```r
# S3 method for class 'dynrCook'
autoplot(object, dynrModel, style = 1, numSubjDemo = 2, idtoPlot = c(), names.state, names.observed, names.regime, shape.values, title, ylab, is.bw = FALSE, colorPalette = "Set2", fillPalette = "Set2", mancolorPalette, manfillPalette, ...)
```
\begin{verbatim}
idtoPlot = c(), names.state, names.observed, names.regime, shape.values,
title, ylab, is.bw = FALSE, colorPalette = "Set2", fillPalette = "Set2",
mancolorPalette, manfillPalette, ...) 
\end{verbatim}

**Arguments**

- `res` The dynr object returned by `dynr.cook()`.
- `dynrModel` The model object to plot.
- `style` The style of the plot. If style is 1 (default), user-selected smoothed state variables are plotted. If style is 2, user-selected observed-versus-predicted values are plotted.
- `numSubjDemo` The number of subjects to be randomly selected for plotting.
- `idtoPlot` Values of the ID variable to plot.
- `names.state` (optional) The names of the states to be plotted, which should be a subset of the state.names slot of the measurement slot of `dynrModel`.
- `names.observed` (optional) The names of the observed variables to be plotted, which should be a subset of the obs.names slot of the measurement slot of `dynrModel`.
- `names.regime` (optional) The names of the regimes to be plotted, which can be missing.
- `shape.values` (optional) A vector of values that correspond to the shapes of the points, which can be missing. See the R documentation on `pch` for details on possible shapes.
- `title` (optional) A title of the plot.
- `ylab` (optional) The label of the y axis.
- `is.bw` Is plot in black and white? The default is FALSE.
- `colorPalette` A color palette for lines and dots. It is a value passed to the palette argument of the `ggplot2::scale_colour_brewer()` function. These palettes are in the R package `RColorBrewer`. One can find them by attaching the package with `library(RColorBrewer)` and run `display.brewer.all()`.
- `fillPalette` A color palette for blocks. It is a value passed to the palette argument of the `ggplot2::scale_fill_brewer()` function. These palettes are in the package `RColorBrewer`. One can find them by attaching the package with `library(RColorBrewer)` and run `display.brewer.all()`.
- `mancolorPalette` (optional) A color palette for manually scaling the colors of lines and dots. It is a vector passed to the values argument of the `ggplot2::scale_colour_manual` function.
- `manfillPalette` (optional) A color palette for manually scaling the colors of filled blocks. It is a vector passed to the values argument of the `ggplot2::scale_fill_manual` function.
- `...` A list of elements that modify the existing ggplot theme. Consult the `ggplot2::theme()` function in the R package `ggplot2` for more options.
- `object` The same as `res`. The dynr object returned by `dynr.cook()`.
**Details**

This function outputs a ggplot layer that can be modified using functions in the package `ggplot2`. That is, one can add layers, scales, coords and facets with the `+` sign. In an example below, the `ggplot2::ylim()` function is used to modify the limits of the y axis of the graph. More details can be found on [http://ggplot2.tidyverse.org](http://ggplot2.tidyverse.org) and [http://ggplot2.tidyverse.org/reference/](http://ggplot2.tidyverse.org/reference/).

The two functions `dynr.ggplot()` and `autoplot()` as identical aliases of one another. The `autoplot()` function is an S3 method from the package `ggplot2` that allows many objects to be plotted and works like the base `plot()` function.

**Examples**

```r
# The following code is part of a demo example in dynr
# One can obtain the yum and rsmod objects needed below by running demo(RSLinearDiscreteYang).
# p <- dynr.ggplot(yum, dynrModel = rsmod, style = 1,
# names.regime = c("Deactivated", "Activated"),
# title = "(B) Results from RS-AR model", numSubjDemo = 1,
# shape.values = c(1),
# text = element_text(size = 16),
# is.bw = TRUE)
# One can modify the limits on the y axis by using '+'
# p + ggplot2::ylim(-2, 4)

# autoplot(yum, dynrModel = rsmod, style = 1,
# names.regime = c("Deactivated", "Activated"),
# title = "(B) Results from RS-AR model", numSubjDemo = 1,
# shape.values = c(1),
# text = element_text(size = 16),
# is.bw = TRUE)
```

---

**dynr.ldl**

*LDL Decomposition for Matrices*

**Description**

LDL Decomposition for Matrices

**Usage**

`dynr.ldl(x)`

**Arguments**

- `x` a numeric matrix

This is a wrapper function around the `chol` function. The goal is to factor a square, symmetric, positive (semi-)definite matrix into the product of a lower triangular matrix, a diagonal matrix, and the transpose of the lower triangular matrix. The value returned is a lower triangular matrix with the elements of D on the diagonal.
**Description**

Multiple Imputation of dynrModel objects

**Usage**

```r
dynr.mi(dynrModel, which.aux = NULL, which.lag = NULL, lag = 0, which.lead = NULL, lead = 0, m = 5, iter = 5, imp.obs = FALSE, imp.exo = TRUE, diag = TRUE, Rhat = 1.1, conf.level = 0.95, verbose = TRUE, seed = NA)
```

**Arguments**

- `dynrModel`: dynrModel object. data and model setup
- `which.aux`: character. names of the auxiliary variables used in the imputation model
- `which.lag`: character. names of the variables to create lagged responses for imputation purposes
- `lag`: integer. number of lags of variables in the imputation model
- `which.lead`: character. names of the variables to create leading responses for imputation purposes
- `lead`: integer. number of leads of variables in the imputation model
- `m`: integer. number of multiple imputations
- `iter`: integer. number of MCMC iterations in each imputation
- `imp.obs`: logical. flag to impute the observed dependent variables
- `imp.exo`: logical. flag to impute the exogenous variables
- `diag`: logical. flag to use convergence diagnostics
- `Rhat`: numeric. value of the Rhat statistic used as the criterion in convergence diagnostics
- `conf.level`: numeric. confidence level used to generate confidence intervals
- `verbose`: logical. flag to print the intermediate output during the estimation process
- `seed`: integer. random number seed to be used in the MI procedure

**Value**

an object of ‘dynrMi’ class that is a list containing: 1. the imputation information, including a data set containing structured lagged and leading variables and a ‘mids’ object from mice() function; 2. the diagnostic information, including trace plots, an Rhat plot and a matrix containing Rhat values; 3. the estimation results, including parameter estimates, standard error estimates and confidence intervals.
dynr.model

Create a dynrModel object for parameter estimation (cooking dynr) using dynr.cook

Description

Create a dynrModel object for parameter estimation (cooking dynr) using dynr.cook

Usage

dynr.model(dynamics, measurement, noise, initial, data, ..., outfile = tempfile())

Arguments

dynamics a dynrDynamics object prepared with prep.formulaDynamics or prep.matrixDynamics
measurement a dynrMeasurement object prepared with prep.loadings or prep.measurement
noise a dynrNoise object prepared with prep.noise
initial a dynrInitial object prepared with prep.initial
data a dynrData object made with dynr.data
... additional arguments specifying other dynrRecipe objects. Argument regimes is for a dynrRegimes object prepared with prep.regimes and argument transform is for a dynrTrans object prepared with prep.tfun.
outfile a character string of the name of the output C script of model functions to be compiled for parameter estimation. The default is the name for a potential temporary file returned by tempfile().

References


Examples

# See the demo, MILinearDiscrete.R, for an illustrative example
# of using dynr.mi to implement multiple imputation with
# a vector autoregressive model
# dynrMi <- dynr.mi(dynrModel, which.aux=c("x1","x2"),
# which.lag=c("wp","hp"), lag=1, which.lead=NULL, lead=0,
# m=5, iter=5, imp.obs=FALSE, imp.exo=TRUE,
# diag = TRUE, Rhat=1.1,
# conf.level=0.95, verbose=FALSE, seed=12345)
Details

A dynrModel is a collection of recipes. The recipes are constructed with the functions prep.measurement, prep.noise, prep.formulaDynamics, prep.matrixDynamics, prep.initial, and in the case of regime-switching models prep.regimes. Additionally, data must be prepared with dynr.data and added to the model.

Several named arguments can be passed into the ... section of the function. These include

- Argument regimes is for a dynrRegimes object prepared with prep.regimes
- Argument transform is for a dynrTrans object prepared with prep.tfun.
- Argument options a list of options. Check the NLopt website http://ab-initio.mit.edu/wiki/index.php/NLopt_Reference#Stopping_criteria for details. Available options for use with a dynrModel object include xtol_rel, stopval, ftol_rel, ftol_abs, maxeval, and max-time, all of which control the termination conditions for parameter optimization. The examples below show a case where options were set.

There are several available methods for dynrModel objects.

- The dollar sign ($) can be used to both get objects out of a model and to set pieces of the model.
- names returns the names of the free parameters in a model.
- printex prints LaTeX expressions for the equations that compose a model. The output can then be readily typeset for inclusion in presentations and papers.
- nobs gives the total number of observations (e.g. all times across all people)
- coef gives the free parameter starting values. Free parameters can also be assigned with coef(model) <- aNamedVectorOfCoefficients

Examples

#rsmod <- dynr.model(dynamics=recDyn, measurement=recMeas, noise=recNoise,  
# initial=recIni, regimes=recReg, data=dd, outfile="RSLinearDiscrete.c")

#Set relative tolerance on function value via 'options':
#rsmod <- dynr.model(dynamics=recDyn, measurement=recMeas, noise=recNoise,  
# initial=recIni, regimes=recReg, data=dd, outfile="RSLinearDiscrete.c",  
# options=list(ftol_rel=as.numeric(1e-6)))

#For a full demo example, see:
#demo(RSLinearDiscrete , package="dynr")

---

dynr.plotFreq

Plot of the estimated frequencies of the regimes across all individuals and time points based on their smoothed regime probabilities

Description

Plot of the estimated frequencies of the regimes across all individuals and time points based on their smoothed regime probabilities
**dynr.plotFreq**

```r
dynr.plotFreq(res, dynrModel, names.regime, title, xlab, ylab, textsize = 12, print = TRUE)
```

**Arguments**

- `res`: The dynr object returned by `dynr.cook()`.
- `dynrModel`: The model object to plot.
- `names.regime`: (optional) Names of the regimes (must match the length of the number of regimes).
- `title`: (optional) Title of the plot.
- `xlab`: (optional) Label of the x-axis.
- `ylab`: (optional) Label of the y-axis.
- `textsize`: (default = 12) Text size for the axis labels and title (= textsize + 2).
- `print`: (default = TRUE) A flag for whether the plot should be printed.

**dynr.taste**

Detect outliers in state space models.

