Package 'DySeq'

March 6, 2017

Type Package

Date 2017-03-05

Version 0.22

Title Functions for Dyadic Sequence Analyses

Author Peter Fuchs

Maintainer Peter Fuchs <Pete.Fuchs@gmail.com>

Depends R (>= 3.2.1)

Imports MASS, boot, TraMineR, graphics

Suggests survival, lme4

Description

Functions for dyadic binary/dichotomous sequence analyses are implemented in this contribution. The focus is on estimating actor-partner-interaction models using various approaches, for instances the approach of Bakeman & Gottman's (1997) <DOI:10.1017/cbo9780511527685>, generalized multi-level models, and basic Markov models. Moreover, coefficients of one model can be translated into those of the other models. Finally, simulation-based power analyses are provided.

License GPL-3

LazyData TRUE

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-03-06 08:28:49

R topics documented:

`APIMtoTrans` *APIMtoTrans*

Description

Transforms APIM beta-coefficients into an equivalent transition matrix. (only implemented for binary dyadic sequences!)

Usage

```
APIMtoTrans(B0_1, AE_1, PE_1, Int_1, B0_2, AE_2, PE_2, Int_2)
```

Arguments

`B0_1`	Intercept if the first sequence is the dependent variable
`AE_1`	Actor-Effect if the first sequence is the dependent variable
`PE_1`	Partner-Effect if the first sequence is the dependent variable
`Int_1`	Actor*Partner-Intercation-Effect if the first sequence is the dependent variable
`B0_2`	Intercept if the second sequence is the dependent variable
`AE_2`	Actor-Effect if the second sequence is the dependent variable
`PE_2`	Partner-Effect if the second sequence is the dependent variable
`Int_2`	Actor*Partner-Intercation-Effect if the second sequence is the dependent variable

Value

myTrans a transition matrix

Examples

```
trans1<-APIMtoTrans(B0_1=0.1, AE_1=0.2, PE_1=0.3, Int_1=0.4,
                    B0_2=0.5, AE_2=0.6, PE_2=0.7, Int_2=0.8)

#inspecting the equivalent matrix
trans1

#backtesting by transforming the matrix back into beta-coefficients
round(TransToAPIM(trans1),6)
```

Basic_Markov_as_APIM *Basic_Markov_as_APIM*

Description

Fits a basic Markov-model on dyadic sequences. The transition matrix is converted into equivalent APIM-beta-coefficients. Bootstrapping is used for approximating p-values. (H1: Effect is different from zero)

Usage

```
Basic_Markov_as_APIM(x, first, second, boot = 1000, SimOut = FALSE,
  CPU = 1, sim = "ordinary", parallel = "no")
```

Arguments

x	Dataframe or matix containing the sequences (not combined!)
first	a vector that indicates all columns of the first sequence
second	a vector that indicates all columns of the second sequence
boot	number of bootstrap samples
SimOut	For simulation purposes: If TRUE output and tansition matrix will be omitted.
CPU	passes argument to boot()
sim	passes argument to boot()
parallel	passes argument to boot()

Examples

```
## Not run:
# Simulating example-data:
trans1<-APIMtoTrans(B0_1=0, AE_1=1, PE_1=0, Int_1=0,
                    B0_2=0, AE_2=0, PE_2=0, Int_2=0)

x<-simSeqSample(trans=trans1, initial=rep(.25,4), length=100, N=100)

# Running the function,
# small boot-size sample only for demonstration purposes!
Basic_Markov_as_APIM(x, 1:100, 101:200, boot=10)

## End(Not run)
```

CouplesCope *Dyadic sequences of 64 heterosexual couples*

Description

Will be used to exemplify a typical data structure and to illustrate the several function of this package.

Usage

```
CouplesCope
```

Format

A data frame with 64 rows and 98 variables:

Details

In the original sample study, which was promoted as a study on close relationship and stress, 198 heterosexual couples living in Switzerland participated.

