

# Package ‘TraMineR’

January 19, 2022

**Version** 2.2-3

**Date** 2022-01-19

**Title** Trajectory Miner: a Toolbox for Exploring and Rendering Sequences

**Depends** R (>= 3.0.0)

**Imports** utils, graphics, grDevices, stats, cluster, colorspace, RColorBrewer, weights, cNORM, boot

**Suggests** xtable, TraMineRextras

**SystemRequirements** C++11

**Description** Toolbox for the manipulation, description and rendering of sequences, and more generally the mining of sequence data in the field of social sciences. Although the toolbox is primarily intended for analyzing state or event sequences that describe life courses such as family formation histories or professional careers, its features also apply to many other kinds of categorical sequence data. It accepts many different sequence representations as input and provides tools for converting sequences from one format to another. It offers several functions for describing and rendering sequences, for computing distances between sequences with different metrics (among which optimal matching), original dissimilarity-based analysis tools, and simple functions for extracting the most frequent subsequences and identifying the most discriminating ones among them. A user's guide can be found on the TraMineR web page.

**License** GPL (>= 2)

**URL** <http://traminer.unige.ch>

**Encoding** UTF-8

**Maintainer** Gilbert Ritschard <[gilbert.ritschard@unige.ch](mailto:gilbert.ritschard@unige.ch)>

**NeedsCompilation** yes

**Author** Alexis Gabadinho [aut, cph],  
Matthias Studer [aut, cph] (<<https://orcid.org/0000-0002-6269-1412>>),  
Nicolas Müller [aut],  
Reto Bürgin [aut],  
Pierre-Alexandre Fonta [aut],  
Gilbert Ritschard [aut, cre, cph]  
(<<https://orcid.org/0000-0001-7776-0903>>)

**Repository** CRAN

**Date/Publication** 2022-01-19 16:52:42 UTC

**R topics documented:**

TraMineR-package . . . . .	4
actcal . . . . .	5
actcal.tse . . . . .	6
alphabet . . . . .	7
bfspell . . . . .	8
biofam . . . . .	9
cpal . . . . .	11
dissassoc . . . . .	12
disscenter . . . . .	14
dissdomassoc . . . . .	16
dissmfacw . . . . .	18
dissrep . . . . .	20
disstree . . . . .	23
disstree2dot . . . . .	25
disstreeleaf . . . . .	28
dissvar . . . . .	29
ex1 . . . . .	30
ex2 . . . . .	31
famform . . . . .	32
is.stslist . . . . .	32
mvad . . . . .	33
plot.seqdiff . . . . .	35
plot.stslist . . . . .	36
plot.stslist.freq . . . . .	38
plot.stslist.meant . . . . .	40
plot.stslist.modst . . . . .	41
plot.stslist.rep . . . . .	43
plot.stslist.statd . . . . .	44
plot.subseqelist . . . . .	46
plot.subseqelistchisq . . . . .	47
print.stslist . . . . .	48
read.tda.mdist . . . . .	49
seqalign . . . . .	50
seqcomp . . . . .	51
seqconc . . . . .	52
seqcost . . . . .	53
seqdecomp . . . . .	57
seqdef . . . . .	58
seqdiff . . . . .	62
seqdim . . . . .	64
seqdist . . . . .	64
seqdistmc . . . . .	71
seqdss . . . . .	73
seqdur . . . . .	75
sequeapplysub . . . . .	76
seqecmpgroup . . . . .	77

seqeconstraint . . . . .	79
seqecontain . . . . .	81
seqecreate . . . . .	82
seqefsub . . . . .	84
seqeid . . . . .	87
seqelength . . . . .	87
seqetm . . . . .	88
seqeweight . . . . .	90
seqfind . . . . .	91
seqformat . . . . .	92
seqfpos . . . . .	97
seqgen . . . . .	98
seqibad . . . . .	99
seqici . . . . .	100
seqient . . . . .	102
seqindic . . . . .	103
seqintegr . . . . .	106
seqipos . . . . .	108
seqistatd . . . . .	110
seqivardur . . . . .	111
seqivolatility . . . . .	112
seqlegend . . . . .	114
seqlength . . . . .	115
seqLLCP . . . . .	116
seqLLCS . . . . .	117
seqlogp . . . . .	118
seqmeant . . . . .	119
seqmodst . . . . .	121
seqmpos . . . . .	122
seqnum . . . . .	123
seqpcplot . . . . .	124
seqplot . . . . .	129
seqpm . . . . .	135
seqprecarity . . . . .	136
seqprecstart . . . . .	141
seqrecode . . . . .	143
seqrep . . . . .	145
seqsep . . . . .	148
seqST . . . . .	149
seqstatd . . . . .	151
seqstatf . . . . .	152
seqstatl . . . . .	153
seqsubsn . . . . .	154
seqtab . . . . .	156
seqtransn . . . . .	157
seqtrate . . . . .	159
seqtree . . . . .	160
seqtreedisplay . . . . .	162

slab . . . . .	164
TraMineR.check.depr.args . . . . .	165
TraMineR.checkupdates . . . . .	166
TraMineRInternal . . . . .	167

<b>Index</b>	<b>168</b>
--------------	------------

---

TraMineR-package	<i>Trajectory Miner: a Toolbox for Exploring and Rendering Sequences</i>
------------------	--

---

## Description

(Version: 2.2-3) Toolbox for the manipulation, description and rendering of sequences, and more generally the mining of sequence data in the field of social sciences. Although the toolbox is primarily intended for analyzing state or event sequences that describe life courses such as family formation histories or professional careers, its features also apply to many other kinds of categorical sequence data. It accepts many different sequence representations as input and provides tools for converting sequences from one format to another. It offers several functions for describing and rendering sequences, for computing distances between sequences with different metrics (among which optimal matching), original dissimilarity-based analysis tools, and simple functions for extracting the most frequent subsequences and identifying the most discriminating ones among them. A user's guide can be found on the TraMineR web page.

## Details

TraMineR provides tools for both state sequences and event sequences. The first step when using the package is to define a state sequence object (with `seqdef`) if you want to explore state sequences, and an event sequence object (with `seqcreate`) if you are interested in event sequencing.

State sequences are defined from a series of variables giving the states at the successive positions, while event sequences are defined from (vertical) time stamped event data. The package, however, can handle many other different data organizations and provides tools to help converting state sequences into event sequences and vice versa.

## Author(s)

Alexis Gabadinho, Matthias Studer, Nicolas S. Müller, Reto Bürgin, Pierre-Alexandre Fonta, and Gilbert Ritschard

## References

- Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.
- Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. Department of Econometrics and Laboratory of Demography, University of Geneva

**Examples**

```
## load the mvad data
library(TraMineR)
data(mvad)

## create a state sequence object from columns 17 to 86
mvad.seq <- seqdef(mvad[,17:86])

## distribution plot by sex (male)
seqdplot(mvad.seq, group=mvad$male, border=NA)

## compute the LCS pairwise distance matrix
## among the first 10 sequences
mvad.lcs <- seqdist(mvad.seq[1:10,], method="LCS")
```

---

actcal

*Example data set: Activity calendar from the Swiss Household Panel*

---

**Description**

This data set contains 2000 individual sequences of monthly activity statuses from January to December 2000.

**Usage**

```
data(actcal)
```

**Format**

A data frame with 2000 rows, 12 state variables, 1 id variable and 11 covariates.

**Details**

The data set is a subsample of the data collected by the Swiss Household Panel (SHP).

The state column (variable) names are ‘jan00’, ‘feb00’, etc... and correspond to columns 13 to 24.

There are four possible states:

A = Full-time paid job (> 37 hours)  
B = Long part-time paid job (19-36 hours)  
C = Short part-time paid job (1-18 hours)  
D = Unemployed (no work)

The data set contains also the following covariates:

age00        (age in 2000)  
educat00    (education level)

civsta00 (civil status)  
nbadul00 (number of adults in household)  
nbkid00 (number of children)  
aoldki00 (age of oldest kid)  
ayouki00 (age of youngest kid)  
region00 (residence region)  
com2.00 (residence commune type)  
sex (sex of respondent)  
birthy (birth year)

### Source

Swiss Household Panel

### References

<https://forscenter.ch/projects/swiss-household-panel/>

---

actcal.tse

*Example data set: Activity calendar from the Swiss Household Panel  
(time stamped event format)*

---

### Description

This data set contains events defined from the state sequences in the actcal data set. It was created with the code shown in the examples section. It is provided to symplify example of event sequence mining.

### Usage

```
data(actcal.tse)
```

### Format

Time stamped events derived from state sequences in the actcal data set.

### Source

Swiss Household Panel

### See Also

[seqformat](#), [actcal](#)

**Examples**

```

data(actcal)
actcal.seq <- seqdef(actcal[,13:24])

## Defining the transition matrix
transition <- seqetm(actcal.seq, method="transition")
transition[1,1:4] <- c("FullTime"           , "Decrease,PartTime",
  "Decrease,LowPartTime", "Stop")
transition[2,1:4] <- c("Increase,FullTime", "PartTime"           ,
  "Decrease,LowPartTime", "Stop")
transition[3,1:4] <- c("Increase,FullTime", "Increase,PartTime",
  "LowPartTime"           , "Stop")
transition[4,1:4] <- c("Start,FullTime"    , "Start,PartTime"    ,
  "Start,LowPartTime"    , "NoActivity")
transition

## Converting STS data to TSE
actcal.tse <- seqformat(actcal, 13:24, from = "STS", to = "TSE",
  tevent = transition)

## Defining the event sequence object
actcal.eseq <- seqcreate(id=actcal.tse$id,
  time=actcal.tse$time, event=actcal.tse$event)

```

---

alphabet

*Get or set the alphabet of a state or event sequence object*


---

**Description**

For state sequences, the function gets or sets the (short) labels associated to the states in the alphabet of a state sequence object (the list of all possible states). The get form also applies to event sequences, while the set form does not work with event sequences.

**Usage**

```

alphabet(seqdata, with.missing=FALSE)
alphabet(seqdata) <- value

```

**Arguments**

seqdata	a state sequence object of class <code>stslst</code> as defined with the <code>seqdef</code> function or, for the get form only, an event sequence object as defined with <code>seqcreate</code> or a probabilistic suffix tree generated with the PST package.
value	For state sequences only. Vector of characters of the same length as the vector returned by the <code>alphabet</code> function, i.e. one label for each state in the alphabet.
with.missing	Logical. When <code>seqdata</code> is a state sequence object ( <code>stslst</code> ), should the returned alphabet include the <code>nr</code> symbol standing for missing states?

## Details

A state sequence object—created with the [seqdef](#) function—stores sequences as a matrix where columns are factors. The levels of the factors include the alphabet plus the codes for missing values and void elements. The `alphabet` function retrieves or sets the “alphabet” attribute of the state sequence object. The state names composing the alphabet are preferably short labels, since they are used for printing sequences. Longer labels for describing more precisely each state in legend are stored in the “labels” attribute of the sequence object.

For an event sequence object—created with [seqcreate](#)—the get form of `alphabet` works as an alias for `levels`. The set form `alphabet <-` does not work and should not be used.

## Value

For ‘`alphabet`’ a character vector containing the alphabet.

For ‘`alphabet <-`’ the updated state sequence object.

## Author(s)

Alexis Gabadinho and Gilbert Ritschard

## See Also

[seqdef](#)

## Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the alphabet
alphabet(actcal.seq)

## Setting the alphabet
alphabet(actcal.seq) <- c("FT", "PT", "LT", "NO")

## Event sequences
actcal.eseq <- seqcreate(actcal.seq)
alphabet(actcal.eseq)
```

---

bfspell

*Example data set: First 20 biofam sequences in SPELL form*

---

## Description

First 20 sequences of the [biofam](#) data set in SPELL form. The data serve to illustrate the use of [seqformat](#) for converting SPELL data into STS (horizontal) form.



**Usage**

```
data(bfspell)
```

**Format**

A data set with two data frames: `bfspell120` with one row per spell and `bfpdata20` with one row per id. The `bfspell120` data frame contains the spell data themselves (4 variables `id`, `begin`, `end`, `states`) and `bfpdata20` the year when aged 15 (2 variables `id`, `when15`).

**Details**

The states are coded with the following short labels

P = "Parent"

L = "Left"

M = "Married"

LM = "Left+Marr"

C = "Child"

LC = "Left+Child"

LMC = "Left+Marr+Child"

D = "Divorced"

The data is a SPELL representation of `biofam[1:20,10:25]`, corresponding to 20 family life sequences between ages 15 and 30.

**See Also**

[biofam](#)

---

biofam

*Example data set: Family life states from the Swiss Household Panel biographical survey*

---

**Description**

2000 16 year-long family life sequences built from the retrospective biographical survey carried out by the Swiss Household Panel (SHP) in 2002.

**Usage**

```
data(biofam)
```

```
data(bfspell)
```

**Format**

A data frame with 2000 rows, 16 state variables, 1 id variable and 7 covariates and 2 weights variables.

## Details

The *biofam* data set was constructed by Müller et al. (2007) from the data of the retrospective biographical survey carried out by the Swiss Household Panel (SHP) in 2002.

The data set contains (in columns 10 to 25) sequences of family life states from age 15 to 30 (sequence length is 16) and a series of covariates. The sequences are a sample of 2000 sequences of those created from the SHP biographical survey. It includes only individuals who were at least 30 years old at the time of the survey. The *biofam* data set describes family life courses of 2000 individuals born between 1909 and 1972.

The states numbered from 0 to 7 are defined from the combination of five basic states, namely Living with parents (Parent), Left home (Left), Married (Marr), Having Children (Child), Divorced:

0 = "Parent"  
 1 = "Left"  
 2 = "Married"  
 3 = "Left+Marr"  
 4 = "Child"  
 5 = "Left+Child"  
 6 = "Left+Marr+Child"  
 7 = "Divorced"

The covariates are:

sex	
birthyr	(birth year)
nat_1_02	(first nationality)
plingu02	(language of questionnaire)
p02r01	(religion)
p02r04	(religious participation)
cspfaj	(father's social status)
cspmoj	(mother's social status)

Two additional weights variables are inserted for illustrative purpose ONLY (since *biofam* is a sub-sample of the original data, these weights are not adapted to the actual data):

wp00tbgp	(weights inflating to the Swiss population)
wp00tbgs	(weights respecting sample size)

## Source

Swiss Household Panel <https://forscenter.ch/projects/swiss-household-panel/>

## References

Müller, N. S., M. Studer, G. Ritschard (2007). Classification de parcours de vie à l'aide de l'optimal matching. In *XIVe Rencontre de la Société francophone de classification (SFC 2007), Paris, 5 - 7 septembre 2007*, pp. 157–160.

---

cpal

*Get or set the color palette of a sequence object*

---

## Description

This function gets or sets the color palette of a sequence object, that is, the list of colors used to represent the states.

## Usage

```
cpal(seqdata)
cpal(seqdata) <- value
```

## Arguments

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
value	a vector containing the colors, of length equal to the number of states in the alphabet. The colors can be passed as character strings representing color names such as returned by the <a href="#">colors</a> function, as hexadecimal values or as RGB vectors using the <a href="#">rgb</a> function. Each color is attributed to the corresponding state in the alphabet, the order being the one returned by the <a href="#">alphabet</a> .

## Details

In the plot functions provided for visualizing sequence objects, a different color is associated to each state of the alphabet. The color palette is defined when creating the sequence object, either automatically or by specifying a user defined color vector. The `cpal` function can be used to get or set the color palette of a previously defined sequence object.

## Value

For `'cpal(seqdata)'` a vector containing the colors.  
For `'cpal(seqdata) <-'` the updated sequence object.

## Author(s)

Alexis Gabadinho

## See Also

[seqdef](#)

**Examples**

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
## The color palette is automatically set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the color palette
cpal(actcal.seq)
seqiplot(actcal.seq)

## Setting a user defined color palette
cpal(actcal.seq) <- c("blue","red", "green", "yellow")
seqiplot(actcal.seq)
```

---

dissassoc

*Analysis of discrepancy from dissimilarity measures*


---

**Description**

Compute and test the share of discrepancy (defined from a dissimilarity matrix) explained by a categorical variable.

**Usage**

```
dissassoc(diss, group, weights=NULL, R=1000,
          weight.permutation="replicate", squared=FALSE)
```

**Arguments**

diss	A dissimilarity matrix or a dist object (see <a href="#">dist</a> )
group	A categorical variable. For a numerical variable use <a href="#">dissmfacw</a> .
weights	optional numerical vector containing weights.
R	Number of permutations for computing the p-value. If equal to 1, no permutation test is performed.
weight.permutation	Weighted permutation method: "diss" (attach weights to the dissimilarity matrix), "replicate" (replicate case using weights), "rounded-replicate" (replicate case using rounded weights), "random-sampling" (random assignment of covariate profiles to the objects using distributions defined by the weights.)
squared	Logical. If TRUE the dissimilarities diss are squared.

## Details

The `dissassoc` function assesses the association between objects characterized by their dissimilarity matrix and a discrete covariate. It provides a generalization of the ANOVA principle to any kind of distance metric. The function returns a pseudo F statistic, a pseudo Brown-Forsythe F<sub>bf</sub> statistic, and a pseudo R-square that can be interpreted as a usual R-square. The statistical significance of the association is computed by means of permutation tests. The function performs also a test of discrepancy homogeneity (equality of within variances) using a generalization of the Levene statistic and the Bartlett statistic.

There are `print` and `hist` methods (the latter producing an histogram of the permuted values used for testing the significance).

If a numeric group variable is provided, it will be treated as categorical, i.e., each different value will be considered as a different category. To measure the ‘linear’ effect of a numerical variable, use `dissmfacw`.

## Value

An object of class `dissassoc` with the following components:

<code>groups</code>	A data frame with the number of cases and the discrepancy of each group
<code>anova.table</code>	The pseudo ANOVA table
<code>stat</code>	The value of the statistics (Pseudo F, Pseudo F <sub>bf</sub> , Pseudo R <sup>2</sup> , Bartlett, and Levene) and their p-values
<code>perms</code>	The permutation object, containing the values computed for each permutation

## Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

## References

- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2010) Discrepancy analysis of complex objects using dissimilarities. In F. Guillet, G. Ritschard, H. Briand, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence, Volume 292, pp. 3-19. Berlin: Springer.
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009). Analyse de dissimilarités par arbre d’induction. In EGC 2009, *Revue des Nouvelles Technologies de l’Information*, Vol. E-15, pp. 7–18.
- Anderson, M. J. (2001) A new method for non-parametric multivariate analysis of variance. *Austral Ecology* **26**, 32–46.
- Batagelj, V. (1988) Generalized Ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67–74.

**See Also**

[dissvar](#) to compute the pseudo variance from dissimilarities and for a basic introduction to concepts of pseudo variance analysis.

[disstree](#) for an induction tree analyse of objects characterized by a dissimilarity matrix.

[disscenter](#) to compute the distance of each object to its group center from pairwise dissimilarities.

[dissmfacw](#) to perform multi-factor analysis of variance from pairwise dissimilarities.

**Examples**

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities (any dissimilarity measure can be used)
mvad.ham <- seqdist(mvad.seq, method="HAM")

## R=1 implies no permutation test
da <- dissassoc(mvad.ham, group=mvad$gcse5eq, R=10)
print(da)
hist(da)
```

---

disscenter

---

*Compute distances to the center of a group*


---

**Description**

Computes the dissimilarity between objects and their group center from their pairwise dissimilarity matrix.

**Usage**

```
disscenter(diss, group=NULL, medoids.index=NULL,
           allcenter = FALSE, weights=NULL, squared=FALSE)
```

**Arguments**

diss	a dissimilarity matrix such as generated by <a href="#">seqdist</a> , or a dist object (see <a href="#">dist</a> )
group	if NULL (default), the whole data set is considered. Otherwise a different center is considered for each distinct value of the group variable
medoids.index	if NULL, returns the dissimilarity to the center. If set to "first", returns the index of the first encountered most central sequence. If group is set, an index is returned per group. When set to "all", indexes of all medoids (one list per group) are returned.
allcenter	logical. If TRUE, returns a data.frame containing the dissimilarity between each object and its group center, each column corresponding to a group.
weights	optional numerical vector containing weights.
squared	Logical. If TRUE diss is squared.

## Details

This function computes the dissimilarity between given objects and their group center. It is possible that the group center does not belong to the space formed by the objects (in the same way as the average of integer numbers is not necessarily an integer itself). This distance can also be understood as the contribution to the discrepancy (see [dissvar](#)). Note that when the dissimilarity measure does not respect the triangle inequality, the dissimilarity between a given object and its group center may be negative

It can be shown that this dissimilarity is equal to (see *Batagelj 1988*):

$$d_{x\bar{g}} = \frac{1}{n} \left( \sum_{i=1}^n d_{xi} - SS \right)$$

where  $SS$  is the sum of squares (see [dissvar](#)).

## Value

A vector with the dissimilarity to the group center for each object, or a list of medoid indexes.

## Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

## References

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2010) Discrepancy analysis of complex objects using dissimilarities. In F. Guillet, G. Ritschard, D. A. Zighed and H. Briand (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence, Volume 292, pp. 3-19. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.

Batagelj, V. (1988) Generalized ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67-74.

## See Also

[dissvar](#) to compute the pseudo variance from dissimilarities and for a basic introduction to concepts of pseudo variance analysis

[dissassoc](#) to test association between objects represented by their dissimilarities and a covariate.

[disstree](#) for an induction tree analyse of objects characterized by a dissimilarity matrix.

[dissmfac](#) to perform multi-factor analysis of variance from pairwise dissimilarities.

## Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities (any dissimilarity measure can be used)
mvad.ham <- seqdist(mvad.seq, method="HAM")

## Compute distance to center according to group gcse5eq
dc <- disscenter(mvad.ham, group=mvad$gcse5eq)

## Plotting distribution of dissimilarity to center
boxplot(dc~mvad$gcse5eq, col="cyan")

## Retrieving index of the first medoids, one per group
dc <- disscenter(mvad.ham, group=mvad$Grammar, medoids.index="first")
print(dc)

## Retrieving index of all medoids in each group
dc <- disscenter(mvad.ham, group=mvad$Grammar, medoids.index="all")
print(dc)
```

---

dissdomassoc

*Domain association measures*


---

## Description

Measures of association between domains are computed as the association between the pairwise dissimilarities in the domains. Measures are: Pearson correlation, Spearman correlation, global Cronbach alpha, and Cronbach alpha for each subset of the domains. The function can also return the share of variance (R-square) of the dissimilarities in one domain that can be reproduced from the dissimilarities in the other domains.

## Usage

```
dissdomassoc(domdiss, jointdiss = NULL, what = c("pearson", "R2"),
             dnames=names(domdiss), weights=NULL, w.rank=FALSE)
```

## Arguments

domdiss	List of symmetrical matrices or dist objects: the pairwise dissimilarities per domain.
jointdiss	NULL (default), matrix or dist object: pairwise dissimilarities for joint dimensions.
what	String or vector of strings: requested association measures among 'pearson', 'spearman', 'R2', 'cronbach', 'cron.subsets'. Can also be 'all' for all measures but 'spearman'.



dnames	String vector of length equal to number of domains. Names of domains. Default is names(domdiss).
weights	Vector of non-negative weights. If NULL, no weights are applied. (Currently Cronbach measures ignore weights!).
w.rank	Logical. If weights are provided and 'spearman' is selected, should we use weighted ranks. Caution: computation of weighted ranks considerably increases computation time!

## Details

The `disssdomassoc` function computes the domain association measures proposed by *Piccarreta (2017)*. These are for each pair of domains the correlation (Pearson or Spearman) between the domain specific pairwise dissimilarities. When 'R2' is requested, the function computes the share of variance of the dissimilarities in one domain that is reproduced by the dissimilarities in all other domains. The Cronbach alpha measures the coherence between the domains. With 'cronbach.subsets', Cronbach alpha is computed for the entire set of domains as well as for each possible subset of two or more domains.

When a `jointdiss` distance matrix or object is provided, correlations of each individual with this joint domain are also computed. The R2 of the joint domain takes account of all domains. However, R2's for the domains remain unchanged, i.e. they ignore the joint domain. Likewise, `jointdiss` does not affect the Cronbach alpha.

Spearman correlations are based on rank values of the distances. The computation of weighted ranks for weighted data is done with the `weighted.rank` function of the `cNORM` package and may take a while for large number of sequences (> 500, i.e. 124750 distances per domain). Therefore, weighted ranks are only used when explicitly requested by setting `w.rank = TRUE`. When `w.rank = FALSE` (default), Spearman is computed as the weighted Pearson correlation between non-weighted ranks.

## Value

An object of class `ddomassoc`, which is a list with the requested association measures. The `summary` method organizes the correlations and their p-values in table form.

## Author(s)

Gilbert Ritschard

## References

Piccarreta (2017). Joint Sequence Analysis: Association and Clustering. *Sociological Methods and Research*, 46(2), 252–287. doi: [10.1177/0049124115591013](https://doi.org/10.1177/0049124115591013).

## Examples

```
data(biofam)

## Building one channel per type of event left, children or married
## Using only first 200 sequences
bf <- as.matrix(biofam[1:200, 10:25])
```

```

children <- bf==4 | bf==5 | bf==6
married <- bf == 2 | bf== 3 | bf==6
left <- bf==1 | bf==3 | bf==5 | bf==6
## weights
weights <- biofam[1:200,"wp00tbgs"]

## Building sequence objects
child.seq <- seqdef(children)
marr.seq <- seqdef(married)
left.seq <- seqdef(left)
## distances by channel
dchild <- seqdist(child.seq, method="OM", sm="INDELSLOG")
dmarr <- seqdist(marr.seq, method="OM", sm="INDELSLOG")
dleft <- seqdist(left.seq, method="OM", sm="INDELSLOG")
dbiofam <- list(dchild,dmarr,dleft)
names(dbiofam) <- c("child","marr","left")

## Association between domains and R2 by domain
rass <- dissdomassoc(dbiofam, weights=weights)

## Joint distance matrix based on domain INDELSLOG costs.
mcdist <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
                    method="OM", sm =list("INDELSLOG", "INDELSLOG", "INDELSLOG"))

rassj <- dissdomassoc(dbiofam, jointdiss=mcdist, what=c("all"), weights=weights)
rassj[["Pearson.Rsquare"]]
rass[["Pearson.Rsquare"]]
summary(rassj)

```

---

dissmfacw

*Multi-factor ANOVA from a dissimilarity matrix*


---

## Description

Perform a multi-factor analysis of variance from a dissimilarity matrix.

## Usage

```
dissmfacw(formula, data, R = 1000, gower = FALSE, squared = FALSE,
           weights = NULL)
```

## Arguments

formula	A regression-like formula. The left hand side term should be a dissimilarity matrix or a dist object.
data	A data frame from which the variables in formula should be taken.
R	Number of permutations used to assess significance.
gower	Logical: Is the dissimilarity matrix already a Gower matrix?
squared	Logical: Should we square the provided dissimilarities?
weights	Optional numerical vector of case weights.

## Details

This method is, in some way, a generalization of `dissassoc` to account for several explanatory variables. The function computes the part of discrepancy explained by the list of covariates specified in the formula. It provides for each covariate the Type-II effect, i.e. the effect measured when removing the covariate from the full model with all variables included.

(The returned F values may slightly differ from those obtained with TraMineR versions older than 1.8-9. Since 1.8-9, the within sum of squares at the denominator is divided by  $n - m$  instead of  $n - m - 1$ , where  $n$  is the sample size and  $m$  the total number of predictors and/or contrasts used to represent categorical factors.)

For a single factor `dissmfacw` is slower than `dissassoc`. Moreover, the latter performs also tests for homogeneity in within-group discrepancies (equality of variances) with a generalization of Levene's and Bartlett's statistics.

Part of the function is based on the Multivariate Matrix Regression with qr decomposition algorithm written in SciPy-Python by Ondrej Libiger and Matt Zapala (See *Zapala and Schork, 2006*, for a full reference.) The algorithm has been adapted for Type-II effects and extended to account for case weights.

## Value

A `dissmultifactor` object with the following components:

<code>mfac</code>	The part of variance explained by each variable (comparing full model to model without the specified variable) and its significance using permutation test
<code>call</code>	Function call
<code>perms</code>	Permutation values as a boot object

## Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

## References

- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2010) Discrepancy analysis of complex objects using dissimilarities. In F. Guillet, G. Ritschard, D. A. Zighed and H. Briand (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence, Volume 292, pp. 3-19. Berlin: Springer.
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009). Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.
- Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology* 26, 32-46.
- McArdle, B. H. and M. J. Anderson (2001). Fitting multivariate models to community data: A comment on distance-based redundancy analysis. *Ecology* 82(1), 290-297.

Zapala, M. A. and N. J. Schork (2006). Multivariate regression analysis of distance matrices for testing associations between gene expression patterns and related variables. *Proceedings of the National Academy of Sciences of the United States of America* 103(51), 19430-19435.

### See Also

[dissvar](#) to compute a pseudo variance from dissimilarities and for a basic introduction to concepts of discrepancy analysis.

[dissassoc](#) to test association between objects represented by their dissimilarities and a covariate.

[disstree](#) for an induction tree analysis of objects characterized by a dissimilarity matrix.

[disscenter](#) to compute the distance of each object to its group center from pairwise dissimilarities.

### Examples

```
## Define the state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])
## Here, we use only first 100 sequences
mvad.seq <- mvad.seq[1:100,]

## Compute dissimilarities (any dissimilarity measure can be used)
mvad.ham <- seqdist(mvad.seq, method="HAM")

## And now the multi-factor analysis
print(dissmfacw(mvad.ham ~ male + Grammar + funemp +
gcse5eq + fmpr + livboth, data=mvad[1:100,], R=10))
```

---

dissrep

*Extracting sets of representative objects using a dissimilarity matrix*

---

### Description

The function extracts a set of representative objects that exhibits the key features of the whole data set, the goal being to get easy sounded interpretation of the latter. The user can set either the desired coverage level (the proportion of objects having a representative in their neighborhood) or the desired number of representatives.

### Usage

```
dissrep(diss, criterion = "density", score = NULL, decreasing = TRUE,
coverage = 0.25, nrep = NULL, pradius = 0.10, dmax = NULL,
weights = NULL, trep, tsim)
```

### Arguments

`diss` A dissimilarity matrix or a `dist` object (see [dist](#))

criterion	the representativeness criterion for sorting the candidate list. One of "freq" (frequency), "density" (neighborhood density) or "dist" (centrality). An optional vector containing the scores for sorting the candidate objects may also be provided. See below and details.
score	an optional vector containing the representativeness scores used for sorting the objects in the candidate list. The length of the vector must be equal to the number of rows/columns in the distance matrix, i.e the number of objects.
decreasing	if a score vector is provided, indicates whether the objects in the candidate list must be sorted in ascending or decreasing order of this score. The first object in the candidate list is supposed to be the most representative.
coverage	controls the size of the representative set by setting the desired coverage level, i.e the proportion of objects having a representative in their neighborhood. Neighborhood radius is defined by pradius.
nrep	number of representatives. If NULL (default), coverage argument is used to control the size of the representative set.
pradius	neighborhood radius as a percentage of the maximum (theoretical) distance dmax. Defaults to 0.1 (10%). Object $y$ is redundant to object $x$ when it is in the neighborhood of $x$ , i.e., within a distance $\text{pradius} \cdot \text{dmax}$ from $x$ .
dmax	maximum theoretical distance. Used to derive the neighborhood radius as $\text{pradius} \cdot \text{dmax}$ . If NULL, the value of dmax is derived from the dissimilarity matrix.
weights	vector of weights of length equal to the number of rows of the dissimilarity matrix. If NULL, equal weights are assigned.
trep	Deprecated. Use coverage instead.
tsim	Deprecated. Use pradius instead.

## Details

The representative set is obtained by an heuristic. Representatives are selected by successively extracting from the sequences sorted by their representativeness score those which are not redundant with already retained representatives. The selection stops when either the desired coverage or the wanted number of representatives is reached. Objects are sorted either by the values provided as score argument, or by specifying one of the following as criterion argument: "freq" (*sequence frequency*), "density" (*neighborhood density*), "dist" (*centrality*).

The *frequency* criterion uses the frequencies as representativeness score. The frequency of an object in the data is computed as the number of other objects with whom the dissimilarity is equal to 0. The more frequent an object the more representative it is supposed to be. Hence, objects are sorted in decreasing frequency order. Indeed, this criterion is the neighborhood (see below) criterion with the neighborhood diameter set to 0.

The *neighborhood density* is the number—density—of sequences in the neighborhood of the object. This requires to set the neighborhood radius pradius. Objects are sorted in decreasing density order.

The *centrality* criterion is the sum of distances to all other objects. The smallest the sum, the most representative the sequence.

Use `criterion="dist"` and `nrep=1` to get the medoid and `criterion="density"` and `nrep=1` to get the densest object pattern.

For more details, see *Gabadinho et al., 2011*.

**Value**

An object of class `diss.rep`. This is a vector containing the indexes of the representative objects with the following additional attributes:

Scores	a vector with the representative score of each object given the chosen criterion.
Distances	a matrix with the distance of each object to its nearest representative.
Statistics	a data frame with quality measures for each representative: number of objects attributed to the representative, number of object in the representative's neighborhood, mean distance to the representative.
Quality	overall quality measure.

Print and summary methods are available.

**Author(s)**

Alexis Gabadinho (with Gilbert Ritschard for the help page)

**References**

Gabadinho A, Ritschard G (2013). "Searching for typical life trajectories applied to child birth histories", In R Lévy, E. Widmer (eds.), *Gendered Life Courses*, pp. 287-312. Vienna: LIT.

Gabadinho A, Ritschard G, Studer M, Müller NS (2011). "Extracting and Rendering Representative Sequences", In A Fred, JLG Dietz, K Liu, J Filipe (eds.), *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 128 of *Communications in Computer and Information Science (CCIS)*, pp. 94-106. Springer-Verlag.