**Description**


**Usage**

```r
dynr.taste(dynrModel, dynrCook = NULL, which.state, which.obs, conf.level = 0.99, alternative = c("two.sided", "less", "greater"), debug_flag = FALSE)
```

**Arguments**

- `dynrModel`: an object of ‘dynrModel’ class.
- `dynrCook`: the ‘dynrCook’ object fitted with ‘debug_flag=TRUE’ for the ‘dynrModel’ object. The default is NULL. If the dynrCook object were not provided, or the object were cooked with ‘debug_flag=FALSE’, `dynr.taste` will fit the dynrModel object with ‘debug_flag=TRUE’ internally.
- `which.state`: a character vector of the names of latent variables. The outlier detection process will be applied only to the chosen variable. If the argument is NA, all the latent variables will be excluded in the outlier detection process. If the argument is missing (default), all the latent variables will be chosen.
- `which.obs`: a character vector of the names of measured or observed variables. The outlier detection process will be applied only to the chosen variable. If the argument is NA, all the measured variables will be excluded in the outlier detection process. If the argument is missing (default), all the measured variables will be chosen.
conf.level  a numeric of confidence level that is used for outliers detection tests (chi-square test and t-test). The default is 0.99.

alternative  a character string specifying the alternative hypothesis of t-test, must be one of “two.sided” (default), “greater” or “less”.

debug_flag  a logical. 'TRUE' for output of by-products related to t-value calculation

Value

an object of ‘dynrTaste’ class that is a list containing lists of results from the outlier detection process. Vectors of ID and measured time points are included for later use, such as in dynr.taste2. The values, p-values, and shock points related to ‘joint’ chi-square, ‘independent’ chi-square, and t statistic for innovative and additive outliers are following in that order. The estimated delta for innovative and additive components are in the last. If debug_flag is TRUE, The by-products of the Kalman filter and smoother (Q, S, s, F_inv, N, u, r) would be added at the end. See the reference for definition of the notations. The t statistic (estimate of an outlier divided by standard error of the outlier) of the last time point is NA, because the Kalman smoothing process starts with setting r and N to zero for the last time point (core elements of calculating estimates and the standard errors of outliers) that lead to 0/0 of the t statistic of the last time point. For the time-varying models, more NAs would appear at the end of times because the Kalman smoother needs more time points to obtain all elements of r nad N from limited number of observed variables in the model.

The ‘delta_chi’ list comprises magnitude of innovative (Latent) and additive (Observed) outliers, ‘delta.L’ and ‘delta.O’, when chi-square statistics is used to detect outliers. The ‘delta_t’ list comprises magnitude of innovative (Latent) and additive (Observed) outliers, ‘delta.L’ and ‘delta.O’, when t statistics is used to detect outliers.

References


Examples

```r
## Not run:
# See the demo for outlier detection, OutlierDetection.R
dynrCook <- dynr.cook(dynrModel)
dynrTaste <- dynr.taste(dynrModel, dynrCook)

# Detect outliers related to 'eta1' out of, say, three latent
# variables c("eta1", "eta2", "eta3"), and all measured variables.
dynrTaste <- dynr.taste(dynrModel, dynrCook, which.state=c("eta1"))

## End(Not run)
```
**dynr.taste2**

Re-fit state-space model using the estimated outliers.

**Description**

The function `dynr.taste2()` updates the `dynrModel` object applying outliers from the `dynrTaste` object, or outliers from users. The function then re-cook the model.

**Usage**

```r
dynr.taste2(dynrModel, dynrCook, dynrTaste, delta_inn = c("t", "ind", "jnt", "null"), delta_add = c("t", "ind", "jnt", "null"), delta_L = NULL, delta_O = NULL, cook = TRUE, verbose = FALSE, newOutfile = "new_taste.c")
```

**Arguments**

- `dynrModel`: an object of `dynrModel` class.
- `dynrCook`: an object of `dynrCook` class.
- `dynrTaste`: an object of `dynrTaste` class. The default is `NULL`.
- `delta_inn`: a character string for a method detecting ‘inn’ovative outliers, which must be one of “t” (default), “ind”, “jnt” or “null”. According to the method, corresponding delta estimates (magnitude of estimated outliers) will be included in the new `dynrModel` in output. ‘t’ represents the t statistic, ‘ind’ represents the independent chi-square statistic, ‘jnt’ represents the joint chi-square statistic. If no outliers are assumed, “null” can be used.
- `delta_add`: a character string for a method detecting ‘add’itive outliers, which must be one of “t” (default), “ind”, “jnt” or “null”. According to the method, corresponding delta estimates will be included in the new `dynrModel`.
- `delta_L`: a data.frame containing user-specified latent outliers. The delta estimates from `dynrTaste` will be ignored. The number of rows should equal to the total time points, and the number of columns should equal to the number of latent variables.
- `delta_O`: a data.frame containing user-specified observed outliers. The delta estimates from `dynrTaste`, and arguments of `delta_inn` and `delta_add` will be ignored. The number of rows should equal to the total time points, and the number of columns should equal to the number of observed variables.
- `cook`: a logical specifying whether the newly built model would be cooked by `dynr.cook` function. The default is `TRUE`. When `cook=FALSE`, only the newly built model will be saved for the output.
- `verbose`: a logical specifying the verbose argument of the new `cook` object. The default is `FALSE`.
- `newOutfile`: a character string for `outfile` argument of `dynr.model` function to create new `dynrModel` object. The default is "new_taste.c".
**Details**

The argument `dynrTaste` should be the `dynrTaste` object that is output of the `dynr.taste` function the argument `dynrModel` is applied.

The argument `dynrTaste` can be `NULL`, if user-specified outliers are offered by the arguments `delta_L` and `delta_O`.

**Value**

a list with the two arguments; a new `dynrModel` object the outliers are applied, and a `dynrCook` object the new `dynrModel` object is cooked.

**Examples**

```r
# dynrTaste2 <- dynr.taste2(dynrModel, dynrCook, dynrTaste)
```

---

**Description**

A Function to perform numerical integration of the chosen ODE system, for a user-specified set of initial conditions. Plots the resulting solution(s) in the phase plane. This function from the phaseR package written by Michael J. Grayling.

**Usage**

```r
dynr.trajectory(deriv, y0 = NULL, n = NULL, tlim, tstep = 0.01, parameters = NULL, system = "two.dim", col = "black", add = TRUE, state.names = c("x", "y"), ...)
```

**Arguments**

- `deriv`: A function computing the derivative at a point for the specified ODE system. See the phaseR package guide for more examples.
- `y0`: The initial condition(s) (ICs). In one-dimensional system, this can either be a single number indicating a single IC or a vector indicating multiple ICs. In two-dimensional system, this can either be a vector of length two reflecting the location of the two dependent variables initially, or it can be matrix where each row reflects a different set of ICs. Alternatively this can be left blank and the user can use locator to specify initial condition(s) on a plot. In this case, for one dimensional systems, all initial conditions are taken at `tlim[1]`, even if not selected so on the graph. Defaults to `NULL`.  

If \( y_0 \) is left NULL so initial conditions can be specified using locator, \( n \) sets the number of initial conditions to be chosen. Defaults to NULL.

Sets the limits of the independent variable for which the solution should be plotted. Should be a vector of length two. If \( \text{tlim}[2] > \text{tlim}[1] \), then \( \text{tstep} \) should be negative to indicate a backwards trajectory.

The step length of the independent variable, used in numerical integration. Defaults to 0.01.

Parameters of the ODE system, to be passed to \( \text{deriv} \). Supplied as a vector; the order of the parameters can be found from the \( \text{deriv} \) file. Defaults to NULL.

Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".

The color(s) to plot the trajectories in. Will be reset accordingly if it is a vector not of the length of the number of initial conditions. Defaults to "black".

Logical. Defaults to TRUE. TRUE = the trajectories added to an existing plot; FALSE = a new plot is created.

State names for the ODE functions that do not use positional states

Additional arguments to be passed to either \( \text{plot} \) or \( \text{arrows} \).

Returns a list with the following components: add, col, deriv, \( n \), parameters, system, \( \text{tlim} \), \( \text{tstep} \), \( t \), \( x \), \( y \), \( ylab \), \( y_0 \). Most of these components correspond simply to their original input values.

The only new elements are: \( t \) = A vector containing the values of the independent variable at each integration step.

\( x \) = In the two dimensional system case, a matrix whose columns are the numerically computed values of the first dependent variable for each set of ICs.

\( y \) = In the two dimensional system case, a matrix whose columns are the numerically computed values of the second dependent variable for each initial condition. In the one dimensional system case, a matrix whose columns are the numerically computed values of the dependent variable for each initial condition.

\( y_0 \) = As per input, but converted to a matrix if supplied as a vector initially.

The phaseR package was taken off cran as off 10/1/2019 so we are exporting some selected functions from phaseR_2.0 published on 8/20/2018. For details of these functions please see original documentations on the phaseR package.

Examples

```r
# Osc <- function(t, y, parameters) {
  # dy <- numeric(2)
  # return(list(dy))
#}
#
# param <- coef(g)
# dynr.flowField(Osc, xlim = c(-3, 3),
#     ylim = c(-3, 3),
#     xlab="x", ylab="dx/dt",
#     main=paste0("Oscillator model"),
#     cex.main=2,
#     parameters = param,
#     points = 15, add = FALSE,
#     col="blue",
#     arrow.type="proportional",
#     arrow.head=.05)
# IC <- matrix(c(-2, -2), ncol = 2, byrow = TRUE) # Initial conditions
# phaseR::trajectory(Osc, y0 = IC, parameters = param, tlim=c(0,10))
```

---

dynr.version  

Current Version String

Description

Current Version String

Usage

dynr.version(verbos=TRUE)

Arguments

- `verbose` If TRUE, print detailed information to the console (default)
  This function returns a string with the current version number of dynr. Optionally (with verbose = TRUE (the default)), it prints a message containing the version of R and the platform. The primary purpose of the function is for bug reporting.

Examples

- `dynr.version()`
- `dynr.version(verbos=FALSE)`
- `packageVersion("dynr")`
The `dynrCook` Class

**Description**
The `dynrCook` Class

**Details**
This is an internal class structure. You should not use it directly. Use `dynr.cook` instead.

The `dynrDynamics` Class

**Description**
The `dynrDynamics` Class

**Details**
This is an internal class structure. The classes `dynrDynamicsFormula-class` and `dynrDynamicsMatrix-class` are subclasses of this. However, you should not use it directly. Use `prep.matrixDynamics` or `prep.formulaDynamics` instead.

The `dynrInitial` Class

**Description**
The `dynrInitial` Class

**Details**
This is an internal class structure. You should not use it directly. Use `prep.initial` instead.

The `dynrMeasurement` Class

**Description**
The `dynrMeasurement` Class

**Details**
This is an internal class structure. You should not use it directly. Use `prep.measurement` or `prep.loadings` instead.
### dynrModel-class

**The dynrModel Class**

**Description**

The dynrModel Class

**Details**

This is an internal class structure. You should not use it directly. Use `dynr.model` instead.

### dynrNoise-class

**The dynrNoise Class**

**Description**

The dynrNoise Class

**Details**

This is an internal class structure. You should not use it directly. Use `prep.noise` instead.