The couples had to have been in the current romantic relationship for at least a year and to use German language as their main communication language. During the study, either the woman, the man, or both partners were stressed using the Trier Social Stress Test (TSST; Kirschbaum, Pirke, Hellhammer, 1993). For exemplification purposes, only those 64 couples are included where only the female partner was stressed. Directly after the stress induction, both partners joint again and the couple was left alone for eight minutes. During this period (a 'fake' waiting condition) the two partners were filmed for 8 minutes divided into 48 intervals of ten seconds length. It was coded if the female partners showed stress communication (SC) within an interval (sequence 1; Colums 50:97) and if the male partner showd dyadic coping reactions (DC; sequence 2; columns 2:49). For rurther insides about dyadic coping and/or stress communication, see Bodenmann (2015).

Coding:

- code: ID variable

- IKCB01-IKCB48: Was stress communication (SC) shown in the time intervalls 1-48?
- DCCB01-DCCB48: Was dyadic coping (DC) shown in the time intervalls 1-48?
- EDCm: Men's self-assessed dyadic coping ability

Source

data: research grants 100013-115948/1 and 100014-115948 from the Swiss National Science Foundation.

References

- Kirschbaum, C., Pirke, K. M., & Hellhammer, D. H. (1993) <DOI: 10.1159/000119004>
- Bodenmann, G. (2015) <DOI: 10.1037/11031-002>

`DySeq` *DySeq Overview*

Description

A collection of functions for dyadic sequence anaylses.

Details

DySeq provides an implementation of Bakeman & Gottman's (1997) approach of aggregated logit models. And some additional functions for data preparation. A commented R-Script with examples can be found at https://github.com/PeFox/DySeq_script .

- Combining two sequences into one, see `StateExpand`.
- Producing state-transition tables from two combined dyadic dichotomous sequences, see `StateTrans`.
- For analyzing multiple state-transition tables using Bakeman & Gottman's (1997) approach, see `LogSeq`.
- For analyzing dyadic sequences using generalized multi-level models, see `MLAP_Trans`.
- For analyzing dyadic sequences using a basic Markov model, see `Basic_Markov_as_APIM`.
- For estimating the expected number of cases with low- or zero-frequencies for a behavioral coding study, see `EstFreq` or `EstTime`.
- For 'decumulating' a cumulated hazard function, see `NonCumHaz`.
- For finding the last occurence of a certain state or event within a sequence (needed to transform sequence data into time-to-event data), see `LastOccur`.

Author(s)

Peter Fuchs

EstFreq *EstFreq*

Description

Simulates k state-transition tables (see: StateTrans) based one state-transtion table containing expected population frequencies (relative frequencies!). If simulations should be conducted for different numbers of time intervalls, please see: EstTime

Usage

```
EstFreq(x, t, min.cell = 5, k = 20000)
```

Arguments

x	a matrix containing the assumed probabilities for the expected transition tables
t	number of time intervalls
min.cell	a single integer defines what counts as a low frequency (5 by convention)
k	Number of simulations (at least 20.000 is recommended)

Details

The matrix must have 2*4 dimensions with the following information:

- First column represents if behavior of interest is shown
- Second column represents if behavior of interest is not shown
- First row shows if behavior of interest was shown by both partners in the previous time interval
- Second row shows if behavior of interest was shown only by the partner in the previous time interval
- Third row shows if behavior of interest was shown only by the actor in the previous time interval
- Second row shows if behavior of interest was not shown in the previous time interval

Examples

```
## Not run:
my.trans.table<-matrix(c(0.57, 0.13,0.05,0.05,0.05, 0.05,0.05,0.05),4,2)

my.cellproblems<-EstFreq(my.trans.table, 100, 5, k=20000)

my.cellproblems

## End(Not run)
```

EstTime *EstTime*

Description

- Simulates k state-transition tables (see: StateTrans) based one state-transtion table containing expected population frequencies using the EstFreq function.
- Repeats simulations for a number of defined time intervals or until a termination criterion is reached.
- The estimated number of cells with low or zero frequencies are computed.
- The proportion of low/zero frequencies is plotted against the number of time points/intervalls.