**See Also**

[seqrep](#), [disscenter](#)

**Examples**

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
"Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## Computing the distance matrix
costs <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", sm=costs)

## Representative set using the neighborhood density criterion
biofam.rep <- dissrep(biofam.om)
biofam.rep
summary(biofam.rep)
```

---

disstree                      *Dissimilarity Tree*

---

### Description

Tree structured discrepancy analysis of objects described by their pairwise dissimilarities.

### Usage

```
disstree(formula, data = NULL, weights = NULL, min.size = 0.05,
         max.depth = 5, R = 1000, pval = 0.01, object = NULL,
         weight.permutation = "replicate", squared = FALSE, first = NULL,
         minSize, maxdepth)
```

### Arguments

formula	Formula with a dissimilarity matrix as left hand side and the candidate partitioning variables on the right side.
data	Data frame where variables in formula will be searched for.
weights	Optional numerical vector of weights.
min.size	Minimum number of cases in a node, will be treated as a proportion if less than 1.
max.depth	Maximum depth of the tree
R	Number of permutations used to assess the significance of the split.
pval	Maximum allowed p-value for a split
object	An optional R object represented by the dissimilarity matrix. This object may be used by the print method or <a href="#">disstree2dot</a> to render specific object type.
weight.permutation	Weight permutation method: "diss" (attach weights to the dissimilarity matrix), "replicate" (replicate cases using weights), "rounded-replicate" (replicate case using rounded weights), "random-sampling" (random assignment of covariate profiles to the objects using distributions defined by the weights.)
squared	Logical: Should the diss dissimilarities be squared?
first	One of the variable in the right-hand side of the formula. This forces the first node of the tree to be split by this variable.
minSize	Deprecated. Use min.size instead.
maxdepth	Deprecated. Use max.depth instead.

### Details

The procedure iteratively splits the data. At each step, the procedure selects the variable and split that explain the greatest part of the discrepancy, i.e., the split for which we get the highest pseudo R<sup>2</sup>. The significance of the retained split is assessed through a permutation test.

[seqtree](#) provides a simpler interface if you plan to use `disstree` for state sequence objects.

**Value**

An object of class `disstree` that contains the following components:

<code>root</code>	A node object, root of the tree
<code>info</code>	General information such as parameters used to build the tree
<code>info\$adjustment</code>	A <a href="#">dissassoc</a> object providing global statistics for tree.
<code>formula</code>	The formula used to generate the tree
<code>data</code>	data used to build the tree
<code>weights</code>	weights

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**References**

- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2010) Discrepancy analysis of complex objects using dissimilarities. In F. Guillet, G. Ritschard, D. A. Zighed and H. Briand (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence, Volume 292, pp. 3-19. Berlin: Springer.
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.
- Anderson, M. J. (2001) A new method for non-parametric multivariate analysis of variance. *Austral Ecology* **26**, 32-46.
- Batagelj, V. (1988) Generalized ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67-74.
- Piccarreta, R. et F. C. Billari (2007) Clustering work and family trajectories by using a divisive algorithm. *Journal of the Royal Statistical Society A* **170**(4), 1061–1078.

**See Also**

- [seqtree](#) to generate a specific `disstree` objects for analyzing state sequences.
- [seqtreedisplay](#) to generate graphic representation of `seqtree` objects when analyzing state sequences.
- [disstreedisplay](#) is a more general interface to generate such representation for other type of objects.
- [dissvar](#) to compute discrepancy using dissimilarities and for a basic introduction to discrepancy analysis.
- [dissassoc](#) to test association between objects represented by their dissimilarities and a covariate.
- [dissmfac](#) to perform multi-factor analysis of variance from pairwise dissimilarities.
- [disscenter](#) to compute the distance of each object to its group center from pairwise dissimilarities.



## Examples

```

data(mvad)

## Defining a state sequence object
mvad.seq <- seqdef(mvad[, 17:86])

## Computing dissimilarities (any dissimilarity measure can be used)
mvad.ham <- seqdist(mvad.seq, method="HAM")
## Grow the tree using using a low R value for illustration.
## For R=10, pval cannot be lower than 0.1
dt <- disstree(mvad.ham~ male + Grammar + funemp + gcse5eq + fmpr + livboth,
  data=mvad, R = 10, pval = 0.1)
print(dt)

## Will only work if GraphViz is properly installed
## See seqtree for simpler way to plot a sequence tree.
## Not run:
disstreedisplay(dt, image.fun = seqdplot, image.data = mvad.seq,
  ## Additional parameters passed to seqdplot
  with.legend = FALSE, axes = FALSE, ylab = "")

## End(Not run)
## Second method, using a specific function
myplotfunction <- function(individuals, seqs, ...) {
  par(font.sub=2, mar=c(3,0,6,0), mgp=c(0,0,0))
  ## using mds to order sequence in seqiplot
  mds <- cmdscale(seqdist(seqs[individuals,], method="HAM"),k=1)
  seqiplot(seqs[individuals,], sortv=mds,...)
}

## If image.data is not set, index of individuals are sent to image.fun
## Not run:
disstreedisplay(dt, image.fun = myplotfunction, cex.main = 3,
  ## additional parameters passed to myplotfunction
  seqs = mvad.seq,
  ## additional parameters passed to seqiplot (through myplotfunction)
  with.legend = FALSE, axes = FALSE, idxs = 0, space = 0, ylab = "", border = NA)

## End(Not run)

```

---

disstree2dot

*Graphical representation of a dissimilarity tree*


---

## Description

Functions to generate a ".dot" file and associated images files that can be used in GraphViz to get a graphical representation of the tree.

**Usage**

```
disstree2dot(tree, filename, digits = 3, image.fun = NULL, image.data = NULL,
  only.leaf = FALSE, device = "jpeg", image.format = "jpg",
  device.args = list(), use.title = TRUE, label.pos = "main",
  node.pos = "main", split.pos = "sub", cex.main = 1,
  legend.text = NULL, image.legend = NULL, image.quality = NULL,
  show.depth = FALSE, title.outer = FALSE,
  imagefunc, imagedata, imgLeafOnly, devicefunc, imageext,
  device.arg, label.loc, node.loc, split.loc, title.cex, legendtext,
  legendimage, qualityimage, showdepth, ...)
```

```
disstree2dotp(tree, filename, image.data = NULL, only.leaf = FALSE,
  image.fun = plot, cex.main = 3, with.quality = TRUE,
  cex.quality = cex.main, title.outer = FALSE,
  imagedata, imgLeafOnly, imagefunc, title.cex, withquality,
  quality.fontsize, ...)
```

```
seqtree2dot(tree, filename, seqdata = tree$info$object, only.leaf = FALSE,
  sortv = NULL, diss = NULL, cex.main = 3, with.legend = "auto",
  cex.legend = cex.main, with.quality = FALSE,
  cex.quality = cex.main, axes = FALSE,
  imgLeafOnly, dist.matrix, title.cex,
  withlegend, withquality, ...)
```

**Arguments**

<code>tree</code>	The tree to be plotted.
<code>filename</code>	A filename, without extension, that will be used to generate image and dot files.
<code>digits</code>	Number of significant digits to plot.
<code>image.fun</code>	A function to plot the individuals in a node, see details.
<code>image.data</code>	a data.frame that will be passed to <code>image.fun</code> , see details.
<code>only.leaf</code>	Logical: If TRUE, only terminal node will be plotted.
<code>device</code>	A device function, "jpeg" by default.
<code>image.format</code>	extension for image files.
<code>device.args</code>	Argument passed to device.
<code>use.title</code>	Logical: If TRUE, node information will be printed using <code>title</code> command, see details.
<code>label.pos</code>	Location of the node label, see <code>title</code> for possible values.
<code>node.pos</code>	Node content location, see <code>title</code> for possible values.
<code>split.pos</code>	Split information location, see <code>title</code> for possible values.
<code>cex.main</code>	cex applied to all calls to <code>title</code> (see <code>use.title</code> ).
<code>title.outer</code>	Logical: If TRUE, the title (see <code>use.title</code> ) is printed in the outer margins.
<code>legend.text</code>	An optional text appearing in a distinct node.

<code>image.legend</code>	An optional image file appearing in a distinct node.
<code>image.quality</code>	An optional image file appearing in a distinct node.
<code>show.depth</code>	Logical. If TRUE, information about depth of the tree is added to the plot.
<code>with.quality</code>	If TRUE, a node displaying fitting measures of the tree is added to the plot.
<code>cex.quality</code>	Numeric. Size of the font of the fitting measures node.
<code>seqdata</code>	a sequence object as defined by the the <a href="#">seqdef</a> function.
<code>sortv</code>	The name of an optional variable used to sort the data before plotting, see <a href="#">seqplot</a> .
<code>diss</code>	The name of an optional dissimilarity matrix used to find representative sequences, <a href="#">seqrplot</a> .
<code>with.legend</code>	defines if and where the legend of the state colors is plotted. The default value "auto" sets the position of the legend automatically. Other possible value is "right". Obsolete value TRUE is equivalent to "auto".
<code>cex.legend</code>	Size of the font of the legend.
<code>axes</code>	if set to "all" (default value) x axes are drawn for each plot in the graphic. If set to "bottom" and <code>group</code> is used, axes are drawn only under the plots located at the bottom of the graphic area. If FALSE, no x axis is drawn.
<code>imagefunc</code>	Deprecated. Use <code>image.fun</code> instead.
<code>imagedata</code>	Deprecated. Use <code>image.data</code> instead.
<code>imgLeafOnly</code>	Deprecated. Use <code>only.leaf</code> instead.
<code>devicefunc</code>	Deprecated. Use <code>device</code> instead.
<code>imageext</code>	Deprecated. Use <code>image.format</code> instead.
<code>device.arg</code>	Deprecated. Use <code>device.args</code> instead.
<code>label.loc</code>	Deprecated. Use <code>label.pos</code> instead.
<code>node.loc</code>	Deprecated. Use <code>node.pos</code> instead.
<code>split.loc</code>	Deprecated. Use <code>split.pos</code> instead.
<code>title.cex</code>	Deprecated. Use <code>cex.main</code> instead.
<code>legendtext</code>	Deprecated. Use <code>legend.text</code> instead.
<code>legendimage</code>	Deprecated. Use <code>image.legend</code> instead.
<code>qualityimage</code>	Deprecated. Use <code>image.quality</code> instead.
<code>showdepth</code>	Deprecated. Use <code>show.depth</code> instead.
<code>withquality</code>	Deprecated. Use <code>with.quality</code> instead.
<code>quality.fontsize</code>	Deprecated. Use <code>cex.quality</code> instead.
<code>dist.matrix</code>	Deprecated. Use <code>diss</code> instead.
<code>withlegend</code>	Deprecated. Use <code>with.legend</code> instead.
<code>...</code>	other parameters that will be passed to <code>image.fun</code> or <a href="#">seqplot</a> (for <code>seqtree2dot</code> ).

**Details**

These functions generate a "dot" file that can be used in GraphViz (<http://www.graphviz.org>). It also generates one image per node through a call to `image.fun` passing the selected lines of `image.data` if present or otherwise a list of indexes (of individuals belonging to a node). These functions are not intended to be used by end-user. See [seqtreedisplay](#) and [disstreedisplay](#) for a much simpler way to generate a graphical representation of a tree ([seqtree](#) or [disstree](#)).

`seqtree2dot` is a shortcut for sequences objects using the plot function [seqplot](#). For each node, it calls [seqplot](#) with the corresponding subset of rows of `seqdata` and the provided [seqplot](#)'s arguments. You should at least specify the type of the plot (e.g. `type="d"`, see [seqplot](#) for more details).

If `use.title` is TRUE, `image.fun` should take care to leave enough space for the title.

`disstree2dotp` is a simplified interface of `disstree2dot` which automatically leaves enough space for the title and subtitles. These functions are intended to be generic.

**Value**

Nothing but generates a "dot" and several image files (one per node) in the current working directory (see [getwd](#) and [setwd](#)).

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**See Also**

[seqtree](#) and [seqtreedisplay](#), [disstree](#) and [disstreedisplay](#).

---

disstreeleaf

*Terminal node membership*

---

**Description**

Return a factor with the terminal node membership of each case.

**Usage**

```
disstreeleaf(tree, label=FALSE)
```

**Arguments**

<code>tree</code>	The tree, a <code>disstree</code> or <code>DisstTreeNode</code> object.
<code>label</code>	Logical. If TRUE, the returned leaf memberships are labelled with the corresponding classifications rules.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**See Also**

[disstree](#) for examples

---

dissvar	<i>Dissimilarity based discrepancy</i>
---------	--

---

**Description**

Compute the discrepancy from the pairwise dissimilarities between objects. The discrepancy is a measure of dispersion of the set of objects.

**Usage**

```
dissvar(diss, weights=NULL, squared = FALSE)
```

**Arguments**

diss	A dissimilarity matrix or a dist object (see <a href="#">dist</a> )
weights	optional numerical vector containing weights.
squared	Logical. If TRUE diss is squared.

**Details**

The discrepancy is an extension of the concept of variance to any kind of objects for which we can compute pairwise dissimilarities. The discrepancy  $s^2$  is defined as:

$$s^2 = \frac{1}{2n^2} \sum_{i=1}^n \sum_{j=1}^n d_{ij}$$

*Mathematical ground:* In the Euclidean case, the sum of squares can be expressed as:

$$SS = \sum_{i=1}^n (y_i - \bar{y})^2 = \frac{1}{2n} \sum_{i=1}^n \sum_{j=1}^n (y_i - y_j)^2$$

The concept of discrepancy generalizes the equation by allowing to replace the  $(y_i - y_j)^2$  term with any measure of dissimilarity  $d_{ij}$ .

**Value**

The discrepancy.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

## References

- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2010) Discrepancy analysis of complex objects using dissimilarities. In F. Guillet, G. Ritschard, D. A. Zighed and H. Briand (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence, Volume 292, pp. 3-19. Berlin: Springer.
- Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.
- Anderson, M. J. (2001) A new method for non-parametric multivariate analysis of variance. *Austral Ecology* **26**, 32-46.
- Batagelj, V. (1988) Generalized ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67-74.

## See Also

- [dissassoc](#) to test association between objects represented by their dissimilarities and a covariate.
- [disstree](#) for an induction tree analyse of objects characterized by a dissimilarity matrix.
- [disscenter](#) to compute the distance of each object to its group center from pairwise dissimilarities.
- [dissmfacw](#) to perform multi-factor analysis of variance from pairwise dissimilarities.

## Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities (any dissimilarity measure can be used)
mvad.ham <- seqdist(mvad.seq, method="HAM")

## Pseudo variance of the sequences
print(dissvar(mvad.ham))
```

---

ex1

*Example data set with missing values and weights*

---

## Description

Example data set used to demonstrate the handling of missing values and weights.

The state columns (variable) are named 'P1' to 'P13'.

The alphabet is made of four possible states: A, B, C and D.

The data set contains also case weights (variable weights). The sum of the weights is 60.

**Usage**

```
data(ex1)
```

**Format**

A data frame with 7 rows, 13 state variables, 1 weight variable.

**Source**

The brain of the TraMineR package team.

---

ex2

*Example data sets with weighted and unweighted sequence data*

---

**Description**

Example data sets used to demonstrate the handling of weights. The 'ex2.weighted' data set contains 6 sequences with weights inflating to 100 sequences (sum of weights is 100). The second data frame 'ex2.unweighted' contains the corresponding 100 sequences.

The sequences are, in both data frames, in the 'seq' column, and weights in the 'weight' column of 'ex2.weighted'.

The alphabet is made of four possible states: A, B, C and D.

These data sets are mainly intended to test and illustrate the handling of weights in TraMineR's functions. Weighted results obtained with 'ex2.weighted' data set should be exactly the same as unweighted results obtained with the 'ex2.unweighted' data set.

**Usage**

```
data(ex2)
```

**Format**

The command `data(ex2)` generates two data frames:

ex2.weighted: a data frame with 6 rows, 1 variable containing sequences as character strings, 1 weight variable.

ex2.unweighted: a data frame with 100 rows, 1 variable containing sequences as character strings.

**Source**

The brain of the TraMineR package team.

**Examples**

```
data(ex2)
```

```
ex2w.seq <- seqdef(ex2.weighted, 1, weights=ex2.weighted$weight)
```

```
ex2u.seq <- seqdef(ex2.unweighted)
```

---

`famform`*Example data set: sequences of family formation*

---

**Description**

This data set contains 5 sequences of family formation histories, used by Elzinga (2008) to introduce several metrics for computing distances between sequences. These sequences don't contain information about the duration spent in each state, they contain only distinct successive states.

**Usage**

```
data(famform)
```

**Format**

A data frame with 5 rows and 1 variable.

**Details**

The sequences are in 'STS' format and stored in character strings with states separated with '-'.  
This data set is used in TraMineR's manual to crosscheck some results with those presented by Elzinga.

**Source**

Elzinga (2008)

**References**

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. Non published manuscript. VU University, Amsterdam.

---

`is.stslist`*Test if is a proper state sequence (stslist) object*

---

**Description**

The function tests whether `x` is of class `stslist` and if its `weights` attribute has the expected length and names.

**Usage**

```
is.stslist(x)
```



**Arguments**

x                    object to be tested.

**Value**

Logical: result of the test.

**Author(s)**

Gilbert Ritschard

**See Also**

[seqdef](#)

**Examples**

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(biofam)
biofam <- biofam[sample(nrow(biofam),300),]
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
               "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam[,10:25], weights=biofam$wp00tbgs)

is.stslist(biofam.seq) #TRUE

attr(biofam.seq,"weights") <- NULL
is.stslist(biofam.seq) #TRUE

attr(biofam.seq,"weights") <- rep(1, nrow(biofam.seq))
is.stslist(biofam.seq) #FALSE

w <- rep(1, nrow(biofam.seq))
names(w) <- rownames(biofam.seq)
attr(biofam.seq,"weights") <- w
is.stslist(biofam.seq) #TRUE
```

---

mvad

*Example data set: Transition from school to work*


---

**Description**

The data comes from a study by McVicar and Anyadike-Danes on transition from school to work. The data consist of static background characteristics and a time series sequence of 72 monthly labour market activities for each of 712 individuals in a cohort survey. The individuals were followed up from July 1993 to June 1999. The monthly states are recorded in columns 15 (Jul . 93) to 86 (Jun . 99).

States are:

employment	(EM)
FE	further education (FE)
HE	higher education (HE)
joblessness	(JL)
school	(SC)
training	(TR)

The data set contains also ids (`id`) and sample weights (`weight`) as well as the following binary covariates:

`male`  
`catholic`  
Belfast, N.Eastern, Southern, S.Eastern, Western (location of school, one of five Education and Library Board areas in Northern Ireland)  
Grammar (type of secondary education, 1=grammar school)  
`funemp` (father's employment status at time of survey, 1=father unemployed)  
`gcse5eq` (qualifications gained by the end of compulsory education, 1=5+ GCSEs at grades A-C, or equivalent)  
`fmpr` (SOC code of father's current or most recent job, 1=SOC1 (professional, managerial or related))  
`livboth` (living arrangements at time of first sweep of survey (June 1995), 1=living with both parents)

### Usage

```
data(mvad)
```

### Format

A data frame containing 712 rows, 72 state variables, 1 id variable and 13 covariates.

### Source

McVicar and Anyadike-Danes (2002)

### References

McVicar, Duncan and Anyadike-Danes, Michael (2002). Predicting Successful and Unsuccessful Transitions from School to Work by Using Sequence Methods, *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 165, 2, pp. 317–334.

---

plot.seqdiff	<i>Plotting a seqdiff object.</i>
--------------	-----------------------------------

---

### Description

Plot method for the sliding values returned by [seqdiff](#). Plots a statistic (the Pseudo R2 by default) along the position axis.

### Usage

```
## S3 method for class 'seqdiff'
plot(x, stat = "Pseudo R2", type = "l", ylab = stat,
     xlab = "", legend.pos = "top", ylim = NULL, xaxis = TRUE, col = NULL,
     xtstep = NULL, tick.last = NULL, legendposition, xaxt, ...)
```

### Arguments

x	an object produced by <a href="#">seqdiff</a>
stat	character. Name of the statistic to be plotted. Can be any of the statistics returned by <a href="#">seqdiff</a> or "discrepancy". See details.
type	the line type, see <a href="#">lines</a>
ylab	character: y-axis label.
xlab	character: x-axis label.
legend.pos	character: position of the line legend, see <a href="#">legend</a>
ylim	numeric: if not NULL, range of the y-axis.
xaxis	logical: if TRUE an x-axis is plotted.
col	list of colors to use for each line.
xtstep	integer: optional step between tick-marks and labels on the x-axis. If unspecified, the xtstep attribute of the sequence object x is used. (see <a href="#">seqdef</a> )
tick.last	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the tick.last attribute of the x object is used.
legendposition	Deprecated. Use legend.pos instead.
xaxt	Deprecated. Use xaxis instead.
...	Additional parameters passed to <a href="#">lines</a>

### Details

The function plots the sliding values of the requested statistic.

You can plot the evolution of two statistics by providing for instance `stat=c("Pseudo R2", "Levene")`.

Use `stat="discrepancy"` to plot the within-discrepancies.

For "discrepancy", a separate line is drawn for the whole set of sequences and for each group. Those two values cannot be paired with another statistic.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**See Also**

[seqdiff](#)

---

plot.stslist

*Plot method for state sequence objects*

---

**Description**

This is the plot method for state sequence objects of class `stslist` created by the `seqdef` function. It produces a sequence index plot.

**Usage**

```
## S3 method for class 'stslist'
plot(x, idxs = NULL, weighted = TRUE, sortv = NULL,
     cpal = NULL, missing.color = NULL, ylab = NULL,
     yaxis = TRUE, xaxis = TRUE, ytlab = NULL, ylas = 0,
     xtlab = NULL, xtstep = NULL, tick.last = NULL, cex.axis = 1,
     tlim, cex.plot, ...)
```

**Arguments**

<code>x</code>	A state sequence object created with the <a href="#">seqdef</a> function.
<code>idxs</code>	Indexes of the sequences to be plotted (default value is <code>1:10</code> ), for instance <code>20:50</code> to plot sequences 20 to 50, <code>c(2, 8, 12, 25)</code> to plot sequences 2,8,12 and 25 in <code>seqdata</code> . If set to <code>0</code> , all sequences in <code>seqdata</code> are plotted.
<code>weighted</code>	Logical: Should the bar representing each sequence be proportional to its weight? Ignored when no weights are assigned to sequences (see <a href="#">seqdef</a> .)
<code>sortv</code>	A sorting variable or a sort method (one of <code>"from.start"</code> or <code>"from.end"</code> ). See details.
<code>cpal</code>	Color palette for the states. A vector of colors of length equal to the number of states in the alphabet. If <code>NULL</code> (default), the <code>cpal</code> attribute of the <code>seqdata</code> sequence object is used (see <a href="#">seqdef</a> ).
<code>missing.color</code>	Color for representing missing values inside the sequences. If <code>NULL</code> (default) the color is taken from the <code>"missing.color"</code> attribute of the <code>x</code> sequence object.
<code>ylab</code>	String. An optional label for the y axis. If set to <code>NA</code> , no label is drawn.
<code>yaxis</code>	Logical. Should the y axis be plotted. When set as <code>TRUE</code> , sequence indexes are displayed.
<code>xaxis</code>	Logical. Should the x (time) axis be plotted? Default is <code>TRUE</code> .

ytlab	the labels of the plotted sequences to display on the y axis. Default is the indexes of the sequences as defined by the <code>idxs</code> argument. Can be set to "id" for displaying the row names (id) of the sequences instead of their indexes; row names can be assigned to the sequence object with the <code>id</code> argument of the <code>seqdef</code> function or afterwards with <code>rownames</code> . Otherwise <code>ytlab</code> can be set to a vector of length equal to the number of sequences to be plotted.
ylas	sets the orientation of the sequence labels appearing on the y axis. Accepted values are the same as for the <code>las</code> standard option 0: always parallel to the axis (default), 1: always horizontal, 2: always perpendicular to the axis, 3: always vertical.
xtlab	optional labels for the x axis ticks labels. If unspecified, the column names of the <code>seqdata</code> sequence object are used (see <code>seqdef</code> ).
xtstep	optional interval at which the tick-marks and labels of the x-axis are displayed. For example, with <code>xtstep=3</code> a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the <code>xtstep</code> attribute of the x object is used.
tick.last	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the <code>tick.last</code> attribute of the x object is used.
cex.axis	Axis annotation magnification. See <code>par</code> .
tlim	Deprecated. Use <code>idxs</code> instead.
cex.plot	Deprecated. Use <code>cex.axis</code> instead.
...	arguments to be passed to the plot function or other graphical parameters.

### Details

This is the default plot method for state sequence objects (produced by the `seqdef` function), i.e., for objects of class `stslist`. It produces a sequence index plot, where individual sequences are rendered with stacked bars depicting the states over time.

This method is called by the generic `seqplot` function (if `type="i"`). The latter produces more sophisticated plots, allowing grouping and automatic display of the state color legend. The `seqiplot` function is a shortcut for calling `seqplot` with `type="i"`.

When a `sortv` variable is provided to `seqiplot` or `seqIplot`, its values define the order in which the sequences are plotted. With `sortv = "from.start"`, sequence are sorted by the elements of the alphabet at the successive positions starting from the beginning of the sequences. The "from.end" method proceeds similarly, but backward from the last position.

The interest of sequence index plots has for instance been stressed by *Scherer (2001)* and *Brzinsky-Fay et al. (2006)*. Notice that such index plots for thousands of sequences result in very heavy graphic files if they are stored in PDF or POSTSCRIPT format. To reduce the size, we suggest saving the figures in bitmap format by using for instance `png` instead of `postscript` or `pdf`.

### Author(s)

Gilbert Ritschard

**See Also**[seqplot](#)**Examples**

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam <- biofam[500:600,] ## using a subsample only
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
  "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## Plot of the 10 most frequent sequences
## with bar width proportional to the frequency
plot(biofam.seq)

## Plotting the whole data set
## with no borders
plot(biofam.seq, idxs=0, space=0, border=NA)

## =====
## Weights
## =====
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)
plot(ex1.seq)
plot(ex1.seq, weighted=FALSE)
```

---

plot.stslist.freq      *Plot method for sequence frequency tables*

---

**Description**

Plot method for output produced by the seqtab function, i.e objects of class stslist.freq.

**Usage**

```
## S3 method for class 'stslist.freq'
plot(x, cpal = NULL, missing.color = NULL, pbarw = TRUE,
  ylab = NULL, yaxis = TRUE, xaxis = TRUE,
  xtlab = NULL, xtstep = NULL, tick.last = NULL, cex.axis = 1,
  cex.plot, ...)
```

**Arguments**

x                      an object of class stslist.freq as produced by the seqtab function.

cpal	alternative color palette to be used for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the x object is used.
missing.color	alternative color for representing missing values inside the sequences. By default, this color is taken from the missing.color attribute of the x object.
pbarw	if pbarw=TRUE (default), the width of the bars are proportional to the sequence frequency in the dataset.
ylab	an optional label for the y axis. If set to NA, no label is drawn.
yaxis	if TRUE or "cum", the y axis is plotted with a label showing the cumulated percentage frequency of the displayed sequences. If "pct", the percentage value for each sequence is displayed.
xaxis	if TRUE (default) the x-axis is plotted.
xtlab	optional labels for the ticks of the x-axis. If unspecified, the names attribute of the x object is used.
xtstep	optional interval at which the tick-marks and labels of the x-axis are displayed. For example, with xtstep=3 a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the xtstep attribute of the x object is used.
tick.last	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the tick.last attribute of the x object is used.
cex.axis	Axis annotation magnification. See <a href="#">par</a> .
...	further graphical parameters. For example border=NA to remove the bars borders, space=0 to remove space between sequences. For more details about the graphical parameter arguments, see <a href="#">barplot</a> and <a href="#">par</a> .
cex.plot	Deprecated. Use cex.axis instead.

## Details

This is the plot method for the output produced by the [seqtab](#) function, i.e. objects of class *stslist.freq*. It produces a plot showing the sequences sorted bottom up according to their frequency in the data set.

This method is called by the generic [seqplot](#) function (if type="f") that produces more sophisticated plots, allowing grouping and automatic display of the state color legend. The [seqfplot](#) function is a shortcut for calling [seqplot](#) with type="f".

## Author(s)

Alexis Gabadinho

## Examples

```
## Loading the 'actcal' example data set
data(actcal)

## Defining a sequence object with data in columns 13 to 24
## (activity status from january to december 2000)
```

```

actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)

## 10 most frequent sequences in the data
actcal.freq <- seqtab(actcal.seq)

## Plotting the object
plot(actcal.freq, main="Sequence frequencies - actcal data set")

## Plotting all the distinct sequences without borders
## and space between sequences
actcal.freq2 <- seqtab(actcal.seq, idxs=0)
plot(actcal.freq2, main="Sequence frequencies - actcal data set",
     border=NA, space=0)

```

---

plot.stslist.meant      *Plot method for objects produced by the seqmeant function*

---

## Description

This is the plot method for objects of class *stslist.meant* produced by the [seqmeant](#) function.

## Usage

```

## S3 method for class 'stslist.meant'
plot(x, cpal = NULL, ylab = NULL, yaxis = TRUE,
     xaxis = TRUE, cex.axis = 1, ylim = NULL, bar.labels = NULL,
     cex.barlab = cex.axis, offset.barlab = .1, cex.plot, ...)

```

## Arguments

x	an object of class <i>stslist.meant</i> as produced by the <i>seqmeant</i> function.
cpal	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the 'seqdata' sequence object is used (see <a href="#">seqdef</a> ).
ylab	an optional label for the y axis. If set to NA, no label is drawn.
yaxis	controls whether the y axis is plotted. Default is TRUE.
xaxis	if TRUE (default) the xaxis is plotted.
cex.axis	Axis annotation magnification. See <a href="#">par</a> .
ylim	an optional vector setting the limits for the y axis. If NULL (default), limits are set to (0, max. sequence length).
bar.labels	Vector of bar labels of length equal to size of alphabet.
cex.barlab	Real. Bar labels magnification. Defaults to <i>cex.axis</i>
offset.barlab	Real. Vertical offset of bar labels as a proportion of $\max(\text{ylim})$ . Default is 0.1.
cex.plot	Deprecated. Use <i>cex.axis</i> instead.
...	further graphical parameters. For more details about the graphical parameter arguments, see <i>barplot</i> and <i>par</i> .



## Details

This is the plot method for the output produced by the `seqmeant` function, i.e., objects of class `stslist.meant`. It produces a plot showing the mean times spent in each state of the alphabet.

When the `"se"` attribute of `x` is `TRUE`, i.e., when `x` contains also the standard errors of the mean times, error bars are automatically displayed on the plot. See the `serr` argument of `seqmeant`.

This method is called by the generic `seqplot` function (if `type="mt"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqmplot` function is a shortcut for calling `seqplot` with `type="mt"`.

## Examples

```
## Loading the mvad data set and creating a sequence object
data(mvad)
mvad.labels <- c("employment", "further education", "higher education",
               "joblessness", "school", "training")
mvad.scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, 15:86, states=mvad.scodes, labels=mvad.labels)

## Computing the mean times
mvad.meant <- seqmeant(mvad.seq)

## Plotting
plot(mvad.meant, main="Mean durations in each state of the alphabet")

## Changing the y axis limits
plot(mvad.meant, main="Mean durations in each state of the alphabet",
     ylim=c(0,40))

## Displaying error bars
mvad.meant.e <- seqmeant(mvad.seq, serr=TRUE)
plot(mvad.meant.e, main="Mean durations in each state of the alphabet",
     ylim=c(0,40))
```

---

plot.stslist.modst      *Plot method for modal state sequences*

---

## Description

Plot method for output produced by the `seqmodst` function, i.e objects of class `stslist.modst`.

## Usage

```
## S3 method for class 'stslist.modst'
plot(x, cpal = NULL, ylab = NULL, yaxis = TRUE,
     xaxis = TRUE, xtlab = NULL, xtstep = NULL, tick.last = NULL,
     info = TRUE, cex.axis = 1, cex.plot, ...)
```

## Arguments

<code>x</code>	an object of class <code>stslist.modst</code> as produced by the <code>seqmodst</code> function.
<code>cpal</code>	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the <code>'cpal'</code> attribute of the <code>x</code> object is used.
<code>ylab</code>	an optional label for the y axis. If set to <code>NA</code> , no label is drawn.
<code>yaxis</code>	if <code>TRUE</code> (default) the y axis is plotted.
<code>xaxis</code>	if <code>TRUE</code> (default) the x axis is plotted.
<code>xtlab</code>	optional labels for the x axis ticks. If unspecified, the <code>names</code> attribute of the <code>x</code> object is used.
<code>xtstep</code>	optional interval at which the tick-marks and labels of the x-axis are displayed. For example, with <code>xtstep=3</code> a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the <code>xtstep</code> attribute of the <code>x</code> object is used.
<code>tick.last</code>	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the <code>tick.last</code> attribute of the <code>x</code> object is used.
<code>info</code>	Logical: should info about frequency of occurrence of the sequence of modal states be displayed?
<code>cex.axis</code>	Axis annotation magnification. See <a href="#">par</a> .
<code>cex.plot</code>	Deprecated. Use <code>cex.axis</code> instead.
<code>...</code>	further graphical parameters. For more details about the graphical parameter arguments, see <code>barplot</code> and <code>par</code> .

## Details

This is the plot method for the output produced by the `seqmodst` function, i.e. objects of class `stslist.modst`. It produces a plot showing the sequence of modal states with bar width proportional to the state frequencies.

This method is called by the generic `seqplot` function (if `type="ms"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqmplot` function is a shortcut for calling `seqplot` with `type="ms"`.

## Examples

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
               "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## Modal state sequence
biofam.modst <- seqmodst(biofam.seq)
plot(biofam.modst)
```

---

plot.stslist.rep      *Plot method for representative sequence sets*

---

### Description

This is the plot method for output produced by the [seqrep](#) function, i.e. for objects of class *stslist.rep*. It produces a representative sequence plot.

### Usage

```
## S3 method for class 'stslist.rep'
plot(x, cpal = NULL, missing.color = NULL, pbarw = TRUE,
     dmax = NULL, stats = TRUE, ylab = NULL, xaxis = TRUE, xtlab = NULL,
     xtstep = NULL, tick.last = NULL, cex.with.axis = 1, cex.plot, ...)
```

### Arguments

<code>x</code>	an object of class <i>stslist.rep</i> as produced by the <a href="#">seqrep</a> function.
<code>cpal</code>	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the <i>x</i> object is used.
<code>missing.color</code>	alternative color for representing missing values inside the sequences. By default, this color is taken from the "missing.color" attribute of the sequence object being plotted.
<code>pbarw</code>	when TRUE, the bar heights are set proportional to the number of represented sequences.
<code>dmax</code>	maximal theoretical distance, used for the <i>x</i> axis limits.
<code>stats</code>	if TRUE (default), mean discrepancy in each subset defined by all sequences attributed to one representative sequence and the mean distance to this representative sequence are displayed.
<code>ylab</code>	an optional label for the <i>y</i> axis. If set to NA, no label is drawn.
<code>xaxis</code>	controls whether a <i>x</i> axis is plotted.
<code>xtlab</code>	optional labels for the <i>x</i> axis ticks labels. If unspecified, the column names of the object being plotted.
<code>xtstep</code>	optional interval at which the tick-marks and labels of the <i>x</i> -axis are displayed. For example, with <code>xtstep=3</code> a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the <code>xtstep</code> attribute of the <i>x</i> object is used.
<code>tick.last</code>	Logical. Should a tick mark be enforced at the last position on the <i>x</i> -axis? If unspecified, the <code>tick.last</code> attribute of the <i>x</i> object is used.
<code>cex.with.axis</code>	Axis annotation and plotting text and symbols magnification. See <a href="#">par</a> .
<code>cex.plot</code>	Deprecated. Use <code>cex.with.axis</code> instead.
<code>...</code>	further graphical parameters. For more details about the graphical parameter arguments, see <a href="#">barplot</a> and <a href="#">par</a> .

## Details

This is the plot method for the output produced by the `seqrep` function, i.e. objects of class `stslist.rep`. It produces a plot where the representative sequences are displayed as horizontal bars with width proportional to the number of sequences assigned to them. Sequences are plotted bottom-up according to their representativeness score.

Above the plot, two parallel series of symbols associated to each representative are displayed horizontally on a scale ranging from 0 to the maximal theoretical distance  $D_{max}$ . The location of the symbol associated to the representative  $r_i$  indicates on axis  $A$  the (pseudo) variance ( $V_i$ ) within the subset of sequences assigned to  $r_i$  and on the axis  $B$  the mean distance  $MD_i$  to the representative.

This method is called by the generic `seqplot` function (if `type="r"`) that produces more sophisticated plots with group splits and automatic display of the color legend. The `seqrplot` function is a shortcut for calling `seqplot` with `type="r"`.

## Author(s)

Alexis Gabadinho (with Gilbert Ritschard for the help page)

## Examples

```
## Loading the mvad data set and creating a sequence object
data(mvad)
mvad.labels <- c("employment", "further education", "higher education",
               "joblessness", "school", "training")
mvad.scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")

## First 36 months trajectories
mvad.seq <- seqdef(mvad, 15:50, states=mvad.scodes, labels=mvad.labels)

## Computing Hamming distances
##
dist.ham <- seqdist(mvad.seq, method="HAM")

## Extracting a representative set using the sequence frequency
## as a representativeness criterion
mvad.rep <- seqrep(mvad.seq, diss=dist.ham)

## Plotting the representative set
plot(mvad.rep)
```

---

plot.stslist.statd      *Plot method for objects produced by the seqstatd function*

---

## Description

This is the plot method for output produced by the `seqstatd` function, i.e for objects of class `stslist.statd`.

**Usage**

```
## S3 method for class 'stslist.statd'
plot(x, type = "d", cpal = NULL, ylab = NULL,
     yaxis = TRUE, xaxis = TRUE, xtlab = NULL, xtstep = NULL, tick.last = NULL,
     cex.axis = 1, space = 0, xlab = NULL, cex.plot, ...)
```

**Arguments**

x	an object of class <code>stslist.statd</code> as produced by the <a href="#">seqstatd</a> function.
type	if "d" (default), a state distribution plot is produced. If "Ht" an entropy index plot is produced.
cpal	alternative color palette to be used for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the x object is used.
ylab	an optional label for the y axis. If set to NA, no label is drawn.
yaxis	if TRUE or "cum", the y axis is plotted with a label showing the cumulated percentage frequency of the displayed sequences. If "pct", the percentage value for each sequence is displayed.
xaxis	if TRUE (default) the x-axis is plotted.
xtlab	optional labels for the ticks of the x-axis. If unspecified, the names attribute of the input x object is used.
xtstep	optional interval at which the tick-marks and labels of the x-axis are displayed. For example, with <code>xtstep=3</code> a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the <code>xtstep</code> attribute of the x object is used.
tick.last	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the <code>tick.last</code> attribute of the x object is used.
cex.axis	Axis annotation magnification. See <a href="#">par</a> .
space	the space between the stacked bars. Default is 0, i.e. no space.
xlab	Optional title for the x axis. See <a href="#">title</a> .
cex.plot	Deprecated. Use <code>cex.axis</code> instead.
...	further graphical parameters such as <code>border=NA</code> to remove the borders of the bars. For more details about the graphical parameter arguments, see <code>barplot</code> and <code>par</code> .

**Details**

This is the plot method for the output produced by the [seqstatd](#) function, i.e. for objects of class `stslist.statd`. If `type="d"` it produces a state distribution plot presenting the sequence of the transversal state frequencies at each successive (time) position, as computed by the [seqstatd](#) function. With `type="Ht"`, the series of entropies of the transversal state distributions is plotted.

This method is called by the generic [seqplot](#) function (if `type="d"` or `type="Ht"`) that produces more sophisticated plots, allowing grouping and automatic display of the state color legend. The `seqdplot` and `seqHtplot` functions are shortcuts for calling `seqplot` with `type="d"` or `type="Ht"` respectively.

**Value**

The plotted values, i.e. for type="d" the cross-sectional distributions, and for type="Ht" the cross-sectional entropies.

**Examples**

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
               "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## State distribution
biofam.statd <- seqstatd(biofam.seq)

## State distribution plot (default type="d" option)
plot(biofam.statd)

## Entropy index plot
plot(biofam.statd, type="Ht")
```

---

plot.subseqelist      *Plot frequencies of subsequences*

---

**Description**

Plot frequencies of subsequences.