### dynrRecipe-class

**The dynrRecipe Class**

**Description**

The dynrRecipe Class

**Details**

This is an internal class structure. You should not use it directly. The following are all subclasses of this class: `dynrMeasurement-class`, `dynrDynamics-class`, `dynrRegimes-class`, `dynrInitial-class`, `dynrNoise-class`, and `dynrTrans-class`. Recipes are the things that go into a dynrModel-class using `dynr.model`. Use the recipe prep functions (`prep.measurement`, `prep.formulaDynamics`, `prep.matrixDynamics`, `prep.regimes`, `prep.initial`, `prep.noise`, or `prep.tfun`) to create these classes instead.
The dynrRegimes Class

Description
The dynrRegimes Class

Details
This is an internal class structure. You should not use it directly. Use prep.regimes instead.

The dynrTrans Class

Description
The dynrTrans Class

Details
This is an internal class structure. You should not use it directly. Use prep.tfun instead.

EMG Single-subject time series of facial electromyography data

Description
A dataset obtained and analyzed in Yang and Chow (2010).

Usage
data(EMG)

Format
A data frame with 695 rows and 4 variables
Details


The variables are as follows:

- id. ID of the participant (= 1 in this case, over 695 time points)
- time Time in seconds
- iEMG. Observed integrated facial electromyography data
- SelfReport. Covariate - the individual’s concurrent self-reports

---

Description

A dataset simulated using an autoregressive model of order (AR(1)) with regime-specific AR weight, intercept, and slope for a covariate. This model is a special case of Model 1 in Yang and Chow (2010) in which the moving average coefficient is set to zero.


Usage

data(EMGsim)

Format

A data frame with 500 rows and 6 variables

Details

The variables are as follows:

- id. ID of the participant (= 1 in this case, over 500 time points)
- EMG. Hypothetical observed facial electromyography data
- self. Covariate - the individual’s concurrent self-reports
- trueregime. The true underlying regime for the individual at each time point
- truestate. The true score of the individual’s EMG at each time point
getdx  

A wrapper function to call functions in the fda package to obtain smoothed estimated derivatives at a specified order

Description

A wrapper function to call functions in the fda package to obtain smoothed estimated derivatives at a specified order

Usage

getdx(theTimes, norder, roughPenaltyMax, lambda, dataMatrix, derivOrder)

Arguments

- theTimes: The time points at which derivative estimation are requested
- norder: Order of Bsplines - usually 2 higher than roughPenaltyMax
- roughPenaltyMax: Penalization order. Usually set to 2 higher than the highest-order derivatives desired
- lambda: A positive smoothing parameter: larger -> more smoothing
- dataMatrix: Data of size total number of time points x total number of subjects
- derivOrder: The order of the desired derivative estimates

Value

A list containing: 1. out (a matrix containing the derivative estimates at the specified order that matches the dimension of dataMatrix); 2. basisCoef (estimated basis coefficients); 3. basis2 (basis functions)

References


Examples

```r
#x = getdx(theTimes,norder,roughPenaltyMax,sp,out2,0)[[1]] #Smoothed level
#dx = getdx(theTimes,norder,roughPenaltyMax,sp,out2,1)[[1]] #Smoothed 1st derivs
#d2x = getdx(theTimes,norder,roughPenaltyMax,sp,out2,2)[[1]] #Smoothed 2nd derivs
```
Do internal model preparation for dynr

Description

Principally, this function takes a host of arguments and gives back a list that importantly includes the function addresses.

Usage

```r
internalModelPrep(num_regime, dim_latent_var, xstart, ub, lb,
                   options = default.model.options, isContinuousTime, infile, outfile,
                   compileLib, verbose)
```

Arguments

- **num_regime**: An integer number of the regimes.
- **dim_latent_var**: An integer number of the latent variables.
- **xstart**: The starting values for parameter estimation.
- **ub**: The upper bounds of the estimated parameters.
- **lb**: The lower bounds of the estimated parameters.
- **options**: A list of NLopt estimation options. By default, xtol_rel=1e-7, stopval=-9999,
  ftol_rel=-1, ftol_abs=-1, maxeval=as.integer(-1), and maxtime=-1.
- **isContinuousTime**: A binary flag indicating whether the model is a continuous-time model (FALSE/0 = no; TRUE/1 = yes)
- **infile**: Input file name
- **outfile**: Output file name
- **compileLib**: Whether to compile the library anew
- **verbose**: Logical flag for verbose output

Value

A list of model statements to be passed to dynr.cook().
LinearOsc

Simulated time series data for a deterministic linear damped oscillator model

Description
The variables are as follows:

Usage
data(LinearOsc)

Format
A data frame with 1000 rows and 3 variables

Details
- ID. ID of the systems (1 to 10)
- x. Latent level variable
- theTimes. Measured time Points

Examples
# The following was used to generate the data
#--------------------------------------
#Osc<- function(t, prevState, parms) {
#  x1 <- prevState[1] # x1[t]
#  x2 <- prevState[2] # x2[t]
#  eta1 = parms[1]
#  zeta1 = parms[2]
#  with(as.list(parms), {
#    dx1 <- x2
#    dx2 <- eta1*x1 + zeta1*x2
#    res<-c(dx1,dx2)
#    list(res)
#  })
#}
#n = 10 #Number of subjects
#T = 100 #Number of time points
#deltaT = .1 #dt
#lastT = deltaT*T #Value of t_{i,T}
#theTimes = seq(0, lastT, length=T) #A list of time values
#
#eta = -.8
#zeta = -.1
#trueOut = matrix(NA,T*n,1)
#out1 = matrix(NA,T*n,1)
#parms = c(eta, zeta)
# for (i in 1:n){
# xstart = c(rnorm(1,0,2),rnorm(1,0,.5))
# out <- lsoda(as.numeric(xstart), theTimes, Osc, parms)
# trueOut[(1+(i-1)*T):((i+1)*T)] = out[,2]
# out1[(1+(i-1)*T):((i+1)*T)] = out[,2]+rnorm(T,0,1)
# }
#
#LinearOsc= data.frame(ID=rep(1:n,each=T),x=out1[,1],
# theTimes=rep(theTimes,n))
#save(LinearOsc,file="LinearOsc.rda")

LogisticSetPointSDE

Simulated time series data for a stochastic linear damped oscillator model with logistic time-varying setpoints

**Description**

A dataset simulated using a continuous-time stochastic linear damped oscillator model. The variables are as follows:

**Usage**

data(LogisticSetPointSDE)

**Format**

A data frame with 2410 rows and 6 variables

**Details**

- id. ID of the systems (1 to 10)
- times. Time index (241 time points for each system)
- x. Latent level variable
- y. Latent first derivative variable
- z. True values of time-varying setpoints
- obsy. Observed level

**Examples**

# The following was used to generate the data
#--------------------------------------
#require(Sim.DiffProc)
#freq <- -1
#damp <- -.1
#mu <- -2
#r <- .5
#b <- .1
#sigma1 <- 0.1
#sigma2 <- 0.1
#fx <- expression(y, freq*(x-z) + damp*y, r*z*(1-b*z))
#gx <- expression(0, sigma1, 0)
#r3dall <- c()
#for (j in 1:10){
#  r3dtemp <- c(-5,0,.1)
#  r3d <- r3dtemp
#  for (i in seq(0.125, 30, by=0.125)){
#    mod3dtemp <- snssde3d(drift=fx, diffusion=gx, M=1, t0=i-0.125,
#      x0=as.numeric(r3dtemp), T=i, N=500, type="str",
#      method="smilstein")
#    r3dtemp <- rsde3d(mod3dtemp,at=i)
#    r3d <- rbind(r3d,r3dtemp)
#  }
#  r3dall <- rbind(r3dall, cbind(r3d, id=j))
#}
#r3dall$obsy <- r3dall$x+rnorm(length(r3dall$x),0,1)
#write.table(r3dall, file="LogisticSetPointSDE.txt")