Usage

```
EstTime(x, t = NA, crit = 0.05, zero = T, min.cell = 5, k = 20000,
  pos = "bottomleft", smoothed = T, show.it = 10, max.it = 10000)
```

Arguments

x	a matrix containing the assumed probabilities (see EstFreq for further detail!)
t	optional: a vector of time intervalls for which frequencies should be simulated. Can be used to decrease simulation time!
crit	optional: but must be sepcified if t is not; Simulations will end if the relative frequency of zero cells is less than crit
zero	optional: if FALSE simulations will end if number of low frequencies (instead of zero frequencies) is less than crit
min.cell	a single integer defines what counts as a low frequency (lower than 5 by convention)
k	Number of simulations (at least 20.000 is recommended)
pos	position of the output's legend. Options are: "bottomleft", "bottomright", "upperleft", and "upperright".
smoothed	logical value. If true, output lines will be smoothed!
show.it	single integer that defines which steps of iteration protocol should be shown. Only active if t is not defined else iteration protocoll is replaced with a progression bar
max.it	single integer that defines the maximum number of iterations if t is not specified.

Details

First vector represents time points, second vector provides rel. frequency of cases with zero's, third vector rel. frequency of cases with low cell frequencies.

Value

EstTime object; a list of three vectors if printed, provides a plot of expected number of low and zero cell frequencies

Examples

```
## Not run:
my.trans.table<-matrix(c(0.57, 0.13,0.05,0.05,0.05, 0.05,0.05,0.05),4,2)
my.cellproblems<-EstTime(my.trans.table, k=500) # low k only for examplification purposes!
my.cellproblems

## End(Not run)
```

`GiveSome` *Dyadic sequences of 45 subjects engaging a social dilemma*

Description

The data set stems from the bachelor thesis of Halstenberg (2016) and contains sequences of 45 subjects that engaged in a 32-rounds-long four-coin dilemma. That is, each player starts with four coins that are worth one point for oneself and two points for the opponent. Both players have to submit zero to four of them to the other player. The decision is made secretly and simultaneously.

Usage

```
GiveSome
```

Format

An object of class `matrix` with 42 rows and 62 columns.

Details

The computer was set to ignore the humans behavior at all. Instead the 32 rounds were divided into eight blocks. Within each block, the computer gave one-times one, two-times two, and one-times three coins in randomized order. The only exception was the very first turn, in which the algorithm always gave two coins followed by one, two and three coins in randomized order.Thus, on average, the algorithm gave two coins. Hence, it was always possible for the human player to give more or fewer coins than the algorithm did before.

For the humnan, it was coded whether the human player gave more (1) coins in his turn than the algorithm did in the last turn or not (0). The same was coded for the computer. Thus, coding started in the second turn resulting in 31 entries for each of both.

The data frame contains 45 rows (subjects) and 62 columns. Columns 1 to 31 correspond to the human behavior, columns 32:61 to the algorithm.

Source

Halstenberg, E. (2016). The effect of social value orientation on cooperation in a four-coin dilemma: a quasi-replication study using the svo slider measure. (Unpublished bachelor thesis). University Bielefeld, Germany.

`LastOccur` *LastOccur*

Description

Returns index of last occurence: Each row is scanned for the last column in which a certain integer is shown. Consider the following example: one row has the values 1-2-2-1-0-4, last occurence of 0 would be the fith column, last occurence of 1 would be the forth column, and so on.

Usage

```
LastOccur(x, y)
```

Arguments

`x`	Dataframe or matix containing one sequence per row
`y`	The value of interest

Value

returns a vector containing the index of the last event occurence for every row.

Examples

```
# Example 1: Small artificial data

 my.data<-matrix(c(1,0,1,1,
                   0,0,1,0,
                   1,0,0,0,
                   0,0,0,1),4,4, TRUE) # create data

 my.data # inspect sampe data
 LastOccur(my.data,1) # last Occurence of one
 LastOccur(my.data,0) # last Occurence of zero


# Example 2: Real data

 data(CouplesCope)
 LastOccur(CouplesCope[,2:49],1)
```

LogSeq *LogSeq*

Description

Implementation of Bakeman & Gottman (1997) for sequence analysis. Kenny, Kashy & Cook (2006) provide further examples.