**Usage**

```
## S3 method for class 'subseqelist'
plot(x, freq=NULL, cex=1, ...)
```

**Arguments**

x	The subsequences to plot (a subseqelist object)
freq	The frequencies to plot, support if NULL
cex	Plotting text and symbols magnification. See <a href="#">par</a> .
...	arguments passed to <a href="#">barplot</a>

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**See Also**[seqfsub](#)**Examples**

```
## loading data
data(actcal.tse)

## creating sequences
actcal.eseq <- seqecreate(actcal.tse)

## Looking for frequent subsequences
fsubseq <- seqfsub(actcal.eseq, pmin.support=0.01)

## Frequency of first ten subsequences
plot(fsubseq[1:10], cex=2)
plot(fsubseq[1:10])
```

---

plot.subseqelistchisq *Plot discriminant subsequences*

---

**Description**

Plot the result of [seqecmpgroup](#)

**Usage**

```
## S3 method for class 'subseqelistchisq'
plot(x, ylim = "uniform", rows = NA, cols = NA,
     resid.levels = c(0.05, 0.01),
     cpal = brewer.pal(1 + 2 * length(resid.levels), "RdBu"), vlegend = NULL,
     cex.legend = 1, ptype = "freq", legend.title = NULL,
     with.legend = TRUE, residlevels, legendcol, legend.cex, ...)
```

**Arguments**

x	The subsequences to plot (a subseqelist object).
ylim	if "uniform" all axes have same limits.
rows	Number of graphic rows
cols	Number of graphic columns
resid.levels	Significance levels used to colorize the Pearson residual
cpal	Color palette used to color the results
vlegend	When TRUE the legend is printed vertically, when FALSE it is printed horizontally. If NULL (default) the best position will be chosen.
cex.legend	Scale parameters for text legend.

ptype	If set to "resid", Pearson residuals are plotted instead of frequencies
legend.title	Legend title.
with.legend	Logical. Should legend be displayed?
residlevels	Deprecated. Use resid.levels instead.
legendcol	Deprecated. Use vlegend instead.
legend.cex	Deprecated. Use cex.legend instead.
...	Additional parameters passed to <a href="#">barplot</a>

**Value**

nothing

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**See Also**

[seqecmpgroup](#)

---

print.stslist                      *Print method for state sequence objects*

---

**Description**

This is the print method for state sequence objects of class `stslist` created by the `seqdef` function.

**Usage**

```
## S3 method for class 'stslist'
print(x, format='STS', extended=FALSE, ...)
```

**Arguments**

x	A state sequence ( <code>stslist</code> ) object.
format	String: print format. One of "STS" (default) or "SPS".
extended	Logical: should the output be printed in extended matrix form?
...	Additional print arguments.

**Author(s)**

Gilbert Ritschard

**See Also**

[seqdef](#), [plot.stslist](#)



## Examples

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam <- biofam[500:600,] ## using a subsample only
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
               "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.shortlab <- c("P", "L", "M", "LM", "C", "LC", "LMC", "D")
biofam.seq <- seqdef(biofam, 10:25, states=biofam.shortlab,
                    labels=biofam.lab)

## Print of first 5 sequences
print(biofam.seq[1:5,])
print(biofam.seq[1:5,], extended=TRUE)
print(biofam.seq[1:5,], format="SPS")
print(biofam.seq[1:5,], format="SPS", SPS.out = list(xfix = "", sdsep = "/"))
```

---

read.tda.mdist

*Read a distance matrix produced by TDA.*

---

## Description

This function reads a distance matrix produced by TDA into an R object. When computing OM distances in TDA, the output is a 'half' matrix stored in a text file as a vector.

## Usage

```
read.tda.mdist(file)
```

## Arguments

file            the path to the file containing TDA output.

## Value

a R matrix containing the distances.

seqalign

*Computation details about a pairwise alignment***Description**

The function provides details about a pairwise alignment.

**Usage**

```
seqalign(seqdata, indices, indel=1, sm, with.missing = FALSE)

## S3 method for class 'seqalign'
plot(x, cpal = NULL, missing.color = NULL, ylab = NULL,
     yaxis = TRUE, xaxis = TRUE, ytlab = NULL, ylas = 0, xtlab = NULL,
     cex.axis = 1, cex.plot, ...)

## S3 method for class 'seqalign'
print(x, digits=3, ...)
```

**Arguments**

seqdata	a state sequence object defined with the <a href="#">seqdef</a> function.
indices	a vector of length 2 giving the indexes of the two sequences
indel	indel cost (see <a href="#">seqdist</a> )
sm	matrix of substitution costs or a method for computing the costs (see <a href="#">seqdist</a> )
with.missing	logical: Should the missing state be considered as an element of the alphabet?
x	an object of class seqalign
cpal	color palette
missing.color	color for missing elements
ylab	y label
yaxis	yaxis
xaxis	xaxis
ytlab	ytlab
ylas	ylas
xtlab	xtlab
cex.axis	Axis annotation magnification. See <a href="#">par</a> .
digits	number of digits for printed output
cex.plot	Deprecated. Use cex.axis instead.
...	additional arguments passed to other functions

**Details**

There are print and plot methods for seqalign objects.

**Value**

Object of class seqalign

**Author(s)**

Alexis Gabadinho ([plot.seqalign](#)) and Matthias Studer ([seqalign](#)) (with Gilbert Ritschard for the help page)

**See Also**

[seqdist](#)

**Examples**

```
data(biofam)
biofam.seq <- seqdef(biofam, 10:25)
costs <- seqsubm(biofam.seq, method="TRATE")
sa <- seqalign(biofam.seq, 1:2, indel=1, sm=costs)
print(sa)
plot(sa)
sa <- seqalign(biofam.seq, c(1,5), indel=0.5, sm=costs)
print(sa)
plot(sa)
```

---

seqcomp

*Compare two state sequences*

---

**Description**

Check whether two state sequences are identical.

**Usage**

```
seqcomp(x, y)
```

**Arguments**

x	a state sequence object containing a single sequence (typically the row of a main sequence object, see <a href="#">seqdef</a> )
y	a state sequence object containing a single sequence (typically the row of a main sequence object, see <a href="#">seqdef</a> )

**Value**

TRUE if sequences are identical, FALSE otherwise

**See Also**

[seqfind](#), [seqfpos](#), [seqpm](#)

**Examples**

```

data(mvad)
mvad.shortlab <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, states=mvad.shortlab, 15:86)

## Comparing sequences 1 and 2 in mvad.seq
seqcomp(mvad.seq[1,],mvad.seq[2,])

## Comparing sequences 176 and 211 in mvad.seq
seqcomp(mvad.seq[176,],mvad.seq[211,])

```

seqconc

*Concatenate vectors of states or events into a character string***Description**

Concatenate vectors of states or events into a character string. In the string, each state is separated by 'sep'. The void elements in the input sequences are eliminated.

**Usage**

```
seqconc(data, var=NULL, sep="-", vname="Sequence", void=NA)
```

**Arguments**

data	A dataframe or matrix containing sequence data.
var	List of the columns containing the sequences. Default is NULL in which case all columns are retained. Whether the sequences are in the compressed (character strings) or extended format is automatically detected by counting the number of columns.
sep	Character used as separator. By default, "-".
vname	an optional name for the variable containing the sequences. By default, "Sequence".
void	the code used for void elements appearing in the sequences (see <i>Gabadinho et al. (2009)</i> for more details on missing values and void elements in sequences). Default is NA.

**Value**

a vector of character strings, one for each row in the input data.

**Author(s)**

Alexis Gabadinho

## References

Gabardinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

## See Also

[seqdecomp](#).

## Examples

```
data(actcal)
actcal.string <- seqconc(actcal,13:24)
head(actcal.string)
```

---

seqcost

*Generate substitution and indel costs*

---

## Description

The function `seqcost` proposes different ways to generate substitution costs (supposed to represent state dissimilarities) and possibly indel costs. Proposed methods are: "CONSTANT" (same cost for all substitutions), "TRATE" (derived from the observed transition rates), "FUTURE" (Chi-squared distance between conditional state distributions lag positions ahead), "FEATURES" (Gower distance between state features), "INDELS", "INDELSLOG" (based on estimated indel costs). The substitution-cost matrix is intended to serve as `sm` argument in the `seqdist` function that computes distances between sequences. `seqsubm` is an alias that returns only the substitution cost matrix, i.e., no indel.

## Usage

```
seqcost(seqdata, method, cval = NULL, with.missing = FALSE, miss.cost = NULL,
  time.varying = FALSE, weighted = TRUE, transition = "both", lag = 1,
  miss.cost.fixed = NULL, state.features = NULL, feature.weights = NULL,
  feature.type = list(), proximities = FALSE)
```

```
seqsubm(...)
```

## Arguments

<code>seqdata</code>	A sequence object as returned by the <code>seqdef</code> function.
<code>method</code>	String. How to generate the costs. One of "CONSTANT" (same cost for all substitutions), "TRATE" (derived from the observed transition rates), "FUTURE" (Chi-squared distance between conditional state distributions lag positions ahead), "FEATURES" (Gower distance between state features), "INDELS", "INDELSLOG" (based on estimated indel costs).

<code>cval</code>	Scalar. For method "CONSTANT", the single substitution cost. For method "TRATE", a base value from which transition probabilities are subtracted. If NULL, <code>cval=2</code> is used, unless transition is "both" and <code>time.varying</code> is TRUE, in which case <code>cval=4</code> .
<code>with.missing</code>	Logical. Should an additional entry be added in the matrix for the missing states? If TRUE, the 'missing' state is also added to the alphabet. Set as TRUE if you want to use the costs for distances between sequences containing non deleted (non void) missing values. Forced as FALSE when there are no non-void missing values in <code>seqdata</code> . See <i>Gabadinho et al. (2010)</i> for more details on the options for handling missing values when creating the state sequence object with <a href="#">seqdef</a> .
<code>miss.cost</code>	Scalar or vector. Cost for substituting the missing state. Default is <code>cval</code> .
<code>miss.cost.fixed</code>	Logical. Should the substitution cost for missing be set as the <code>miss.cost</code> value. When NULL (default) it will be set as FALSE when <code>method = "INDELS"</code> or "INDELSLOG", and TRUE otherwise.
<code>time.varying</code>	Logical. If TRUE return an <a href="#">array</a> with a distinct matrix for each time unit. Time is the third dimension (subscript) of the returned array. Time varying works only with <code>method='CONSTANT', 'TRATE', 'INDELS', and 'INDELSLOG'</code> .
<code>weighted</code>	Logical. Should weights in <code>seqdata</code> be used when applicable?
<code>transition</code>	String. Only used if <code>method="TRATE"</code> and <code>time.varying=TRUE</code> . On which transition are rates based? Should be one of "previous" (from previous state), "next" (to next state) or "both".
<code>lag</code>	Integer. For methods TRATE and FUTURE only. Time ahead to which transition rates are computed (default is <code>lag=1</code> ).
<code>state.features</code>	Data frame with features values for each state.
<code>feature.weights</code>	Vector of feature weights with length equal to the number of columns of <code>state.features</code> .
<code>feature.type</code>	List of feature types. See <a href="#">daisy</a> for details.
<code>proximities</code>	Logical: should state proximities be returned instead of substitution costs?
<code>...</code>	Arguments passed to <code>seqcost</code>

## Details

The substitution-cost matrix has dimension  $ns * ns$ , where  $ns$  is the number of states in the [alphabet](#) of the sequence object. The element  $(i, j)$  of the matrix is the cost of substituting state  $i$  with state  $j$ . It represents the dissimilarity between the states  $i$  and  $j$ . The indel cost of the cost of inserting or deleting a state.

With method CONSTANT, the substitution costs are all set equal to the `cval` value, the default value being 2.

With method TRATE (transition rates), the transition probabilities between all pairs of states is first computed (using the [seqtrate](#) function). Then, the substitution cost between states  $i$  and  $j$  is obtained with the formula

$$SC(i, j) = cval - P(i|j) - P(j|i)$$

where  $P(i|j)$  is the probability of transition from state  $j$  to  $i$  lag positions ahead. Default `cval` value is 2. When `time.varying=TRUE` and `transition="both"`, the substitution cost at position  $t$  is set as

$$SC(i, j, t) = cval - P(i|j, t - 1) - P(j|i, t - 1) - P(i|j, t) - P(j|i, t)$$

where  $P(i|j, t - 1)$  is the probability to transit from state  $j$  at  $t - 1$  to  $i$  at  $t$ . Here, the default `cval` value is 4.

With method `FUTURE`, the cost between  $i$  and  $j$  is the Chi-squared distance between the vector ( $d(alphabet|i)$ ) of probabilities of transition from states  $i$  and  $j$  to all the states in the alphabet lag positions ahead:

$$SC(i, j) = ChiDist(d(alphabet|i), d(alphabet|j))$$

With method `FEATURES`, each state is characterized by the variables `state.features`, and the cost between  $i$  and  $j$  is computed as the Gower distance between their vectors of `state.features` values.

With methods `INDELS` and `INDELSLOG`, values of indels are first derived from the state relative frequencies  $f_i$ . For `INDELS`,  $indel_i = 1/f_i$  is used, and for `INDELSLOG`,  $indel_i = \log[2/(1 + f_i)]$ . Substitution costs are then set as  $SC(i, j) = indel_i + indel_j$ .

For all methods but `INDELS` and `INDELSLOG`, the indel is set as  $\max(sm)/2$  when `time.varying=FALSE` and as 1 otherwise.

## Value

For `seqcost`, a list of two elements, `indel` and `sm` or `prox`:

<code>indel</code>	The indel cost. Either a scalar or a vector of size $ns$ . When <code>time.varying=TRUE</code> and method is one of "INDELS" or "INDELSLOG", a matrix with indels per time point in columns.
<code>sm</code>	The substitution-cost matrix (or array) when <code>proximities = FALSE</code> (default).
<code>prox</code>	The state proximity matrix when <code>proximities = TRUE</code> .

`sm` and `prox` are, when `time.varying = FALSE`, a matrix of size  $ns * ns$ , where  $ns$  is the number of states in the alphabet of the sequence object. When `time.varying = TRUE`, they are a three dimensional array of size  $ns * ns * L$ , where  $L$  is the maximum sequence length.

For `seqsubm`, only one element, the matrix (or array) `sm`.

## Author(s)

Gilbert Ritschard and Matthias Studer (and Alexis Gabadinho for first version of `seqsubm`)

## References

Gabardinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

Gabardinho, A., G. Ritschard, M. Studer and N. S. Müller (2010). Mining Sequence Data in R with the TraMineR package: A user's guide. Department of Econometrics and Laboratory of Demography, University of Geneva.

Studer, M. & Ritschard, G. (2016), "What matters in differences between life trajectories: A comparative review of sequence dissimilarity measures", *Journal of the Royal Statistical Society, Series A*. **179**(2), 481-511. doi: [10.1111/rssa.12125](https://doi.org/10.1111/rssa.12125)

Studer, M. and G. Ritschard (2014). "A Comparative Review of Sequence Dissimilarity Measures". *LIVES Working Papers*, **33**. NCCR LIVES, Switzerland, 2014. doi: [10.12682/lives.2296-1658.2014.33](https://doi.org/10.12682/lives.2296-1658.2014.33)

## See Also

[seqtrate](#), [seqdef](#), [seqdist](#).

## Examples

```
## Defining a sequence object with columns 10 to 25
## of a subset of the 'biofam' example data set.
data(biofam)
biofam.seq <- seqdef(biofam[501:600,10:25])

## Indel and substitution costs based on log of inverse state frequencies
lifcost <- seqcost(biofam.seq, method="INDELSLOG")
## Here lifcost$indel is a vector
biofam.om <- seqdist(biofam.seq, method="OM", indel=lifcost$indel, sm=lifcost$sm)

## Optimal matching using transition rates based substitution-cost matrix
## and the associated indel cost
## Here trcost$indel is a scalar
trcost <- seqcost(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", indel=trcost$indel, sm=trcost$sm)

## Using costs based on FUTURE with a forward lag of 4
fucost <- seqcost(biofam.seq, method="FUTURE", lag=4)
biofam.om <- seqdist(biofam.seq, method="OM", indel=fucost$indel, sm=fucost$sm)

## Optimal matching using a unique substitution cost of 2
## and an insertion/deletion cost of 3
ccost <- seqsubm(biofam.seq, method="CONSTANT", cval=2)
biofam.om.c2 <- seqdist(biofam.seq, method="OM", indel=3, sm=ccost)

## Displaying the distance matrix for the first 10 sequences
biofam.om.c2[1:10,1:10]

## =====
## Example with weights and missings
## =====
```



```

data(ex1)
ex1.seq <- seqdef(ex1[,1:13], weights=ex1$weights)

## Unweighted
subm <- seqcost(ex1.seq, method="INDELSLOG", with.missing=TRUE, weighted=FALSE)
ex1.om <- seqdist(ex1.seq, method="OM", indel=subm$indel, sm=subm$sm, with.missing=TRUE)

## Weighted
subm.w <- seqcost(ex1.seq, method="INDELSLOG", with.missing=TRUE, weighted=TRUE)
ex1.omw <- seqdist(ex1.seq, method="OM", indel=subm.w$indel, sm=subm.w$sm, with.missing=TRUE)

ex1.om == ex1.omw

```

---

seqdecomp

---

*Convert a character string into a vector of states or events*


---

### Description

States can be represented by any substring that does not include the sep value. An empty separator sep = "" can only be used when each state is coded with a single character.

### Usage

```
seqdecomp(data, var=NULL, sep='-', miss="NA", vnames=NULL)
```

### Arguments

data	a dataframe or matrix containing sequence data.
var	the list of columns containing the sequences. Default is NULL, ie all the columns. Whether the sequences are in the compressed (character strings) or extended format is automatically detected by counting the number of columns.
sep	the between states/events separator used in the input data set. Default is '-'. cant be empty.
miss	the symbol for missing values (if any) used in the input data set. Default is NA.
vnames	optional names for the column/variables of the output data set. Default is NULL.

### See Also

[seqconc](#).

### Examples

```

# 1 sequence of length 4
seqdecomp("A-BB-C-DD")

# 2 sequences of length 6
seqdecomp(c("ABBCDD", "BCCAD"), sep="")

```

seqdef

*Create a state sequence object***Description**

Create a state sequence object with attributes such as alphabet, color palette and state labels. Most TraMineR functions for state sequences require such a state sequence object as input argument. There are specific methods for plotting, summarizing and printing state sequence objects.

**Usage**

```
seqdef(data, var=NULL, informat="STS", stsep=NULL,
       alphabet=NULL, states=NULL, id=NULL, weights=NULL, start=1,
       left=NA, right="DEL", gaps=NA, missing=NA, void="%", nr="*",
       cnames=NULL, xtstep=1, tick.last=FALSE, cpal=NULL,
       missing.color="darkgrey", labels=NULL, ...)
```

**Arguments**

data	a data frame or matrix containing sequence data.
var	the list of columns containing the sequences. Default is NULL, i.e. all the columns. The function detects automatically whether the sequences are in the compressed (successive states in a character string) or extended format.
informat	format of the original data. Default is "STS". Other available formats are: "SPS" and "SPELL", in which case the <a href="#">seqformat</a> function is called to convert the data into the "STS" format (see TraMineR user's manual ( <i>Gabadinho et al., 2010</i> ) for a description of these formats). A better solution is nonetheless to convert first your data with <a href="#">seqformat</a> , so as to have better control over the conversion process and visualize the intermediate "STS" formatted data.
stsep	the character used as separator in the original data if input format is successive states in a character string. If NULL (default value), the <a href="#">seqfcheck</a> function is called for detecting automatically a separator among "-" and ":". Other separators must be specified explicitly.
alphabet	optional vector containing the alphabet (the list of all possible states). Use this option if some states in the alphabet don't appear in the data or if you want to reorder the states. The specified vector MUST contain AT LEAST all the states appearing in the data. It may possibly contain additional states not appearing in the data. If NULL, the alphabet is set to the distinct states appearing in the data as returned by the <a href="#">seqstat1</a> function. See details.
states	an optional vector containing the short state labels. Must have a length equal to the size of the alphabet and the labels must be ordered conformably with alphanumeric ordered values returned by the <a href="#">seqstat1</a> function, or, when alphabet= is set, with the thus newly defined alphabet.

<code>id</code>	optional argument for setting the rownames of the sequence object. If NULL (default), the rownames are taken from the input data. If set to "auto", sequences are numbered from 1 to the number of sequences. A vector of rownames of length equal to the number of sequences may be specified as well.
<code>weights</code>	optional numerical vector containing weights, which are accounted for by plotting and statistical functions when applicable.
<code>start</code>	starting time. For instance, if sequences begin at age 15, you can specify 15. At this stage, used only for labelling column names.
<code>left</code>	the behavior for missing values appearing before the first (leftmost) valid state in each sequence. When NA (default), left missing values are treated as 'real' missing values and converted to the internal missing value code defined by the <code>nr</code> option. Other options are "DEL" to delete the positions containing missing values or a state code (belonging to the alphabet or not) to replace the missing values. See <i>Gabadinho et al. (2010)</i> for more details on the options for handling missing values when defining sequence objects.
<code>right</code>	the behavior for missing values appearing after the last (rightmost) valid state in each sequence. Same options as for the <code>left</code> argument. Default is 'DEL'.
<code>gaps</code>	the behavior for missing values appearing inside the sequences, i.e. after the first (leftmost) valid state and before the last (rightmost) valid state of each sequence. Same options as for the <code>left</code> argument. Default is NA.
<code>missing</code>	the code used for missing values in the input data. Default is NA. When any other value, all cells containing this value are treated as NAs and replaced by <code>nr</code> or void code according to the <code>left</code> , <code>gaps</code> , and <code>right</code> options.
<code>void</code>	the internal code used by TraMineR for representing void elements in the sequences. Default is "%". Must be different from <code>left</code> , <code>gaps</code> , and <code>right</code> .
<code>nr</code>	the internal code used by TraMineR for representing real missing elements in the sequences. Default is "*".
<code>cnames</code>	optional names for the columns composing the sequence data. Those names will be used by default in the graphics as axis labels. If NULL (default), names are taken from the original column names in the data.
<code>xtstep</code>	step between displayed tick-marks and labels on the time x-axis of state sequence plots. If not overridden by the user, plotting functions retrieve this parameter from the <code>xtstep</code> attribute of the sequence object. For example, with <code>xtstep=3</code> a tick-mark is displayed at positions 1, 4, 7, etc... Default value is 1; i.e., a tick mark is displayed at each position. The display of the corresponding labels depends on the available space and is dealt with automatically.
<code>tick.last</code>	Logical. Should a tick mark be enforced at the last position on the time x-axis?
<code>cpal</code>	an optional color palette for representing the states in the graphics. If NULL (default), a color palette is created by means of the <code>brewer.pal</code> function of the <code>RColorBrewer</code> package for number of states up to 12. When the number of states is less or equal than 8, the "Accent" palette is used. If number of states is between 8 and 12, the "Set3" palette is used. When the number of states is greater than 12, colors are set using <code>hcl.colors</code> with the "Set 3" palette. To specify your own palette use e.g. the <code>colors</code> function, or the <code>RColorBrewer</code> or <code>colorspace</code> packages.

<code>missing.color</code>	alternative color for representing missing values inside the sequences. Defaults to "darkgrey".
<code>labels</code>	optional state labels used for the color legend of TraMineR's graphics. If NULL (default), the state names in the alphabet are used as state labels as well.
<code>...</code>	options passed to the <code>seqformat</code> function for handling input data that is not in STS format.

### Details

Applying subscripts to sequence objects (eg. `seq[, 1:5]` or `seq[1:10, ]`) returns a state sequence object with some attributes preserved (`alphabet`, `missing`) and some others (`start`, `column names`, `weights`) adapted to the selected column or row subset. When the number of columns selected is 1, the returned object is a factor.

For reordering the states use the `alphabet` argument. This may for instance be useful to compare data from different sources with different codings of similar states. Using `alphabet` permits to order the states conformably in all sequence objects. Otherwise, the default state order is the alphanumeric order returned by the `seqstat1` function which may differ when you have different original codings.

### Value

An object of class `stslst`.

There are `print`, `plot`, `rbind`, `summary`, and subsetting `[, ]` methods for such objects.

### Author(s)

Alexis Gabadinho and Gilbert Ritschard

### References

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2010). Mining Sequence Data in R with the TraMineR package: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

### See Also

`plot.stslst` plot method for state sequence objects,  
`print.stslst` print method for state sequence objects,  
`is.stslst` to test whether an object is a proper `stslst` object,  
`seqplot` for high level plots of state sequence objects,  
`seqcreate` to create an event sequence object,  
`seqformat` for converting between various longitudinal data formats.

**Examples**

```

## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24,
labels=c("> 37 hours", "19-36 hours", "1-18 hours", "no work"))

## Displaying the first 10 rows of the sequence object
actcal.seq[1:10,]

## Displaying the first 10 rows of the sequence object
## in SPS format
print(actcal.seq[1:10,], format="SPS")

## Plotting the first 10 sequences
plot(actcal.seq)

## Re-ordering the alphabet
actcal.seq <- seqdef(actcal,13:24,alphabet=c("B","A","D","C"))
alphabet(actcal.seq)

## Adding a state not appearing in the data to the
## alphabet
actcal.seq <- seqdef(actcal,13:24,alphabet=c("A","B","C","D","E"))
alphabet(actcal.seq)

## Adding a state not appearing in the data to the
## alphabet and changing the states labels
actcal.seq <- seqdef(actcal,13:24,
  alphabet=c("A","B","C","D","E"),
  states=c("FT","PT","LT","NO","TR"))
alphabet(actcal.seq)

## rbind and summary
seq1 <- actcal.seq[1:10,]
seq2 <- actcal.seq[20:25,]
seq <- rbind(seq1,seq2)
summary(seq)

## =====
## Example with missing values
## =====
data(ex1)

## With right="DEL" default value
seqdef(ex1,1:13)

## Eliminating 'left' missing values
seqdef(ex1,1:13, left="DEL")

## Eliminating 'left' missing values and gaps
seqdef(ex1,1:13, left="DEL", gaps="DEL")

```

```
## =====
## Example with weights
## =====
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## weighted sequence frequencies
seqtab(ex1.seq)
```

---

seqdiff

*Position-wise discrepancy analysis between groups of sequences*


---

### Description

The function analyses how the differences between groups of sequences evolve along the positions. It runs a sequence of discrepancy analyses on sliding windows.

### Usage

```
seqdiff(seqdata, group, cmprange = c(0, 1),
        seqdist.args = list(method = "LCS", norm = "auto"), with.missing = FALSE,
        weighted = TRUE, squared = FALSE, seqdist_arg)
```

### Arguments

seqdata	a state sequence object created with the <a href="#">seqdef</a> function.
group	The group variable.
cmprange	Vector of two integers: Time range of the sliding windows. Comparison at $t$ is computed on the window $(t+cmprange[1], t+cmprange[2])$ .
seqdist.args	List of arguments passed to <a href="#">seqdist</a> for computing the distances.
with.missing	Logical. If TRUE, missing values are considered as an additional state. If FALSE subsequences with missing values are removed from the analysis.
weighted	Logical. If TRUE, seqdiff uses the weights specified in seqdata.
squared	Logical. If TRUE the dissimilarities are squared for computing the discrepancy.
seqdist_arg	Deprecated. Use seqdist.args instead.

### Details

The function analyses how the part of discrepancy explained by the group variable evolves along the position axis. It runs successively discrepancy analyses within a sliding time-window of range cmprange). At each position  $t$ , the method uses [seqdist](#) to compute a distance matrix over the time-window  $(t+cmprange[1], t+cmprange[2])$  and then derives the explained discrepancy on that window with [dissassoc](#).

There are print and plot methods for the returned value.

**Value**

A seqdiff object, with the following items:

stat	A data.frame with five statistics (Pseudo F, Pseudo Fbf, Pseudo R2, Bartlett, and Levene) for each time stamp of the sequence (see <a href="#">dissassoc</a> )
discrepancy	A data.frame with, at each time position $t$ , the discrepancy within the whole set of sequences and within each group (defined by the group variable).

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**References**

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2010) Discrepancy analysis of complex objects using dissimilarities. In F. Guillet, G. Ritschard, D. A. Zighed and H. Briand (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence, Volume 292, pp. 3-19. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.

**See Also**

[dissassoc](#) to analyse the association of the group variable with the whole sequence

**Examples**

```
## Define a state sequence object
data(mvad)
## First 12 months of first 100 trajectories
mvad.seq <- seqdef(mvad[1:100, 17:28])

## Position-wise discrepancy analysis using
## centered sliding windows of length 5.
mvad.diff <- seqdiff(mvad.seq, group=mvad$gcse5eq[1:100], cmprange=c(-2,2))
print(mvad.diff)
plot(mvad.diff, stat=c("Pseudo R2", "Levene"))
plot(mvad.diff, stat="discrepancy")
```

---

seqdim	<i>Dimension of a set of sequences</i>
--------	--

---

**Description**

Returns the number of sequences (rows) and the maximum length of a set of sequences.

**Usage**

```
seqdim(seqdata)
```

**Arguments**

seqdata          a set of sequences.

**Details**

The function will first search for separators '-' or ':' in the sequences in order to detect whether they are in the compressed or extended format.

**Value**

a vector with the number of sequences and the maximum sequence length.

**Author(s)**

Alexis Gabadinho

---

seqdist	<i>Distances (dissimilarities) between sequences</i>
---------	--

---

**Description**

Computes pairwise dissimilarities between sequences or dissimilarity from a reference sequence. Several dissimilarity measures can be chosen, including optimal matching (OM) and many of its variants, distance based on the count of common attributes, and distances between sequence state distributions.

**Usage**