---

**Description**

Extract the log likelihood from a dynrCook Object

**Usage**

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

## S3 method for class 'dynrCook'
deviance(object, ...)
```

**Arguments**

- `object` The dynrCook object for which the log likelihood is desired
- `...` further named arguments, ignored for this method

**Details**

The 'df' attribute for this object is the number of freely estimated parameters. The 'nobs' attribute is the total number of rows of data, adding up the number of time points for each person.

The deviance method returns minus two times the log likelihood.
Value

In the case of logLik, an object of class logLik.

See Also

Other S3 methods coef.dynrCook

Examples

# Let cookedModel be the output from dynr.cook
# logLik(cookedModel)

names,dynrCook-method  Extract the free parameter names of a dynrCook object

Description

Extract the free parameter names of a dynrCook object

Usage

## S4 method for signature 'dynrCook'
names(x)

Arguments

x  The dynrCook object from which the free parameter names are desired

names,dynrModel-method

Extract the free parameter names of a dynrModel object

Description

Extract the free parameter names of a dynrModel object

Usage

## S4 method for signature 'dynrModel'
names(x)

Arguments

x  The dynrModel object from which the free parameter names are desired
nobs.dynrCook

Extract the number of observations for a dynrCook object

Description

Extract the number of observations for a dynrCook object

Usage

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

Arguments

object A fitted model object
...
Further named arguments. Ignored.

Details

We return the total number of rows of data, adding up the number of time points for each person. For some purposes, you may want the mean number of observations per person or the number of people instead. These are not currently supported via nobs.

Value

A single number. The total number of observations across all IDs.

Examples

# Let cookedModel be the output from dynr.cook
#nobs(cookedModel)

nobs.dynrModel

Extract the number of observations for a dynrModel object

Description

Extract the number of observations for a dynrModel object

Usage

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

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

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

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

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

## S3 method for class 'dynrModel'
nobs(object, ...)
Arguments

object            An unfitted model object
...          Further named arguments. Ignored.

Details

We return the total number of rows of data, adding up the number of time points for each person.
For some purposes, you may want the mean number of observations per person or the number of
people instead. These are not currently supported via nobs.

Value

A single number. The total number of observations across all IDs.

Examples

# Let rawModel be the output from dynr.model
# nobs(rawModel)

NonlinearDFAsim

Simulated multi-subject time series based on a dynamic factor analysis
model with nonlinear relations at the latent level

Description

A dataset simulated using a discrete-time nonlinear dynamic factor analysis model with 6 observed
indicators for identifying two latent factors: individuals’ positive and negative emotions. Proposed
by Chow and Zhang (2013), the model was inspired by models of affect and it posits that the two
latent factors follow a vector autoregressive process of order 1 (VAR(1)) with parameters that vary
between two possible regimes: (1) an "independent" regime in which the lagged influences between
positive and negative emotions are zero; (2) a "high-activation" regime to capture instances on which
the lagged influences between PA and NA intensify when an individual’s previous levels of positive
and negative emotions were unusually high or low (see Model 2 in Chow & Zhang).

models. Psychometrika, 78(4), 740-768.

Usage

data(NonlinearDFAsim)

Format

A data frame with 3000 rows and 8 variables
oscData

Details

• id. ID of the participant (1 to 10)
• time. Time index (300 time points from each subject)
• y1-y3. Observed indicators for positive emotion
• y4-y6. Observed indicators for negative emotion

oscData Another simulated multilevel multi-subject time series of a damped oscillator model

Description

The variables are as follows:

Usage
data(oscData)

Format

A data frame with 1,800 rows and 6 variables

Details

• id. Person ID
• times. Continuous time of measurement
• y1. Observed score 1
• u1. Covariate 1
• u2. Covariate 2
• trueb. True value of person-specific random effect

Oscillator Simulated time series data of a damped linear oscillator

Description

A dataset simulated using a damped linear oscillator model in continuous time with 1 observed indicator for identifying two latent factors (position and velocity). The variables are as follows:

Usage
data(Oscillator)
Format

A data frame with 1000 rows and 5 variables

Details

• id. ID of the systems (1 to 1 because this is a single person)
• y1. Noisy observed position
• times. Time index (1000 time points) spaced at one unit intervals
• x1. True latent position
• x2. True latent velocity

Examples

# The following was used to generate the data
#--------------------------------------
# Data Generation
#require(mvtnorm)
#require(Matrix)
#
#xdim <- 2
#udim <- 1
#ydim <- 1
#tdim <- 1000
#set.seed(315)
#tA <- matrix(c(0, -.3, 1, -.7), xdim, xdim)
#tB <- matrix(c(0), xdim, udim)
#tC <- matrix(c(1, 0), ydim, xdim)
#tD <- matrix(c(0), ydim, udim)
#tQ <- matrix(c(0), xdim, xdim); diag(tQ) <- c(0, 2.2)
#tR <- matrix(c(0), ydim, ydim); diag(tR) <- c(1.5)
#
#x0 <- matrix(c(0, 1), xdim, 1)
#P0 <- diag(c(1), xdim)
#tdx <- matrix(0, xdim, tdim+1)
#tx <- matrix(0, xdim, tdim+1)
#tu <- matrix(0, udim, tdim)
#ty <- matrix(0, ydim, tdim)
#
#tT <- matrix(0:tdim, nrow=1, ncol=tdim+1)
#
#tI <- diag(1, nrow=tdim, ncol=tdim+1)
#
#tx[,1] <- x0
#for(i in 2:(tdim+1)){
#  q <- t(rmvnorm(1, rep(0, xdim), tQ))
#  tdx[,i] <- tA %*% tx[,i-1] + tB %*% tu[,i-1] + q
#  expA <- as.matrix(expm(tA * (tT[,i]-tT[,i-1])))
#  intA <- solve(tA) %*% (expA - tI)
#  tx[,i] <- expA %*% tx[,i] + intA %*% tB %*% tu[,i-1] + intA %*% q
#  ty[,i-1] <- tC %*% tx[,i] + tD %*% tu[,i-1] + t(rmvnorm(1, rep(0, ydim), tR))}
Outliers

Simulated time series data for detecting outliers.

Description
This is a list object containing true outliers, the dataset, and the saved result from running dynr.taste.

Usage
data(Outliers)

Format
A data frame with 6000 rows and 6 variables

Details
The true outliers for observed variables are saved in `Outliers$generated$shockO`.

- id. Six outliers were added for each ID.
- time_O. Time points where the outliers were added.
- obs. Variable indices where the outliers were added.
- shock_O. The magnitude of outliers.

The true outliers for state variables are saved in `Outliers$generated$shockL`.

- id. Three outliers were added for each ID.
- time_L. Time points where the outliers were added.
- lat. Variable indices where the outliers were added.
- shock_L. The magnitude of outliers.

A dataset simulated based on state-space model including the outliers. The data is saved in `Outliers$generated$y`. The variables are as follows:

- id. ID of the systems (1 to 100)
- times. Time indices (100 time points for each participant)
• V1 - V6. observed variables

The detected innovative outliers from dynr.taste for this dataset, which is used for testing whether the dynr.taste replicate the same result. The data is saved in ‘Outliers$detect_O’. The variables are as follows:

• id. IDs
• time_L. Time points where the outliers were detected
• obs. Variable indices for observed variables where the outliers were detected

The detected additive outliers from dynr.taste for this dataset, which is used for testing whether the dynr.taste replicate the same result. The data is saved in ‘Outliers$detect_L’. The variables are as follows:

• id. IDs
• time_L. Time points where the outliers were detected
• obs. Variable indices for latent variables where the outliers were detected

Examples

# The following was used to generate the data
#--------------------------------------
# lambda <- matrix(c(1.0, 0.0,
# 0.9, 0.0,
# 0.8, 0.0,
# 0.0, 1.0,
# 0.0, 0.9,
# 0.0, 0.8), ncol=2, byrow=TRUE)
# psi <- matrix(c(0.3, -0.1,
# -0.1, 0.3), ncol=2, byrow=TRUE)
# beta <- matrix(c(0.8, -0.2,
# -0.2, 0.7), ncol=2, byrow=TRUE)
# theta <- diag(c(0.2, 0.2, 0.2, 0.2, 0.2, 0.2), ncol=6, nrow=6)
# nlat <- 2; nobs <- 6
# mean_0 <- rep(0, nlat)
# psi_inf <- diag(1, 2*2) - kronecker(beta, beta)
# psi_inf_inv <- solve(psi_inf)
# psi_0 <- psi_inf_inv %*% as.vector(psi)
# dim(psi_0) <- c(2, 2)
# # measurement error covariance matrix
# mea_cov <- lambda %*% psi_0 %*% t(lambda) + theta
# resl <- lapply(1:100, function(subj) {
# # initial state
# eta_0 <- rmvnorm(1, mean=mean_0, sigma=psi_0)[1,nlat]
# zeta_0 <- rmvnorm(1, mean=rep(0, nlat), sigma=psi)
# eta <- matrix(0, nrow=time, ncol=nlat)
# eta[1, ] <- beta %*% t(eta_0) + t(zeta_0)
# zeta <- rmvnorm(time, mean=rep(0, nlat), sigma=psi)
# # random shock generation
# to avoid shock appearing too early or late (first and last 3)
shkLat_time <- sample(4:(time-3), nshockLat)
shk_lat <- sample(1:nlat, nshockLat, replace=TRUE)
shockLatIdx <- matrix(c(shkLat_time, shk_lat), ncol=2)
colnames(shockLatIdx) <- c("time_L","lat")
shockLatV <- shockSignL*( shockMag*sqrt(diag(shockPsi)))[shockLatIdx[,"lat"]]
shockLatM <- matrix(0, time, nlat)
shockLatM[shockLatIdx] <- shockLatV
shkObs_time <- sample(4:(time-3), nshockObs)
shk_obs <- sample(1:nobs, nshockObs, replace=TRUE)
shockObsIdx <- matrix(c(shkObs_time, shk_obs), ncol=2)
colnames(shockObsIdx) <- c("time_O","obs")
shockObsV <- shockSignO*( shockMag*sqrt(diag(mea_cov)))[shockObsIdx[,"obs"]]
shockObsM <- matrix(0, time, nobs)
shockObsM[shockObsIdx] <- shockObsV

# generate state process WITH shock
for (t in 1:(time-1)) {
  eta[t+1, ] <- shockLatM[t, ] + beta %*% eta[t, ] + zeta[t, ]
}

# generate observed process
y <- shockObsM + eta %*% t(lambda) +
mvtnorm::rmvnorm(time, mean=rep(0, nobs), sigma=theta)# epsilon

---

**PFAsim**  
*Simulated time series data of a multisubject process factor analysis*

**Description**

A multiple subject dataset simulated using a two factor process factor analysis model in discrete time with 6 observed indicators for identifying two latent factors. The variables are as follows:

**Usage**

data(PFAsim)

**Format**

A data frame with 2,500 rows and 10 variables

**Details**

- **ID.** Person ID variable (1 to 50 because there are 50 simulated people)
- **Time.** Time ID variable (1 to 50 because there are 50 time points)
- **V1.** Noisy observed variable 1
- **V2.** Noisy observed variable 2
• V3. Noisy observed variable 3
• V4. Noisy observed variable 4
• V5. Noisy observed variable 5
• V6. Noisy observed variable 6
• F1. True latent variable 1 scores
• F2. True latent variable 2 scores

Variables V1, V2, and V3 load on F1, whereas variables V4, V5, V6 load on F2. The true values of the factor loadings are 1, 2, 1, 1, 2, and 1, respectively. The true measurement error variance is 0.5 for all variables. The true dynamic noise covariance has F1 with a variance of 2.77, F2 with a variance of 8.40, and their covariance is 2.47. The across-time dynamics have autoregressive effects of 0.5 for both F1 and F2 with a cross-lagged effect from F1 to F2 at 0.4. The cross-lagged effect from F2 to F1 is zero. The true initial latent state distribution as mean zero and a diagonal covariance matrix with var(F1) = 2 and var(F2) = 1. The generating model is the same for all individuals.

Examples

require(dynr)
# Load the data with
data(PFAsim)
# Create a dynr data object with
dd <- dynr.data(PFAsim, id="ID", time="Time", observed=paste0("V",1:6))

# The following was used to generate the data
#set.seed(12345678)
#library(mvtnorm)
## setting up matrices
time <- 50
## Occasions to throw out to wash away the effects of initial condition
npad <- 0
np <- 50
# Number of latent variables
ne <- 2
# Number of manifest variables
ny <- 6
## Residual variance-covariance matrix
psi <- matrix(c(2.77, 2.47,
    2.47, 8.40),
    ncol = ne, byrow = T)
## Lambda matrix containing contemporaneous relations among
## observed variables and 2 latent variables.
lambda <- matrix(c(1, 0,
    2, 0,
    1, 0,
    0, 1,
    0, 2,
    0, 1),
    ncol = ne, byrow = TRUE)
## Measurement error variances
theta <- diag(.5, ncol = ny, nrow = ny)
## Lagged directed relations among variables
beta <- matrix(c(0.5, 0,
# 0.4, 0.5),
ncol = ne, byrow = TRUE)
#a0  <- mvtnorm::rmvnorm(1, mean = c(0, 0),
#  sigma = matrix(c(2,0,0,1),ncol=ne))
#yall <- matrix(0,nrow = time*np, ncol = ny)
#eall <- matrix(0,nrow = time*np, ncol = ne)
#for (p in 1:np){
# Latent variable residuals
# zeta <- mvtnorm::rmvnorm(time+npad, mean = c(0, 0), sigma = psi)
# Measurement errors
# epsilon <- rmvnorm(time, mean = c(0, 0, 0, 0, 0, 0), sigma = theta)
# Set up matrix for contemporaneous variables
# etaC  <- matrix(0, nrow = ne, ncol = time + npad)
# Set up matrix for lagged variables
# etaL  <- matrix(0, nrow = ne, ncol = time + npad + 1)
#
# etaL[,1] <- a0
# etaC[,1] <- a0
# generate factors
# for (i in 2:(time+npad)){
# etaL[, i] <- etaC[,i-1]
# etaC[,i] <- beta %*% etaL[,i] + zeta[i,]
# }
# etaC  <- etaC[(npad+1):(npad+time)]
# eta <- t(etaC)
#
# generate observed series
# y <- matrix(0, nrow = time, ncol = ny)
# for (i in 1:nrow(y)){
# y[i, ] <- lambda %*% eta[i,] + epsilon[i,]
# }
# yall[(1+(p-1)*time):(p*time),] <- y
# eall[(1+(p-1)*time):(p*time),] <- eta
#}
#yall <- cbind(rep(1:np,each=time),rep(1:time,np),yall)
#yall <- cbind(yall,eall)
#write.table(yall,'PFAsim.txt',row.names=FALSE,
# col.names=c("ID", "Time", paste0("V", 1:ny), paste0("F", 1:ne)))

plot.dynrCook

Plot method for dynrCook objects

Description

Plot method for dynrCook objects

Usage

## S3 method for class 'dynrCook'
plot(x, dynrModel, style = 1, names.state, names.observed,
       printDyn = TRUE, printMeas = TRUE, textsize = 4, ...)
Arguments

x dynrCook object
dynrModel model object
style The style of the plot in the first panel. If style is 1 (default), user-selected smoothed state variables are plotted. If style is 2, user-selected observed-versus-predicted values are plotted.
names.state (optional) The names of the states to be plotted, which should be a subset of the state.names slot of the measurement slot of dynrModel.
names.observed (optional) The names of the observed variables to be plotted, which should be a subset of the obs.names slot of the measurement slot of dynrModel.
printDyn A logical value indicating whether or not to plot the formulas for the dynamic model
printMeas A logical value indicating whether or not to plot the formulas for the measurement model
textsize numeric. Font size used in the plot.
... Further named arguments

Details

This is a wrapper around dynr.ggplot. A great benefit of it is that it shows the model equations in a plot.

plotFormula

Plot the formula from a model

Description

Plot the formula from a model

Usage

plotFormula(dynrModel, ParameterAs, printDyn = TRUE, printMeas = TRUE, printRS = FALSE, textsize = 4)

Arguments

dynrModel The model object to plot.
ParameterAs The parameter values or names to plot. The underscores in parameter names are saved for use of subscripts. Greek letters can be specified as corresponding LaTeX symbols without backslashes (e.g., "lambda") and printed as greek letters.
printDyn A logical value indicating whether or not to plot the formulas for the dynamic model.
plotGCV

printMeas  A logical value indicating whether or not to plot the formulas for the measurement model
printRS    logical. Whether or not to print the regime-switching model. The default is FALSE.
textsize   The text size use in the plot.

Details
This function typesets a set of formulas that represent the model. Typical inputs to the ParameterAs argument are (1) the starting values for a model, (2) the final estimated values for a model, and (3) the parameter names. These are accessible with (1) model$xstart, (2) coef(cook), and (3) model$param.names or names(coef(cook)), respectively.

plotGCV A function to evaluate the generalized cross-validation (GCV) values associated with derivative estimates via Bsplines at a range of specified smoothing parameter (lambda) values

Description
A function to evaluate the generalized cross-validation (GCV) values associated with derivative estimates via Bsplines at a range of specified smoothing parameter (lambda) values

Usage
plotGCV(theTimes, norder, roughPenaltyMax, dataMatrix, lowLambda, upLambda, lambdaInt, isPlot)

Arguments
theTimes    The time points at which derivative estimation are requested
norder      Order of Bsplines - usually 2 higher than roughPenaltyMax
roughPenaltyMax  Penalization order. Usually set to 2 higher than the highest-order derivatives desired
dataMatrix  Data of size total number of time points x total number of subjects
lowLambda   Lower limit of lambda values to be tested. Here, lambda is a positive smoothing parameter, with larger values resulting in greater smoothing
upLambda    Upper limit of lambda
lambdaInt   The interval of lambda values to be tested.
isPlot      A binary flag on whether to plot the gcv values (0 = no, 1 = yes)

Value
A data frame containing: 1. lambda values; 2. edf (effective degrees of freedom); 3. GCV (Generalized cross-validation value as averaged across units (e.g., subjects))
References


Examples