Usage

```
LogSeq(x, delta = 0.5, subgroups = NA, single.case = FALSE)
```

Arguments

x	a state.trans object or a list of 4*2 state-transition tables
delta	constant added to every cell, required if zero frequencies occur!
subgroups	an optional vector containing groupmembership if estimates should be compared between groups
single.case	should p-values be computed for single case analysis

Details

- Runs logit models over a multiple number of state-transition tables, see: StateTrans
- Aggregates coefficients of all logit models and tests them against zero.
- If subgroups are defined, coefficients are tested to be different between groups.
- Print-function displays mean logit-coefficients and p-values.

References

- Bakeman, R., & Gottman, J. M. (1997) <DOI: 10.1017/cbo9780511527685 >
- Kenny, D. A., Kashy, D. A., & Cook, W. L. (2006) <DOI: 10.1177/1098214007300894>

Examples

```
## Not run:
data(CouplesCope)
my.states<-StateExpand(CouplesCope, 2:49, 50:97)
my.trans<-StateTrans(my.states, FALSE)
my.logseq<-LogSeq(my.trans, single.case=TRUE)
my.logseq

plot(my.logseq) # interaction can be plotted

single.LogSeq(my.logseq, 41) # for single case analysis

## End(Not run)
```

MLAP_Trans *MLAP_Trans*

Description

Transforms data, which has been transformed by the function ML_Trans from sequence data into multi-level transitions, into lagged partner and lagged actor effects. This transformation is often required before fitting a multi-level actor-partner-interaction model (APIM).

Usage

```
MLAP_Trans(x)
```

Arguments

x	the output of the ML_Trans function

Examples

```
# Example: Applying a APIM on the example data

# Transforms Sequences into Multi-Level data
ML_data<-ML_Trans(CouplesCope, 2:49, 50:97)

# Transforms transitions into lagged Actor and Partner effects
MLAP_data<-MLAP_Trans(ML_data)

# Data preparation

# In example data first seq referred to females
# and second to males

names(MLAP_data)[1]<-"sex"
MLAP_data$sex<-as.factor(MLAP_data$sex)
levels(MLAP_data$sex)<-c("female", "male")

# Effectcoding
MLAP_data$Partner[MLAP_data$Partner==0]<-(-1)
MLAP_data$Actor[MLAP_data$Actor==0]<-(-1)

# Fits a multi-level APIM using lme4
# Here a random intercept-only model
## Not run:
## make sure lme4 is installed!
## and loaded!
#install.packages("lme4")
# library(lme4)

set.seed(1234)
```

```
glmer(DV~1+sex+Actor+Partner+Actor*Partner+
       sex*Actor+sex*Partner+sex*Actor*Partner+
       (1|ID),
     data=MLAP_data,
     family=binomial)

## End(Not run)
```

ML_Trans *ML_Trans*

Description

Transforms transition tables into multi-level data. Each transition between states is handled as a single observation, which is nested within dyads.

Usage

```
ML_Trans(data, first, second)
```

Arguments

data	a data.frame, which contains dyadic sequences in a bride format
first	column-index that defines the first sequence of each dyad
second	column-index that defines the second sequence of each dyad

Examples

```
# Example: Sequences from couples cope into multi-level data

data(CouplesCope)
ML_data<-ML_Trans(CouplesCope, 2:49, 50:97)
```

NonCumHaz *NonCumHaz*

Description

Computes the non-cumulated hazard from a vector containing the cumulated hazard. Can be applied directly to survfit-object (no need to extract the hazard first!). If the vector contains only hazard information for some but not all time intervalls, e.g. intervals with a hazard of zero are left out, a second vector is needed to match the hazard to the corresonding time intervals.

Usage

```
NonCumHaz(x, t = NA, plot = FALSE)
```

Arguments

x	a vector containing cumulated hazard or a survfit-object from the 'survival'-package
t	optional: vector containting time reference for x (is required for plot)
plot	logical value indicating if non-cumulated plot should be generated