```
seqdist(seqdata, method, refseq = NULL, norm = "none", indel = "auto", sm = NULL,
  with.missing = FALSE, full.matrix = TRUE, kweights = rep(1.0, ncol(seqdata)),
  tpow = 1.0, expcost = 0.5, context, link = "mean", h = 0.5, nu,
  transindel = "constant", otto, previous = FALSE, add.column = TRUE,
  breaks = NULL, step = 1, overlap = FALSE, weighted = TRUE,
  global.pdotj = NULL, prox = NULL, check.max.size=TRUE)
```



**Arguments**

seqdata	State sequence object of class <code>stsl</code> . The sequence data to use. Use <code>seqdef</code> to create such an object.
method	String. The dissimilarity measure to use. It can be "OM", "OMloc", "OMslen", "OMspell", "OMstran", "HAM", "DHD", "CHI2", "EUCLID", "LCS", "LCP", "RLCP", "NMS", "NMSMST", "SVRspell", or "TWED". See the Details section.
refseq	NULL, Integer, State Sequence Object, or List. Default: NULL. The baseline sequence to compute the distances from. When an integer, the index of a sequence in <code>seqdata</code> or 0 for the most frequent sequence. When a state sequence object, it must contain a single sequence and have the same alphabet as <code>seqdata</code> . When a list, it must be a list of two sets of indexes of <code>seqdata</code> rows.
norm	String. Default: "none". The normalization to use when <code>method</code> is one of "OM", "OMloc", "OMslen", "OMspell", "OMstran", "TWED", "HAM", "DHD", "LCS", "LCP", "RLCP", "CHI2", "EUCLID". It can be "none", "auto", or, except for "CHI2" and "EUCLID", "maxlength", "gmean", "maxdist", or "YujianBo". "auto" is equivalent to "maxlength" when <code>method</code> is one of "OM", "HAM", or "DHD", to "gmean" when <code>method</code> is one of "LCS", "LCP", or "RLCP", to <code>YujianBo</code> when <code>method</code> is one of "OMloc", "OMslen", "OMspell", "OMstran", "TWED". See the Details section.
indel	Double, Vector of Doubles, or String. Default: "auto". Insertion/deletion cost(s). Applies when <code>method</code> is one of "OM", "OMslen", "OMspell", or "OMstran". The single state-independent insertion/deletion cost when a double. The state-dependent insertion/deletion costs when a vector of doubles. The vector should contain an indel cost by state in the order of the alphabet. When "auto", the indel is set as $\max(sm)/2$ when <code>sm</code> is a matrix and is computed by means of <code>seqcost</code> when <code>sm</code> is a string specifying a cost method.
sm	NULL, Matrix, Array, or String. Substitution costs. Default: NULL. The substitution-cost matrix when a matrix and <code>method</code> is one of "OM", "OMloc", "OMslen", "OMspell", "OMstran", "HAM", or "TWED". The series of the substitution-cost matrices when an array and <code>method</code> = "DHD". They are grouped in a 3-dimensional array with the third index referring to the position in the sequence. One of the strings "CONSTANT", "INDELS", "INDELSLOG", or "TRATE". Designates a <code>seqcost</code> method to build <code>sm</code> . "CONSTANT" is not relevant for "DHD". <code>sm</code> is mandatory when <code>method</code> is one of "OM", "OMloc", "OMslen", "OMspell", "OMstran", or "TWED". <code>sm</code> is autogenerated when <code>method</code> is one of "HAM" or "DHD" and <code>sm</code> = NULL. See the Details section. Note: With <code>method</code> = "NMS" or <code>method</code> = "SVRspell", use <code>prox</code> instead.
with.missing	Logical. Default: FALSE. Should the non-deleted missing value be added to the alphabet as an additional state? If FALSE and <code>seqdata</code> or <code>refseq</code> contains such gaps, an error is raised.

full.matrix	Logical. Default: TRUE. When refseq = NULL, if TRUE, the full distance matrix is returned, if FALSE, an object of class <code>dist</code> is returned, that is, a vector containing only values from the upper triangle of the distance matrix. Objects of class <code>dist</code> are smaller and can be passed directly as arguments to most clustering functions.
kweights	Vector of Doubles. Default: vector of 1.0. The weights applied to subsequences when method is one of "NMS", "NMSMST", or "SVRspell". It contains at position $k$ the weight applied to the subsequences of length $k$ . It must be positive. Its length must be equal to the number of columns of seqdata.
tpow	Double. Default: 1.0. The exponential weight of spell length when method is one of "OMspell", "NMSMST", or "SVRspell".
expcost	Double. Default: 0.5. The cost of spell length transformation when method = "OMloc" or method = "OMspell". It must be positive. The exact interpretation is distance-dependent.
context	Double. Default: 1-2*expcost. The cost of local insertion when method = "OMloc". It must be positive.
link	String. Default: "mean". The function used to compute substitution costs when method = "OMslen". One of "mean" (arithmetic average) or "gmean" (geometric mean as in the original proposition of Halpin 2010).
h	Double. Default: 0.5. It must be greater than or equal to 0. The exponential weight of spell length when method = "OMslen". The gap penalty when method = "TWED". It corresponds to the lambda in <i>Halpin (2014), p 88</i> . It is usually chosen in the range [0,1]
nu	Double. Stiffness when method = "TWED". It must be strictly greater than 0 and is usually less than 1. See <i>Halpin (2014), p 88</i> .
transindel	String. Default: "constant". Method for computing transition indel costs when method = "OMstran". One of "constant" (single indel of 1.0), "subcost" (based on substitution costs), or "prob" (based on transition probabilities).
otto	Double. The origin-transition trade-off weight when method = "OMstran". It must be in [0, 1].
previous	Logical. Default: FALSE. When method = "OMstran", should we also account for the transition from the previous state?
add.column	Logical. Default: TRUE. When method = "OMstran", should the last column (and also the first column when previous = TRUE) be duplicated? When sequences have different lengths, should the last (first) valid state be duplicated.
breaks	NULL, List of pairs Integers. Default: NULL. The list of the possibly overlapping intervals when method = "CHI2" or method = "EUCLID".
step	Integer. Default: 1. The length of the intervals when method = "CHI2" or method = "EUCLID" and breaks = NULL. It must be positive. It must also be even when overlap = TRUE.
overlap	Logical. Default: FALSE. When method = "CHI2" or method = "EUCLID" and breaks = NULL, should the intervals overlap?
weighted	Logical. Default: TRUE. When method is "CHI2" or when sm is a string (method), should the distributions of the states account for the sequence weights in seqdata? See <a href="#">seqdef</a> .

global.pdotj	Numerical vector, "obs", or NULL. Default: NULL. Only for method = "CHI2". The vector of state proportions to be used as marginal distribution. When NULL, the state distribution on the corresponding interval is used. When "obs", the overall state distribution in seqdata is used for all intervals. When a vector of proportions, it is used as marginal distribution for all intervals.
prox	NULL or Matrix. Default: NULL. The matrix of state proximities when method = "NMS" or method = "SVRspell".
check.max.size	Logical. Should seqdist stop when maximum allowed number of unique sequences is exceeded? Caution, setting FALSE may produce unexpected results or even crash R.

## Details

The seqdist function returns a matrix of distances between sequences or a vector of distances from the reference sequence when refseq is set. The available metrics (see method option) include:

- *Edit distances*: optimal matching ("OM"), localized OM ("OMloc"), spell-length-sensitive OM ("OMslen"), OM of spell sequences ("OMspell"), OM of transition sequences ("OMstran"), Hamming ("HAM"), dynamic Hamming ("DHD"), and the time warp edit distance ("TWED").
- *Metrics based on counts of common attributes*: distance based on the longest common subsequence ("LCS"), on the longest common prefix ("LCP"), on the longest common suffix ("RLCP"), on the number of matching subsequences ("NMS"), on the number of matching subsequences weighted by the minimum shared time ("NMSMST") and, the subsequence vectorial representation distance ("SVRspell").
- *Distances between state distributions*: Euclidean ("EUCLID"), Chi-squared ("CHI2").

See *Studer and Ritschard (2014, 2016)* for a description and the comparison of the above dissimilarity measures except "TWED" for which we refer to *Marteau (2009)* and *Halpin (2014)*.

Each method can be controlled with the following parameters:

method	parameters
OM	sm, indel, norm
OMloc	sm, expcost, context, norm
OMslen	sm, indel, link, h, norm
OMspell	sm, indel, norm, tpow, expcost, norm
OMstran	sm, indel, transindel, otto, previous, add.column, norm
HAM, DHD	sm, norm
CHI2	breaks, step, overlap, norm, weighted, global.pdotj, norm
EUCLID	breaks, step, overlap, norm
LCS, LCP, RLCP	norm
NMS	prox, kweights
NMSMST	kweights, tpow
SVRspell	prox, kweights, tpow
TWED	sm, (indel), h, nu, norm

"LCS" is "OM" with a substitution cost of 2 (sm = "CONSTANT", cval = 2) and an indel of 1.0.

"HAM" is "OM" without indels. "DHD" is "HAM" with specific substitution costs at each position.

"HAM" and "DHD" apply only to sequences of equal length.

For "TWED", the (single) indel serves only for empty sequences. The distance to an empty sequence is set as  $n * \text{indel}$ , where  $n$  is the length of the non empty sequence. By default (`indel="auto"`), `indel` is set as  $2 * \max(\text{sm}) + \text{nu} + \text{h}$ .

When `sm = NULL`, the substitution-cost matrix is automatically created for "HAM" with a single substitution cost of 1 and for "DHD" with the costs derived from the transition rates at the successive positions, i.e. with `sm = "TRATE"`.

Some distances can optionally be normalized by means of the `norm` argument. Let  $d$  be the distance,  $m$  the maximum possible of the distance given the lengths  $p$  and  $q$  of the two sequences, and  $k$  the length of the longer sequence. Normalization "maxlength" is  $d/k$  (Abbott's normalization), "gmean" is  $1 - (m - d)/(p * q)^{.5}$  (Elzinga's normalization), "maxdist" is  $d/m$ , and "YujianBo" is  $2 * d/(m + d)$ . For more details, see *Gabadinho et al. (2009, 2011)*. Actually, to avoid negative outcomes, the length  $p$ ,  $q$ , and  $k$  are set as (max) indel times the corresponding length. For some distances,  $m$  is only a possibly non-reachable upper bound.

When `norm="auto"`, "gmean" is applied to "LCS", "LCP" and "RLCP" distances, "maxlength" is applied to "OM", "HAM" and "DHD", and the normalization "YujianBo" of *Yujian and Bo (2007)* that preserves the triangle inequality is used in the other cases except "CHI2" and "EUCLID". For the latter two, the square of the distances are normalized by the number of intervals and the maximal distance on each interval. Note that for 'CHI2' the maximal distance on each interval depends on the state distribution on the interval.

When sequences contain gaps and the `left = NA`, `gaps = NA`, or `right = NA` option was passed to `seqdef` (i.e. when there are non deleted missing values), the `with.missing` argument should be set as `TRUE`. If left as `FALSE` the function stops when it encounters a gap. This is to make the user aware that there are gaps in the sequences. For methods that need an `sm` value, `seqdist` expects a substitution-cost matrix with a row and a column entry for the missing state (symbol defined with the `nr` option of `seqdef`). Substitution-cost matrices returned by `seqcost` (and so `seqsubm`) include these additional entries when the function is called with `with.missing = TRUE`. More details on how to compute distances with sequences containing gaps can be found in *Gabadinho et al. (2009)*.

## Value

When `refseq` is `NULL` (default), the whole matrix of pairwise distances between sequences or, if `full.matrix = FALSE`, the corresponding `dist` object of pairwise distances between sequences.

When `refseq` is a `list` of two sets of indexes, the matrix of distances from the first set of sequences (rows) to the second set (columns).

Otherwise, a vector with distances between the sequences in the state sequence object and the reference sequence specified with `refseq`.

## Author(s)

Matthias Studer, Pierre-Alexandre Fonta, Alexis Gabadinho, Nicolas S. Müller, Gilbert Ritschard.

## References

Studer, M. and G. Ritschard (2016), "What matters in differences between life trajectories: A comparative review of sequence dissimilarity measures", *Journal of the Royal Statistical Society, Series*

A. **179**(2), 481-511, doi: [10.1111/rssa.12125](https://doi.org/10.1111/rssa.12125)

Studer, M. and G. Ritschard (2014). "A Comparative Review of Sequence Dissimilarity Measures". *LIVES Working Papers*, **33**. NCCR LIVES, Switzerland, doi: [10.12682/lives.22961658.2014.33](https://doi.org/10.12682/lives.22961658.2014.33)

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1–37.

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. Department of Econometrics and Laboratory of Demography, University of Geneva

Halpin, B. (2014). Three Narratives of Sequence Analysis, in Blanchard, P., Bühlmann, F. and Gauthier, J.-A. (Eds.) *Advances in Sequence Analysis: Theory, Method, Applications*, Vol 2 of Series *Life Course Research and Social Policies*, pages 75–103, Heidelberg: Springer. doi: [10.1007/9783319049694\\_5](https://doi.org/10.1007/9783319049694_5)

Marteau, P.-F. (2009). Time Warp Edit Distances with Stiffness Adjustment for Time Series Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **31**(2), 306–318. doi: [10.1109/TPAMI.2008.76](https://doi.org/10.1109/TPAMI.2008.76)

Yujian, L. and Bo, L. (2007). A normalized Levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **29**(6), 1091–1095. doi: [10.1109/TPAMI.2007.1078](https://doi.org/10.1109/TPAMI.2007.1078)

See also all references in *Studer and Ritschard (2014, 2016)*

### See Also

[seqcost](#), [seqsubm](#), [seqdef](#), and for multichannel distances [seqdistmc](#).

### Examples

```
## =====
## Examples without missings
## =====

## Defining a sequence object with columns 10 to 25
## of a subset of the 'biofam' data set
data(biofam)
biofam.seq <- seqdef(biofam[501:600, 10:25])

## OM distances using the vector of indels and substitution
## costs derived from the estimated state frequencies
costs <- seqcost(biofam.seq, method = "INDELSLOG")
biofam.om <- seqdist(biofam.seq, method = "OM",
                    indel = costs$indel, sm = costs$sm)

## OM between sequences of transitions
biofam.omstran <- seqdist(biofam.seq, method = "OMstran",
                        indel = costs$indel, sm = costs$sm,
                        otto=.3, transindel="subcost")

## Normalized LCP distances
biofam.lcp.n <- seqdist(biofam.seq, method = "LCP",
                       norm = "auto")
```

```

## Normalized LCS distances to the most frequent sequence
biofam.dref1 <- seqdist(biofam.seq, method = "LCS",
                      refseq = 0, norm = "auto")

## LCS distances to an external sequence
ref <- seqdef(as.matrix("(0,5)-(3,5)-(4,6)"), informat = "SPS",
             alphabet = alphabet(biofam.seq))
biofam.dref2 <- seqdist(biofam.seq, method = "LCS",
                      refseq = ref)

## LCS distances between two subsets of sequences
set1 <- 1:10
set2 <- 31:36
biofam.dref2 <- seqdist(biofam.seq, method = "LCS",
                      refseq = list(set1,set2))

## Chi-squared distance over the full observed timeframe
biofam.chi.full <- seqdist(biofam.seq, method = "CHI2",
                         step = max(seqlength(biofam.seq)))

## Chi-squared distance over successive overlapping
## intervals of length 4
biofam.chi.ostep <- seqdist(biofam.seq, method = "CHI2",
                          step = 4, overlap = TRUE)

## =====
## Examples with missings
## =====
data(ex1)
## Ignore empty row 7
ex1.seq <- seqdef(ex1[1:6, 1:13])

## OM with indel and substitution costs based on
## log of inverse state frequencies
costs.ex1 <- seqcost(ex1.seq, method = "INDELSLOG",
                   with.missing = TRUE)
ex1.om <- seqdist(ex1.seq, method = "OM",
                indel = costs.ex1$indel, sm = costs.ex1$sm,
                with.missing = TRUE)

## Localized OM
ex1.omloc <- seqdist(ex1.seq, method = "OMloc",
                   sm = costs.ex1$sm, expcost=.1, context = .4,
                   with.missing = TRUE)

## OMspell with a scalar indel
indel <- max(costs.ex1$indel)
## OM of spells
ex1.omspell <- seqdist(ex1.seq, method = "OMspell",
                    indel = indel, sm = costs.ex1$sm,
                    with.missing = TRUE)

```

```
## Distance based on number of matching subsequences
ex1.nms <- seqdist(ex1.seq, method = "NMS",
                  with.missing = TRUE)

## Using the sequence vectorial representation metric
costs.fut <- seqcost(ex1.seq, method = "FUTURE", lag = 4,
                   proximities = TRUE, with.missing = TRUE)
ex1.svr <- seqdist(ex1.seq, method = "SVRspell",
                  prox = costs.fut$prox, with.missing = TRUE)
```

---

seqdistmc

*Multichannel distances between sequences*


---

### Description

Compute multichannel pairwise optimal matching (OM) distances between sequences by deriving the substitution costs from the costs of the single channels. Works with OM and its following variants: distance based on longest common subsequence (LCS), Hamming distance (HAM), and Dynamic Hamming distance (DHD).

### Usage

```
seqdistmc(channels, method, norm="none", indel="auto", sm=NULL,
          with.missing=FALSE, full.matrix=TRUE, link="sum", cval=2,
          miss.cost=2, cweight=NULL, what="diss", ch.sep="@@@TraMinerSep@@@")
```

### Arguments

channels	A list of state sequence objects defined with the <a href="#">seqdef</a> function, each state sequence object corresponding to a "channel".
method	a character string indicating the metric to be used. One of "OM" (Optimal Matching), "LCS" (Longest Common Subsequence), "HAM" (Hamming distance), "DHD" (Dynamic Hamming distance).
norm	String. Default: "none". The normalization method to use. See <a href="#">seqdist</a> .
indel	Double, vector of doubles, or list with an insertion/deletion cost or a vector of state dependent indel costs for each channel. Can also be "auto" (default), in which case the indel cost of each channel is automatically set in accordance with the sm value of the channel. See indel argument of <a href="#">seqdist</a> .
sm	A list with a substitution-cost matrix for each channel or a list of method names for generating the substitution-costs (see <a href="#">seqcost</a> ). Ignored when method="LCS".
with.missing	Logical or vector of logical. Must be TRUE for channels with non deleted gaps (missing values). See details.
full.matrix	Logical. If TRUE (default), the full distance matrix is returned. If FALSE, an object of class <a href="#">dist</a> is returned.

link	Character string. One of "sum" or "mean". Method to compute the "link" between channels. Default is to sum the substitution costs.
cval	Double. Substitution cost for "CONSTANT" matrix, see <a href="#">seqcost</a> .
miss.cost	Double. Missing values substitution cost, see <a href="#">seqcost</a> .
cweight	A vector of channel weights. Default is 1 (same weight for each channel).
what	Character string. What output should be returned? One of "diss", "sm", "seqmc".
ch.sep	Character string. Separator used for building state names of the expanded alphabet.

### Details

The `seqdistmc` function builds a state sequence by combining the channels, derives the multichannel indel and substitution costs from the indel and substitution costs of each channel (following the strategy proposed by *Pollock, 2007*), and computes the multichannel distances using the multichannel costs. The available metrics (see `method` argument) are optimal matching ("OM"), longest common subsequence ("LCS"), Hamming distance ("HAM"), and Dynamic Hamming Distance ("DHD"). For other edit distances, extract the combined state sequence object (by setting `what="seqmc"`) and the multichannel substitution and indel costs (by setting `what="sm"`). Then use these outcomes as input in a call to `seqdist`. See [seqdist](#) for more information about available distances measures.

Normalization may be useful when dealing with sequences that are not all of the same length. For details on the applied normalization, see [seqdist](#).

### Value

When `what="diss"`, a matrix of pairwise distances between multichannel sequences.

When `what="sm"`, the matrix of substitution costs with three attributes: `indel` the indel cost(s), `alphabet` the alphabet of the combined state sequences, and `cweight` the channel weights used.

When `what="seqmc"`, the combined state sequence object.

### Author(s)

Matthias Studer and Gilbert Ritschard

### References

Pollock, Gary (2007) Holistic trajectories: a study of combined employment, housing and family careers by using multiple-sequence analysis. *Journal of the Royal Statistical Society: Series A* **170**, Part 1, 167–183.

### See Also

[seqcost](#), [seqdef](#), [seqdist](#).

### Examples

```
data(biofam)

## Building one channel per type of event left, children or married
```



```

bf <- as.matrix(biofam[, 10:25])
children <- bf==4 | bf==5 | bf==6
married <- bf == 2 | bf== 3 | bf==6
left <- bf==1 | bf==3 | bf==5 | bf==6

## Building sequence objects
child.seq <- seqdef(children)
marr.seq <- seqdef(married)
left.seq <- seqdef(left)

## Using transition rates to compute substitution costs on each channel
mcdist <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
  method="OM", sm =list("INDELSLOG", "INDELSLOG", "TRATE"))

## Using a weight of 2 for children channel and specifying substitution-cost
smatrix <- list()
smatrix[[1]] <- seqsubm(child.seq, method="CONSTANT")
smatrix[[2]] <- seqsubm(marr.seq, method="CONSTANT")
smatrix[[3]] <- seqsubm(left.seq, method="TRATE")
mcdist2 <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
  method="OM", sm =smatrix, cweight=c(2,1,1))

## Retrieving the multichannel sequences
mcseq <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
  method="OM", sm =smatrix, cweight=c(2,1,1), what="seqmc", ch.sep="+")
alphabet(mcseq)

## Retrieving the multichannel substitution costs
mcsm <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
  method="OM", sm =smatrix, cweight=c(2,1,1), what="sm", ch.sep="+")

```

---

seqdss

*Extract sequences of distinct successive states*


---

## Description

Extract the sequence of distinct successive states from each sequence in a object.

## Usage

```
seqdss(seqdata, with.missing=FALSE)
```

## Arguments

seqdata	a sequence object as defined by the <a href="#">seqdef</a> function.
with.missing	Should non-void missing values be considered as regular states? See Details.

**Details**

Returns a sequence object containing the sequences of distinct successive states (DSS). The spell durations are not taken into account. E.g., the DSS contained in 'D-D-D-D-A-A-A-A-A-A-D' is 'D-A-D'. Associated durations can be extracted with the [seqdur](#) function.

When {with.missing=TRUE}, non-void missing values are considered as a regular state of the alphabet. For example, the DSS of A-A-\*-\*-\*B-B-C-C-D is A-\*B-C-D.

When with.missing=FALSE (default) missing values are ignored and a substring A-A-\*-\*-\*A for example will be considered as a single spell in A while the DSS of this substring would be A-\*A with with.missing=TRUE.

See [seqdef](#) on options for handling missing values when creating sequence objects.

**Value**

a sequence object containing the distinct state sequence (DSS) for each sequence in the object given as argument.

**Author(s)**

Alexis Gabadinho and Gilbert Ritschard

**See Also**

[seqdur](#).

**Examples**

```
## Creating a sequence object from columns 13 to 24
## in the 'actcal' example data set
## Here we retain the first 10 sequences only.
data(actcal)
actcal.seq <- seqdef(actcal[1:10,13:24])

## Retrieving the DSS
actcal.dss <- seqdss(actcal.seq)

## Displaying the DSS for the first 10 sequences
actcal.dss

## Example with with.missing argument
data(ex1)
ex1.seq <- seqdef(ex1[, 1:13])

seqdss(ex1.seq)
seqdss(ex1.seq, with.missing=TRUE)
```

---

seqdur	<i>Extract state durations from a sequence object.</i>
--------	--

---

## Description

Extracts states durations from a sequence object. Returns a matrix containing the states durations for the sequences. The states durations in 'D-D-D-D-A-A-A-A-A-A-D' are 4,7,1. Distinct states can be extracted with the [seqdss](#) function.

## Usage

```
seqdur(seqdata, with.missing=FALSE)
```

## Arguments

`seqdata` a sequence object as defined by the [seqdef](#) function.  
`with.missing` Should non-void missing values be considered as regular states? See Details.

## Details

When `with.missing=FALSE` (default) missing values are ignored and a substring AA\*\*\*A for example will be considered as a spell AAA of duration 3. When `with.missing=TRUE`, durations are also computed for spells of missing values (gaps in sequences).

See [seqdef](#) on options for handling missing values when creating sequence objects.

## Value

a matrix containing the states durations for each distinct state in each sequence.

## Author(s)

Alexis Gabadinho and Gilbert Ritschard

## See Also

[seqdss](#).

## Examples

```
## Creating a sequence object from columns 13 to 24
## in the 'actcal' example data set
## Here we retain the first 10 sequences only.
data(actcal)
actcal.seq <- seqdef(actcal[1:10,13:24])

## Retrieving the spell durations
actcal.dur <- seqdur(actcal.seq)
```

```
## Displaying the durations for the first 10 sequences
actcal.dur
```

---

seqeapplysub

*Checking for the presence of given event subsequences*


---

## Description

Checks occurrences of the subsequences `subseq` among the event sequences and returns the result according to the selected method.

## Usage

```
seqeapplysub(subseq, method = NULL, constraint = NULL,
             rules=FALSE)
```

## Arguments

<code>subseq</code>	list of subsequences (an event subsequence object) such as created by <a href="#">seqefsub</a>
<code>method</code>	type of result, should be one of "count", "presence" or "age"
<code>constraint</code>	Time constraints overriding those used to compute <code>subseq</code> . See <a href="#">sequeconstraint</a>
<code>rules</code>	If set to TRUE, instead of checking occurrences of the subsequences among the event sequences, check the occurrence of the subsequences inside the subsequences (internally used by <code>sequerules</code> )

## Details

There are three methods implemented: "count" counts the number of occurrence of each given subsequence in each event sequence; "presence" returns 1 if the subsequence is present, 0 otherwise; "age" returns the age of appearance of each subsequence in each event sequence. In case of multiple possibilities, the age of the first occurrence is returned. When the subsequence is not in the sequence, -1 is returned.

## Value

The return value is a matrix where each row corresponds to a sequence (row names are set accordingly) and each column corresponds to a subsequence (col names are set accordingly). The cells of the matrix contain the requested values (count, presence-absence indicator or age).

## Author(s)

Matthias Studer and Reto Bürgin (alternative counting methods) (with Gilbert Ritschard for the help page)

## References

Gabardinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. Department of Econometrics and Laboratory of Demography, University of Geneva.

## See Also

[seqecreate](#) for more information on event sequence object and *Gabardinho et al. (2009)* on how to use the event sequence analysis module.

## Examples

```
## Loading data
data(actcal.tse)

## Creating the event sequence object
actcal.eseq <- seqecreate(actcal.tse)

## Printing sequences
actcal.eseq[1:10]

## Looking for frequent subsequences
fsubseq <- seqefsub(actcal.eseq, pmin.support=0.01)

## Counting the number of occurrences of each subsequence
msubcount <- seqeapplysub(fsubseq, method="count")
## First lines...
msubcount[1:10, 1:10]
## Presence-absence of each subsequence
msubpres <- seqeapplysub(fsubseq, method="presence")
## First lines...
msubpres[1:10, 1:10]

## Age at first appearance of each subsequence
msubage <- seqeapplysub(fsubseq, method="age")

## First lines...
msubage[1:10, 1:10]
```

---

seqecmpgroup

*Identifying discriminating subsequences*

---

## Description

Identify and sort the most discriminating subsequences by their discriminating power.

## Usage

```
seqecmpgroup(subseq, group, method="chisq", pvalue.limit=NULL,
             weighted = TRUE)
```

**Arguments**

subseq	A subseqelist object (list of subsequences) such as produced by <a href="#">seqefsub</a>
group	Group membership, i.e., a variable or factor defining the groups which we want to discriminate
method	The discrimination method; one of "bonferroni" or "chisq"
pvalue.limit	Can be used to filter the results. Only subsequences with a p-value lower than this parameter are selected. If NULL all subsequences are returned (regardless of their p-values).
weighted	Logical. If TRUE, seqecmpgroup uses the weights specified in subseq, (see <a href="#">seqefsub</a> ).

**Details**

The following discrimination test functions are implemented: `chisq`, the Pearson Independence Chi-squared test, and `bonferroni`, the Pearson Independence Chi-squared test with Bonferroni correction.

**Value**

An object of type `subseqelistchisq` (subtype of `subseqelist`) with the following elements

subseq	Sorted list of found discriminating subsequences
eseq	The event sequence object on which the tests were computed
constraint	Time constraints used for searching the subsequences (see <a href="#">seqeconstraint</a> )
labels	Levels (value labels) of the target group variable
type	Type of test used
data	A data frame with columns <code>support</code> , <code>index</code> (original rank of the subsequence, i.e., its position in the inputted subseq) and a pair of frequency and Pearson residual columns for each group

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**References**

Studer, M., Müller, N.S., Ritschard, G. & Gabadinho, A. (2010), "Classer, discriminer et visualiser des séquences d'événements", In *Extraction et gestion des connaissances (EGC 2010)*, *Revue des nouvelles technologies de l'information RNTI*. Vol. E-19, pp. 37-48.

Ritschard, G., Bürgin, R., and Studer, M. (2014), "Exploratory Mining of Life Event Histories", In McArdle, J.J. & Ritschard, G. (eds) *Contemporary Issues in Exploratory Data Mining in the Behavioral Sciences*. Series: Quantitative Methodology, pp. 221-253. New York: Routledge.

**See Also**

See also [plot.subseqelistchisq](#) to plot the results

**Examples**

```

data(actcal.tse)
actcal.eseq <- segecreate(actcal.tse)

##Searching for frequent subsequences, that is, appearing at least 20 times
fsubseq <- seqefsub(actcal.eseq, pmin.support=0.01)

##searching for subsequences discriminating the most men and women
data(actcal)
discr <- seqecmpgroup(fsubseq, group=actcal$sex, method="bonferroni")
##Printing the six most discriminating subsequences
print(discr[1:6])
##Plotting the six most discriminating subsequences
plot(discr[1:6])

```

---

segeconstraint

*Setting time constraints and the counting method*


---

**Description**

Function used to set time constraints and the counting method in methods (sege...) for event sequences such as [seqefsub](#) for searching frequent subsequences or [segeapplysub](#) for checking occurrences of subsequences.

**Usage**

```

segeconstraint(max.gap = -1, window.size = -1, age.min = -1, age.max = -1,
  age.max.end = -1, count.method = 1, maxGap, windowSize, ageMin, ageMax,
  ageMaxEnd, countMethod)

```

**Arguments**

max.gap	The maximum time gap between two events
window.size	The maximum time span accepted for subsequences
age.min	Minimal start time position allowed for subsequences. Ignored when equal to -1 (default).
age.max	Maximal start time position allowed for subsequences. Ignored when equal to -1 (default).
age.max.end	Maximal end time position allowed for subsequences. Ignored when equal to -1 (default).
count.method	By default, subsequences are counted only one time by sequence ('COBJ' method). Alternative counting methods are 'CDIST_0', 'CWIN', 'CMINWIN' or 'CDIST' respectively. See details.
maxGap	Deprecated. Use max.gap instead.
windowSize	Deprecated. Use window.size instead.

ageMin	Deprecated. Use age.min instead.
ageMax	Deprecated. Use age.max instead.
ageMaxEnd	Deprecated. Use age.max.end instead.
countMethod	Deprecated. Use count.method instead.

## Details

max.gap, window.size, age.min, age.max and age.max.end. If so, two events should not be separated by more than max.gap and the whole subsequence should not exceed a window.size time span. The other parameters specify the start and end age of the subsequence, it should start between age.min and age.max and finish before age.max.end. Parameters age.min, age.max and age.max.end are interpreted as the number of positions (time units) from the beginning of the sequence.

There are 5 options for the count.method argument. (1) By default, the count is the number of sequences that contain the subsequence ("COBJ" method). Alternatives are (2) "CDIST\_0" (counts all distinct occurrences in each sequence including possibly overlapping occurrences, i.e., occurrences sharing a same event occurrence), (3) "CWIN" (number of slidden windows of length window.size that contain an occurrence of the subsequence), (4) "CMINWIN" (number of minimal windows of occurrence) and (5) "CDIST" (distinct occurrences without event occurrences overlap). See references.

## Value

A constraint object containing one item per constraint type.

## Author(s)

Matthias Studer, Nicolas S. Müller and Reto Bürigin (alternative counting methods) (with Gilbert Ritschard for the help page)

## References

Joshi, Mahesh V., George Karypis, and Vipin Kumar (2001) A Universal Formulation of Sequential Patterns *Proceedings of the KDD'2001 Workshop on Temporal Data Mining*, San Francisco.

Ritschard, G., A. Gabadinho, N.S. Müller and M. Studer (2008), Mining event sequences: A social science perspective, *International Journal of Data Mining, Modelling and Management, IJDMMM*, 1(1), 68-90.

## See Also

[seqfsub](#), [sequeapplysub](#)



---

sequecontain	<i>Check if sequence contains events</i>
--------------	--

---

## Description

Check if an event sequence or subsequence contains given events

## Usage

```
sequecontain(eseq, event.list, unknown.exclude = FALSE,  
             seq, eventList, exclude)
```

## Arguments

eseq	A event sequence object (seqelist) or a an event subsequence object (subseqelist)
event.list	A list of events
unknown.exclude	if TRUE the search is exclusive and returns FALSE for any subsequence containing an event that is not in event.list
seq	Deprecated. Use eseq instead.
eventList	Deprecated. Use event.list instead.
exclude	Deprecated. Use unknown.exclude instead.

## Details

Checks, for each provided event sequence, if it contains one of the events in event.list. If unknown.exclude is TRUE, sequecontain looks if all events of the subsequence are in event.list.

## Value

A logical vector.

## Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

## See Also

[seqcreate](#) for creating event sequence objects and [seqefsub](#) for creating event subsequence objects.

**Examples**

```

data(actcal.tse)
actcal.eseq <- seqcreate(actcal.tse)

##Searching for frequent subsequences, that is appearing at least 20 times
fsubseq <- seqefsub(actcal.eseq,min.support=20)

##looking for subsequence with FullTime
seqecontain(fsubseq,c("FullTime"))

```

---

seqcreate	<i>Create event sequence objects.</i>
-----------	---------------------------------------

---

**Description**

Create an event sequence object either from time stamped events or from a state sequence object.

**Usage**

```

seqcreate(data = NULL, id = NULL,timestamp = NULL, event = NULL,
  end.event = NULL, tevent = "transition", use.labels = TRUE,
  weighted = TRUE, endEvent)

```

**Arguments**

data	A state sequence object (see <a href="#">seqdef</a> ) or a data frame
id	Integer. The sequence 'id' column when data are provided in TSE format (ignored if data argument is provided).
timestamp	Double. The event 'timestamp' column when data are provided in TSE format, i.e., the time at which events occur (ignored if data argument is provided).
event	Character or factor. The 'event' column when data are provided in TSE format, i.e., the events occurring at the specified time stamps (ignored if data argument is provided).
end.event	Character. If specified this event indicates the end of observation time (total length of event sequences) when it is not followed by any other valid event. The event is ignored when occurring in between two valid events.
tevent	Either a transition matrix or a method to generate events from state sequences (see <a href="#">seqetm</a> ). Used only when data is a state sequence object.
use.labels	Logical. If TRUE, transitions names are built from long state labels rather than from the short state names of the alphabet.
weighted	Logical. If TRUE and data is a state sequence object, use the weights specified in data (see <a href="#">seqdef</a> )
endEvent	Deprecated. Use end.event instead.

## Details

There are several ways to create an event sequence object. The first one is by providing the events in TSE format (see [seqformat](#)), i.e. by providing three paired lists: id, timestamp and event, such that each triplet (id, timestamp, event) defines the event that occurs at time timestamp for case id. Several events at the same time for a same id are allowed. The lists can be provided with the arguments id, timestamp and event. An alternative is by providing a data frame as data argument in which case the function takes the required information from the "id", "timestamp" and "event" columns of that data frame. In any case with TSE format, **listed events should be grouped by id** and an error will be thrown otherwise. Such grouping can be achieved by ordering the data according to the id column using the [order](#) function (e.g., `data[order(data$id),]`).

The other way is to pass a state sequence object (as data argument) and to perform an automatic state-to-event conversion. The simplest way to make a conversion is by means of a predefined method (see [seqetm](#)), such as "transition" (one distinct event per possible transition), "state" (a new event for each entering in a new state) and "period" (a pair of events, one start-state event and one end-state event for each found transition). For a more customized conversion, you can specify a transition matrix in the same way as in [seqformat](#). Function [seqetm](#) can help you in creating your transition matrix.

Event sequence objects as created by `seqcreate` are required by most other 'seqe' methods, such as [seqefsub](#) or [seqeapplysub](#) for example.

## Value

An object of class `seqelist`. There are `print`, and `plot` methods for such objects.

## Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

## References

Ritschard, G., Bürgin, R., and Studer, M. (2014), "Exploratory Mining of Life Event Histories", In McArdle, J.J. & Ritschard, G. (eds) *Contemporary Issues in Exploratory Data Mining in the Behavioral Sciences*. Series: Quantitative Methodology, pp. 221-253. New York: Routledge.

Ritschard, G., A. Gabadinho, M. Studer and N. S. Müller. Converting between various sequence representations. in Ras, Z. & Dardzinska, A. (eds.) *Advances in Data Management*, Springer, 2009, 223, 155-175.

## See Also

[seqformat](#) for converting between sequence formats, [seqeweight](#) for retrieving or assigning weights, [seqefsub](#) for searching frequent subsequences, [seqecmpgroup](#) to search for discriminant subsequences, [seqeapplysub](#) for counting subsequence occurrences, [seqelength](#) for information about length (observation time) of event sequences, [seqdef](#) to create a state sequence object.

## Examples

```
##Starting with states sequences
##Loading data
```

```

data(biofam)
## Creating state sequences
biofam.seq <- seqdef(biofam,10:25, informat='STS')
## Creating event sequences from biofam
biofam.eseq <- seqcreate(biofam.seq)

## Loading data
data(actcal.tse)
## Creating sequences
actcal.eseq <- seqcreate(id=actcal.tse$id, timestamp=actcal.tse$time,
event=actcal.tse$event)
##printing sequences
actcal.eseq[1:10]
## Using the data argument
actcal.eseq <- seqcreate(data=actcal.tse)

## Example with missings
data(ex1) ## STS data with missing values

## Creating the state sequence object with by default
## the left missings and gaps coded as '*' and
## end missings coded as void ('%')
sqex1 <- seqdef(ex1[,1:13])
## and without ignoring right missings (coded as '*')
sqex1b <- seqdef(ex1[,1:13], right=NA)

## Compare the outcome
seqcreate(sqex1)
seqcreate(sqex1, tevent='state')
seqcreate(sqex1, tevent='state', end.event=attr(sqex1,'void'))
seqcreate(sqex1b, tevent='state')

```

---

seqefsub

*Searching for frequent subsequences*


---

## Description

Returns the list of subsequences with minimal support sorted in decreasing order of support. Various time constraints can be set to restrict the search to specific time periods or subsequence durations. The function permits also to get information on specified subsequences.

## Usage