```r
#outMatrix = plotGCV(theTimes,norder,roughPenaltyMax,out2,lambdaLow, #lambdaHi,lambdaBy,isPlot)
```

PPsim

Simulated time series data for multiple eco-systems based on a predator-and-prey model

Description

A dataset simulated using a continuous-time nonlinear predator-and-prey model with 2 observed indicators for identifying two latent factors. The variables are as follows:

Usage

```r
data(PPsim)
```

Format

A data frame with 1000 rows and 6 variables

Details

- id. ID of the systems (1 to 20)
- time. Time index (50 time points for each system)
- prey. The true population of the prey species
- predator. The true population of the predator species
- x. Observed indicator for the population of the prey species
- y. Observed indicator for the population of the predator species
**Description**

Recipe function for specifying dynamic functions using formulas

**Usage**

```r
prep.formulaDynamics(formula, startval = numeric(0),
                      isContinuousTime = FALSE, jacobian, ...)
```

**Arguments**

- `formula` a list of formulas specifying the drift or state-transition equations for the latent variables in continuous or discrete time, respectively.
- `startval` a named vector of starting values of the parameters in the formulas for estimation with parameter names as its name. If there are no free parameters in the dynamic functions, leave startval as the default `numeric(0)`.
- `isContinuousTime` if True, the left hand side of the formulas represent the first-order derivatives of the specified variables; if False, the left hand side of the formulas represent the current state of the specified variable while the same variable on the righ hand side is its previous state.
- `jacobian` (optional) a list of formulas specifying the analytic jacobian matrices containing the analytic differentiation function of the dynamic functions with respect to the latent variables. If this is not provided, dynr will invoke an automatic differentiation procedure to compute the jacobian functions.
- `...` further named arguments. Some of these arguments may include:
  - `theta.formula` = a list consisting of formula(s) of the form list (par ~ 1 * b_0 + covariate_1 * b_1 + ... + covariate_p * b_p + 1 * rand_par), where par is a parameter is a unit- (e.g., person-) specific that appears in a dynamic formula and is assumed to follow a linear mixed effects structure. Here, b_p are fixed effects parameters; covariate_1, ..., covariate_p are known covariates as ??pdeclared in dynr.data, and b_p is a random effect component representing unit i’s random deviation in par value from that predicted by b_0 + covariate_1*b_1 + ... + covariate_p*b_p.
  - `random.names` = names of random effect components in the theta.formula
  - `random.params.inicov` = names of elements in the covariance matrix of the random effect components
  - `random.values.inicov` = starting values of elements in the covariance matrix of the random effect components
Details

This function defines the dynamic functions of the model either in discrete time or in continuous time. The function can be either linear or nonlinear, with free or fixed parameters, numerical constants, covariates, and other mathematical functions that define the dynamics of the latent variables. Every latent variable in the model needs to be defined by a differential (for continuous time model), or difference (for discrete time model) equation. The names of the latent variables should match the specification in prep.measurement(). For nonlinear models, the estimation algorithm generally needs a Jacobian matrix that contains elements of first differentiations of the dynamic functions with respect to the latent variables in the model. For most nonlinear models, such differentiations can be handled automatically by dynr. However, in some cases, such as when the absolute function (abs) is used, the automatic differentiation would fail and the user may need to provide his/her own Jacobian functions. When theta.formula and other accompanying elements in "..." are provided, the program automatically inserts the random effect components specified in random.names as additional latent (state) variables in the model, and estimate (cook) this expanded model. Do check that the expanded model satisfies conditions such as observability for the estimation to work.

Examples

# In this example, we present how to define the dynamics of a bivariate dual change score model # (McArdle, 2009). This is a linear model and the user does not need to worry about # providing any jacobian function (the default).

# We start by creating a list of formula that describes the model. In this model, we have four # latent variables, which are "readLevel", "readSlope", "mathLevel", and "math Slope". The right- # hand side of each formula gives a function that defines the dynamics.

formula = list(
    list(readLevel ~ (1+beta.read)*readLevel + readSlope + gamma.read*mathLevel,
         readSlope ~ readSlope,
         mathLevel ~ (1+beta.math)*mathLevel + mathSlope + gamma.math*readLevel,
         mathSlope ~ mathSlope ))

# Then we use prep.formulaDynamics() to define the formula, starting value of the parameters in # the model, and state the model is in discrete time by setting isContinuousTime=FALSE.

dynm <- prep.formulaDynamics(formula=formula,
                            startval=c(beta.read = -.5, beta.math = -.5,
                                       gamma.read = .3, gamma.math = .03 ),
                            isContinuousTime=FALSE)

# For a full demo example of regime switching nonlinear discrete time model, you # may refer to a tutorial on # 
# \url{https://quantdev.ssri.psu.edu/tutorials/dynr-rsnonlineardiscreteexample}

# Not run:
# For a full demo example that uses user-supplied analytic jacobian functions see: # demo(RSNonlinearDiscrete, package="dynr")

formula <- list(
    list(
prep.initial

Recipe function for preparing the initial conditions for the model.

Description
Recipe function for preparing the initial conditions for the model.

Usage
prep.initial(values.inistate, params.inistate, values.inicov, params.inicov,
values.regimep = 1, params.regimep = 0, covariates, deviation = FALSE,
refRow)

Arguments
values.inistate
a vector or list of vectors of the starting or fixed values of the initial state vector
in one or more regimes. May also be a matrix or list of matrices.

params.inistate
a vector or list of vectors of the parameter names that appear in the initial state
vector in one or more regimes. If an element is 0 or "fixed", the corresponding
element is fixed at the value specified in the values vector; Otherwise, the
corresponding element is to be estimated with the starting value specified in the
values vector. May also be a matrix or list of matrices.
values.inicov a positive definite matrix or a list of positive definite matrices of the starting or fixed values of the initial error covariance structure(s) in one or more regimes. If only one matrix is specified for a regime-switching dynamic model, the initial error covariance structure stays the same across regimes. To ensure the matrix is positive definite in estimation, we apply LDL transformation to the matrix. Values are hence automatically adjusted for this purpose.

params.inicov a matrix or list of matrices of the parameter names that appear in the initial error covariance(s) in one or more regimes. If an element is 0 or "fixed", the corresponding element is fixed at the value specified in the values matrix; Otherwise, the corresponding element is to be estimated with the starting value specified in the values matrix. If only one matrix is specified for a regime-switching dynamic model, the process noise structure stays the same across regimes. If a list is specified, any two sets of the parameter names as in two matrices should be either the same or totally different to ensure proper parameter estimation.

values.regimep a vector/matrix of the starting or fixed values of the initial probabilities of being in each regime. By default, the initial probability of being in the first regime is fixed at 1.

params.regimep a vector/matrix of the parameter indices of the initial probabilities of being in each regime. If an element is 0 or "fixed", the corresponding element is fixed at the value specified in the "values" vector/matrix; Otherwise, the corresponding element is to be estimated with the starting value specified in the values vector/matrix.

covariates character vector of the names of the (person-level) covariates

deviation logical. Whether to use the deviation form or not. See Details.

refRow numeric. Which row is treated at the reference. See Details.

Details

The initial condition model includes specifications for the initial state vector, initial error covariance matrix, initial probabilities of being in each regime and all associated parameter specifications. The initial probabilities are specified in multinomial logistic regression form. When there are no covariates, this implies multinomial logistic regression with intercepts only. In particular, the initial probabilities not not specified on a 0 to 1 probability scale, but rather a negative infinity to positive infinity log odds scale. Fixing an initial regime probability to zero does not mean zero probability. It translates to a comparison log odds scale against which other regimes will be judged.

The structure of the initial state vector and the initial probability vector depends on the presence of covariates. When there are no covariates these should be vectors, or equivalently single-column matrices. When there are covariates they should have $c + 1$ columns for $c$ covariates. For values.regimep and params.regimep the number of rows should be the number of regimes. For inistate and inicov the number of rows should be the number of latent states. Of course, inicov is a square and symmetric so its number of rows should be the same as its number of columns.

When deviation=FALSE, the non-deviation form of the multinomial logistic regression is used. This form has a separate intercept term for each entry of the initial probability vector. When deviation=TRUE, the deviation form of the multinomial logistic regression is used. This form has an intercept term that is common to all rows of the initial probability vector. The rows are then distinguished by their own individual deviations from the common intercept. The deviation form
prep.initial requires the same reference row constraint as the non-deviation form (described below). By default the reference row is taken to be the row with all zero covariate effects. Of course, if there are no covariates and the deviation form is desired, then the user must provide the reference row.

The refRow argument determines which row is used as the intercept row. It is only used in the deviation form (i.e. deviation=TRUE). In the deviation form, one row of values.regimep and params.regimep contains the intercepts, other rows contain deviations from these intercepts. The refRow argument says which row contains the intercept terms. The default behavior for refRow is to detect the reference row automatically based on which parameters are fixed. If we have problems detecting which is the reference row, then we provide error messages that are as helpful as we can make them.

See Also

Methods that can be used include: print, printex, show

Examples

#### No-covariates

# Single regime, no covariates
# latent states are position and velocity
# initial position is free and called 'inipos'
# initial slope is fixed at 1
# initial covariance is fixed to a diagonal matrix of 1s
initialNoC <- prep.initial(
  values.inistate=c(0, 1),
  params.inistate=c('inipos', 'fixed'),
  values.inicov=diag(1, 2),
  params.inicov=diag('fixed', 2))

#### One covariate

# Single regime, one covariate on the initial mean
# latent states are position and velocity
# initial covariance is fixed to a diagonal matrix of 1s
# initial latent means have
# nrow = numLatentState, ncol = numCovariates + 1
# initial position has free intercept and free u1 effect
# initial slope is fixed at 1
initialOneC <- prep.initial(
  values.inistate=matrix(c(0, .5, 1, 0), byrow=TRUE, nrow=2, ncol=2),
  params.inistate=matrix(c('iniPosInt', 'iniPosSlopeU1', 'fixed', 'fixed'), byrow=TRUE, nrow=2, ncol=2),
  values.inicov=diag(1, 2),
  params.inicov=diag('fixed', 2),
  covariates='u1')

#### Regime-switching, one covariate
# initial covariance is fixed to a diagonal matrix of 1s
# initial latent means have
# nrow = numLatentState, ncol = numCovariates + 1
# initial position has free intercept and free u1 effect
# initial slope is fixed at 1
# There are 3 regimes but the mean and covariance
# are not regime-switching.
initialRSOneC <- prep.initial(
  values.regimep=matrix(
    c(1, 1,
      0, 1,
      0, 0), byrow=TRUE,
  nrow=3, ncol=2),
  params.regimep=matrix(
    c('r1int', 'r1slopeU1',
      'r2int', 'r2slopeU2',
      'fixed', 'fixed'), byrow=TRUE,
  nrow=3, ncol=2),
  values.inistate=matrix(
    c(0, .5,
      1, 0), byrow=TRUE,
  nrow=2, ncol=2),
  params.inistate=matrix(
    c('iniPosInt', 'iniPosSlopeU1',
      'fixed', 'fixed'), byrow=TRUE,
  nrow=2, ncol=2),
  values.inicov=diag(1, 2),
  params.inicov=diag('fixed', 2),
  covariates='u1')

prep.loadings

Recipe function to quickly create factor loadings

Description

Recipe function to quickly create factor loadings

Usage

prep.loadings(map, params = NULL, idvar, exo.names = character(0),
  intercept = FALSE)

Arguments

  map          list giving how the latent variables map onto the observed variables
  params       parameter numbers
  idvar        names of the variables used to identify the factors
prepData

### Examples