Examples

```
# Example 1: Short artificial data
# example cumulated hazard with time referenz
cumhaz<-c(0.2 ,0.21 ,0.31 ,0.44 ,1.1  ,1.1  ,1.12 ,1.2)
time<-c(4     ,5    ,6    ,7    ,10   ,14   ,15   ,16)

NonCumHaz(cumhaz, time, plot=TRUE)



# Example 2: Every hazard entry represents one point of time

# if every hazard entry repesents one point of time
NonCumHaz(cumhaz, 1:8, plot=TRUE)



# Example 3: real data and real researchquestion
## Not run: #install.packages("survival")
library(survival)

# How long till the last stress signal
my.last<-LastOccur(CouplesCope[,50:97],1)

# If last stress signal was in time intervall 48,
# stress did not end till the observation duration
event<-rep(1,length(my.last))
event[my.last==48]<-0

# Coxregression
my.surv<-Surv(my.last,event) # creates a object for survival time analysis
my.fit<-survfit(coxph(my.surv~1)) # fits coxregression without covariates on the data

plot(my.fit) # survival curve
plot(my.fit, fun="cumhaz") # cumulated survival curve

# Different uses for NonCumHaz
NonCumHaz(my.fit, plot=TRUE)
NonCumHaz(my.fit$cumhaz, my.fit$time, plot=TRUE) # if over packages than 'survival'are used
```

```
## End(Not run)
```

NumbOccur *NumbOccur*

Description

Returns the Number of occurences for sequences with different lenghts. Outside the context of sequence analysis this means that for each frequency of one specifics integer will be computed.

Usage

```
NumbOccur(x, y, t = NA, prop = TRUE)
```

Arguments

x	Dataframe or matix containing one sequence per row
y	single integer: represents the occurence that should be counted
t	optional vector that contains the lengths of sequences
prop	if TRUE: proportion will be computed,if FALSE: sum will be computed

Value

returns a vector containing containing the number of occurences.

Examples

```
# Example 1: Small artificial example

# Creating data, if sequence ends, rest should be 'NA'
seq1<-c(1,0,0,0,1,0,1, NA, NA, NA) # 3 out of 7 Entrys should be round about .43
seq2<-c(1,1,1,1, NA, NA, NA, NA, NA, NA) # 4 out of 4 should be 1
seq3<-c(1,0,0,0,1,1, NA, NA, NA, NA) # 3 out of 6 should be .50
my.data<-rbind(seq1,seq2,seq3)

# Determine the proportion of ones in my.data
NumbOccur(my.data,1)
NumbOccur(my.data,1, prop=FALSE) # compute absolute frequencies


# Example 2: Real data dyadic sequences
# A researcher is interested in how often was a certain behavior
# shown till another one stopped completely

my.last<-LastOccur(CouplesCope[,2:49],1) # how long till stress ended?
NumbOccur(CouplesCope[,50:97],1, my.last) # how often did dyadic coping occure in this time?
```

`plot.LogSeq` *plot.LogSeq*

Description

Generates interaction diagram for LogSeq Objects, see: `LogSeq`

Usage

```
## S3 method for class 'LogSeq'
plot(x, y, ...)
```

Arguments

`x`	a LogSeq object, that should be printed.
`y`	further arguments passed to or from other methods.
`...`	further arguments passed to or from other methods.

`print.EstFreq` *print.state.trans*

Description

Generates output for EstFrag object, see: `EstFreq`

Usage

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

Arguments

`x`	a EstFrag object, that should be printed. See help(EstFreq)
`...`	further arguments passed to or from other methods.

print.EstTime *print.EstTime*

Description

Generates output for EstTime object, see: EstTime

Usage

```
## S3 method for class 'EstTime'
print(x, pos = NA, ...)
```

Arguments

x	a EstTime object, printing it will result in a frequency plot. See help(EstTime)
pos	position of legend, same arguments avaible as in the argument pos from the plot() function.
...	further arguments passed to or from other methods.

print.LogSeq *print.LogSeq*

Description

Generates output for LogSeq Objects, see: LogSeq

Usage

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

Arguments

x	a LogSeq object, that should be printed
...	further arguments passed to or from other methods.

print.state.trans *print.state.trans*

Description

Generates output for state.trans object, see: StateTrans

Usage

```
## S3 method for class 'state.trans'
print(x, ...)
```

Arguments

x	a state.trans object, that should be printed
...	further arguments passed to or from other methods.

simSeq *simSeq*

Description

Simulates a single sequence with four possible states.