```

seqefsub(eseq, str.subseq = NULL, min.support = NULL,
pmin.support = NULL, constraint = seqconstraint(), max.k = -1,
weighted = TRUE, seq, strsubseq, minSupport, pMinSupport, maxK)

```

**Arguments**

<code>eseq</code>	A list of event sequences
<code>str.subseq</code>	A list of specific subsequences to look for. See details.
<code>min.support</code>	The minimum support (in number of sequences)
<code>pmin.support</code>	The minimum support (in percentage, corresponding count will be rounded)
<code>constraint</code>	A time constraint object as returned by <a href="#">sequeconstraint</a>
<code>max.k</code>	The maximum number of events allowed in a subsequence
<code>weighted</code>	Logical. Should seqefsub use the weights specified in eseq (see <a href="#">seqeweight</a> ).
<code>seq</code>	Deprecated. Use <code>eseq</code> instead.
<code>strsubseq</code>	Deprecated. Use <code>str.subseq</code> instead.
<code>minSupport</code>	Deprecated. Use <code>min.support</code> instead.
<code>pMinSupport</code>	Deprecated. Use <code>pmin.support</code> instead.
<code>maxK</code>	Deprecated. Use <code>max.k</code> instead.

**Details**

There are two usages of this function. The first is for searching subsequences satisfying a support condition. By default, the support is counted per sequence and not per occurrence, i.e. when a sequence contains several occurrences of a same subsequence it is counted only once. Use the `count.method` argument of [sequeconstraint](#) to change that. The minimal required support can be set with `pmin.support` as a proportion (between 0 and 1) in which case the support will be rounded, or through `min.support` as a number of sequences. Time constraints can also be imposed with the `constraint` argument, which must be the outcome of a call to the [sequeconstraint](#) function.

The second possibility is for searching sequences that contain specified subsequences. This is done by passing the list of subsequences with the `str.subseq` argument. The subsequences must contain only events from the alphabet of events of `eseq` and must be in the same format as that used to display subsequences (see [str.seqelist](#)). Each transition (group of events) should be enclosed in parentheses () and separated with commas, and the succession of transitions should be denoted by a '-' indicating a time gap. For instance "(FullTime)-(PartTime, Children)" stands for the subsequence "FullTime" followed by the transition defined by the two simultaneously occurring events "PartTime" and "Children".

To get information such as the number of occurrences of the subsequences returned by `seqefsub` or the sequences that contain each subsequence use the function [sequeapplysub](#).

Subsets of the returned `subseqelist` can be accessed with the `[]` operator (see example). There are `print` and `plot` methods for `subseqelist`.

**Value**

A `subseqelist` object with at least the following attributes:

<code>eseq</code>	The list of sequences in which the subsequences were searched (a <code>seqelist</code> event sequence object).
<code>subseq</code>	A list of subsequences (a <code>seqelist</code> event sequence object).
<code>data</code>	A data frame containing details (support, frequency, ...) about the subsequences
<code>constraint</code>	The constraint object used when searching the subsequences.
<code>type</code>	The type of search: 'frequent' or 'user'

**Author(s)**

Matthias Studer and Reto Bürgin (alternative counting methods) (with Gilbert Ritschard for the help page)

**References**

Ritschard, G., Bürgin, R., and Studer, M. (2014), "Exploratory Mining of Life Event Histories", In McArdle, J.J. & Ritschard, G. (eds) *Contemporary Issues in Exploratory Data Mining in the Behavioral Sciences*. Series: Quantitative Methodology, pp. 221-253. New York: Routledge.

**See Also**

See [plot.subseqelist](#) to plot the result. See [seqecreate](#) for creating event sequences. See [seqeapplysub](#) to count the number of occurrences of frequent subsequences in each sequence. See [is.seqelist](#) about seqelist.

**Examples**

```
data(actcal.tse)
actcal.eseq <- seqecreate(actcal.tse)

## Searching for subsequences appearing at least 20 times
fsubseq <- seqefsub(actcal.eseq, min.support=20)
## The same using a percentage
fsubseq <- seqefsub(actcal.eseq, pmin.support=0.01)
## Getting a string representation of subsequences
## First ten most frequent subsequences
fsubseq[1:10]

## Using time constraints
## Looking for subsequences starting in Summer (between June and September)
fsubseq <- seqefsub(actcal.eseq, min.support=10,
  constraint=seqeconstraint(age.min=6, age.max=9))
fsubseq[1:10]

##Looking for subsequences occurring in Summer (between June and September)
fsubseq <- seqefsub(actcal.eseq, min.support = 10,
  constraint=seqeconstraint(age.min=6, age.max=9, age.max.end=9))
fsubseq[1:10]

##Looking for subsequence enclosed in a 6 month period
## and with a maximum gap of 2 month
fsubseq <- seqefsub(actcal.eseq, min.support=10,
  constraint=seqeconstraint(max.gap=2, window.size=6))
fsubseq[1:10]
```

---

seqeid	<i>Retrieve unique ids from an event sequence object.</i>
--------	---

---

**Description**

Retrieve the unique ids from an event sequence object or from a list of event sequence object.

**Usage**

```
seqeid(eseq, s)
```

**Arguments**

eseq	An event sequence object (as created with <a href="#">seqecreate</a> ) or a list of event sequence objects
s	Deprecated. Use eseq instead.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**Examples**

```
data(actcal.tse)
actcal.eseq <- seqecreate(actcal.tse)
sid <- seqeid(actcal.eseq)
length(sid)
head(sid)
```

---

seqlength	<i>Lengths of event sequences</i>
-----------	-----------------------------------

---

**Description**

The length of an event sequence is its time span, i.e., the total time of observation. This information is useful to perform for instance a survival analysis. The function `seqlength` retrieves the lengths of the provided sequences, while `seqlength <-` sets the length of the sequences.

**Usage**

```
seqlength(eseq, s)
seqlength(eseq, s) <- value
```

**Arguments**

<code>eseq</code>	An event sequence object (seqelist).
<code>value</code>	A list of sequence lengths.
<code>s</code>	Deprecated. Use <code>eseq</code> instead.

**Value**

A numeric vector with the lengths of the sequences.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**Examples**

```
data(actcal.tse)
actcal.eseq <- seqecreate(actcal.tse)
## Since end.event is not specified, contains no sequence lengths
## We set them manually as 12 for all sequences
sl <- numeric()
sl[1:2000] <- 12
seqelength(actcal.eseq) <- sl
actcal.eseq[1:10]
## Retrieve lengths
slen <- seqelength(actcal.eseq)
summary(slen)
```

---

seqetm

*Create a transition-definition matrix*

---

**Description**

This function automatically creates a transition-definition matrix from a state sequence object to transform the state sequences into time stamped event sequences (in TSE format).

**Usage**

```
seqetm(seqdata, method = "transition", use.labels = TRUE, sep = ">",
       bp = "", ep = "end", seq)
```

**Arguments**

<code>seqdata</code>	State sequence object from which transition events will be determined
<code>method</code>	The method to use. One of "transition", "period" or "state".
<code>use.labels</code>	If TRUE, transition names are built from state labels rather than from the alphabet.
<code>sep</code>	Separator to be used between the from-state and to-state that define the transition ("transition" method).



bp	Prefix for beginning of period event names ("period" method)
ep	Prefix for end of period event names ("period" method)
seq	Deprecated. Use seqdata instead.

### Details

Warning!!!: State labels should not contain commas ", " which are reserved for separating multiple events of a same transition!

One of three methods can be selected with the method argument:

"transition" generates a single (from-state > to-state) event for each found transition and a distinct start-state event for each different sequence start;

"period" generates a pair of events (end-state-event, start-state-event) for each found transition, a start-state event for the beginning of the sequence and an end-state event for the end of the sequence; names used for end-state and start-state names can be controlled with the bp and ep arguments;

"state" generates only the to-state event of each found transition (useful for analysing state sequences with methods for event sequences);

### Value

The transition-definition matrix.

### Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

### See Also

[seqformat](#) for converting to TSE format, [seqcreate](#) for creating an event sequence object, [seqdef](#) for creating a state sequence object.

### Examples

```
## Creating a state sequence object from columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24,
  labels=c("FullTime", "PartTime", "LowPartTime", "NoWork"))
## Creating a transition matrix, one event per transition
seqetm(actcal.seq,method = "transition")

## Creating a transition matrix, single to-state events
seqetm(actcal.seq,method = "state")

## Creating a transition matrix, two events per transition
seqetm(actcal.seq,method = "period")

## changing the prefix of period start event.
seqetm(actcal.seq,method = "period", bp="begin")
```

seqeweight

*Setting or retrieving weights of an event sequence object.*

---

**Description**

Event sequence objects can be weighted. Weights are used by other functions such as [seqefsub](#) or [seqecmpgroup](#) to compute weighted statistics.

**Usage**

```
seqeweight(eseq, s)
seqeweight(eseq, s) <- value
```

**Arguments**

eseq	An event sequence object (seqelist).
value	Numerical vector containing weights
s	Deprecated. Use eseq instead.

**Value**

seqeweight returns a numerical vector containing the weights associated to each event sequence.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**Examples**

```
##Starting with states sequences
##Loading data
data(biofam)
## Creating state sequences
biofam.seq <- seqdef(biofam,10:25,informat='STS')

## Creating event sequences from biofam
biofam.eseq <- seqcreate(biofam.seq, weighted=FALSE)

## Using the weights
seqeweight(biofam.eseq) <- biofam$w00tbgs

## Now seqefsub accounts for weights unless weighted is set to FALSE
fsubseq <- seqefsub(biofam.eseq, pmin.support=0.01)

## Searching for weighted subsequences which best
## discriminate the birth cohort
discr <- seqecmpgroup(fsubseq, group=biofam$birthyr>=1940)
plot(discr[1:15])
```

---

seqfind	<i>Indexes of state sequence(s) x in state sequence object y</i>
---------	--

---

**Description**

Finds the row indexes of state sequence(s) x in the state sequence object y.

**Usage**

```
seqfind(x, y)
```

**Arguments**

x                    a state sequence object containing one or more sequences ([seqdef](#)).  
y                    a state sequence object.

**Value**

row index(es) of sequence(s) x in the set of sequences y.

**Author(s)**

Alexis Gabadinho (with Gilbert Ritschard for the help page)

**See Also**

.

**Examples**

```
data(mvad)
mvad.shortlab <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, states=mvad.shortlab, 15:86)

## Finding occurrences of sequence 176 in mvad.seq
seqfind(mvad.seq[176,],mvad.seq)

## Finding occurrences of sequence 1 to 8 in mvad.seq
seqfind(mvad.seq[1:8,],mvad.seq)
```

seqformat

*Conversion between sequence formats***Description**

Convert a sequence data set from one format to another.

**Usage**

```
seqformat(data, var = NULL, from, to, compress = FALSE, nrep = NULL, tevent,
  stsep = NULL, covar = NULL, SPS.in = list(xfix = "()", sdsep = ","),
  SPS.out = list(xfix = "()", sdsep = ","), id = 1, begin = 2, end = 3,
  status = 4, process = TRUE, pdata = NULL, pvar = NULL, limit = 100,
  overwrite = TRUE, fillblanks = NULL, tmin = NULL, tmax = NULL, missing = "*",
  with.missing = TRUE, right="DEL", compressed, nr)
```

**Arguments**

data	Data Frame, Matrix, or State Sequence Object. The data to use. A data frame or a matrix with sequence data in one or more columns when from = "STS" or from = "SPS". If sequence data are in a single column, they are assumed to be in the compressed form (see stsep). A data frame with sequence data in one or more columns when from = "SPELL". If sequence data has not four columns ordered as individual ID, spell start time, spell end time, and spell state status, use var or id / begin / end / status. A state sequence object when from = "STS" or from is not specified.
var	NULL, List of Integers or Strings. Default: NULL. The indexes or the names of the columns with the sequence data in data. If NULL, all columns are considered.
from	String. The format of the input sequence data. It can be "STS", "SPS", or "SPELL". It is not needed if data is a state sequence object.
to	String. The format of the output data. It can be "STS", "DSS", "SPS", "SRS", "SPELL", or "TSE".
compress	Logical. Default: FALSE. When to = "STS", to = "DSS", or to = "SPS", should the sequences (row vector of states) be concatenated into strings? See <a href="#">seqconc</a> .
nrep	Integer. The number of shifted replications when to = "SRS".
tevent	Matrix. The transition-definition matrix when to = "TSE". It should be of size $d * d$ where $d$ is the number of distinct states appearing in the sequences. The cell $(i, j)$ lists the events associated with a transition from state $i$ to state $j$ . It can be created with <a href="#">seqetm</a> .
stsep	NULL, Character. Default: NULL. The separator between states in the compressed form (strings) when from = "STS" or from = "SPS". If NULL, <a href="#">seqfcheck</a> is called for detecting automatically a separator among "-" and ":". Other separators must be specified explicitly. See <a href="#">seqdecomp</a> .

covar	List of Integers or Strings. The indexes or the names of additional columns in data to include as covariates in the output when to = "SRS". The covariates are replicated across the shifted replicated rows.
SPS.in	List. Default: <code>list(xfix = "()", sdsep = ",")</code> . The specifications for the state-duration couples in the input data when from = "SPS". The first specification, xfix, specifies the prefix/suffix character. Use a two-character string if the prefix and the suffix differ. Use xfix = "" when no prefix/suffix are present. The second specification, sdsep, specifies the state/duration separator.
SPS.out	List. Default: <code>list(xfix = "()", sdsep = ",")</code> . The specifications for the state-duration couples in the output data when to = "SPS". See SPS.in above.
id	<p>NULL, Integer, String, List of Integers or Strings. Default: 1.</p> <p>When from = "SPELL", the index or the name of the column containing the individual IDs in data (after var filtering).</p> <p>When to = "TSE", the index or the name of the column containing the individual IDs in data (after var filtering) or the unique individual IDs. If id is not manually specified, id is set as NULL for backward compatibility with TraMineR 1.8-13 behaviour. If id is manually or automatically set as NULL, the original individual IDs are ignored and replaced by the indexes of the sequences in the input data.</p> <p>When from = "SPELL" and to = "TSE", the index or the name of the column containing the individual IDs in data (after var filtering). The TSE output will use the original individual IDs.</p>
begin	Integer or String. Default: 2. The index or the name of the column containing the spell start times in data (after var filtering) when from = "SPELL". Start times should be positive integers.
end	Integer or String. Default: 3. The index or the name of the column containing the spell end times in data (after var filtering) when from = "SPELL". End times should be positive integers.
status	Integer or String. Default: 4. The index or the name of the column containing the spell statuses in data (after var filtering) when from = "SPELL".
process	<p>Logical. Default: TRUE. When from = "SPELL", if TRUE, create sequences on a process time axis, if FALSE, create sequences on a calendar time axis.</p> <p>This process argument as well as the associated pdata and pvar arguments are intended for data containing spell data with calendar begin and end times. When those times are ages, use process = FALSE with pdata=NULL to use those ages as process times. Option process = TRUE does currently not work for age times.</p>
pdata	<p>NULL, "auto", or Data Frame. Default: NULL.</p> <p>If NULL, the start and end times of each spell are supposed to be, if process = TRUE, ages, if process = FALSE, years when from = "SPELL".</p> <p>If "auto", ages are computed using the start time of the first spell of each individual as her/his birthdate when from = "SPELL" and process = TRUE. For from = "SPELL" and process = FALSE, "auto" is equivalent to NULL.</p> <p>A data frame containing the ID and the birth time of the individuals when from = "SPELL" or to = "SPELL". Use pvar to specify the column names. The ID is</p>

	used to match the birth time of each individual with the sequence data. The birth time should be integer. It is the start time from which the positions on the time axis are computed. It also serves to compute <code>tmin</code> and to guess <code>tmax</code> when the latter are NULL, <code>from = "SPELL"</code> , and <code>process = FALSE</code> .
<code>pvar</code>	List of Integers or Strings. The indexes or names of the columns of the data frame <code>pdata</code> that contain the ID and the birth time of the individuals in that order.
<code>limit</code>	Integer. Default: 100. The maximum age of age sequences when <code>from = "SPELL"</code> and <code>process = TRUE</code> . Age sequences will be considered to start at 1 and to end at <code>limit</code> .
<code>overwrite</code>	Logical. Default: TRUE. When <code>from = "SPELL"</code> , if TRUE, the most recent episode overwrites the older one when they overlap each other, if FALSE, in case of overlap, the most recent episode starts after the end of the previous one.
<code>fillblanks</code>	Character. The value to fill gaps between episodes when <code>from = "SPELL"</code> .
<code>tmin</code>	NULL or Integer. Default: NULL. The start time of the axis when <code>from = "SPELL"</code> and <code>process = FALSE</code> . If NULL, the value is the minimum of the spell start times (see <code>begin</code> ) or the minimum of the birth time of the individuals (see <code>pdata</code> when it is a data frame and <code>process = FALSE</code> ).
<code>tmax</code>	NULL or Integer. Default: NULL. The end time of the axis when <code>from = "SPELL"</code> and <code>process = FALSE</code> . If NULL, the value is the maximum of the spell end times (see <code>end</code> ) or the sum of the maximum of the spell end times and of the maximum of the birth time of the individuals (see <code>pdata</code> when it is a data frame and <code>process = FALSE</code> ).
<code>missing</code>	String. Default: "*". The code for missing states in data. It will be replaced by NA in the output data. Ignored when data is a state sequence object (see <a href="#">seqdef</a> ), in which case the attribute <code>nr</code> is used as missing value code.
<code>with.missing</code>	Logical. Default: TRUE. When <code>to = "SPELL"</code> , should the spells of missing states be included?
<code>right</code>	One of "DEL" or NA. Default: "DEL". When <code>to = "SPELL"</code> and <code>with.missing=TRUE</code> , set <code>right=NA</code> to include the end spells of missing states.
<code>compressed</code>	Deprecated. Use <code>compress</code> instead.
<code>nr</code>	Deprecated. Use <code>missing</code> instead.

## Details

The `seqformat` function is used to convert data from one format to another. The input data is first converted into the STS format and then converted to the output format. Depending on input and output formats, some information can be lost in the conversion process. The output is a matrix or a data frame, NOT a sequence `stslst` object. To process, print or plot the sequences with TraMineR functions, you will have to first transform the data frame into a `stslst` state sequence object with [seqdef](#). See *Gabadinho et al. (2009)* and *Ritschard et al. (2009)* for more details on longitudinal data formats and converting between them.

When data are in "SPELL" format (`from = "SPELL"`), the `begin` and `end` times are expected to be positions in the sequences. Therefore they should be strictly positive integers. With `process=TRUE`, the outcome sequences will be aligned on ages (process duration since birth), while with `process=FALSE`

they will be aligned on dates (position on the calendar time). If process=TRUE, values in the begin and end columns of data are assumed to be ages when pdata is NULL and integer dates otherwise. If process=FALSE, begin and end values are assumed to be integer dates when pdata is NULL and ages otherwise.

### Value

A data frame for SRS, TSE, and SPELL, a matrix otherwise.

When from="SPELL", outcome has an attribute issues with indexes of sequences with issues (truncated sequences, missing start time, spells before birth year, ...)

### Author(s)

Alexis Gabadinho, Pierre-Alexandre Fonta, Nicolas S. Müller, Matthias Studer, and Gilbert Ritschard.

### References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. Department of Econometrics and Laboratory of Demography, University of Geneva.

Ritschard, G., A. Gabadinho, M. Studer and N. S. Müller. Converting between various sequence representations. in Ras, Z. & Dardzinska, A. (eds.) *Advances in Data Management*, Springer, 2009, 223, 155-175.

### See Also

[seqdef](#)

### Examples

```
## =====
## Examples with raw STS sequences as input
## =====

## Loading a data frame with sequence data in the columns 13 to 24
data(actcal)

## Converting to SPS format
actcal.SPS.A <- seqformat(actcal, 13:24, from = "STS", to = "SPS")
head(actcal.SPS.A)

## Converting to compressed SPS format with no
## prefix/suffix and with "/" as state/duration separator
actcal.SPS.B <- seqformat(actcal, 13:24, from = "STS", to = "SPS",
  compress = TRUE, SPS.out = list(xfix = "", sdsep = "/"))
head(actcal.SPS.B)

## Converting to compressed DSS format
actcal.DSS <- seqformat(actcal, 13:24, from = "STS", to = "DSS",
  compress = TRUE)
head(actcal.DSS)
```

```

## =====
## Examples with a state sequence object as input
## =====

## Loading a data frame with sequence data in the columns 10 to 25
data(biofam)

## Limiting the number of considered cases to the first 20
biofam <- biofam[1:20, ]

## Creating a state sequence object
biofam.labs <- c("Parent", "Left", "Married", "Left/Married",
  "Child", "Left/Child", "Left/Married/Child", "Divorced")
biofam.short.labs <- c("P", "L", "M", "LM", "C", "LC", "LMC", "D")
biofam.seq <- seqdef(biofam, 10:25, alphabet = 0:7,
  states = biofam.short.labs, labels = biofam.labs)

## Converting to SPELL format
bf.spell <- seqformat(biofam.seq, from = "STS", to = "SPELL",
  pdata = biofam, pvar = c("idhous", "birthyr"))
head(bf.spell)

## =====
## Examples with SPELL sequences as input
## =====

## Loading two data frames: bfpell20 and bfpdata20
## bfpell20 contains the first 20 biofam sequences in SPELL format
## bfpdata20 contains the IDs and the years at which the
## considered individuals were aged 15
data(bfpell)

## Converting to STS format with alignment on calendar years
bf.sts.y <- seqformat(bfpell20, from = "SPELL", to = "STS",
  id = "id", begin = "begin", end = "end", status = "states",
  process = FALSE)
head(bf.sts.y)

## Converting to STS format with alignment on ages
bf.sts.a <- seqformat(bfpell20, from = "SPELL", to = "STS",
  id = "id", begin = "begin", end = "end", status = "states",
  process = TRUE, pdata = bfpdata20, pvar = c("id", "when15"),
  limit = 16)
names(bf.sts.a) <- paste0("a", 15:30)
head(bf.sts.a)

## =====
## Examples for TSE and SPELL output
## in presence of missing values

```



```
## =====

data(ex1) ## STS data with missing values
## creating the state sequence object with by default
## the end missings coded as void ('%')
sqex1 <- seqdef(ex1[,1:13])
as.matrix(sqex1)

## Creating state-event transition matrices
ttrans <- seqetm(sqex1, method='transition')
tstate <- seqetm(sqex1, method='state')

## Converting into time stamped events
seqformat(sqex1, from = "STS", to = "TSE", tevent = ttrans)
seqformat(sqex1, from = "STS", to = "TSE", tevent = tstate)

## Converting into vertical spell data
seqformat(sqex1, from = "STS", to = "SPELL", with.missing=TRUE)
seqformat(sqex1, from = "STS", to = "SPELL", with.missing=TRUE, right=NA)
seqformat(sqex1, from = "STS", to = "SPELL", with.missing=FALSE)
```

---

seqfpos

*Search for the first occurrence of a given element in a sequence*


---

### Description

Returns a vector containing the position of the first occurrence of the given element in each of the sequences in the data set.

### Usage

```
seqfpos(seqdata, state)
```

### Arguments

seqdata	a sequence object (see <a href="#">seqdef</a> function).
state	the state element to search in the sequences

### Details

the state to search for has to be passed as a character string, and must be one of the state returned by the [alphabet](#) function. If the state is not contained in a sequence, NA is returned for this sequence.

### Author(s)

Alexis Gabadinho

### Examples

```
data(biofam)
biofam.seq <- seqdef(biofam,10:25)

## Searching for the first occurrence of state 1
## in each of the 5 first sequence of the biofam data set.
seqfpos(biofam.seq[1:5,],"1")
```

---

seqgen

*Random sequences generation*

---

### Description

Generates random sequences.

### Usage

```
seqgen(n, length, alphabet, p)
```

### Arguments

n	number of sequences to generate
length	sequences length
alphabet	the alphabet from which the sequences are generated
p	an optional vector of probabilities for the states in the alphabet. Must be of the same length as the alphabet. If not specified, equal probabilities are used.

### Details

Each sequence is generated by choosing a set of random numbers (with min=1 and max=length of the alphabet) using the `runif` function. When the probability distribution is not specified, the uniform probability distribution giving same probability to each state is used to generate the sequences.

### Value

a sequence object.

### Author(s)

Alexis Gabadinho (with Gilbert Ritschard for the help page)

### Examples

```
seq <- seqgen(1000,10,1:4,c(0.2,0.1,0.3,0.4))
seqstatd(seqdef(seq))
```

---

seqibad	<i>Badness index</i>
---------	----------------------

---

### Description

Badness index of each sequence, i.e. the sum of undesirableness of each state weighted by the potential to integrate that state in the sequence.

### Usage

```
seqibad(seqdata, pow=1, with.missing=FALSE, ...)
```

### Arguments

seqdata	a state sequence object (stslist) as returned by <a href="#">seqdef</a> .
pow	real. Exponent applied to the position in the sequence. Higher value increase the importance of recency (see <a href="#">seqintegration</a> ). Default is 1.
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.
...	arguments such as <code>stprec</code> or <code>state.order</code> required by <a href="#">seqprecstart</a> to determine/normalize the state undesirableness degrees.

### Details

For each sequence, the badness is the sum of the undesirableness of each state weighted by the potential to integrate the state. As long as `pow` is strictly greater than zero, the undesirableness of states occurring at the end of the sequence get higher weights than those at the beginning. The index reaches its maximum 1 for a sequence made of a single spell in the worst state and the minimum 0 for a sequence made of a single spell in the most favorable state.

### Value

A vector with the badness index for each sequence.

### Author(s)

Gilbert Ritschard

### References

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

### See Also

[seqintegr](#), [seqidegrad](#), [seqprecarity](#)

**Examples**

```

data(ex1)
sx <- seqdef(ex1[,1:13], right="DEL")

seqibad(sx) ## using original alphabet order
seqibad(sx, stprec=c(1,2,3,6)) ## user defined undesirableness values
seqibad(sx, with.missing=TRUE, state.order=c('A','B','C','D'))

```

seqici

*Complexity index of individual sequences***Description**

Computes the complexity index, a composite measure of sequence complexity. The index uses the number of transitions in the sequence as a measure of the complexity induced by the state ordering and the longitudinal entropy as a measure of the complexity induced by the state distribution in the sequence.

**Usage**

```
seqici(seqdata, with.missing=FALSE, silent=TRUE)
```

**Arguments**

seqdata	a sequence object as returned by the the <a href="#">seqdef</a> function.
with.missing	if set to TRUE, missing status (gaps in sequences) is handled as an additional state when computing the state distribution and the number of transitions in the sequence.
silent	logical: should messages about running operations be displayed?

**Details**

The *complexity index*  $C(s)$  of a sequence  $s$  is

$$C(s) = \sqrt{\frac{q(s)}{q_{max}} \frac{h(s)}{h_{max}}}$$

where  $q(s)$  is the number of transitions in the sequence,  $q_{max}$  the maximum number of transitions,  $h(s)$  the within entropy, and  $h_{max}$  the theoretical maximum entropy which is  $h_{max} = -\log 1/|A|$  with  $|A|$  the size of the alphabet.

The index  $C(s)$  is the geometric mean of its two normalized components and is, therefore, itself normalized. The minimum value of 0 can only be reached by a sequence made of one distinct state, thus containing 0 transitions and having an entropy of 0. The maximum 1 of  $C(s)$  is reached when the two following conditions are fulfilled: i) Each of the state in the alphabet is present in the sequence, and the total durations are uniform, i.e. each state occurs  $\ell/|A|$  times, and ii) the number of transitions in the sequence is  $\ell - 1$ , meaning that the length  $\ell_d$  of the DSS is equal to the length of the sequence  $\ell$ .

**Value**

a single-column matrix of length equal to the number of sequences in seqdata containing the complexity index value of each sequence.

**Author(s)**

Alexis Gabadinho (with Gilbert Ritschard for the help page)

**References**

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

Gabadinho, A., Ritschard, G., Studer, M. and Müller, N.S. (2010). "Indice de complexité pour le tri et la comparaison de séquences catégorielles", In *Extraction et gestion des connaissances (EGC 2010)*, *Revue des nouvelles technologies de l'information RNTI*. Vol. E-19, pp. 61-66.

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

**See Also**

[seqindic](#), [seqient](#), [seqipos](#). For alternative measures of sequence complexity see [seqST](#), [seqivolatility](#).

**Examples**

```
## Creating a sequence object from the mvad data set
data(mvad)
mvad.labels <- c("employment", "further education", "higher education",
               "joblessness", "school", "training")
mvad.scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, 15:86, states=mvad.scodes, labels=mvad.labels)

##
mvad.ci <- seqici(mvad.seq)
summary(mvad.ci)
hist(mvad.ci)

## Example using with.missing argument
data(ex1)
ex1.seq <- seqdef(ex1, 1:13)
seqici(ex1.seq)
seqici(ex1.seq, with.missing=TRUE)
```

---

seqient	<i>Within sequence entropies</i>
---------	----------------------------------

---

**Description**

Computes normalized or non-normalized within sequence entropies

**Usage**

```
seqient(seqdata, norm=TRUE, base=exp(1), with.missing=FALSE, silent=TRUE)
```

**Arguments**

seqdata	a sequence object as returned by the the <a href="#">seqdef</a> function.
norm	logical: should the entropy be normalized? TRUE by default. (see details)
base	real positive value: base of the logarithm used in the entropy formula (see details). Default is $\exp(1)$ , i.e., the natural logarithm is used.
with.missing	logical: if TRUE, the missing state (gap in sequences) is handled as an additional state when computing the state distribution in the sequence.
silent	logical: should messages about running operations be displayed?

**Details**

The seqient function returns the Shannon entropy of each sequence in seqdata. The entropy of a sequence is computed using the formula

$$h(\pi_1, \dots, \pi_s) = - \sum_{i=1}^s \pi_i \log \pi_i$$

where  $s$  is the size of the alphabet and  $\pi_i$  the proportion of occurrences of the  $i$ th state in the considered sequence. The base of the log is controlled with the base argument. By default the natural logarithm, i.e. the logarithm in base  $e$ , is used. The entropy can be interpreted as the ‘uncertainty’ of predicting the states in a given sequence. If all states in the sequence are the same, the entropy is equal to 0. For example, the maximum entropy for a sequence of length 12 with an alphabet of 4 states is 1.386294 and is attained when each of the four states appears 3 times.

Normalization can be requested with the norm=TRUE option, in which case the returned value is the entropy divided by the entropy of the alphabet. The latter is an upper bound for the entropy of sequences made from this alphabet. It is exactly the maximal entropy when the sequence length is a multiple of the alphabet size. The value of the normalized entropy is independent of the chosen logarithm base.

**Value**

a single-column matrix with an entropy value for each sequence in seqdata; the column length is equal to the number of sequences.

**Author(s)**

Alexis Gabadinho

**References**

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

**See Also**

[seqindic](#), [seqici](#), [seqST](#), and [seqstatd](#) for the entropy of the cross-sectional state distributions by positions in the sequence.

**Examples**

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Summarize and plots an histogram
## of the within sequence entropy
actcal.ient <- seqient(actcal.seq)
summary(actcal.ient)
hist(actcal.ient)

## Examples using with.missing argument
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

seqient(ex1.seq)
seqient(ex1.seq, with.missing=TRUE)
```

---

seqindic

*Sequence indicators*

---

**Description**

Table of per sequence values of selected indicators.

**Usage**

```
seqindic(seqdata, indic=c("visited","trans","entr","cplx","turb2n"), with.missing=FALSE,
         ipos.args=list(), prec.args=list(), w=.5)
```

## Arguments

seqdata	a state sequence object (class <code>stslst</code> ) as returned by <a href="#">seqdef</a> .
indic	vector of character strings. List of selected indicators among "lgth" (sequence length), "norm" (number of non-missing elements), "dlgth" (number of spells), "visited" (number of visited states), "visitp" (proportion of states visited), "trans" (number of state changes), "transp" (number of state changes as a proportion of maximum possible transitions), "meand" (mean spell duration), "dustd" (duration standard deviation), "meand2" (mean spell duration taking non-visited states into account), "dustd2" (duration standard deviation taking non-visited states into account), "entr" (longitudinal normalized entropy), "nsubs" (number of subsequences of the DSS sequence), "volat" (objective volatility), "cplx" (complexity index), "turb" (turbulence), "turbn" (normalized turbulence), "turb2" (turbulence taking non-visited states into account), "turb2n" (normalized turbulence taking non-visited states into account), "ppos" (proportion of positive states), "nvolat" (normative volatility), "vpos" objective volatility of positive-negative state sequences, "integr" (integrative potential), "degrad" (degradation), "bad" (badness), "prec" (precarity index), "insec" (insecurity). Can also be "all", "basic", "diversity", "complexity", "binary" or "ranked" that each selects a subset of the indicators. Setting <code>indic="all"</code> selects all indicators but "ppos", "vpos", "nvolat", "integr", "degrad", "bad", "prec", and "insec" that require an additional argument list.
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.
ipos.args	list: when any of "ppos", "nvolat", "vpos", or "integr" is requested, the arguments passed to <code>seqipos</code> together with <code>seqdata</code> and <code>with.missing</code> . See <a href="#">seqipos</a>
prec.args	list: when any of "degrad", "bad", "prec", or "insec" is requested, the arguments passed to the underlying functions together with <code>seqdata</code> and <code>with.missing</code> . See <a href="#">seqprecarity</a> and <code>link{seqibad}</code> .
w	real in range [0,1]: when <code>volat</code> weight coefficient passed to <a href="#">seqivolatility</a> . Default is .5.

## Details

The number of visited states is the number of different elements in the sequence, e.g. 2 for aababba. The recurrence index 'recu' is the average number of visits of visited states, i.e.  $Dlgh/Visited$ , the number of spells on the number of visited states.

The sequence length, number of transitions, longitudinal entropy, duration standard deviation, volatility, complexity, turbulence, degradation, badness, precarity, and insecurity are computed respectively with functions [seqlength](#), [seqtransn](#), [seqient](#), [seqivardur](#), [seqivolatility](#), [seqici](#), [seqST](#), [seqidegrad](#), [seqibad](#), [seqprecarity](#), and [seqinsecurity](#). The proportion of positive states, normative volatility, and integrative potential are computed with [seqipos](#). See corresponding help pages for details.

The proportion of positive states ('ppos') and the normative volatility ('nvolat') are the proportions of positive elements in respectively the original sequences and the DSS. They ignore the value of dss in the `ipos.args` list.



The `with.missing` argument applies to all indicators but the length. `'lgth'` returns the length obtained with `with.missing=TRUE`, and `'nonm'` the length obtained with `with.missing=FALSE`.

### Value

A data frame with the selected indicators. Names are:

`Lght`: Length of the sequence

`NonM`: Number of non-missing elements

`Dlght`: Number of spells (length of DSS)

`Visited`: Number of visited states

`Visitp`: Proportion of states visited

`Recu`: Recurrence: average number of visits to visited states

`Trans`: Number of transitions (state changes)

`Transp`: Number of state changes as a proportion of maximum number of transitions

`Entr`: Longitudinal entropy

`Meand`: Mean spell duration

`Dustd`: Duration standard deviation

`Meand2`: Mean spell duration taking non visited states into account

`Dustd2`: Duration standard deviation taking non visited states into account

`Nsubs`: Number of subsequences of the DSS sequence

`Volat`: Objective volatility

`Cplx`: Complexity

`Turb`: Turbulence

`Turbn`: Normalized turbulence

`Turb2`: Turbulence taking non visited states into account

`Turbn2`: Normalized turbulence taking non visited states into account

`Ppos`: Proportion of positive states

`Nvolat`: Normative volatility (proportion of positive spells)

`Vpos`: Objective volatility of positive-negative state sequences

`Integr`: Integrative capacity (potential)

`Degrad`: Degradation

`Bad`: Badness

`Prec`: Precarity

`Insec`: Insecurity

### Author(s)

Gilbert Ritschard

### References

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

### See Also

[seqlength](#), [seqtransn](#), [seqient](#), [seqivardur](#), [seqivolatility](#), [seqici](#), [seqST](#), [seqidegrad](#), [seqibad](#), [seqprecarity](#), [seqinsecurity](#), [seqipos](#).