```r
# Single factor model with one latent variable fixing first loading
prep.loadings(list(eta1=paste0('y', 1:4), paste0("lambda\_", 2:4))

# Single factor model with one latent variable fixing the fourth loading
prep.loadings(list(eta1=paste0('y', 1:4), paste0("lambda\_", 1:3), idvar='y4')

# Single factor model with one latent variable freeing all loadings
prep.loadings(list(eta1=paste0('y', 1:4), paste0("lambda\_", 1:4), idvar='eta1')

# Single factor model with one latent variable fixing first loading
# and freely estimated intercept
prep.loadings(list(eta1=paste0('y', 1:4), paste0("lambda\_", 2:4),
               intercept=TRUE)

# Single factor model with one latent variable fixing first loading
# and freely estimated covariate effects for u1 and u2
prep.loadings(list(eta1=paste0('y', 1:4),
               paste0("lambda\_", 2:4),
               exo.names=paste0('u', 1:2))

# Two factor model with simple structure
prep.loadings(list(eta1=paste0('y', 1:4),
                 eta2=paste0('y\_5:7'),
                 paste0("lambda\_", c(2:4, 6:7))

# Two factor model with repeated use of a free parameter
prep.loadings(list(eta1=paste0('y', 1:4),
                 eta2=paste0('y\_5:8'),
                 paste0("lambda\_", c(2:4, 6:7, 4)))

# Two factor model with a cross loading
prep.loadings(list(eta1=paste0('y', 1:4),
                 eta2=diff(c('y5', 'y2', 'y6')),
                 paste0("lambda\_", c("21", "31", "41", "22", "62")))```
**prep.matrixDynamics**

Recipe function for creating Linear Dynamics using matrices

---

### Description

Recipe function for creating Linear Dynamics using matrices

### Usage

```r
dyn = prep.matrixDynamics(params.dyn = NULL, values.dyn, params.exo = NULL, values.exo = NULL, params.int = NULL, values.int = NULL, covariates, isContinuousTime)
```

### Arguments

- `params.dyn`: the matrix of parameter names for the transition matrix in the specified linear dynamic model
- `values.dyn`: the matrix of starting/fixed values for the transition matrix in the specified linear dynamic model
- `params.exo`: the matrix of parameter names for the regression slopes of covariates on the latent variables (see details)
- `values.exo`: matrix of starting/fixed values for the regression slopes of covariates on the latent variables (see details)
- `params.int`: vector of names for intercept parameters in the dynamic model specified as a matrix or list of matrices.
- `values.int`: vector of intercept values in the dynamic model specified as a matrix or list of matrices. Contains starting/fixed values of the intercepts.
- `covariates`: the names or the index numbers of the covariates used in the dynamic model
- `isContinuousTime`: logical. When TRUE, use a continuous time model. When FALSE use a discrete time model.

### Details

A recipe function for specifying the deterministic portion of a set of linear dynamic functions as:

- **Discrete-time model**: \( \text{eta}(t+1) = \text{int} + \text{dyn} \times \text{eta}(t) + \text{exo} \times \text{x(t)} \), where \( \text{eta}(t) \) is a vector of latent variables, \( \text{x(t)} \) is a vector of covariates, \( \text{int, dyn, and exo} \) are vectors and matrices specified via the arguments *.int, *.dyn, and *.exo.

- **Continuous-time model**: \( d/dt \text{eta}(t) = \text{int} + \text{dyn} \times \text{eta}(t) + \text{exo} \times \text{x(t)} \), where \( \text{eta}(t) \) is a vector of latent variables, \( \text{x(t)} \) is a vector of covariates, \( \text{int, dyn, and exo} \) are vectors and matrices specified via the arguments *.int, *.dyn, and *.exo.

The left-hand side of the dynamic model consists of a vector of latent variables for the next time point in the discrete-time case, and the vector of derivatives for the latent variables at the current time point in the continuous-time case.
For models with regime-switching dynamic functions, the user will need to provide a list of the *.int, * .dyn, and *.exo arguments. (when they are specified to take on values other than the default of zero vectors and matrices), or if a single set of vectors/matrices are provided, the same vectors/matrices are assumed to hold across regimes.

prep.matrixDynamics serves as an alternative to prep.formulaDynamics.

See Also

Methods that can be used include: print, show

Examples

# Single-regime, continuous-time model. For further details run:
# demo(RSNonlinearDiscrete, package="dynr")
dynamics <- prep.matrixDynamics(
  values.dyn=matrix(c(0, -0.1, 1, -0.2), 2, 2),
  params.dyn=matrix(c('fixed', 'spring', 'fixed', 'friction'), 2, 2),
  isContinuousTime=TRUE)

# Two-regime, continuous-time model. For further details run:
# demo(RSNonlinearDiscrete, package="dynr")
dynamics <- prep.matrixDynamics(
  values.dyn=list(matrix(c(0, -0.1, 1, -0.2), 2, 2),
    matrix(c(0, -0.1, 1, 0), 2, 2)),
  params.dyn=list(matrix(c('fixed', 'spring', 'fixed', 'friction'), 2, 2),
    matrix(c('fixed', 'spring', 'fixed', 'fixed'), 2, 2)),
  isContinuousTime=TRUE)

prep.measurement

Prepare the measurement recipe

Description

Prepare the measurement recipe

Usage

prep.measurement(values.load, params.load = NULL, values.exo = NULL, params.exo = NULL, values.int = NULL, params.int = NULL, obs.names, state.names, exo.names)

Arguments

values.load matrix of starting or fixed values for factor loadings. For models with regime-specific factor loadings provide a list of matrices of factor loadings.

params.load matrix or list of matrices. Contains parameter names of the factor loadings.

values.exo matrix or list of matrices. Contains starting/fixed values of the covariate regression slopes.
prep.noise

Recipe function for specifying the measurement error and process noise covariance structures

Description

Recipe function for specifying the measurement error and process noise covariance structures