Usage

```
simSeq(trans, initial, length)
```

Arguments

trans	a 4x4 matrix containing transition probabilities
initial	a four element vector containing initial states probabilities
length	single value specifying the length of the simulated sequence

Examples

```
test1<-matrix(c(0.5 , 0.2 , 0.2 , 0.1,
                0.8 , 0.05, 0.05, 0.1,
                0.5 , 0.1 , 0.2 , 0.2,
                0.1 , 0.1 , 0.1 , 0.7) , 4 , 4 , byrow = TRUE)

initial<-c(.25 , .25 , .25 , .25)

simSeq(test1, initial, 30)
```

simSeqSample *simSeqSample*

Description

Simulates N single sequences with four possible states.

Usage

```
simSeqSample(trans, initial, length, N)
```

Arguments

trans	a 4x4 matrix containing transition probabilities
initial	a four element vector containing initial states probabilities
length	single value specifying the length of the simulated sequence (columns)
N	number of sequences which should be simulated (rows)

Examples

```
test1<-matrix(c(0.5 , 0.2 , 0.2 , 0.1,
                0.8 , 0.05, 0.05, 0.1,
                0.5 , 0.1 , 0.2 , 0.2,
                0.1 , 0.1 , 0.1 , 0.7) , 4 , 4 , byrow = TRUE)

initial<-c(.25 , .25 , .25 , .25)

simSeq(test1, initial, 30)
```

single.LogSeq *single.LogSeq*

Description

prints estimates vor single case out of an LogSeq object, see: LogSeq

Usage

```
single.LogSeq(x, case)
```

Arguments

x	a LogSeq object, that should be printed.
case	determines which case should be shown.

Examples

```
data(CouplesCope)
my.states<-StateExpand(CouplesCope, 2:49, 50:97)
my.trans<-StateTrans(my.states, FALSE)
my.logseq<-LogSeq(my.trans, single.case=TRUE)
my.logseq
single.LogSeq(my.logseq, 41)  # prints estimates for case 41
```

`StateExpand` *StateExpand*

Description

Transforms dyadic binary data into state-expand-sequences (combines two corresponding sequences into one for every row of a dataframe)

Usage

```
StateExpand(x, pos1, pos2, replace.na = FALSE)
```

Arguments

`x`	Dataframe or matix containing the sequences that should be combined
`pos1`	a vector that indicates all columns of the first sequence
`pos2`	a vector that indicates all columns of the second sequence
`replace.na`	a numeric that is used for replacement. If FALSE: no replacement will take place! 0 is handled as zero in this case not als FALSE!

Details

Takes a dataframe or matrix with dyadic binary data in wide data format, that is:

- one observation unit (for example one couple) is represented by one row
- every observation unit has two sequences with the same length
- entrys must be corresponding to each other (for example they represent same time intervalls)
- every sequence contains only zeros and/or ones (for example behavior is shown or not)

and transforms it into state-expand-sequences:

- one sequence per observation unit that contains the same information as before
- every entry represents the combination of the corresponding previous entrys
- 0 represents two former zeros,
- 1 represents a one in the first sequence and a zeros in the second
- 2 represents a zero in the first sequence and a one in the second

- 3 represents a one in both former sequeces

why/how to use:

- Most packages are only suited for univariat sequence analysis.
- This function transforms dyadic dequences into univariate sequences.
- state-expand-sequences are needed for some of the other functions of this package.

Value

returns a matrix with the combined sequences.

References

- Bakeman, R., & Gottman, J. M. (1997) <DOI: 10.1017/cbo9780511527685 >
- Kenny, D. A., Kashy, D. A., & Cook, W. L. (2006) <DOI: 10.1177/1098214007300894>