## Examples

```
data(ex1)
sx <- seqdef(ex1[,1:13], right="DEL")
print(sx, format='SPS')
seqindic(sx, indic=c("lgth", "nonm", "visited", "turbn", "cplx"))
seqindic(sx, indic=c("lgth", "nonm", "visited", "turbn", "cplx"), with.missing=TRUE)
seqindic(sx, indic=c("lgth", "dlgth", "ppos", "integr", "prec"), with.missing=TRUE,
         ipos.args=list(pos.states=c("A", "B")),
         prec.args=list(state.order=c("A", "B", "C"), state.equiv=list(c("C", "D"))))
seqindic(sx, indic=c("volat", "binary"), ipos.args=list(pos.states=c("A", "B")))
seqindic(sx, indic=c("basic", "integr"), ipos.args=list(pos.states="D"))
```

---

seqintegr

*Integrative potential*


---

## Description

Returns the index of integrative potential (capability) for each sequence, either a table with the index for each state or a vector with the index for the selected state.

## Usage

```
seqintegr(seqdata, state=NULL, pow=1, with.missing=FALSE)
```

## Arguments

seqdata	a state sequence object (stslst) as returned by <a href="#">seqdef</a> .
state	character string. The state for which to compute the integrative index (see Details). When NULL the index is computed for each state.
pow	real. Exponent applied to the position in the sequence. Higher value increase the importance of recency (see Details). Default is 1.
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.

## Details

The index of integrative potential or capability (*Brzinsky-Fay, 2007, 2018*) measures the capacity to integrate the selected state within the sequence, i.e. the tendency to reach the selected state and end up in it. The index is defined as the sum of the position numbers occupied by the selected state in the sequence over the sum of all position numbers. Formally, for a sequence  $s$  of length  $L$ , and numbering the positions  $i$  from 1 to  $L$ , the index is

$$integr = \sum_{(i|s_i=state)} i^{pow} / \sum_i i^{pow}$$

where *state* is the selected state. This same index has also been independently developed by *Manzoni and Mooi-Reci (2018)* under the name of quality index.

The recency exponent *pow* permits to control the focus given on the latest positions in the sequence. The higher *pow*, the higher the importance of the last positions relative to the first ones.

When `with.missing = FALSE`, the index is obtained by using the sum of the positions numbers of the non-missing elements as denominator. To compute the index for the missing state, `with.missing` should be set as `TRUE`.

For capability to integrate a set of states see [seqipos](#).

### Value

when `state=NULL`, a numeric matrix with a row for each sequence and a column by state. When a state is provides, a single column.

### Author(s)

Gilbert Ritschard

### References

Brzinsky-Fay, C. (2007) Lost in Transition? Labour Market Entry Sequences of School Leavers in Europe, *European Sociological Review*, 23(4). doi: [10.1093/esr/jcm011](https://doi.org/10.1093/esr/jcm011)

Brzinsky-Fay, C. (2018) Unused Resources: Sequence and Trajectory Indicators. International Symposium on Sequence Analysis and Related Methods, Monte Verita, TI, Switzerland, October 10-12, 2018.

Manzoni, A and I. Mooi-Reci (2018) Measuring Sequence Quality, in Ritschard and Studer (eds), *Sequence Analysis and Related Approaches. Innovative Methods and Applications*, Springer, 2018, pp 261-278.

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

### See Also

[seqipos](#), [seqivolatility](#), [seqindic](#)

### Examples

```
data(ex1)
sx <- seqdef(ex1[,1:13], right="DEL")

seqintegr(sx)
seqintegr(sx, with.missing=TRUE)
seqintegr(sx, state="B")
seqintegr(sx, state="B", pow=1.5)
```

seqipos

*Indicators for binary sequences of positive and negative states.***Description**

Indicators for the underlying binary sequences of positive and negative states. Possible indicators are the proportion of positive states within each sequence, i.e. of positive spells if computed on the sequences of distinct successive states (DSS), objective volatility of the binary sequences, and capacity to integrate a positive state.

**Usage**

```
seqipos(seqdata, dss=NULL, pos.states=NULL, neg.states=NULL, index="share",
        pow=1, w=.5, with.missing=FALSE)
```

**Arguments**

seqdata	a state sequence object (stslist) as returned by <a href="#">seqdef</a> .
dss	logical. Should the proportion be computed inside the DSS sequences? Defaults to TRUE when index="share" and to FALSE otherwise.
pos.states	vector of positive states.
neg.states	vector of negative states.
index	character string. One of "share" (proportion of positive states or spells), "volatility" (objective volatility), or "integr" (capability to integrate a positive state). See Details. Default is "share".
pow	real. Recency exponent passed to <a href="#">seqintegr</a> . Only used when index="integr". Default is 1.
w	real in range [0,1]. Relative weight of proportion of visited states in volatility (see <a href="#">seqivolatility</a> ). Only used when index="volatility". Default is .5.
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.

**Details**

The function transforms the provided state sequences into binary sequences of positive and negative states. When dss = TRUE, the counts of positive and negative elements give the number of positive and negative spells. The binary state sequence object is returned as an attribute.

When neg.states=NULL, states not listed on the pos.states argument are considered as negative and conversely when pos.states=NULL. When with.missing=TRUE, the missing state nr will be considered as positive if included in the pos.states list or as negative if in neg.states. When with.missing=FALSE (default) missing states are ignored. However, when missing values appear within a spell, such as in AA\*ABB, the spell is split into two consecutive spells in a same state. For the example we would have AAB, i.e. if A is positive and B negative, a proportion of 2/3 of positive spells.

When both `pos.states` and `neg.states` are provided, states of the alphabet (including the `nr` code when `with.missing=TRUE`) that belong nor to `pos.states` nor to `neg.states` are ignored.

For `index="share"`, letting  $n_{pos}$  be the number of positive states and  $n_{neg}$  the number of negative states in the sequence, the function returns the value of  $n_{pos}/(n_{pos} + n_{neg})$  for each sequence. With `dss=TRUE`, this is the normative volatility of Brzinsky-Fay (2007,2018).

For `index="volatility"`, the function returns the objective volatility of the binary sequences. See [seqivolatility](#).

For `index="integr"`, the function returns the index of integrative potential for the positive state. See [seqintegr](#).

### Value

A numeric vector of

- when `index="share"`, the proportion of positive states (or spells) within each sequence;
- when `index="volatility"`, the objective volatility of the binary sequences;
- when `index="integration"`, the index of integration into a positive state.

The binary sequence as an attribute `sbinary`.

### Author(s)

Gilbert Ritschard

### References

Brzinsky-Fay, C. (2007) Lost in Transition? Labour Market Entry Sequences of School Leavers in Europe, *European Sociological Review*, 23(4). doi: [10.1093/esr/jcm011](https://doi.org/10.1093/esr/jcm011)

Brzinsky-Fay, C. (2018) Unused Resources: Sequence and Trajectory Indicators. International Symposium on Sequence Analysis and Related Methods, Monte Verita, TI, Switzerland, October 10-12, 2018.

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

### See Also

[seqindic](#)

### Examples

```
data(ex1)
sx <- seqdef(ex1[,1:13], right="DEL")
nr <- attr(sx,'nr') ## code for missing values

seqipos(sx, pos.states=c("A","B"))
seqipos(sx, pos.states=c("A","B"), with.missing=TRUE)

## ignoring state "D"
seqipos(sx, pos.states=c("A","B"), neg.states=c("C",nr), with.missing=TRUE)

seqipos(sx, pos.states=c("A","B"), dss=FALSE)
```

```

seqipos(sx, pos.states=c("A","B",nr), dss=FALSE, with.missing=TRUE)

seqipos(sx, pos.states=c("A","B"), index="volatility")
seqipos(sx, pos.states=c("A","B"), index="integr")

## retrieving the binary sequences
ip <- seqipos(sx, pos.states=c("A","B"))
attr(ip,"sbinary")
ip <- seqipos(sx, pos.states=c("A","B"), with.missing=TRUE)
attr(ip,"sbinary")

```

---

seqistatd

*State frequencies in each individual sequence*


---

### Description

Returns the state frequencies (total durations) for each sequence in the sequence object.

### Usage

```
seqistatd(seqdata, with.missing=FALSE, prop=FALSE)
```

### Arguments

seqdata	a sequence object (see <a href="#">seqdef</a> function).
with.missing	logical: if set as TRUE, total durations are also computed for the missing status (gaps in the sequences). See <a href="#">seqdef</a> on options for handling missing values when creating sequence objects.
prop	logical: if TRUE, proportions of time spent in each state are returned instead of absolute values. This option is specially useful when sequences contain missing states, since the sum of the state durations may not be the same for all sequences.

### Author(s)

Alexis Gabadinho

### References

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

**Examples**

```

data(actcal)
actcal.seq <- seqdef(actcal,13:24)
seqistatd(actcal.seq[1:10,])

## Example using "with.missing" argument
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

seqistatd(ex1.seq)
seqistatd(ex1.seq, with.missing=TRUE)

```

---

seqivardur

*Variance of spell durations in individual sequences*


---

**Description**

Variance of spell durations of individual state sequences.

**Usage**

```
seqivardur(seqdata, type=1, with.missing=FALSE)
```

```
## S3 method for class 'seqivardur'
print(x, stat='var', ...)
```

**Arguments**

seqdata	a state sequence object (stslist) as returned by <a href="#">seqdef</a> .
type	either 1 or 2. The default type=1 ignores non visited states. Type 2 takes into account the 0-time spent in non-visited states (see <a href="#">Details</a> ).
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.
x	an outcome of seqivardur.
stat	string or vector of strings. Duration statistic to be printed. Either 'mean' (mean duration), 'std' (standard deviation), 'var' (variance), 'vmax' (maximum variance for number of spells), or 'all'. Default is 'var'.
...	further arguments such as <code>digits</code> passed to the next <a href="#">print</a> method.

**Details**

The returned variance is a population variance, i.e. the sum of squares divided by the number of terms.

When type=1, this is the variance of the observed spells in the sequence. When type=2, in addition to the observed spells one spell of length 0 is considered for each non-visited state. The mean duration is computed the same way.

**Value**

A numeric vector with the variance of spell duration of each sequence.

The returned object has two attributes:

vmax	the maximum value that the variance of each sequence could reach given its number of spells.
meand	the mean spell duration of each sequence.

**Author(s)**

Gilbert Ritschard

**References**

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/004912412111036156](https://doi.org/10.1177/004912412111036156).

**See Also**

[seqdur](#), [seqindic](#)

**Examples**

```
sq.dat <- c('AAAA', 'AAAC', 'ABC', 'ABAA', 'AC')
sq <- seqdef(seqdecomp(sq.dat, sep=''), right=NA)

seqivardur(sq, type=1)
seqivardur(sq, type=1, with.missing=TRUE)
vd2 <- seqivardur(sq, type=2)
print(vd2, stat='all')
vd2m <- seqivardur(sq, type=2, with.missing=TRUE)
print(vd2m, stat=c('var', 'vmax'))
```

---

seqivolatility

*Volatility of individual state sequences*

---

**Description**

Returns Brzinsky-Fay's objective volatility of each sequence.

**Usage**

```
seqivolatility(seqdata, w=.5, with.missing=FALSE, adjust=TRUE)
```



**Arguments**

seqdata	a state sequence object (stslist) as returned by <a href="#">seqdef</a> .
adjust	Logical. Should the indicator be adjusted such that it can reach its bounds 0 and 1. Default is TRUE (see Details).
w	Real in the range [0, 1]. Default is 0.5. Weight given to the proportion of states visited (see Details).
with.missing	Logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.

**Details**

The (objective) volatility is the weighted average between the proportion *pvisited* of states visited and the frequency *ftrans* of transitions (state changes). Formally,

$$volatility = w \cdot pvisited + (1 - w) \cdot ftrans$$

The proportion of states visited is computed as  $(visited - 1) / (|a| - 1)$  when `adjust=TRUE` and as  $visited / |a|$  when `adjust=FALSE`. Here, *visited* is the number of states visited and  $|a|$  the size of the alphabet.

The frequency of transition is  $ftrans = \frac{transn}{max.transn}$  where *transn* is the number of transitions (state changes) within the sequence, and *max.transn* the maximum possible transitions in the sequence.

For the normative volatility, see [seqipos](#). For alternative measures of sequence complexity see [seqST](#), [seqici](#), [seqindic](#).

**Value**

A numeric vector with the volatility of each sequence.

**Author(s)**

Gilbert Ritschard

**References**

- Brzinsky-Fay, C. Unused Resources: Sequence and Trajectory Indicators. International Symposium on Sequence Analysis and Related Methods, Monte Verita, TI, Switzerland, Oct 10-11, 2018
- Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

**See Also**

[seqintegr](#), [seqipos](#), [seqindic](#)

**Examples**

```

data(ex1)
sx <- seqdef(ex1[,1:13], right="DEL")

seqvolatility(sx)
seqvolatility(sx, adjust=FALSE)
seqvolatility(sx, with.missing=TRUE)
seqvolatility(sx, w=.7, with.missing=TRUE)

```

seqlegend

*Plot a legend for the states in a sequence object***Description**

Plots a legend for the states in a sequence object. Useful if several graphics are plotted together and only one legend is necessary. Unless specified by the user, the *cpal* and *labels* attributes of the sequence object are used for the colors and text appearing in the legend (see [seqdef](#)).

**Usage**

```

seqlegend(seqdata, with.missing = "auto", cpal = NULL, missing.color = NULL,
  ltext = NULL, position = "topleft", cex = 1, boxes=TRUE, fontsize, ...)

```

**Arguments**

<code>seqdata</code>	a state sequence object (of class <code>stslst</code> ) as returned by the <a href="#">seqdef</a> function.
<code>with.missing</code>	logical: if set to "auto" (default), a legend for the missing state is added automatically if one or more of the sequences in <code>seqdata</code> contains a missing state. If TRUE a legend for the missing state is added in any case. Setting to FALSE omits the legend for the missing state.
<code>cpal</code>	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of distinct states. By default, the 'cpal' attribute of the 'seqdata' sequence object is used (see <a href="#">seqdef</a> ).
<code>missing.color</code>	alternative color for representing missing values inside the sequences. By default, this color is taken from the "missing.color" attribute of the sequence object being plotted.
<code>ltext</code>	optional description of the states to appear in the legend. Must be a vector of character strings with number of elements equal to the number of distinct states. If unspecified, the 'labels' attributes of the 'seqdata' sequence object is used (see <a href="#">seqdef</a> ).
<code>position</code>	the position of the legend in the graphic area. For accepted values, see the <code>x</code> argument of <a href="#">legend</a> . Defaults to "topleft".
<code>cex</code>	size of the font for the labels. A value less than 1 decreases the font size, a value greater than 1 increases the font size. Defaults to 1.

boxes	logical: should the colors be displayed in small square boxes? Default is TRUE. If FALSE, colors are shown using small line segments of by default length .4 and width 15. The latter can be changed by means of the lwd and seg.len <a href="#">par</a> arguments.
fontsize	Deprecated. Use cex instead.
...	optional arguments passed to the <a href="#">legend</a> function.

**Details**

When x, legend, or col arguments are provided, they supersede respectively the position, ltext, and cpal values.

**Value**

Coordinate values returned by the [legend](#) function.

**Author(s)**

Alexis Gabadinho and Gilbert Ritschard

**Examples**

```
## Loading the 'actcal' example data set
## and defining a sequence object with
## activity statuses from Jan. to Dec. 2000 (columns 13 to 24)
## of first 100 cases.
data(actcal)
actcal.seq <- seqdef(actcal[1:100,13:24],
  labels=c("> 37 hours", "19-36 hours", "1-18 hours", "no work"))

## Plotting the sequences frequency,
## the states distribution
## and the legend
par(mfrow=c(2,2))
seqIplot(actcal.seq, sortv="from.start", with.legend=FALSE)
seqfplot(actcal.seq, pbarw=TRUE, with.legend=FALSE)
seqdplot(actcal.seq, with.legend=FALSE)
seqlegend(actcal.seq)
```

---

seqlength

*Sequence length*

---

**Description**

Returns the length of sequences.

**Usage**

```
seqlength(seqdata, with.missing=TRUE)
```

**Arguments**

seqdata            a sequence object created with the `seqdef` function.  
 with.missing      logical: should non-void missing values be treated as a regular state? Default is TRUE. If FALSE missing values are considered as void.

**Details**

The length of a sequence is computed by counting its number of non void elements, i.e. including non-void missing values. The `seqlength` function returns a vector containing the length of each sequence in the provided sequence object.

**Author(s)**

Alexis Gabadinho and Gilbert Ritschard

**Examples**

```
## Loading the 'famform' example data set
data(famform)

## Defining a sequence object with the 'famform' data set
ff.seq <- seqdef(famform)

## Retrieving the length of the first 10 sequences
## in the ff.seq sequence object
seqlength(ff.seq)
```

---

 seqLLCP

---

*Compute the length of the longest common prefix of two sequences*


---

**Description**

Returns the length of the longest common prefix of two sequences. This attribute is described in *Elzinga (2008)*.

**Usage**

```
seqLLCP(seq1, seq2)
```

**Arguments**

seq1              a sequence from a sequence object.  
 seq2              a sequence from a sequence object.

**Value**

an integer being the length of the longest common prefix of the two sequences.

**References**

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. *Technical Report*, Department of Social Science Research Methods, Vrije Universiteit, Amsterdam.

**See Also**

[seqdist](#)

**Examples**

```
data(famform)
famform.seq <- seqdef(famform)

## The LCP's length between sequences 1 and 2
## in the famform sequence object is 2
seqLLCP(famform.seq[1,], famform.seq[2,])
```

---

seqLLCS	<i>Compute the length of the longest common subsequence of two sequences</i>
---------	--

---

**Description**

Returns the length of the longest common subsequence of two sequences. This attribute is described in *Elzinga (2008)*.

**Usage**

```
seqLLCS(seq1, seq2)
```

**Arguments**

seq1	a sequence from a sequence object
seq2	a sequence from a sequence object

**Value**

an integer being the length of the longest common subsequence of the two sequences.

**References**

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. *Technical Report*, Department of Social Science Research Methods, Vrije Universiteit, Amsterdam.

**See Also**

[seqdist](#)

**Examples**

```
LCS.ex <- c("S-U-S-M-S-U", "U-S-SC-MC", "S-U-M-S-SC-UC-MC")
LCS.ex <- seqdef(LCS.ex)
seqLLCS(LCS.ex[1,],LCS.ex[3,])
```

seqlogp

*Logarithm of the probabilities of state sequences***Description**

Logarithm of the probabilities of state sequences. The probability of a sequence is defined as the product of the probabilities of the successive states in the sequence. State probabilities can either be provided or be computed with one of a few basic models.

**Usage**

```
seqlogp(seqdata, prob="trate", time.varying=TRUE,
        begin="freq", weighted=TRUE, with.missing=FALSE)
```

**Arguments**

seqdata	A state sequence object as produced by <a href="#">seqdef</a> .
prob	String or numeric array. If a string, either "trate" or "freq" to select a probability model to compute the state probabilities. If a numeric array, a matrix or 3-dimensional array of transition probabilities. See details.
time.varying	Logical. If TRUE, the probabilities (transitions or frequencies) are computed separately for each time $t$ point.
begin	String or numeric vector. Distribution used to determine the probability of the first state. If a vector, the probabilities to use. If a string, either "freq" or <code>global.freq</code> . With <code>freq</code> , the observed distribution at first position is used. If <code>global.freq</code> , the overall distribution is used. Default is "freq".
weighted	Logical. Should we account for the weights when present in seqdata? Default is TRUE.
with.missing	Logical. Should non void missing states be treated as regular values? Default is FALSE.

**Details**

The sequence likelihood  $P(s)$  is defined as the product of the probability with which each of its observed successive state is supposed to occur at its position. Let  $s = s_1 s_2 \cdots s_\ell$  be a sequence of length  $\ell$ . Then

$$P(s) = P(s_1, 1) \cdot P(s_2, 2) \cdots P(s_\ell, \ell)$$

with  $P(s_t, t)$  the probability to observe state  $s_t$  at position  $t$ .

There are different ways to determine the state probabilities  $P(s_t, t)$ . The method is chosen by means of the `prob` argument.

With `prop = "freq"`, the probability  $P(s_t, t)$  is set as the observed relative frequency at position  $t$ . In that case, the probability does not depend on the probabilities of transition. By default (`time.varying=TRUE`), the relative frequencies are computed separately for each position  $t$ . With `time.varying=FALSE`, the relative frequencies are computed over the entire covered period, i.e. the same frequencies are used at each  $t$ .

Option `prop = "trate"` assumes that each  $P(s_t, t)$ ,  $t > 1$  is set as the transition probability  $p(s_t|s_{t-1})$ . The state distribution used to determine the probability of the first state  $s_1$  is set by means of the `begin` argument (see below). With the default `time.varying=TRUE`, the transition probabilities are estimated separately at each position, yielding an array of transition matrices. With `time.varying=FALSE`, the transition probabilities are assumed to be constant over the successive positions and are estimated over the entire sequence duration, i.e. from all observed transitions.

Custom transition probabilities can be provided by passing a matrix or a 3-dimensional array as `prob` argument.

The distribution used at the first position is set by means of the `begin` argument. You can either pass the distribution (probabilities of the states in the alphabet including the missing value when `with.missing=TRUE`), or specify `"freq"` for the observed distribution at the first position, or `global.freq` for the overall state distribution.

The likelihood  $P(s)$  being generally very small, `seqlogp` returns  $-\log P(s)$ . The latter quantity is minimal when  $P(s)$  is equal to 1.

### Value

Vector of the negative logarithm  $-\log P(s)$  of the sequence probabilities.

### Author(s)

Matthias Studer, Alexis Gabadinho, and Gilbert Ritschard

### Examples

```
## Creating the sequence objects using weights
data(biofam)
biofam.seq <- seqdef(biofam, 10:25, weights=biofam$wprob)

## Computing sequence probabilities
biofam.prob <- seqlogp(biofam.seq)
## Comparing the probability of each cohort
cohort <- biofam$birthyr>1940
boxplot(biofam.prob~cohort)
```

---

seqmeant

*Mean durations in each state*

---

### Description

Compute the mean total time spent in each state of the alphabet for the set of sequences given as input.

**Usage**

```
seqmeant(seqdata, weighted=TRUE, with.missing=FALSE, prop=FALSE, serr=FALSE)
```

**Arguments**

seqdata	a sequence object as defined by the <a href="#">seqdef</a> function.
weighted	logical: if TRUE, the weights (weights attribute) attached to the sequence object are used for computing weighted mean total time.
with.missing	logical: if set to TRUE, cumulated durations are also computed for the missing status (gaps in the sequences). See <a href="#">seqdef</a> on options for handling missing values when creating sequence objects.
prop	logical: if TRUE, proportions of time spent in each state are returned instead of absolute values. This option is especially useful when sequences contain missing states, since the sum of the state durations may not be the same for all sequences.
serr	logical: if TRUE, the variance and standard deviation of the total time spent in the states, as well as the standard error of the mean are also computed.

**Value**

An object of class *stslst.meant*. There are print and plot methods for such objects.

**Author(s)**

Alexis Gabadinho

**References**

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

**See Also**

[plot.stslst.meant](#) for basic plots of *stslst.meant* objects and [seqmtplot](#) ([seqplot](#) with type="mt") argument for more sophisticated plots of the mean durations allowing grouping and legend.

**Examples**

```
## Defining a sequence object with columns 13 to 24
## in the actcal example data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal,13:24,labels=actcal.lab)

## Computing the mean time in the different states
seqmeant(actcal.seq)

## Mean times with their standard error
seqmeant(actcal.seq, serr=TRUE)
```



---

seqmodst	<i>Sequence of modal states</i>
----------	---------------------------------

---

### Description

Sequence made of the modal state at each position.

### Usage

```
seqmodst(seqdata, weighted=TRUE, with.missing=FALSE)
```

### Arguments

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
weighted	if TRUE, distributions account for the weights assigned to the state sequence object (see <a href="#">seqdef</a> ). Set as FALSE if you want ignore the weights.
with.missing	If FALSE (default value), returned distributions ignore missing values.

### Details

In case of multiple modal states at a given position, the first one is taken. Hence, the result may vary with the alphabet order.

### Value

an object of class *stslst.modst*. This is actually a state sequence object (containing a single state sequence) with additional attributes, among which the `Frequencies` attribute containing the transversal frequency of each state in the sequence. There are print and plot methods for such objects. More sophisticated plots can be produced with the `seqplot` function.

### Author(s)

Alexis Gabadinho

### References

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

### See Also

[plot.stslst.modst](#) for default plot method, [seqplot](#) for higher level plots.

**Examples**

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
"Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## Modal state sequence
seqmodst(biofam.seq)

## Examples using weights and with.missing arguments
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

seqmodst(ex1.seq)
seqmodst(ex1.seq, weighted=FALSE)
seqmodst(ex1.seq, weighted=FALSE, with.missing=TRUE)
```

---

seqmpos

*Number of matching positions between two sequences.*


---

**Description**

Returns the number of common elements, i.e., same states appearing at the same position in the two sequences.

**Usage**

```
seqmpos(seq1, seq2, with.missing=FALSE)
```

**Arguments**

seq1	a sequence from a sequence object.
seq2	a sequence from a sequence object.
with.missing	if TRUE, gaps appearing at the same position in both sequences are also considered as common elements.

**Author(s)**

Alexis Gabadinho (with Gilbert Ritschard for help page)

**See Also**

[seqLLCP](#), [seqLLCS](#) .

**Examples**

```

data(famform)
famform.seq <- seqdef(famform)

seqmpos(famform.seq[1,], famform.seq[2,])
seqmpos(famform.seq[2,], famform.seq[4,])

## Example with gaps in sequences
a <- c(NA, "A", NA, "B", "C")
b <- c(NA, "C", NA, "B", "C")

ex1.seq <- seqdef(rbind(a,b))

seqmpos(ex1.seq[1,], ex1.seq[2,])
seqmpos(ex1.seq[1,], ex1.seq[2,], with.missing=TRUE)

```

---

seqnum

*Transform into a sequence object with numerical alphabet.*


---

**Description**

The function `seqnum` transforms the provided state sequence object into an equivalent sequence object in which the original alphabet is replaced with an alphabet of numbers ranging from 0 to  $(nbstates-1)$ .

**Usage**

```
seqnum(seqdata, with.missing=FALSE)
```

**Arguments**

<code>seqdata</code>	a state sequence object as defined by the <code>seqdef</code> function.
<code>with.missing</code>	logical: Should missing elements in the sequences be turned into numerical values as well? The code for missing values in the sequences is retrieved from the 'nr' attribute of <code>seqdata</code> .

**Details**

The first state (for example 'A') is coded with the value 0, the second state (for example 'B') is coded with the value 1, etc... The function returns a sequence object containing the original sequences coded with the new numerical alphabet ranging from 0 to  $(nbstates-1)$

**Author(s)**

Alexis Gabadinho

**See Also**

[seqdef](#), [alphabet](#)

**Examples**

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## The first 10 sequences in the actcal.seq
## sequence object
actcal.seq[1:10,]
alphabet(actcal.seq)

## The first 10 sequences in the actcal.seq
## sequence object with numerical alphabet
seqnum(actcal.seq[1:10,])

## states A,B,C,D are now coded 0,1,2,3
alphabet(seqnum(actcal.seq))
```

---

seqpcplot

*Parallel coordinate plot for sequence data*

---

**Description**

A decorated parallel coordinate plot to render the order of the successive elements in sequences. The sequences are displayed as jittered frequency-weighted parallel lines. The plot is also embedded as the `type="pc"` option of the [seqplot](#) function and serves as plot method for `eseq` and `seqelist` objects.

**Usage**

```
seqpcplot(seqdata, group = NULL, weights = NULL, cex = 1, lwd = 1/4,
  cpal = NULL, grid.scale = 1/5, ltype = "unique",
  embedding = "most-frequent", lorder = NULL, lcourse = "upwards",
  filter = NULL, hide.col = "grey80", alphabet = NULL,
  missing = "auto", order.align = "first", main = NULL, xlab = NULL,
  ylab = NULL, xaxis = TRUE, yaxis = TRUE, axes = "all", xtlab = NULL,
  cex.lab = 1, rows = NA, cols = NA, plot = TRUE, seed = NULL,
  weighted = TRUE, with.missing = TRUE,
  title, cex.plot, ...)

seqpcfilter(method = c("minfreq", "cumfreq", "linear"), level = 0.05)
```

**Arguments**

seqdata	The sequence data. Either an event sequence object of class <code>seqelist</code> (see <a href="#">seqcreate</a> ) or a state sequence object of class <code>stslst</code> (see <a href="#">seqdef</a> ).
group	a vector (numeric or factor) of group memberships of length equal the number of sequences. When specified, one plot is generated for each different membership value.
weights	a numeric vector of weights of length equal the number of sequences. When NULL, the weights are taken from the <code>seqdata</code> object.
cex	Plotting text and symbols magnification. See <a href="#">par</a> .
lwd	expansion factor for line widths. The expansion is relative to the size of the squared symbols.
cpal	color palette vector for line coloring.
grid.scale	Expansion factor for the translation zones.
ltype	the type of sequence that is drawn. Either "unique" to render unique patterns or "non-embeddable" to render non-embeddable sequences.
embedding	The method for embedding sequences embeddable in multiple non-embeddable sequences. Either "most-frequent" (default) or "uniformly". Relevant only with <code>ltype = "non-embeddable"</code> .
lorder	line ordering. Either "background" or "foreground".
lcourse	Method to connect simultaneous elements with the preceding and following ones. Either "upwards" (default) or "downwards".
filter	list of line coloring options. See details.
hide.col	Color for sequences filtered-out by the <code>filter</code> specification.
alphabet	a vector of response levels in the order they should appear on the y-axis. This argument is solely relevant for <code>seqelist</code> objects.
missing	character. Whether and how missing values should be displayed. Available are "auto", "show" and "hide". If "auto", the plot will show missings only if present. "hide" will fade out missings and "show" will always show missings. If <code>with.missing=FALSE</code> , <code>missing</code> is turned into "hide". If <code>with.missing=TRUE</code> and <code>missing="hide"</code> , <code>missing</code> is turned into "auto".
order.align	Aligning method. For aligning on order positions use either "first" (default) or "last". Option "first" numbers the positions from the beginning while "last" numbers them from the end. With <code>order.align = "time"</code> , the elements in the sequences are aligned on their rounded timestamps.
main	title for the graphic.
xlab	label for the x axis
ylab	label for the y axis
xaxis	logical: Should x-axis be plotted?
yaxis	logical: Should y-axis be plotted?
axes	if set as "all" (default value) x-axes are drawn for each plot in the graphic. If set as "bottom" and <code>group</code> is used, axes are drawn only under the plots at the bottom of the graphic area. If FALSE, no x-axis is drawn.

<code>xtlab</code>	labels for the x-axis ticks.
<code>cex.lab</code>	x and y labels magnification. See <a href="#">par</a> .
<code>rows,cols</code>	integers. Number of rows and columns of the plot panel.
<code>plot</code>	logical. Should the plot be displayed? Set as FALSE to retrieve the seqpcplot object without plotting it.
<code>seed</code>	integer. Start seed value.
<code>weighted</code>	logical. Should weights be accounted for? Default is TRUE.
<code>with.missing</code>	logical. Should we care about possible missings? Default is TRUE. See also the <a href="#">missing</a> argument.
<code>method</code>	character string. Defines the filtering function. Available are "minfreq", "cumfreq" and "linear".
<code>level</code>	numeric scalar between 0 and 1. The frequency threshold for the filtering methods "minfreq" and "cumfreq".
<code>title</code>	Deprecated. Use <code>main</code> instead.
<code>cex.plot</code>	Deprecated. Use <code>cex.lab</code> instead.
<code>...</code>	arguments to be passed to other methods, such as graphical parameters (see <a href="#">par</a> ).

## Details

For plots by groups specified with the `group` argument, plotted line widths and point sizes reflect relative frequencies within group.

The `filter` argument serves to specify filters to gray less interesting patterns. The filtered-out patterns are displayed in the `hide.col` color. The `filter` argument expects a list with at least `type` and `value`. The following types are implemented:

Type "sequence": colors a specific pattern, for example assign  
`filter = list(type = "sequence", value = "(Leaving Home,Union)-(Child)")`.

Type "subsequence": colors patterns which include a specific subsequence, for example  
`filter = list(type = "subsequence", value = "(Child)-(Marriage)")`.

Type "value": gradually colors the patterns according to the numeric vector (of length equal to the number of sequences) provided as "value" element in the list. You can give something like `filter = list(type = "value", value = c(0.2, 1, ...))` or provide the distances to the medoid as value vector for example.

Type "function": colors the patterns depending on the values returned by a [0,1] valued function of the frequency  $x$  of the pattern. Three native functions can be used: "minfreq", "cumfreq" and "linear". Use `filter = list(type = "function", value = "minfreq", level = 0.05)` to color patterns with a support of at least 5% (within group). Use `filter = list(type = "function", value = "cumfreq", level = 0.5)` to highlight the 50% most frequent patterns (within group). Or, use `filter = list(type="function", value="linear")` to use a linear gradient for the color intensity (the most frequent trajectory gets 100% intensity). Other user-specified functions can be provided by giving something like  
`filter = list(type="function", value=function(x, arg1, arg2) {return(x/max(x) * arg1/arg2)}), arg1 = 1, arg2 = 1)`. This latter function adjusts gradually the color intensity of patterns according to the frequency of the pattern.

The function `seqpcfilter` is a convenience function for type "function". The three examples above can be imitated by `seqpcfilter("minfreq",0.05)`, `seqpcfilter("cumfreq",0.5)` and `seqpcfilter("linear")`.

If a numeric scalar is assigned to `filter`, the "minfreq" filter is used.

### Value

An object of class "seqpcplot" with various information necessary for constructing the plot, e.g. coordinates. There is a summary method for such objects.

### Author(s)

Reto Bürgin (with Gilbert Ritschard for the help page)

### References

Bürgin, R. and G. Ritschard (2014), A decorated parallel coordinate plot for categorical longitudinal data, *The American Statistician* 68(2), 98-103.