Usage

prep.noise(values.latent, params.latent, values.observed, params.observed)
**Arguments**

values.latent a positive definite matrix or a list of positive definite matrices of the starting or fixed values of the process noise covariance structure(s) in one or more regimes. If only one matrix is specified for a regime-switching dynamic model, the process noise covariance structure stays the same across regimes. To ensure the matrix is positive definite in estimation, we apply LDL transformation to the matrix. Values are hence automatically adjusted for this purpose.

params.latent a matrix or list of matrices of the parameter names that appear in the process noise covariance(s) in one or more regimes. If an element is 0 or "fixed", the corresponding element is fixed at the value specified in the values matrix; Otherwise, the corresponding element is to be estimated with the starting value specified in the values matrix. If only one matrix is specified for a regime-switching dynamic model, the process noise structure stays the same across regimes. If a list is specified, any two sets of the parameter names as in two matrices should be either the same or totally different to ensure proper parameter estimation. See Details.

values.observed a positive definite matrix or a list of positive definite matrices of the starting or fixed values of the measurement error covariance structure(s) in one or more regimes. If only one matrix is specified for a regime-switching measurement model, the measurement noise covariance structure stays the same across regimes. To ensure the matrix is positive definite in estimation, we apply LDL transformation to the matrix. Values are hence automatically adjusted for this purpose.

params.observed a matrix or list of matrices of the parameter names that appear in the measurement error covariance(s) in one or more regimes. If an element is 0 or "fixed", the corresponding element is fixed at the value specified in the values matrix; Otherwise, the corresponding element is to be estimated with the starting value specified in the values matrix. If only one matrix is specified for a regime-switching dynamic model, the process noise structure stays the same across regimes. If a list is specified, any two sets of the parameter names as in two matrices should be either the same or totally different to ensure proper parameter estimation. See Details.

**Details**

The arguments of this function should generally be either matrices or lists of matrices. Lists of matrices are used for regime-switching models with each list element corresponding to a regime. Thus, a list of three matrices implies a three-regime model. Single matrices are for non-regime-switching models. Some checking is done to ensure that the number of regimes implied by one part of the model matches that implied by the others. For example, the noise model (prep.noise) cannot suggest three regimes when the measurement model (prep.measurement) suggests two regimes. An exception to this rule is single-regime (i.e. non-regime-switching) components. For instance, the noise model can have three regimes even though the measurement model implies one regime. The single-regime components are simply assumed to be invariant across regimes.

Care should be taken that the parameters names for the latent covariances do not overlap with the parameters in the observed covariances. Likewise, the parameter names for the latent covariances
in each regime should either be identical or completely distinct. Because the LDL’ transformation is applied to the covariances, sharing a parameter across regimes may cause problems with the parameter estimation.

Use $ to show specific arguments from a dynrNoise object (see examples).

See Also

printex to show the covariance matrices in latex.

Examples

# Two latent variables and one observed variable in a one-regime model
Noise<-prep.noise(values.latent=diag(c(0.8, 1)), params.latent=diag(c('fixed', "e_x")),
values.observed=diag(1.5,1), params.observed=diag("e_y", 1))
# For matrices that can be import to latex:
printex(Noise,show=TRUE)
# If you want to check specific arguments you've specified, for example,
# values for variance structure of the latent variables
Noise$values.latent
# [[1]]
# [,1] [,2]
# [1,] 0.8 0
# [2,] 0.0 1

# Two latent variables and one observed variable in a two-regime model
Noise<-prep.noise(values.latent=list(diag(c(0.8, 1)),diag(c(0.8, 1))),
params.latent=list(diag(c('fixed', "e_x1")),diag(c('fixed', "e_x2"))),
values.observed=list(diag(1.5,1),diag(0.5,1)),
params.observed=list(diag("e_y1", 1),diag("e_y2",1)))
# If the error and noise structures are assumed to be the same across regimes,
# it is okay to use matrices instead of lists.

prep.regimes

Recipe function for creating regime switching (Markov transition) functions

Description

Recipe function for creating regime switching (Markov transition) functions

Usage

prep.regimes(values, params, covariates, deviation = FALSE, refRow)

Arguments

values    matrix giving the values. Should have (number of Regimes) rows and (number of regimes x number of covariates) columns
params    matrix of the same size as "values" consisting of the names of the parameters
prep.regimes

**covariates**

- a vector of the names of the covariates to be used in the regime-switching functions

**deviation**

- logical. Whether to use the deviation form or not. See Details.

**refRow**

- numeric. Which row is treated at the reference. See Details.

### Details

Note that each row of the transition probability matrix must sum to one. To accomplish this fix at least one transition log odds parameter in each row of "values" (including its intercept and the regression slopes of all covariates) to 0.

When deviation=FALSE, the non-deviation form of the multinomial logistic regression is used. This form has a separate intercept term for each entry of the transition probability matrix (TPM). When deviation=TRUE, the deviation form of the multinomial logistic regression is used. This form has an intercept term that is common to each column of the TPM. The rows are then distinguished by their own individual deviations from the common intercept. The deviation form requires the same reference column constraint as the non-deviation form; however, the deviation form also requires one row to be indicated as the reference row (described below). By default the reference row is taken to be the same as the reference column.

The refRow argument determines which row is used as the intercept row. It is only used in the deviation form (i.e. deviation=TRUE). In the deviation form, one row of values and params contains the intercepts, other rows contain deviations from these intercepts. The refRow argument says which row contains the intercept terms. The default behavior for refRow is to be the same as the reference column. The reference column is automatically detected. If we have problems detecting which is the reference column, then we provide error messages that are as helpful as we can make them.

### See Also

Methods that can be used include: print, printex, show

### Examples

```r
#Two-regime example with a covariate, x; log odds (LO) parameters represented in default form,
#2nd regime set to be the reference regime (i.e., have LO parameters all set to 0).
#The values and params matrices are of size 2 (numRegimes=2) x 4 (numRegimes*(numCovariates+1)).
# The LO of staying within the 1st regime (corresponding to the (1,1) entry in the
# 2 x 2 transition probability matrix for the 2 regimes) = a_11 + d_11*x
# The log odds of switching from the 1st to the 2nd regime (the (1,2) entry in the
# transition probability matrix) = 0
# The log odds of moving from regime 2 to regime 1 (the (2,1) entry) = a_21 + d_21*x
# The log odds of staying within the 2nd regime (the (2,2) entry) = 0
b <- prep.regimes(
  values=matrix(c(8,-1,rep(0,2),
               -4,.1,rep(0,2)),
               nrow=2, ncol=4, byrow=TRUE),
  params=matrix(c("a_11","d_11x",rep("fixed",2),
                "a_21","d_21x",rep("fixed",2)),
               nrow=2, ncol=4, byrow=TRUE),
  covariates=c("x"))
```
# Same example as above, but expressed in deviation form by specifying 'deviation = TRUE'
# The log odds of staying within the 1st regime (corresponding to the (1,1) entry in the
# 2 x 2 transition probability matrix for the 2 regimes) = a_21 + a_11 + d_11*x
# The log odds of switching from the 1st to the 2nd regime (the (1,2) entry in the
# transition probability matrix) = 0
# The log odds of moving from regime 2 to regime 1 (the (2,1) entry) = a_21 + d_21*x
# The log odds of staying within the 2nd regime (the (2,2) entry) = 0
b <- prep.regimes(values=matrix(c(8,-1,rep(0,2),
     -4,.1,rep(0,2)),
    nrow=2, ncol=4, byrow=TRUE),
params=matrix(c("a_11","d_11x",rep("fixed",2),
    "a_21","d_21x",rep("fixed",2)),
    nrow=2, ncol=4, byrow=TRUE), covariates=c("x"), deviation = TRUE)

# An example of regime-switching with no covariates. The diagonal entries are fixed
# at zero for identification purposes
b <- prep.regimes(values=matrix(0, 3, 3),
params=matrix(c("p11",'fixed', 'p13',
    'p21', 'fixed', 'p23',
    'p31', 'p32', 'fixed'), 3, 3, byrow=TRUE))

# An example of regime-switching with no covariates. The parameters for the second regime are
# fixed at zero for identification purposes, making the second regime the reference regime.
b <- prep.regimes(values=matrix(0, 3, 3),
params=matrix(c('p11', 'fixed', 'p13',
    'p21', 'fixed', 'p23',
    'p31', 'fixed', 'p33'), 3, 3, byrow=TRUE))

# 2 regimes with three covariates
b <- prep.regimes(values=matrix(c(0), 2, 8),
params=matrix(c(paste0('p', 8:15), rep(0, 8)), 2, 8),
covariates=c('x1', 'x2', 'x3'))

---

**prep.tfun**

Create a dynrTrans object to handle the transformations and inverse transformations of model parameters

**Description**

Create a dynrTrans object to handle the transformations and inverse transformations of model parameters

**Usage**

prep.tfun(formula.trans, formula.inv, transCcode = TRUE)
Arguments

formula.trans a list of formulae for transforming freed parameters other than variance-covariance parameters during the optimization process. These transformation functions may be helpful for transforming parameters that would normally appear on a constrained scale to an unconstrained scale (e.g., parameters that can only take on positive values can be subjected to exponential transformation to ensure positivity.)

formula.inv a list of formulae that inverse the transformation on the free parameters and will be used to calculate the starting values of the parameters.

transCcode a logical value indicating whether the functions in formula.trans need to be transformed to functions in C. The default for transCcode is TRUE, which means that the formulae will be translated to C functions and utilized during the optimization process. If transCcode = FALSE, the transformations are only performed at the end of the optimization process for standard error calculations but not during the optimization process.

Details

Prepares a dynr recipe that specifies the names of the parameters that are to be subjected to user-supplied transformation functions and the corresponding transformation and reverse-transformation functions. This can be very handy in fitting dynamic models in which certain parameters can only take on permissible values in particular ranges (e.g., a parameter may have to positive). Note that all variance-covariance parameters in the model are automatically subjected to transformation functions to ensure that the resultant covariance matrices are positive-definite. Thus, no additional transformation functions are needed for variance-covariance parameters.

Examples

# Specifies a transformation recipe, r20, that subjects the parameters 
# 'r10' and 'r20' to exponential transformation to ensure that they are positive.
trans <- prep.tfun(formula.trans = list(r10~exp(r10), r20~exp(r20)),
                   formula.inv = list(r10~log(r10), r20~log(r20)))

printex

The printex Method

Description

The printex Method

Usage

printex(object, ParameterAs, printDyn = TRUE, printMeas = TRUE,
        printInit = FALSE, printRS = FALSE, outFile, show, ...)