Examples

```
# Example 1
data(CouplesCope) # Load sample data
CouplesCope[1:5,] # inspect first five cases

my.expand<-StateExpand(CouplesCope, 2:49, 50:97)
my.expand[1:5,] # inspect first five cases of the combined sequences



# Example 2: with NA replacement
data(CouplesCope)

# copy part of the example data
# excluding code and EDCm for simplification
na.CouplesCope<-CouplesCope[,2:97]


# fill it with 10% NA's as an example:
na.CouplesCope[matrix(sample(c(TRUE, rep(FALSE,9)),64*96, TRUE), 64, 96)]<-NA
na.CouplesCope[1:5,] # inspect the first 5 cases

# demonstrate na.replace: combine states and fill NA's with zeros!
my.expand<-StateExpand(na.CouplesCope, 1:48, 49:96, replace.na=0)
my.expand[1:5,] # inspect the first 5 cases



## Not run:
 # Example 3: Use StateExpand for further analyis
 #            or plotting using the Package TraMineR

 # install.packages("TraMineR") # install "TraMineR" for graphical analysis
 library(TraMineR) #load TraMineR
```

```
my.expand<-StateExpand(CouplesCope, 2:49, 50:97) # create combined sequences

# create labels for plot
couple.labels <-c("no reaction", "stress only", "coping only", "both reactions")

# create a sequence object (the way TraMineR represents sequences)
couple.seq <- seqdef(my.expand, labels = couple.labels)
seqdplot(couple.seq)

detach(TraMineR) # unloading TraMineR

## End(Not run)
```

StateTrans *StateTrans*

Description

Produces a state transition table for dyadic binary sequences.

Usage

```
StateTrans(x, first = TRUE, dep.lab = c("1", "0"), indep.lab = c("1-1",
  "1-0", "0-1", "0-0"))
```

Arguments

x	Dataframe or matix containing combined sequences, see help(StateExpand)
first	logical value indicating if the first sequence should used as dependend variable (TRUE) or the second (FALSE)
dep.lab	two-element string vector with labels for dependend variable (first entry corresponds to the value zero, the second to one)
indep.lab	four-element string vector with labels for the combined variable (order corresponds to the order of the StateExpand function)

Details

That is, the behavior of interest in interval t, is mapped against the combination of the observed behaviors in the preceding interval (t - 1). Hence, the total absolute frequency equals the number of time intervals minus 1. And the number of obtained tables is equal the number of sequence-pairs.

printing the output will display mean frequencies. For inspecting individual cases use [[original-rownumber]].

For an extensive overview see Kenny, Kashy and Cook (2006). The original idea stems from (to our knowledge) Bakeman and Gottman (1997).

References

- Bakeman, R., & Gottman, J. M. (1997) <DOI: 10.1017/cbo9780511527685 >
- Kenny, D. A., Kashy, D. A., & Cook, W. L. (2006) <DOI: 10.1177/1098214007300894>

Examples

```
# Example 1: Sequences from couples cope

data(CouplesCope)
my.s<-StateExpand(CouplesCope, 2:49, 50:97)

# First sequence is dependend variable
# - what behavior preceeds stress signals?
StateTrans(my.s)

# Second sequence is dependend variable
# - what behavior preceeds dyadic coping signals?
StateTrans(my.s, FALSE)

# investigating a single case
StateTrans(my.s, FALSE)[[41]]
```

`TransToAPIM` *TransToAPIM*

Description

Transforms a transition matrix into equivalent APIM beta-coefficients. (only implemented for binary dyadic sequences!)

Usage

```
TransToAPIM(M)
```

Arguments

`M`	a transition matrix which should be converted to APIM like beta-coefficients

Value

B0_1 Intercept if the first sequence is the dependent variable

AE_1 Actor-Effect if the first sequence is the dependent variable

PE_1 Partner-Effect if the first sequence is the dependent variable

Int_1 Actor*Partner-Intercation-Effect if the first sequence is the dependent variable

B0_2 Intercept if the second sequence is the dependent variable

AE_2 Actor-Effect if the second sequence is the dependent variable

PE_2 Partner-Effect if the second sequence is the dependent variable

Int_2 Actor*Partner-Intercation-Effect if the second sequence is the dependent variable

Examples

```
test<-matrix(c(0.41 , 0.28 , 0.19 , 0.12,
               0.18 , 0.18 , 0.32 , 0.32,
               0.18 , 0.22 , 0.27 , 0.33,
               0.05 , 0.09 , 0.30 , 0.55) , 4 , 4 , byrow = TRUE)

x<-TransToAPIM(test)
# inspecting the beta-coefficients
x

#backtesting (last row will show minor errors caused by rounding)
round(APIMtoTrans(x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]),2)
```

Index