### See Also

[seqplot](#), [seqdef](#), [seqcreate](#)

### Examples

```
## =====
## plot biofam data
## =====

data(biofam)
lab <- c("Parent", "Left", "Married", "Left+Marr", "Child", "Left+Child",
        "Left+Marr+Child", "Divorced")

## plot state sequences in STS representation
## =====

## creating the weighted state sequence object.
biofam.seq <- seqdef(data = biofam[,10:25], labels = lab,
                    weights = biofam$wpo0tbgs)

## select the first 20 weighted sequences (sum of weights = 18)
biofam.seq <- biofam.seq[1:20, ]

par(mar=c(4,8,2,2))
seqpcplot(seqdata = biofam.seq, order.align = "time")

## .. or
seqplot(seqdata = biofam.seq, type = "pc", order.align = "time")

## Distinct successive states (DSS)
```

```

## =====

seqplot(seqdata = biofam.seq, type = "pc", order.align = "first")

## .. or (equivalently)

biofam.DSS <- seqdss(seqdata = biofam.seq) # prepare format
seqpcplot(seqdata = biofam.DSS)

## plot event sequences
## =====

biofam.eseq <- seqcreate(biofam.seq, tevent = "state") # prepare data

## plot the time in the x-axis
seqpcplot(seqdata = biofam.eseq, order.align = "time", alphabet = lab)

## ordering of events
seqpcplot(seqdata = biofam.eseq, order.align = "first", alphabet = lab)

## ... or
plot(biofam.eseq, order.align = "first", alphabet = lab)

## additional arguments
## =====

## non-embeddable sequences
seqpcplot(seqdata = biofam.eseq, ltype = "non-embeddable",
          order.align = "first", alphabet = lab)

## align on last event
par(mar=c(4,8,2,2))
seqpcplot(seqdata = biofam.eseq, order.align = "last", alphabet = lab)

## use group variables
seqpcplot(seqdata = biofam.eseq, group = biofam$sex[1:20],
          order.align = "first", alphabet = lab)

## color patterns (Parent)-(Married) and (Parent)-(Left+Marr+Child)
par(mfrow = c(1, 1))
seqpcplot(seqdata = biofam.eseq,
          filter = list(type = "sequence",
                        value=c("(Parent)-(Married)",
                                "(Parent)-(Left+Marr+Child)")),
          alphabet = lab, order.align = "first")

## color subsequence pattern (Parent)-(Left)
seqpcplot(seqdata = biofam.eseq,
          filter = list(type = "subsequence",
                        value = "(Parent)-(Left)"),
          alphabet = lab, order.align = "first")

## color sequences over 10% (within group) (function method)

```



```
seqpcplot(seqdata = biofam.eseq,
          filter = list(type = "function",
                        value = "minfreq",
                        level = 0.1),
          alphabet = lab, order.align = "first", seed = 1)

## .. same result using the convenience functions
seqpcplot(seqdata = biofam.eseq,
          filter = 0.1,
          alphabet = lab, order.align = "first", seed = 1)

seqpcplot(seqdata = biofam.eseq,
          filter = seqpcfilter("minfreq", 0.1),
          alphabet = lab, order.align = "first", seed = 1)

## highlight the 50% most frequent sequences
seqpcplot(seqdata = biofam.eseq,
          filter = list(type = "function",
                        value = "cumfreq",
                        level = 0.5),
          alphabet = lab, order.align = "first", seed = 2)

## .. same result using the convenience functions
seqpcplot(seqdata = biofam.eseq,
          filter = seqpcfilter("cumfreq", 0.5),
          alphabet = lab, order.align = "first", seed = 2)

## linear gradient
seqpcplot(seqdata = biofam.eseq,
          filter = list(type = "function",
                        value = "linear"),
          alphabet = lab, order.align = "first", seed = 2)

seqpcplot(seqdata = biofam.eseq,
          filter = seqpcfilter("linear"),
          alphabet = lab, order.align = "first", seed = 1)
```

---

seqplot

*Plot state sequence objects*

---

### **Description**

High level plot functions for state sequence objects that can produce state distribution (chronograms), frequency, index, transversal entropy, sequence of modes, meant time, and representative plots.

**Usage**

```
seqplot(seqdata, group = NULL, type = "i", main = NULL, cpal = NULL,
        missing.color = NULL, ylab = NULL, yaxis = TRUE, axes = "all",
        xtlab = NULL, cex.axis = 1, with.legend = "auto", ltext = NULL,
        cex.legend = 1, use.layout = (!is.null(group) | with.legend != FALSE),
        legend.prop = NA, rows = NA, cols = NA, title, cex.plot, withlegend, ...)
```

```
seqdplot(seqdata, group = NULL, main = NULL, ...)
seqfplot(seqdata, group = NULL, main = NULL, ...)
seqiplot(seqdata, group = NULL, main = NULL, ...)
seqIplot(seqdata, group = NULL, main = NULL, ...)
seqHtplot(seqdata, group = NULL, main = NULL, ...)
seqmsplot(seqdata, group = NULL, main = NULL, ...)
seqmtplot(seqdata, group = NULL, main = NULL, ...)
seqrplot(seqdata, group = NULL, main = NULL, ...)
```

**Arguments**

<code>seqdata</code>	State sequence object created with the <a href="#">seqdef</a> function.
<code>group</code>	Grouping variable of length equal to the number of sequences. When not NULL, a distinct plot is generated for each level of group.
<code>type</code>	the type of the plot. Available types are "d" for state distribution plots (chronograms), "f" for sequence frequency plots, "Ht" for transversal entropy plots, "i" for selected sequence index plots, "I" for whole set index plots, "ms" for plotting the sequence of modal states, "mt" for mean times plots, "pc" for parallel coordinate plots and "r" for representative sequence plots.
<code>main</code>	Character string. Title for the graphic. Default is NULL.
<code>cpal</code>	Color palette used for the states. By default, the <code>cpal</code> attribute of the <code>seqdata</code> sequence object is used (see <a href="#">seqdef</a> ). If user specified, a vector of colors of length and order corresponding to <code>alphabet(seqdata)</code> .
<code>missing.color</code>	Color for representing missing values inside the sequences. By default, this color is taken from the <code>missing.color</code> attribute of <code>seqdata</code> .
<code>ylab</code>	Character string. an optional label for the y-axis. If set to NA, no label is drawn.
<code>yaxis</code>	Logical. Should the y-axis be plotted? When set as TRUE (default value), sequence indexes are displayed for "i" and "I", mean time values for "mt", percentages for "d" and "f", and state/event labels for "pc". Ignored for "r".
<code>axes</code>	Character string or logical. If set as "all" (default value) x axes are drawn for each plot in the graphic. If set as "bottom" and group is used, axes are drawn only under the plots located at the bottom of the graphic area. If FALSE, no x-axis is drawn.
<code>xtlab</code>	Vector of length equal to the number of columns of <code>seqdata</code> . Optional labels for the x-axis tick labels. If unspecified, the column names of the <code>seqdata</code> sequence object are used (see <a href="#">seqdef</a> ).
<code>cex.axis</code>	Real value. Axis annotation magnification. When <code>type = "r"</code> and for <code>seqrplot()</code> , it also determines the magnification of the plotted text and symbols. See <a href="#">par</a> .

<code>with.legend</code>	Character string or logical. Defines if and where the legend of the state colors is plotted. The default value "auto" sets the position of the legend automatically. Other possible value is "right". Obsolete value TRUE is equivalent to "auto".
<code>ltext</code>	Vector of character strings of length and order corresponding to <code>alphabet(seqdata)</code> . Optional description of the states to appear in the legend. If unspecified, the label attribute of the <code>seqdata</code> sequence object is used (see <a href="#">seqdef</a> ).
<code>cex.legend</code>	Real. Legend magnification. See <a href="#">legend</a> .
<code>use.layout</code>	Logical. Should <code>layout</code> be used to arrange plots when using the group option or plotting a legend? When layout is activated, the standard <code>'par(mfrow=...)'</code> for arranging plots does not work. With <code>with.legend=FALSE</code> and <code>group=NULL</code> , layout is automatically deactivated and <code>'par(mfrow=...)'</code> can be used.
<code>legend.prop</code>	Real in range [0,1]. Proportion of the graphic area devoted to the legend plot when <code>use.layout=TRUE</code> and <code>with.legend=TRUE</code> . Default value is set according to the place (bottom or right of the graphic area) where the legend is plotted.
<code>rows,cols</code>	Integers. Number of rows and columns of the plot panel when <code>use.layout=TRUE</code> .
<code>title</code>	Deprecated. Use <code>main</code> instead.
<code>cex.plot</code>	Deprecated. Use <code>cex.axis</code> instead.
<code>withlegend</code>	Deprecated. Use <code>with.legend</code> instead.
<code>...</code>	arguments to be passed to the function called to produce the appropriate statistics and the associated plot method (see details), or other graphical parameters. For example the <code>weighted</code> argument can be passed to control whether (un)weighted statistics are produced or <code>with.missing</code> argument to take missing values into account when computing transversal or longitudinal state distributions.

## Details

`seqplot` is the generic function for high level plots of state sequence objects with group splits and automatic display of the color legend. Many different types of plots can be produced by means of the `type` argument. Except for sequence index plots, `seqplot` first calls the specific function producing the required statistics and then the plot method for objects produced by this function (see below). For sequence index plots, the state sequence object itself is plotted by calling the `plot.stslist` method. When splitting by groups and/or displaying the color legend, the `layout` function is used for arranging the plots.

The `seqdplot`, `seqfplot`, `seqiplot`, `seqIplot`, `seqHtplot`, `seqmsplot`, `seqmtplot`, `seqpcplot` and `seqrplot` functions are aliases for calling `seqplot` with `type` argument set respectively to "d", "f", "i", "I", "Ht", "ms", "mt", "pc" or "r".

A *State distribution plot* (`type="d"`) represents the sequence of the cross-sectional state frequencies by position (time point) computed by the `seqstatd` function and rendered with the `plot.stslist.statd` method. Such plots are also known as chronograms.

A *Sequence frequency plots* (`type="f"`) displays the most frequent sequences, each one with an horizontal stack bar of its successive states. Sequences are displayed bottom-up in decreasing order of their frequencies (computed by the `seqtab` function). The `plot.stslist.freq` plot method is called for producing the plot.

The `idxs` optional argument may be specified for selecting the sequences to be plotted (default is 1:10, i.e. the 10 most frequent sequences). The width of the bars representing the sequences is by

default proportional to their frequencies, but this can be disabled with the `pbarw=FALSE` optional argument. If weights have been specified when creating `seqdata`, weighted frequencies are used unless you set the `weighted=TRUE` option. See examples below, the `seqtab` and `plot.stslist.freq` manual pages for a complete list of optional arguments and Müller *et al.*, (2008) for a description of sequence frequency plots.

In *sequence index plots* (`type="i"` or `type="I"`), the requested individual sequences are rendered with horizontal stacked bars depicting the states over successive positions (time). Optional arguments are `idxs` for specifying the indexes of the sequences to be plotted (when `type="i"` defaults to the first ten sequences, i.e. `idxs=1:10`). For plotting nicely a (big) whole set one can use `type="I"` which is the same as using `idxs=0` together with the additional graphical parameters `border=NA` and `space=0` to suppress bar borders and space between bars. The `sortv` argument can be used to pass a vector of numerical values for sorting the sequences or to specify a sorting method. See `plot.stslist` for a complete list of optional arguments and their description.

The interest of sequence index plots has, for instance, been stressed by Scherer (2001) and Brzinsky-Fay *et al.* (2006). Notice that index plots for thousands of sequences result in very heavy PDF or POSTSCRIPT graphic files. Dramatic file size reduction may be achieved by saving the figures in bitmap format with using for instance the `png` graphic device instead of `postscript` or `pdf`.

The *transversal entropy plot* (`type="Ht"`) displays the evolution over positions of the cross-sectional entropies (Billari, 2001). Cross-sectional entropies are computed by calling `seqstatd` function and then plotted by calling the `plot.stslist.statd` plot method.

The *modal state sequence plot* (`type="ms"`) displays the sequence of the modal states with each mode proportional to its frequency at the given position. The `seqmodst` function is called which returns the sequence and the result is plotted by calling the `plot.stslist.modst` plot method.

The *mean time plot* (`type="mt"`) displays the mean time spent in each state of the alphabet as computed by the `seqmeant` function. The `plot.stslist.meant` plot method is used to plot the resulting statistics. Set `serr=TRUE` to display error bars on the mean time plot. Bar labels can be specified by passing the `bar.labels` among the `...` arguments. In that case, `bar.labels` must be either a matrix with group specific labels in columns or a single vector to display the same labels for all groups.

The *representative sequence plot* (`type="r"`) displays a reduced, non redundant set of representative sequences extracted from the provided state sequence object and sorted according to a representativeness criterion. The `seqrep` function is called to extract the representative set which is then plotted by calling the `plot.stslist.rep` method. A distance matrix is required that is passed with the `diss` argument or by calling the `seqdiss` function if `diss=NULL`. The `criterion` argument sets the representativeness criterion used to sort the sequences. Refer to the `seqrep` and `plot.stslist.rep` manual pages for a complete list of optional arguments. See Gabadinho and Ritschard (2013) for more details on the extraction of representative sets. Also look at the examples below.

For *decorated parallel coordinate plots* (`type="pc"`) see the specific manual page of `seqpcplot`.

## Author(s)

Alexis Gabadinho and Gilbert Ritschard

## References

- Billari, F. C. (2001). The analysis of early life courses: Complex description of the transition to adulthood. *Journal of Population Research* **18**(2), 119-142.
- Brzinsky-Fay C., U. Kohler, M. Luniak (2006). Sequence Analysis with Stata. *The Stata Journal*, **6**(4), 435-460.
- Gabardinho, A., and G. Ritschard (2013), "Searching for typical life trajectories applied to childbirth histories", In Levy, R. & Widmer, E. (eds) *Gendered life courses - Between individualization and standardization. A European approach applied to Switzerland*, pp. 287-312. Vienna: LIT.
- Gabardinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.
- Gabardinho A, Ritschard G, Studer M, Müller NS (2011). "Extracting and Rendering Representative Sequences", In A Fred, JLG Dietz, K Liu, J Filipe (eds.), *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 128 of *Communications in Computer and Information Science (CCIS)*, pp. 94-106. Springer-Verlag.
- Müller, N. S., A. Gabardinho, G. Ritschard and M. Studer (2008). Extracting knowledge from life courses: Clustering and visualization. In *Data Warehousing and Knowledge Discovery, 10th International Conference DaWaK 2008, Turin, Italy, September 2-5*, LNCS 5182, Berlin: Springer, 176-185.
- Scherer S (2001). Early Career Patterns: A Comparison of Great Britain and West Germany. *European Sociological Review*, **17**(2), 119-144.

## See Also

[plot.stslist.statd](#), [plot.stslist.freq](#), [plot.stslist](#), [plot.stslist.modst](#), [plot.stslist.meant](#), [plot.stslist.rep](#), [seqrep](#), [seqpcplot](#), [seqplot](#).

## Examples

```
## =====
## Creating state sequence objects from example data sets
## =====

## biofam data set
data(biofam)
## We use only a sample of 300 cases
set.seed(10)
biofam <- biofam[sample(nrow(biofam),300),]
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
               "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## actcal data set
data(actcal)
## We use only a sample of 300 cases
set.seed(1)
actcal <- actcal[sample(nrow(actcal),300),]
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)
```

```

## ex1 using weights
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## =====
## Sequence frequency plots
## =====

## Plot of the 10 most frequent sequences
seqplot(biofam.seq, type="f")

## Grouped by sex
seqfplot(actcal.seq, group=actcal$sex)

## Unweighted vs weighted frequencies
seqfplot(ex1.seq, weighted=FALSE)
seqfplot(ex1.seq, weighted=TRUE)

## =====
## Modal states sequence
## =====
seqplot(biofam.seq, type="ms")
## same as
seqmsplot(biofam.seq)

## =====
## Representative plots
## =====

## Computing a distance matrix
## with OM metric
costs <- seqcost(biofam.seq, method="INDELSLOG")
biofam.om <- seqdist(biofam.seq, method="OM", sm=costs$sm, indel=costs$indel)

## Plot of the representative sets grouped by sex
## using the default density criterion
seqrplot(biofam.seq, group=biofam$sex, diss=biofam.om)

## Plot of the representative sets grouped by sex
## using the "dist" (centrality) criterion
seqrplot(biofam.seq, group=biofam$sex, criterion="dist", diss=biofam.om)

## =====
## Sequence index plots
## =====

## First ten sequences
seqiplot(biofam.seq)

## All sequences sorted by age in 2000
## grouped by sex
## using 'border=NA' and 'space=0' options to have a nicer plot

```

```

seqiplot(actcal.seq, group=actcal$sex, idxs=0, border=NA, space=0,
         sortv=actcal$age00)

## =====
## State distribution plot
## =====

## biofam grouped by sex
seqplot(biofam.seq, type="d", group=actcal$sex)

## actcal grouped by sex
seqplot(actcal.seq, type="d", group=actcal$sex)

## =====
## Cross-sectional entropy plot
## =====
seqplot(biofam.seq, type="Ht", group=biofam$sex)

## =====
## Mean time plot
## =====

## actcal data set, grouped by sex
seqplot(actcal.seq, type="mt", group=actcal$sex)

## displaying mean times as bar labels
group <- factor(actcal$sex)
blab <- NULL
for (i in 1:length(levels(group))){
  blab <- cbind(blab, seqmeant(actcal.seq[group==levels(group)[i],]))
}
seqmtplot(actcal.seq, group=group,
          bar.labels = round(blab,digits=2), cex.barlab=1.2)

```

---

seqpm

*Find substring patterns in sequences*


---

## Description

Search for a pattern (substring) into sequences.

## Usage

```
seqpm(seqdata, pattern, sep="")
```

**Arguments**

seqdata	a sequence object as defined by the <a href="#">seqdef</a> function.
pattern	a character string representing the pattern (substring) to search for.
sep	state separator used in the pattern definition.

**Details**

This function searches a pattern (a character string) into a set of sequences and returns the results as a list with two elements: 'Nbmatch' the number of occurrences of the pattern and 'MatchesIndex' the vector of indexes (row numbers) of the sequences that match the pattern (see examples below).

**Value**

a list with two elements (see details).

**Author(s)**

Alexis Gabadinho

**Examples**

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## search for pattern "DAAD"
## (no work-full time work-full time work-no work)
## results are stored in the 'daad' object
daad <- seqpm(actcal.seq,"DAAD")

## Looking at the sequences
## containing the pattern
actcal.seq[daad$MIndex,]

## search for pattern "AD"
## (full time work-no work)
seqpm(actcal.seq,"AD")
```

**Description**

The degradation index returned by `seqidegrad` is a normalized transformation of the difference between the proportion of downward and upward state changes (transitions). The precarity and insecurity indexes are composite measures of complexity, degradation tendency, and undesirableness of the first state in the sequence.



**Usage**

```

seqidegrad(seqdata, state.order=alphabet(seqdata, with.missing), state.equiv=NULL,
  stprec=NULL, with.missing=FALSE,
  penalized="BOTH", method="RANK", weight.type="ADD",
  pow=1, border.effect=10)

seqprecarity(seqdata, correction=NULL,
  state.order=alphabet(seqdata, with.missing), state.equiv=NULL,
  stprec=NULL, with.missing=FALSE,
  otto=.2, a=1, b=1.2, method = "TRATEDSS",
  ...)

seqinsecurity(seqdata, correction=NULL,
  state.order=alphabet(seqdata, with.missing), state.equiv=NULL,
  stprec=NULL, with.missing=FALSE,
  pow = 1, spow=pow, bound=FALSE, method = "RANK",
  ...)

seqprecorr(...)

```

**Arguments**

seqdata	a state sequence object (class <code>stslst</code> ) as returned by the <a href="#">seqdef</a> function.
correction	Vector of non-negative correction factor values. If <code>NULL</code> (default), the correction factor is set as the degradation value returned by <code>seqidegrad</code> . See details.
state.order	Vector of short state labels defining the order of the states. First the less precarious (most positive) state and then the other states in increasing precariousness order. States of the alphabet that are not included here (and are not equivalent to one of the listed state) define the non-comparable states.
state.equiv	List of state equivalence classes. Each class in the list is given as the vector of the short labels of the states forming the class.
stprec	Vector of state undesirableness degrees. If <code>NULL</code> the values are derived from the state order using <a href="#">seqprecstart</a> . If not <code>NULL</code> , the values in <code>stprec</code> should conform the order of the alphabet. Use negative values to indicate non-comparable states. See details.
with.missing	Logical. Should the missing state be considered as an element of the alphabet?
otto	Scalar in the range $[0,1]$ . Trade-off weight between the precarity degree of the initial state and the corrected complexity. Default is <code>otto=.2</code>
a	Non-negative real value. Exponent weight of the complexity. Default is 1.
b	Non-negative real value. Exponent weight of the correction factor. Default is 1.2.
penalized	One of 'BOTH' (default), 'NEG', 'POS', or 'NO'. What should be penalized or rewarded? 'NEG' only negative transitions, POS only positive transitions (with negative penalization), 'BOTH' penalize negative transitions and reward positive ones, NO no penalization. Can also be logical with <code>TRUE</code> equivalent to 'BOTH' and <code>FALSE</code> equivalent to 'NO'.

method	One of 'FREQ', 'FREQ+', 'TRATE', 'TRATE+', 'TRATEDSS' (default for seqprecarity for backward compatibility), 'TRATEDSS+', 'RANK' (default for seqinsecurity and seqidegrad), 'RANK+', or 'ONE'. Method for determining transition weights. Weights based on transition probabilities: 'FREQ' overall frequency of the transitions, 'TRATE' transition probabilities, and 'TRATEDSS' transition probabilities in the DSS sequences. 'RANK' differences between state undesirableness degrees. 'ONE' no weight. With the + form the returned penalty is adjusted by the mean transition weight in the sequence.
weight.type	One of 'ADD' (default), 'INV', or 'LOGINV'. When method is one of 'FREQ', 'TRATE' or 'TRATEDSS', how weights are derived from the transition probabilities: 'ADD' additive (1-p), 'INV' inverse (1/p), and 'LOGINV' log of inverse. Ignored when any other method is selected.
pow	Real or logical. Recency weight exponent for potential to integrate the next spell. If real, transition weights are adjusted by the potential to integrate the next state using the pow value. pow=TRUE is equivalent to pow=1. (See details.)
spow	Real. Recency weight exponent for potential to integrate the first spell.
bound	Logical. Should the insecurity index be bounded by undesirableness degrees of best and worst states in the sequence?
border.effect	Real. Value (strictly greater than 1) used to adjust estimated transition probabilities to avoid border effect. Default is 10. See details.
...	Arguments passed to seqidegrad when correction=NULL

## Details

The seqidegrad function returns for each sequence  $x$  the difference  $q(x)$  between the proportions of downward and upward transitions (state changes).

The argument penalized allows to choose between three strategies for computing  $q(x)$ : only penalizing negative weights (in which case  $q(x)$  is the proportion of negative transitions), only rewarding (with negative penalties) positive transitions, and applying both positive and negative penalties. The transitions can be weighted and the type of transition weights used is selected with the method argument. For weights based on transition probabilities, the way how these probabilities are transformed into weights is controlled with weight.type. To avoid a border effect, when any computed transition probability  $p$  is close from 1 ( $p > 1 - .1/d$ ), all  $p$ 's are adjusted as  $p - p/d$ , where  $d$  is the border.effect parameter. With method="RANK", the weights are set as the differences between the to and from state undesirableness. When pow is not FALSE, the weight of each transition is multiplied by the potential to integrate the next spell using the provided pow value.

The precarity and insecurity indexes of a sequence  $x$  are both based on the complexity index (Gabadinho et al., 2010)  $c(x)$  (See the seqici function) and the undesirableness degree  $\pi(x_1)$  of the starting state.

The precarity applies a multiplicative correction to the complexity. It is defined as

$$prec(x) = \lambda\pi(x_1) + (1 - \lambda)(1 + r(x))^\beta c(x)^\alpha$$

where  $r(x)$  is the correction factor (argument correction) for the sequence. The  $\lambda$  parameter (argument otto) determines the trade-off between the importance of the undesirableness of the

starting state and of the corrected complexity index. Parameters  $\alpha$  and  $\beta$  (arguments a and b) are exponent weights of respectively the complexity and the correction factor.

The insecurity index applies an additive correction of the complexity:

$$insec(x) = \pi(x_1)integr(x, sp1) + r(x) + c(x)$$

where  $integr(x, sp1)$  is the potential to integrate the first spell (proportion of sequence length covered by first spell when  $spow=0$ ).

When `correction = NULL` (default),  $r(x)$  is set as the degradation index  $q(x)$  provided by `seqidegrad`. The degradation is computed with `pow=FALSE` for the precarity and using the provided `pow` value for the insecurity.

When `stprec` is a vector, negative values indicate non-comparable states that receive each the mean positive undesirableness value. After this transformation, the vector is normalized such that the minimum is 0 and the maximum 1.

When equivalent classes are provided, the class mean undesirableness degree is assigned to each state of the class (see `seqprecstart`). For the count of transitions a same state value is assigned to all equivalent states.

Non-comparable states (those not listed on the `state.order` argument and not equivalent to a listed state) all receive the mean undesirableness value. For the count of transitions, transitions from and to non-comparable states are ignored and replaced by a transition between the states that immediately precede and follow a spell in non-comparable states.

When there are missing states in the sequences, set `with.missing = TRUE` to treat the missing state as an additional state. In that case the missing state will be considered as non-comparable unless you include the `nr` attribute of `seqdata` in `state.order` or `state.equiv`. With `with.missing = FALSE`, transitions to and from the missing state will just be ignored and the undesirableness value of the first valid state will be used as starting undesirableness.

The earlier `seqprecorr` function is obsolete, use `seqidegrad` with `pow=FALSE` and `method='TRATEDSS'` instead.

## Value

For `seqprecarity` and `seqinsecurity`, an object of class `seqprec` with the value of the precarity or insecurity index for each sequence. The returned object has an attribute `stprec` that contains the state precarity degree used at the starting position. The associated print method (`print.seqprec`) prints the state precarity values without the additional attribute.

For `seqidegrad` an object of class `seqidegrad` with the degradation index  $q(x)$

and as additional attributes: `tr` the used transition weights; `signs` the transitions signs; `state.noncomp` the non-comparable states; and `state.order` the used state order. The associated print method (`print.seqidegrad`) prints the outcome values without the additional attributes.

## Author(s)

Gilbert Ritschard

## References

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

Ritschard, G., Bussi, M., and O'Reilly, J. (2018), "An index of precarity for measuring early employment insecurity", in G. Ritschard, and M. Studer, *Sequence Analysis and Related Approaches: Innovative Methods and Applications*, Series Life Course Research and Social Policies, Vol. 10, pp 279-295. Cham: Springer, doi: [10.1007/9783319954202\\_16](https://doi.org/10.1007/9783319954202_16).

Gabadinho, A., Ritschard, G., Studer, M. and Müller, N.S. (2010), "Indice de complexité pour le tri et la comparaison de séquences catégorielles", In *Extraction et gestion des connaissances (EGC 2010)*, *Revue des nouvelles technologies de l'information RNTI*. Vol. E-19, pp. 61-66.

## See Also

[seqici](#), [seqibad](#), [seqprecstart](#), [seqindic](#).

## Examples

```
## Defining a sequence object with columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal <- actcal[1:20,] ## Here, only a subset
actcal.seq <- seqdef(actcal[,13:24], alphabet=c('A','B','C','D'))

## insecurity and precarity using the original state order
insec <- seqinsecurity(actcal.seq)
prec <- seqprecarity(actcal.seq)
ici <- seqici(actcal.seq) ## complexity

seqn <- seqformat(actcal.seq, to="SPS", compress=TRUE)
tab <- data.frame(seqn,ici,insec,prec)
names(tab) <- c("seqn","ici","insec","prec")
tab

## Assuming A and B as equivalent regarding insecurity
insec2 <- seqinsecurity(actcal.seq, state.equiv=list(c('A','B')))
tab <- cbind(tab,insec2)
names(tab)[ncol(tab)] <- "insec2"

## and letting C be non-comparable
insec3 <- seqinsecurity(actcal.seq, state.order=c("A","B","D"),
  state.equiv=list(c('A','B')))
tab <- cbind(tab,insec3)
names(tab)[ncol(tab)] <- "insec3"

## bounding insecurity with undesirableness of best and worst state in sequence
insec4 <- seqinsecurity(actcal.seq, state.order=c("A","B","D"),
  state.equiv=list(c('A','B')), bound=TRUE)
tab <- cbind(tab,insec4)
names(tab)[ncol(tab)] <- "insec4"

## degrading index
```

```

degr <- seqidegrad(actcal.seq, state.order=c("A","B","D"),
  state.equiv=list(c('A','B')))
tab <- cbind(tab,degr)
names(tab)[ncol(tab)] <- "degr"
tab

## Precarity with transition weights based on differences in state undesirableness
prec.rank <- seqprecarity(actcal.seq, method='RANK')

#####
## Indexes in presence of missing values:
## missing state treated as an additional state
data(ex1)
## by default right missings are dropped from the sequences
s.ex1 <- seqdef(ex1[,1:13])
state.order=c("A","B","C","D") ## missing left as non-comparable
seqprecarity(s.ex1, state.order=state.order, with.missing=TRUE)
seqinsecurity(s.ex1, state.order=state.order, with.missing=TRUE)

## same result using the correction argument
dgp <- seqidegrad(s.ex1, with.missing=TRUE, state.order=state.order, method='TRATEDSS')
seqprecarity(s.ex1, state.order=state.order, with.missing=TRUE, correction=dgp)

dg <- seqidegrad(s.ex1, with.missing=TRUE, state.order=state.order)
seqinsecurity(s.ex1, state.order=state.order, with.missing=TRUE, correction=dg)

## bounding insecurity with undesirableness of best and worst state in sequence
seqinsecurity(s.ex1, state.order=state.order, with.missing=TRUE, bound=TRUE)

```

---

seqprecstart

*State precarity*


---

### Description

Determines the state undesirableness degree from the state ordering or conversely the state undesirableness values from the state ordering. (See details.)

### Usage

```

seqprecstart(seqdata,
  state.order=alphabet(seqdata, with.missing), state.equiv=NULL,
  stprec=NULL, with.missing=FALSE)

```

### Arguments

seqdata            a state sequence object (class `stslst`) as returned by the [seqdef](#) function.

<code>state.order</code>	Vector of short state labels defining the order of the states. First the most desirable (most positive) state and then the other states in decreasing desirability order. States of the alphabet that are not included here define the non-comparable states.
<code>state.equiv</code>	List of state equivalence classes. The classes in the list are each set as the vector of the short labels of the states forming the class.
<code>stprec</code>	Vector of user defined state undesirableness degrees ordered conformably with the alphabet of <code>seqdata</code> . Use negative values to indicate non-comparable states.
<code>with.missing</code>	Logical. Should the missing state be considered as an element of the alphabet?

### Details

When `stprec` is provided, the order is determined from the `stprec` values and overwrites `state.order`.

When `stprec=NULL`, the initial state undesirableness degrees are set as equivalently spaced values between 0 and 1 assigned to the states in specified order by `state.order`.

Equivalent states get the mean value of the states in the equivalence class.

Incomparable states (those not on the `state.order` list and not member of an equivalent class having a state listed in `state.order`) receive the average of all state undesirableness degrees.

When `stprec` is user defined, the provided vector is normalized into a vector with minimum 0 and maximum 1 and mean value of states in a class are assigned to all class members.

Only one state per class needs to be on the state order list. If more than one member is on the state order list they should be listed consecutively. Note that currently no check is performed.

When `with.missing=TRUE`, use the `seqdata`, "nr" argument to possibly include it in the `state.order` or `state.equiv`.

### Value

The vector of assigned undesirableness degrees sorted according to the original order of states in the alphabet.

### Author(s)

Gilbert Ritschard

### References

Ritschard, G., Bussi, M., and O'Reilly, J. (2018), "An index of precarity for measuring early employment insecurity", in G. Ritschard, and M. Studer, *Sequence Analysis and Related Approaches: Innovative Methods and Applications*, Series Life Course Research and Social Policies, Vol. 10, pp 279-295. Cham: Springer.

### See Also

[seqprecarity](#).

**Examples**

```
## Defining a sequence object with columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal <- actcal[1:200,] ## Here, only a subset
actcal.seq <- seqdef(actcal[,13:24])

## state precarious levels using the original state order
seqprecstart(actcal.seq)

## Assuming A and B as equivalent regarding precarious
seqprecstart(actcal.seq, state.equiv=list(c('A','B')))

## and letting C be non-comparable
seqprecstart(actcal.seq, state.order=c("A","B","D"), state.equiv=list(c('A','B')))
```

seqrcode

*Recoding state sequence objects and factors***Description**

Utilities for recoding factors or state sequence objects created with [seqdef](#).

**Usage**

```
seqrcode(seqdata, recodes, otherwise = NULL,
         labels = NULL, cpal = NULL)
recodef(x, recodes, otherwise=NULL, na=NULL)
```

**Arguments**

seqdata	The state sequence object to be recoded (created with <a href="#">seqdef</a> ).
recodes	A list specifying the recoding operations where each element is in the form <code>newcode=oldcode</code> or <code>newcode=c(oldcode1,oldcode2,...)</code> . The rules are treated in the same order as they appear, hence subsequent rules may modify the first ones.
otherwise	NULL or Character. Level given to cases uncovered by the recodes list. If NULL, old states remain unchanged.
labels	optional state labels used for the color legend of TraMineR's graphics. If NULL (default), the state names in the alphabet are also used as state labels (see <a href="#">seqdef</a> ).
cpal	an optional color palette for representing the newly defined alphabet in graphics. If NULL (default), a color palette is created from the colors in seqdata by assigning to newcode the color of the first old state listed as oldcode and by leaving the colors of the other states unchanged.
x	A factor to be recoded.
na	Character vector. If not NULL, the list of states that should be recoded as NA (missing values).

**Value**

The recoded factor or state sequence object.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**See Also**

[seqdef](#) to create a state sequence object.

**Examples**

```
## Recoding a state sequence object with seqrecode
data(actcal)
## Creating a state sequence object
actcal.seq <- seqdef(actcal,13:24, labels=c("> 37 hours", "19-36 hours",
  "1-18 hours", "no work"))
## Regrouping states B and C and setting the whole alphabet to A BC D
actcal.new <-seqrecode(actcal.seq,
  recodes = list("A"="A", "BC"=c("B", "C"), "D"="D"))
## Crosstabulate the first column of the recoded and
## original state sequence objects
table(actcal.new[,1], actcal.seq[,1])

## Same as before but using automatically original
## codes for unspecified states.
actcal.new2 <-seqrecode(actcal.seq,
  recodes = list("BC"=c("B", "C")))
table(actcal.new2[,1], actcal.seq[,1])

## Same as before but using otherwise
actcal.new3 <-seqrecode(actcal.seq, recodes = list("A"="A", "D"="D"),
  otherwise="BC")
table(actcal.new3[,1], actcal.seq[,1])

## Recoding factors
## Recoding the marital status to oppose married to all other case
maritalstatus <- recodef(actcal$civsta00,
  recodes=list("Married"="married"), otherwise="Single")
summary(maritalstatus)
table(maritalstatus, actcal$civsta00)

## Recoding the number of kids in the household
## -2 is a missing value
nbkids <- recodef(actcal$nbkid00,
  recodes=list("None"=0, "One"=1, "Two or more"=2:10), na=-2)
table(nbkids, actcal$nbkid00, useNA="always")
```



---

seqrep *Extracting sets of representative sequences*

---

### Description

Returns either an as small as possible set of non redundant representatives covering (having in their neighborhood) a desired percentage of all sequences, or a given number of patterns with highest coverage. Special cases are single representatives such as the medoid or the sequence pattern with densest neighborhood. See [plot.stslist.rep](#) for the plot method and [seqplot](#) for other plot options.

### Usage

```
seqrep(seqdata, criterion = "density", score = NULL, decreasing = TRUE,
       coverage = 0.25, nrep = NULL, pradius = 0.10, dmax = NULL, diss = NULL,
       weighted = TRUE, trep, tsim, dist.matrix, ...)
```

### Arguments

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
criterion	the representativeness criterion for sorting the candidate list. One of "freq" (sequence frequency), "density" (neighborhood density), "mscore" (mean state frequency), "dist" (centrality) and "prob" (sequence likelihood). See details.
score	an optional vector of representativeness scores for sorting the sequences in the candidate list. The length of the vector must be equal to the number of sequences in the sequence object.
decreasing	if a score vector is provided, indicates whether the objects in the candidate list must be sorted in ascending or descending order of this score. Default is TRUE, i.e. descending. The first object in the candidate list is then supposed to be the most representative.
coverage	coverage threshold, i.e., minimum proportion of sequences that should have a representative in their neighborhood (neighborhood radius is defined by pradius).
nrep	number of representative sequences. If NULL (default), the size of the representative set is controlled by coverage.
pradius	neighborhood radius as a percentage of the maximum (theoretical) distance dmax. Defaults to 0.1 (10%). Sequence $y$ is redundant to sequence $x$ when it is in the neighborhood of $x$ , i.e., within a distance $\text{pradius} \cdot \text{dmax}$ from $x$ .
dmax	maximum theoretical distance. Used to derive the neighborhood radius as $\text{pradius} \cdot \text{dmax}$ . If NULL, the value of dmax is derived from the dissimilarity matrix.
diss	matrix of pairwise dissimilarities between sequences in seqdata. If NULL, the matrix is computed by calling the <a href="#">seqdist</a> function. In that case, optional arguments to be passed to the seqdist function (see ... hereafter) should also be provided.
weighted	logical: Should weights assigned to the state sequence object be accounted for? (See <a href="#">seqdef</a> .) Set as FALSE to ignore the weights.

trep	Deprecated. Use coverage instead.
tsim	Deprecated. Use pradius instead.
dist.matrix	Deprecated. Use diss instead.
...	optional arguments to be passed to the seqdist function, mainly dist.method specifying the metric for computing the distance matrix, norm for normalizing the distances, indel and sm for indel and substitution costs when Optimal Matching metric is chosen. See <a href="#">seqdist</a> manual page for details.

## Details

The representative set is obtained by an heuristic. Representatives are selected by successively extracting from the sequences sorted by their representativeness score those which are not redundant with already retained representatives. The selection stops when either the desired coverage or the wanted number of representatives is reached. Sequences are sorted either by the values provided as score argument or by specifying one of the following as criterion argument: "freq" (*sequence frequency*), "density" (*neighborhood density*), "mscore" (*mean state frequency*), "dist" (*centrality*) and "dist" (*sequence likelihood*).

With the *sequence frequency* criterion, the more frequent a sequence the more representative it is supposed to be. Therefore, sequences are sorted in decreasing frequency order.

The *neighborhood density* is the number—density—of sequences in the neighborhood of the sequence. This requires to set the neighborhood radius pradius. Sequences are sorted in decreasing density order.

The *mean state frequency* criterion is the mean value of the transversal frequencies of the successive states. Let  $s = s_1 s_2 \cdots s_\ell$  be a sequence of length  $\ell$  and  $(f_{s_1}, f_{s_2}, \dots, f_{s_\ell})$  the frequencies of the states at (time-)position  $(t_1, t_2, \dots, t_\ell)$ . The mean state frequency is the sum of the state frequencies divided by the sequence length

$$MSF(s) = \frac{1}{\ell} \sum_{i=1}^{\ell} f_{s_i}$$

The lower and upper boundaries of  $MSF$  are 0 and 1.  $MSF$  is equal to 1 when all the sequences in the set are identical, i.e. when there is a single sequence pattern. The most representative sequence is the one with the highest score.

The *centrality* criterion is the sum of distances to all other sequences. The smallest the sum, the most representative the sequence.

The *sequence likelihood*  $P(s)$  is defined as the product of the probability with which each of its observed successive state is supposed to occur at its position. Let  $s = s_1 s_2 \cdots s_\ell$  be a sequence of length  $\ell$ . Then

$$P(s) = P(s_1, 1) \cdot P(s_2, 2) \cdots P(s_\ell, \ell)$$

with  $P(s_t, t)$  the probability to observe state  $s_t$  at position  $t$ .

The question is how to determinate the state probabilities  $P(s_t, t)$ . One commonly used method for computing them is to postulate a Markov Chain model, which can be of various order. The implemented criterion considers the probabilities derived from the first order Markov model, that is each  $P(s_t, t)$ ,  $t > 1$  is set to the transition rate  $p(s_t | s_{t-1})$  estimated across sequences from the observations at positions  $t$  and  $t - 1$ . For  $t = 1$ , we set  $P(s_1, 1)$  to the observed frequency of the state  $s_1$  at position 1.

The likelihood  $P(s)$  being generally very small, we use  $-\log P(s)$  as sorting criterion. The latter quantity reaches its minimum for  $P(s)$  equal to 1, which leads to sort the sequences in ascending order of their score.

Use `criterion="dist"` and `nrep=1` to get the medoid and `criterion="density"` and `nrep=1` to get the densest sequence pattern.

For more details, see *Gabadinho & Ritschard, 2013*.

## Value

An object of class `stslist.rep`. This is actually a state sequence object (containing a list of state sequences) with the following additional attributes:

Scores	a vector with the representative score of each sequence in the original set given the chosen criterion.
Distances	a matrix with the distance of each sequence to its nearest representative.
Statistics	a data frame with quality measures for each representative sequence: number <i>na</i> of sequences attributed to the representative, number <i>nb</i> of sequences in the representative's neighborhood, mean distance <i>MD</i> to the representative and a few other indexes.
Quality	overall quality measure.

Print, plot and summary methods are available. More elaborated plots are produced by the `seqplot` function using the `type="r"` argument, or the `seqrplot` alias.

## Author(s)

Alexis Gabadinho (with Gilbert Ritschard for the help page)

## References

Gabadinho A, Ritschard G (2013). "Searching for typical life trajectories applied to child birth histories", In R Lévy, E. Widmer (eds.), *Gendered Life Courses*, pp. 287-312. Vienna: LIT.

Gabadinho A, Ritschard G, Studer M, Müller NS (2011). "Extracting and Rendering Representative Sequences", In A Fred, JLG Dietz, K Liu, J Filipe (eds.), *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 128 of *Communications in Computer and Information Science (CCIS)*, pp. 94-106. Springer-Verlag.

## See Also

`seqplot`, `plot.stslist.rep`, `dissrep`, `disscenter`

## Examples

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
"Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)
```

```
## Computing the distance matrix
costs <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", sm=costs)

## Representative set using the neighborhood density criterion
biofam.rep <- seqrep(biofam.seq, diss=biofam.om, criterion="density")
biofam.rep
summary(biofam.rep)
plot(biofam.rep)
```

---

seqsep

*Adds separators to sequences stored as character string*

---

## Description

Adds separators to sequences stored as character string.