Arguments

- **object**: The dynr object (recipe, model, or cooked model).
- **ParameterAs**: The parameter values or names to plot. The underscores in parameter names are saved for use of subscripts. Greek letters can be specified as corresponding LaTeX symbols without `##' backslashes (e.g., "lambda") and printed as greek letters.
- **printDyn**: logical. Whether or not to print the dynamic model. The default is TRUE.
- **printMeas**: logical. Whether or not to print the measurement model. The default is TRUE.
- **printInit**: logical. Whether or not to print the initial conditions. The default is FALSE.
- **printRS**: logical. Whether or not to print the regime-switching model. The default is FALSE.
- **outFile**: The name of the output tex file.
- **show**: logical indicator of whether or not to show the result in the console.
- **...**: Further named arguments, passed to internal method. **AsMatrix** is a logical indicator of whether to put the object in matrix form.

Details

This is a general way of getting a LaTeX string for recipes, models, and cooked models. It is a great way to check that you specified the model or recipe you think you did before estimating its free parameters (cooking). After the model is cooked, you can use it to get LaTeX code with the estimated parameters in it.

Typical inputs to the **ParameterAs** argument are (1) the starting values for a model, (2) the final estimated values for a model, and (3) the parameter names. These are accessible with (1) `model$xstart`, (2) `coef(cook)`, and (3) `model$param.names` or `names(coef(cook))`, respectively.

See Also

A way to put this in a plot with **plotFormula**

---

**RSPPsim**

*Simulated time series data for multiple eco-systems based on a regime-switching predator-and-prey model*

Description

A dataset simulated using a regime-switching continuous-time nonlinear predator-and-prey model with 2 observed indicators for identifying two latent factors. The variables are as follows:

Usage

```r
data(RSPPsim)
```
Format
A data frame with 6000 rows and 8 variables

Details
• id. ID of the systems (1 to 20)
• time. Time index (300 time points for each system)
• prey. The true population of the prey species
• predator. The true population of the predator species
• x. Observed indicator for the population of the prey species
• y. Observed indicator for the population of the predator species
• cond. A time-varying covariate indicating the conditions of the respective eco-system across time which affects the regime-switching transition matrix
• regime. The true regime indicators across time (1 and 2).

substituteFormula A internal-use only function for substituting formula. If the RHS of formula has terms in the LHS of term.formula, this function replaces any appearance with the RHS of term.formula

Description
A internal-use only function for substituting formula. If the RHS of formula has terms in the LHS of term.formula, this function replaces any appearance with the RHS of term.formula

Usage
substituteFormula(formula, term.formula)

Arguments
formula a list of original formulas
term.formula a list of term formulas

Value
a list of formulas after the replacement

Examples
#substitutedformula <- substituteFormula(formula, term.formula)
### summary.dynrCook

*Get the summary of a dynrCook object*

**Description**

Get the summary of a dynrCook object.

**Usage**

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

**Arguments**

- `object`: The dynrCook object for which the summary is desired.
- `...`: Further named arguments, passed to the print method (e.g., `digits` and `signif.stars`).

**Details**

The summary gives information on the free parameters estimated: names, parameter values, numerical Hessian-based standard errors, t-values (values divided by standard errors), and standard-error based confidence intervals. Additionally, the likelihood, AIC, and BIC are provided.

Note that an exclamation point (!) in the final column of the summary table indicates that the standard error and confidence interval for this parameter may not be trustworthy. The corresponding element of the (transformed, inverse) Hessian was negative and an absolute value was taken to make it positive.

### theta_plot

*A function to plot simple slopes and region of significance.*

**Description**

A function to plot simple slopes and region of significance.

**Usage**

```r
theta_plot(.lm, predictor, moderator, alpha = 0.05, jn = F, title0, predictorLab, moderatorLab)
```
Arguments

.. lm
   A regression object from running a linear model of the form: \( \text{lm}(y \sim x1+x2+x1:x2) \), yielding: \( y = b0 + b1 \times x1 + b2 \times x2 + b3 \times x1 \times x2 + \text{residual} \). In this case, one may rewrite the \( \text{lm} \) as \( y = b0 + (b1+b3 \times x2) \times x1 + b2 \times x2 + \text{residual} \), where \( (b1+b3 \times x2) \) is referred to as the simple slope of \( x1 \), \( x1 \) is the predictor, and \( x2 \) is the moderator whose values yield different simple slope values for \( x1 \).

   predictor
   The independent variable for which simple slope is requested

   moderator
   The moderator whose values affect the simple slopes of the predictor. Appears on the horizontal axis.

   alpha
   The designated alpha level for the Johnson-Neyman technique

   jn
   A binary flag requesting the Johnson-Neyman test (T or F)

   title0
   Title for the plot

   predictorLab
   Label for the predictor

   moderatorLab
   Label for the moderator

Value

A region of significance plot with simple slopes of the predictor on the vertical axis, and values of the moderator on the horizontal axis.

References

Adapted from functions written by Marco Bachl to perform the Johnson-Neyman test and produce a plot of simple slopes and region of significance available at: https://rpubs.com/bachl/jn-plot

Examples

```r
# g = lm(y~x1:x2,data=data)
# theta_plot(g, predictor = "x1", moderator = "x2",
#           alpha = .05, jn = T, title0=" ",
#           predictorLab = "x1", moderatorLab = "x2")
```

Description

A dataset simulated using methods described in the reference below.


Usage

```r
data(TrueInit_Y14)
```
**Format**

A data frame with 60,000 rows and 10 variables

**Details**

The variables are as follows:

- batch. Batch number from simulation
- kk. Unclear
- trueInit. True initial condition
- id. Person ID
- time. Continuous time of measurement
- y1. Observed score 1
- y2. Observed score 2
- y3. Observed score 3
- co1. Covariate 1
- co2. Covariate 2

---

**VARsim**  
*Simulated time series data for multiple imputation in dynamic modeling.*

**Description**

A dataset simulated using a vector autoregressive (VAR) model of order 1 with two observed variables and two covariates. Data are generated following the simulation design illustrated by Ji and colleagues (2018). Specifically, missing data are generated following the missing at random (MAR) condition under which the probability of missingness in both dependent variables and covariates is conditioned on two completely observed auxiliary variables.

**Usage**

data(VARsim)

**Format**

A data frame with 10,000 rows and 8 variables
Details

The variables are as follows:

- ID. ID of the participant (1 to 100)
- Time. Time index (100 time points from each subject)
- ca. Covariate 1
- cn. Covariate 2
- wp. Dependent variable 1
- hp. Dependent variable 2
- x1. Auxiliary variable 1
- x2. Auxiliary variable 2

References

vdpData

Another simulated multilevel multi-subject time series of a Van der Pol Oscillator

Description

A dataset simulated using methods described in the reference below.

Usage

data(vdpData)

Format

A data frame with 60,000 rows and 11 variables

Details

The variables are as follows:

- batch. Batch number from simulation
- kk. Unclear
- trueInit. True initial condition
- id. Person ID
- time. Continuous time of measurement
- y1. Observed score 1
- y2. Observed score 2
- y3. Observed score 3
- u1. Covariate 1
- u2. Covariate 2
- trueb. True value of person-specific random effect
Index

*Topic datasets

EMG, 31
EMGsim, 32
LinearOsc, 35
LogisticSetPointSDE, 36
NonlinearDFAsim, 40
oscData, 41
Oscillator, 41
Outliers, 43
PFAsim, 45
PPsim, 50
RSPPsim, 66
TrueInit_Y14, 69
VARsim, 70
vdpData, 72
$,dynrCook-method (dynrCook-class), 29
$,dynrModel-method (dynrModel-class), 30
$,dynrRecipe-method (dynrRecipe-class), 30
$<-,dynrModel-method (dynrModel-class), 30

autoplot, 14
autoplot.dynrCook (dynr.ggplot), 17
autoplot.dynrTaste, 9

chol, 19
coeff, 14
coeff.dynrCook, 38
coeff.dynrCook (coeff.dynrModel), 9
coeff.dynrModel, 9
coeff<- (coeff.dynrModel), 9
confint, 14
confint.dynrCook, 10

deviance, 14
deviance.dynrCook (logLik.dynrCook), 37
diag, 12
diag (diag, character-method), 12
diag.character (diag, character-method), 12
dynr (dynr-package), 3
dynr.package, 3
dynr.cook, 12, 14, 21, 29
dynr.data, 14, 21, 22
dynr.flowField, 15
dynr.ggplot, 17, 48
dynr.ldl, 19
dynr.mi, 20
dynr.model, 21, 30
dynr.plotFreq, 22
dynr.taste, 23
dynr.taste2, 25
dynr.trajecory, 26
dynr.version, 28
dynrCook-class, 29
dynrDebug-class (dynrCook-class), 29
dynrDynamics-class, 29
dynrDynamicsFormula-class (dynrDynamics-class), 29
dynrDynamicsMatrix-class (dynrDynamics-class), 29
dynrInitial-class, 29
dynrMeasurement-class, 29
dynrModel-class, 30
dynrNoise-class, 30
dynrRecipe-class, 30
dynrRegimes-class, 31
dynrTrans-class, 31

EMG, 31
EMGsim, 32

getdx, 33

initialize, 14
internalModelPrep, 34

LinearOsc, 35
LogisticSetPointSDE, 36
logLik, 14
logLik.dynrCook, 10, 37
names, 14
names.dynrCook-method, 38
names.dynrModel-method, 38
nobs, 14
nobs.dynrCook, 39
nobs.dynrModel, 39
NonlinearDFAsim, 40
oscData, 41
Oscillator, 41
Outliers, 43
PFAsim, 45
plot, 14
plot.dynrCook, 47
plotFormula, 48, 66
plotGCV, 49
PPsim, 50
prep.formulaDynamics, 21, 22, 29, 30, 51, 59
prep.initial, 21, 22, 29, 30, 53
prep.loadings, 21, 29, 56
prep.matrixDynamics, 21, 22, 29, 30, 58
prep.measurement, 21, 22, 29, 30, 57, 59, 61
prep.noise, 21, 22, 30, 57, 60
prep.regimes, 21, 22, 30, 31, 62
prep.tfun, 21, 22, 30, 31, 64
print, 14, 55, 59, 60, 63
print.dynrCook-method (dynrCook-class), 29
print.dynrModel-method (dynrModel-class), 30
print.dynrRecipe-method (dynrRecipe-class), 30
printex, 22, 55, 60, 62, 63, 65
printex, dynrCook-method (printex), 65
printex, dynrDynamicsFormula-method (printex), 65
printex, dynrDynamicsMatrix-method (printex), 65
printex, dynrInitial-method (printex), 65
printex, dynrMeasurement-method (printex), 65
printex, dynrNoise-method (printex), 65
printex, dynrRegimes-method (printex), 65
RSPPsim, 66
show, 14, 55, 59, 60, 63
show, dynrCook-method (dynrCook-class), 29
show, dynrModel-method (dynrModel-class), 30
show, dynrRecipe-method (dynrRecipe-class), 30
substituteFormula, 67
summary, 14
summary.dynrCook, 68
theta_plot, 68
TrueInit_Y14, 69
VArSim, 70
cov, 14
cov.dynrCook, 71
dvpData, 72