## Usage

```
seqsep(seqdata, sl=1, sep="-")
```

## Arguments

seqdata	a dataframe or matrix containing sequence data, as vectors of states or events.
sl	the length of the states (the number of characters used to represent them). Default is 1.
sep	the character used as separator. Set by default as "-".

## See Also

[seqdecomp](#).

## Examples

```
seqsep("ABAAAAAD")
```

---

seqST	<i>Sequences turbulence</i>
-------	-----------------------------

---

### Description

Elzinga's turbulence for each sequence in a sequence data set.

### Usage

```
seqST(seqdata, norm=FALSE, silent=TRUE, with.missing=FALSE, type=1)
```

### Arguments

seqdata	a state sequence object as returned by the the <a href="#">seqdef</a> function.
norm	logical: should the turbulence index be normalized?
silent	logical: should messages about running operations (extracting dss and durations, computing turbulence) be displayed?
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.
type	either 1 or 2. Type of duration variance. The default type=1 ignores non visited states. Type 2 takes into account the 0-time spent in non-visited states (see <a href="#">seqivardur</a> ).

### Details

Sequence turbulence is a measure proposed by *Elzinga & Liefbroer (2007)*. It is based on the number  $\phi(x)$  of distinct subsequences that can be extracted from the distinct successive state (DSS) sequence and the variance of the consecutive times  $t_i$  spent in the distinct states. For a sequence  $x$ , the formula is

$$T(x) = \log_2\left(\phi(x) \frac{s_{t,max}^2(x) + 1}{s_t^2(x) + 1}\right)$$

where  $s_t^2(x)$  is the variance of the successive state durations in sequence  $x$  and  $s_{t,max}^2(x)$  is the maximum value that this variance can take given the number of spells and the total duration of the sequence. For type=1, this maximum is computed as

$$s_{t,max}^2 = (d - 1)(1 - \bar{t})^2$$

where  $\bar{t}$  is the mean consecutive time spent in the distinct states, i.e. the sequence duration  $t$  divided by the number  $d$  of distinct states in the sequence. For type=2, the variance takes into account the 0-time spent in non-visited states and the maximum is adjusted for the maximum number of non-visited states for the number of spells (see *Ritschard, 2021*).

When `with.missing=TRUE`, the function searches for missing states in the sequences and if found, adds the missing state to the alphabet for the computation of the turbulence. In this case the [seqdss](#)

and `seqdur` functions for extracting the distinct successive state sequences and the associated durations are called with the `{with.missing=TRUE}` argument. Thus, a missing state in a sequence is considered as the occurrence of an additional symbol of the alphabet and two or more consecutive missing states are considered as two or more occurrences of this additional state. E.g. the DSS of A-A-\*-\*-\*B-B-C-C-D is A-\*B-C-D and the associated durations are 2-3-2-2-1.

The normalized value is obtained by subtracting 1 to the index and then dividing by the resulting value for a sequence made by the successive repetition of the alphabet up to the maximal length in `seqdata` (Ritschard, 2021).

### Value

a single-column matrix of length equal to the number of sequences in `seqdata` containing the turbulence value of each sequence. Normalized values are returned when `norm=TRUE`.

### Author(s)

Alexis Gabadinho and Gilbert Ritschard

### References

Elzinga, Cees H. and Liefbroer, Aart C. (2007). De-standardization of Family-Life Trajectories of Young Adults: A Cross-National Comparison Using Sequence Analysis. *European Journal of Population*, 23, 225-250.

Ritschard, G. (2021), "Measuring the nature of individual sequences", *Sociological Methods and Research*, doi: [10.1177/00491241211036156](https://doi.org/10.1177/00491241211036156).

### See Also

[seqdss](#), [seqdur](#), [seqsubsn](#). For alternative measures of sequence complexity see [seqivolatility](#), [seqici](#), [seqindic](#).

### Examples

```
## Loading the 'actcal' example data set
data(actcal)
## Here we consider only the first 10 sequences
actcal <- actcal[1:10,]

## Defining a sequence object with data in columns 13 to 24
## (activity status from January to December 2000)
actcal.seq <- seqdef(actcal[,13:24], informat='STS')

## Computing the sequences turbulence
turb <- seqST(actcal.seq)

## Normalized turbulence
turb.norm <- seqST(actcal.seq, norm=TRUE)

## Normalized turbulence taking non-visited states into account.
turb2.norm <- seqST(actcal.seq, norm=TRUE, type=2)
```

---

seqstatd	<i>Sequence of transversal state distributions and their entropies</i>
----------	--

---

### Description

Returns the state frequencies, the number of valid states and the entropy of the state distribution at each position in the sequence.

### Usage

```
seqstatd(seqdata, weighted=TRUE, with.missing=FALSE, norm=TRUE)
```

### Arguments

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
weighted	if TRUE, distributions account for the weights assigned to the state sequence object (see <a href="#">seqdef</a> ). Set as FALSE if you want ignore the weights.
with.missing	If FALSE (default value), returned distributions ignore missing values.
norm	if TRUE (default value), entropy is normalized, ie divided by the entropy of the alphabet. Set as FALSE if you want the entropy without normalization.

### Details

In addition to the state distribution at each position in the sequence, the `seqstatd` function provides also for each time point the number of valid states and the Shannon entropy of the observed state distribution. Letting  $p_i$  denote the proportion of cases in state  $i$  at the considered time point, the entropy is

$$h(p_1, \dots, p_s) = - \sum_{i=1}^s p_i \log(p_i)$$

where  $s$  is the size of the alphabet. The log is here the natural (base e) logarithm. The entropy is 0 when all cases are in the same state and is maximal when the same proportion of cases are in each state. The entropy can be seen as a measure of the diversity of states observed at the considered time point. An application of such a measure (but with aggregated transversal data) can be seen in *Billari (2001)* and *Fussell (2005)*.

### Author(s)

Alexis Gabadinho (with Gilbert Ritschard for the help page)

### References

Billari, F. C. (2001). The analysis of early life courses: complex descriptions of the transition to adulthood. *Journal of Population Research* 18 (2), 119-24.

Fussell, E. (2005). Measuring the early adult life course in Mexico: An application of the entropy index. In R. Macmillan (Ed.), *The Structure of the Life Course: Standardized? Individualized? Differentiated?*, Advances in Life Course Research, Vol. 9, pp. 91-122. Amsterdam: Elsevier.

**See Also**

[plot.stslist.statd](#) the plot method for objects of class `stslist.statd`,  
[seqdplot](#) for higher level plot of transversal distributions and  
[seqHtplot](#) for plotting the transversal entropy over sequence positions.

**Examples**

```
data(biofam)
biofam.seq <- seqdef(biofam,10:25)
sd <- seqstatd(biofam.seq)
## Plotting the state distribution
plot(sd, type="d")

## Plotting the entropy indexes
plot(sd, type="Ht")

## =====
## example with weights
## =====
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## Unweighted
seqstatd(ex1.seq, weighted=FALSE)

seqstatd(ex1.seq, weighted=TRUE)
```

---

seqstatf

*State frequencies in the whole sequence data set*


---

**Description**

Overall frequency of each state of the alphabet in the state sequence object.

**Usage**

```
seqstatf(seqdata, weighted = TRUE, with.missing=FALSE)
```

**Arguments**

seqdata	a sequence object as defined by the <a href="#">seqdef</a> function.
weighted	Logical. Should frequencies account for weights when present in the state sequence object (see <a href="#">seqdef</a> ). Default is TRUE.
with.missing	Logical. Should non void missing states be treated as regular values? Default is FALSE.



**Details**

The seqstatf function computes the (weighted) count and frequency of each state of the alphabet in seqdata, i.e., the (weighted) sum of the occurrences of a state in seqdata.

**Value**

A data frame with as many rows as states in the alphabet and two columns, one for the count (Freq) and one for the percentage frequencies (Percent).

**Author(s)**

Alexis Gabadinho

**See Also**

[seqstatd](#) for the state distribution by time point (position), [seqistatd](#) for the state distribution within each sequence.

**Examples**

```
## Creating a sequence object from the actcal data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)

## States frequencies
seqstatf(actcal.seq)

## Example with weights
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## Unweighted
seqstatf(ex1.seq, weighted=FALSE)

## Weighted
seqstatf(ex1.seq, weighted=TRUE)
```

---

seqstatf

*List of distinct states or events (alphabet) in a sequence data set.*

---

**Description**

Returns a list containing distinct states or events found in a data frame or matrix containing sequence data, the alphabet.

**Usage**

```
seqstatf(data, var=NULL, format='STS')
```

**Arguments**

data	a data frame or matrix containing sequence data.
var	the list of columns containing the sequences. Default NULL means all columns. Whether the sequences are in the compressed (character strings) or extended format is automatically detected from the number of columns..
format	the format of the sequence data set. One of "STS", "SPS", "DSS". Default is "STS". The seqstat1 function uses the <a href="#">seqformat</a> function to translate between formats when necessary.

**Author(s)**

Alexis Gabadinho

**References**

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with the TraMineR package: A user's guide. Department of Econometrics and Laboratory of Demography, University of Geneva.

**See Also**

[seqformat](#)

**Examples**

```
data(actcal)
seqstat1(actcal, 13:24)
```

---

seqsubsn

*Number of distinct subsequences in a sequence.*

---

**Description**

Computes the number of distinct subsequences in a sequence using Elzinga's algorithm.

**Usage**

```
seqsubsn(seqdata, DSS=TRUE, with.missing=FALSE)
```

## Arguments

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
DSS	if TRUE, the sequences of Distinct Successive States (DSS, see <a href="#">seqdss</a> ) are first extracted (e.g., the DSS contained in 'D-D-D-D-A-A-A-A-A-A-D' is 'D-A-D'), and the number of distinct subsequences in the DSS is computed. If FALSE, the number of distinct subsequences is computed from sequences as they appear in the input sequence object. Hence the number of distinct subsequences is in most cases much higher with the DSS=FALSE option.
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.

## Details

The function first searches for missing states in the sequences and if found, adds the missing state to the alphabet for the extraction of the distinct subsequences. A missing state in a sequence is considered as the occurrence of an additional symbol of the alphabet, and two or more consecutive missing states are considered as two or more occurrences of the same state. The `with.missing=TRUE` argument is used for calling the [seqdss](#) function when `DSS=TRUE`.

## Value

Vector with the number of distinct subsequences for each sequence in the input state sequence object.

## Author(s)

Alexis Gabadinho (with Gilbert Ritschard for the help page)

## See Also

[seqdss](#).

## Examples

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Number of subsequences with DSS=TRUE
seqsubsn(actcal.seq[1:10,])

## Number of subsequences with DSS=FALSE
seqsubsn(actcal.seq[1:10,],DSS=FALSE)
```

---

seqtab	<i>Frequency table of the sequences</i>
--------	---

---

**Description**

Computes the frequency table of the sequences (count and percent of each sequence).

**Usage**

```
seqtab(seqdata, idxs = 1:10, weighted = TRUE, format = "SPS", tlim)
```

**Arguments**

seqdata	a sequence object as defined by the <code>seqdef</code> function.
idxs	returns the table for the sequences at ranks 'idxs' in the list of distinct sequences sorted in decreasing order of their frequencies. Default is 1:10, i.e. the 10 most frequent sequences. Can be any subset, like 5:10 (fifth to tenth most frequent sequences) or c(2, 10) (second and tenth most frequent sequences). Set idxs=0 to get the table for the whole set of distinct sequences.
weighted	if TRUE (default), frequencies account for the weights, if any, assigned to the state sequence object (see <code>seqdef</code> ). Set to FALSE for ignoring weights.
format	format used for displaying the rownames (the sequences) in the output table. Default is SPS format, which yields shorter and more readable sequence representations. Alternatively, "STS" may be specified.
tlim	Deprecated. Use idxs instead.

**Details**

The `weighted` argument has no effect when no weights were assigned to the state sequence object since weights default in that case to 1.

**Value**

An object of class `stslist.freq`. This is actually a state sequence object (containing a list of state sequences) with added attributes, among others the `freq` attribute containing the frequency table. There are `print` and `plot` methods for such objects. More sophisticated plots can be produced with the `seqplot` function.

**Author(s)**

Alexis Gabadinho (with Gilbert Ritschard for the help page)

**References**

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

**See Also**

[seqplot](#), [plot.stslist.freq](#).

**Examples**

```
## Creating a sequence object from the actcal data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)

## 10 most frequent sequences in the data
seqtab(actcal.seq)

## With idxs=0, we get all distinct sequences in the data set
## sorted in decreasing order of their frequency
stab <- seqtab(actcal.seq, idxs=0)
head(stab)
tail(stab)

## Example with weights
## from biofam data set using weights
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## Unweighted frequencies
seqtab(ex1.seq, weighted=FALSE)

## Weighted frequencies
seqtab(ex1.seq, weighted=TRUE)
```

---

seqtransn

*Number of transitions in a sequence*

---

**Description**

Computes the number of transitions (state changes) in each sequence of a sequence object.

**Usage**

```
seqtransn(seqdata, with.missing=FALSE, norm=FALSE, pweight=FALSE)
```

**Arguments**

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
with.missing	logical: should non-void missing values be treated as a regular state? If FALSE (default) missing values are ignored.
norm	logical. If set as TRUE, the number of transitions is divided by its theoretical maximum, the length of the sequence minus 1. When the length of the sequence is 1, the normalized value is set as 0.

`pweight` logical. EXPERIMENTAL! If set as TRUE, the count of transition weights each transition by its probability of transition in the data.

### Details

A transition in a sequence is a state change between time/position  $t$  and  $t + 1$ . For example, the sequence "A-A-A-A-B-B-A-D-D-D" contains 3 transitions. The maximum number of transitions a sequence can contain is  $\ell - 1$  where  $\ell$  is the length of the sequence. The number of transitions is obtained by subtracting 1 to the length of the the Distinct Successive State (DSS) sequence.

### Value

a state sequence object containing the number of transitions of each sequence in the object given as argument.

### Author(s)

Alexis Gabadinho (with Gilbert Ritschard for the help page)

### References

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

### See Also

[seqdss](#).

### Examples

```
## Creating a sequence object from columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Computing the number of transitions
actcal.trans <- seqtransn(actcal.seq)

## Displaying the DSS for the first 10 sequences
actcal.trans[1:10]

## Example with with.missing argument
data(ex1)
ex1.seq <- seqdef(ex1, 1:13)

seqtransn(ex1.seq)
seqtransn(ex1.seq, with.missing=TRUE)
```

---

seqtrate	<i>Compute transition rates between states</i>
----------	--

---

### Description

Returns a matrix with transition rates between states, computed from a set of sequences.

### Usage

```
seqtrate(seqdata, sel.states = NULL, time.varying = FALSE, weighted = TRUE,
         lag = 1, with.missing = FALSE, count = FALSE, stat1)
```

### Arguments

seqdata	a sequence object as defined by the <a href="#">seqdef</a> function.
sel.states	a list of states or events for which the transition rates will be computed. If omitted (default), transition rates are computed between the distinct states in seqdata (obtained with the <a href="#">alphabet</a> function).
time.varying	Logical. If TRUE, return an <a href="#">array</a> containing a distinct matrix for each time unit. The time is the third dimension (subscript).
weighted	Logical. If TRUE, compute transition rates using weights specified in seqdata.
lag	Integer. Time between the two states considered to compute transition rates (one by default).
with.missing	Logical. If FALSE (default value), returned transition rates ignore missing values.
count	Logical. Should counts of transition be returned instead of transition probabilities. Default is FALSE.
stat1	Deprecated. Use sel.states instead.

### Details

Transition rates are the probabilities of transition from one state to another observed in the sequence data. Substitution costs based on transition rates can be used when computing distances between sequences with the optimal matching method (see [seqdist](#)).

### Value

a matrix of dimension  $ns * ns$ , where  $ns$  is the number of states in the [alphabet](#) of the sequence object.

### Author(s)

Matthias Studer, Alexis Gabadinho, and Gilbert Ritschard

### References

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

**See Also**

[seqdist seqsubm alphabet.](#)

**Examples**

```
## Loading the 'actcal' example data set
data(actcal)

## Defining a sequence object with data in columns 13 to 24
## (activity status from January to December 2000)
actcal.seq <- seqdef(actcal[,13:24])

## Computing transition rates
seqrate(actcal.seq)

## Computing transition rates between states "A" and "B" only
seqrate(actcal.seq, c("A","B"))

## =====
## Example with weights
## =====
data(ex1)
ex1.seq <- seqdef(ex1[,1:13], weights=ex1$weights)

seqrate(ex1.seq, weighted=FALSE)
seqrate(ex1.seq, weighted=FALSE, count=TRUE)

## weights are accounted for by default
seqrate(ex1.seq)
seqrate(ex1.seq, count=TRUE)
```

---

seqtree

*Tree structured analysis of a state sequence object.*

---

**Description**

Facility for growing a regression tree for a state sequence object.

**Usage**

```
seqtree(formula, data = NULL, weighted = TRUE, min.size = 0.05,
max.depth = 5, R = 1000, pval = 0.01, weight.permutation = "replicate",
seqdist.args = list(method = "LCS", norm = "auto"), diss = NULL,
squared = FALSE, first = NULL, minSize, maxdepth, seqdist_arg)
```



**Arguments**

formula	a formula where the left hand side is a state sequence object (see <a href="#">seqdef</a> ) and the right hand specifies the candidate variables for partitioning the set of sequences.
weighted	Logical. If TRUE, use the weights of the state sequence object.
data	a data frame where variables in the formula will be searched
min.size	minimum number of cases in a node, in percentage if less than 1.
max.depth	maximum depth of the tree.
R	Number of permutations used to assess the significance of the split.
pval	Maximum p-value, in percent.
weight.permutation	Weights permutation method: "diss" (attach weights to the dissimilarity matrix), "replicate" (replicate case according to the weights arguments), "rounded-replicate" (replicate case according to the rounded weights arguments), "random-sampling" (random assignment of covariate profiles to the objects using distributions defined by the weights.)
seqdist.args	list of arguments directly passed to <a href="#">seqdist</a> , only used if diss=NULL
diss	An optional dissimilarity matrix. If not provided, a dissimilarity matrix is computed using <a href="#">seqdist</a> and seqdist.args
squared	Logical. If TRUE, the dissimilarity matrix is squared
first	Character. An optional variable name to force the first split.
minSize	Deprecated. Use min.size instead.
maxdepth	Deprecated. Use max.depth instead.
seqdist_arg	Deprecated. Use seqdist.args instead.

**Details**

The function provides a simplified interface for applying [disstree](#) on state sequence objects.

The seqtree objects can be "plotted" with [seqtreedisplay](#). A print method is also available which prints the medoid sequence for each terminal node.

**Value**

A seqtree object with same attributes as [disstree](#) objects.

The leaf membership is in the first column of the fitted attribute. For example, the leaf memberships for a tree dt are in `dt$fitted[,1]`.

**Author(s)**

Matthias Studer (with Gilbert Ritschard for the help page)

**References**

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2011). Discrepancy analysis of state sequences, *Sociological Methods and Research*, Vol. 40(3), 471-510, doi: [10.1177/0049124111415372](https://doi.org/10.1177/0049124111415372).

**See Also**

[seqtreedisplay](#), [disstree](#)

**Examples**

```
data(mvad)

## Defining a state sequence object
mvad.seq <- seqdef(mvad[, 17:86])

## Growing a seqtree from Hamming distances:
## Warning: The R=10 used here to save computation time is
## much too small and will generate strongly unstable results.
## We recommend to set R at least as R=1000.
## To comply with this small R value, we set pval = 0.1.
seqt <- seqtree(mvad.seq~ male + Grammar + funemp + gcse5eq + fmpr + livboth,
  data=mvad, R=10, pval=0.1, seqdist.arg=list(method="HAM", norm="auto"))
print(seqt)

## Growing a seqtree from an existing distance matrix
mvad.dhd <- seqdist(mvad.seq, method="DHD")
seqt <- seqtree(mvad.seq~ male + Grammar + funemp + gcse5eq + fmpr + livboth,
  data=mvad, R=10, pval=0.1, diss=mvad.dhd)
print(seqt)

### Following commands only work if GraphViz is properly installed
## Not run:
seqtreedisplay(seqt, type="d", border=NA)
seqtreedisplay(seqt, type="I", sortv=cmdscale(mvad.dhd, k=1))

## End(Not run)
```

---

seqtreedisplay

*Graphical rendering of a sequence regression tree*

---

**Description**

Generate a graphical representation of a regression tree of state sequence data.

**Usage**

```
seqtreedisplay(tree, filename = NULL, seqdata = tree$info$object,
  only.leaf = FALSE, sortv = NULL, diss = NULL, cex.main = 3,
  with.legend = "auto", cex.legend = cex.main, axes = FALSE,
  image.format = "png", with.quality = TRUE, cex.quality = cex.main,
  legend.text = NULL, show.tree = TRUE, show.depth = FALSE,
  imgLeafOnly, dist.matrix, title.cex, withlegend, legend.fontsize,
```

```
imageformat, withquality, quality.fontsize, legendtext, showtree,
showdepth, ...)
```

```
disstreedisplay(tree, filename = NULL, image.data= NULL, image.fun = plot,
  only.leaf = FALSE, cex.main = 3, image.format = "png",
  with.quality = TRUE, cex.quality = cex.main,
  legend.text = NULL, show.tree = TRUE, show.depth = FALSE,
  imagedata, imagefunc, imgLeafOnly, title.cex, imageformat,
  withquality, quality.fontsize, legendtext, showtree, showdepth, ...)
```

## Arguments

tree	A seqtree object (as produced by <a href="#">seqtree</a> ) for seqtreedisplay. A disstree object (as produced by <a href="#">disstree</a> ) for disstreedisplay.
filename	The name of a file where to save the plot (overwriting existing file). If NULL, a temporary file is created.
seqdata	The sequence object containing the state sequences plotted in the nodes.
only.leaf	Logical. If TRUE sequences are plotted only in terminal nodes.
sortv	Argument passed to <a href="#">seqplot</a>
diss	Argument passed to <a href="#">seqplot</a>
cex.main	Node title magnification. See <a href="#">par</a> .
with.legend	Logical. Should the color legend be displayed on the plot?
cex.legend	Legend magnification. See <a href="#">par</a> . If not specified, use the value of cex.main.
axes	Argument passed to <a href="#">seqplot</a>
image.format	Image format of the output file (filename)
with.quality	If TRUE, a node displaying fitting measures of the tree is added to the plot.
cex.quality	Fitting measure text magnification. See <a href="#">par</a> . If not specified, use the value of cex.main.
legend.text	Character. Optional text information that should be added.
show.tree	Logical. Should the tree be shown on the screen?
show.depth	Logical. If TRUE, the splits are ordered according to their global pseudo-R2.
image.fun	A function to plot the individuals in a node, see details.
image.data	a data.frame that will be passed to image.fun.
imgLeafOnly	Deprecated. Use only.leaf instead.
dist.matrix	Deprecated. Use diss instead.
title.cex	Deprecated. Use cex.main instead.
withlegend	Deprecated. Use with.legend instead.
legend.fontsize	Deprecated. Use cex.legend instead.
imageformat	Deprecated. Use image.format instead.
withquality	Deprecated. Use with.quality instead.

quality.fontsize	Deprecated. Use cex.quality instead.
legendtext	Deprecated. Use legend.text instead.
showtree	Deprecated. Use show.tree instead.
showdepth	Deprecated. Use show.depth instead.
imagedata	Deprecated. Use image.data instead.
imagefunc	Deprecated. Use image.fun instead.
...	additional arguments passed to seqplot

### Details

This function generates a tree image. For each node, it invokes [seqplot](#) for the selected lines of seqdata as argument. You should at least specify the type of the plot to use (type="d" for instance, see [seqplot](#) for more details).

The plot is actually not generated as an R plot, but with GraphViz ([www.graphviz.org](http://www.graphviz.org)). Hence, seqtreedisplay only works when GraphViz is correctly installed.

Conversion to image formats other than "jpeg" or "png" is done using ImageMagick ([www.imagemagick.org](http://www.imagemagick.org)). To use this feature, ImageMagick ([www.imagemagick.org](http://www.imagemagick.org)) should hence also be installed.

### Value

None

### Author(s)

Matthias Studer (with Gilbert Ritschard for the help page)

### See Also

See [seqtree](#) and [disstree](#) for examples, and [disstree2dot](#) for generating "dot" files.

---

stlab

*Get or set the state labels of a sequence object*

---

### Description

This function gets or sets the state labels of a sequence object, that is, the long labels used when displaying the state legend in plotting functions.

### Usage

```
stlab(seqdata)
stlab(seqdata) <- value
```

## Arguments

seqdata	a state sequence object as defined by the <a href="#">seqdef</a> function.
value	a vector of character strings containing the labels, of length equal to the number of states in the alphabet. Each string is attributed to the corresponding state in the alphabet, the order being the one returned by the <a href="#">alphabet</a> .

## Details

The state legend is plotted either automatically by the plot functions provided for visualizing sequence objects or with the [seqlegend](#) function. A long label is associated to each state of the alphabet and displayed in the legend. The state labels are defined when creating the sequence object, either automatically using the values found in the data or by specifying a user defined vector of labels. The `stlab` function can be used to get or set the state labels of a previously defined sequence object.

## Value

For 'stlab' a vector containing the labels.

For 'stlab<-' the updated sequence object.

## See Also

[seqdef](#)

## Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
## The color palette is automatically set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the color palette
stlab(actcal.seq)
seqiplot(actcal.seq)

## Changing the state labels
stlab(actcal.seq) <- c("Full time","Part time (19-36 hours)",
  "Part time (1-18 hours)", "No work")
seqiplot(actcal.seq)
```

**Description**

Checks the presence of deprecated arguments, assigns value of a deprecated argument to the corresponding new argument name, and issues warning messages.

**Usage**

```
TraMineR.check.depr.args(arg.pairs)
```

**Arguments**

<code>arg.pairs</code>	List of pairs of old and new argument names (e.g. <code>alist(newname1 = oldname1, newname2 = oldname2)</code> )
------------------------	---

**Details**

To be used inside functions. For developers only.

For each specified pair of new and old argument names, the function checks if the old argument name is specified. If so and the new one is not, a warning message is raised and the argument value is assigned to the new argument name. If one of the names declared in `check.depr.args()` arguments is not an argument of the parent function or if both the new and old argument names are specified an error is raised.

The function does not detect when the new and the old argument names are specified together and the new argument value is its default value. In this case, the value associated with the old argument name is assigned to the new name and a warning message is raised.

The function works whether the argument names are explicitly declared or not in the call to the checked function.

The only requirement for the function to work is that the deprecated arguments should be listed WITHOUT default values in the definition of the checked function.

**Value**

None.

**Author(s)**

Pierre-Alexandre Fonta, Gilbert Ritschard

---

TraMineR.checkupdates *Check for TraMineR updates*

---

**Description**

Check if the installed version of TraMineR is up-to-date. This function only prints a message and does not need any argument. It connects to the TraMineR webserver (<http://traminer.unige.ch>).

**Usage**

```
TraMineR.checkupdates()
```

**Value**

Return your current version number of TraMineR and the latest stable and development version number if more recent versions are available.

**Author(s)**

Nicolas S. Müller

---

TraMineRInternal      *Access to TraMineR internal functions*

---

**Description**

Functions allowing other packages to access some TraMineR internal functions. Corresponding functions are respectively TraMineR.setlayout, TraMineR.Legend, DTNInit, seageage, seqgbar, DTNsplit, and tmrWeightedInertiaDist. For experts only.

**Usage**

```
TraMineRInternalLayout(...)  
TraMineRInternalLegend(...)  
TraMineRInternalNodeInit(...)  
TraMineRInternalSeageage(...)  
TraMineRInternalSeqgbar(...)  
TraMineRInternalSplitInit(...)  
TraMineRInternalWeightedInertiaDist(diss, diss.size, is.dist, individuals, sweights, var)
```

**Arguments**

...	Arguments passed to or from other methods.
diss	See tmrWeightedInertiaDist().
diss.size	See tmrWeightedInertiaDist().
is.dist	See tmrWeightedInertiaDist().
individuals	See tmrWeightedInertiaDist().
sweights	See tmrWeightedInertiaDist().
var	See tmrWeightedInertiaDist().

# Index

## \* Data handling

- read.tda.mdist, 49
- seqcomp, 51
- seqconc, 52
- seqdecomp, 57
- seqdef, 58
- seqcreate, 82
- seqetm, 88
- seqfind, 91
- seqformat, 92
- seqgen, 98
- seqnum, 123
- seqrecode, 143
- seqsep, 148
- seqstatl, 153

## \* Datasets

- actcal, 5
- actcal.tse, 6
- bfspell, 8
- biofam, 9
- ex1, 30
- ex2, 31
- famform, 32
- mvad, 33

## \* Dissimilarity measures

- seqcost, 53
- seqdist, 64
- seqdistmc, 71
- seqLLCP, 116
- seqLLCS, 117
- seqmpos, 122

## \* Dissimilarity-based analysis

- dissassoc, 12
- disscenter, 14
- dissdomassoc, 16
- dissmfacw, 18
- disrep, 20
- disstree, 23
- disstree2dot, 25

- disstreeleaf, 28

- dissvar, 29

- plot.seqdiff, 35

- seqalign, 50

- seqdiff, 62

- seqrep, 145

- seqtree, 160

- seqtreedisplay, 162

## \* Event sequences

- plot.subseqelist, 46

- plot.subseqelistchisq, 47

- sequeapplysub, 76

- seqecmpgroup, 77

- sequeconstraint, 79

- sequecontain, 81

- seqcreate, 82

- seqefsub, 84

- seqeid, 87

- seqelength, 87

- seqetm, 88

- seqeweight, 90

- seqpcplot, 124

## \* Global characteristics

- seqmeant, 119

- seqstatf, 152

- seqtrate, 159

## \* Longitudinal characteristics

- seqdss, 73

- seqdur, 75

- seqelength, 87

- seqfpos, 97

- seqibad, 99

- seqici, 100

- seqient, 102

- seqindic, 103

- seqintegr, 106

- seqipos, 108

- seqistatd, 110

- seqivardur, 111



- seqivolatility, 112
- seqlength, 115
- seqlogp, 118
- seqprecarity, 136
- seqST, 149
- seqsubsn, 154
- seqtransn, 157
- \* **Method**
  - plot.seqdiff, 35
  - plot.stslist, 36
  - plot.stslist.freq, 38
  - plot.stslist.meant, 40
  - plot.stslist.modst, 41
  - plot.stslist.rep, 43
  - plot.stslist.statd, 44
  - plot.subseqelist, 46
  - plot.subseqelistchisq, 47
  - print.stslist, 48
- \* **Plot**
  - disstree2dot, 25
  - seqlegend, 114
  - seqpcplot, 124
  - seqplot, 129
  - seqtreedisplay, 162
- \* **Sequence analysis**
  - seqprecstart, 141
- \* **Sequence-object attributes**
  - alphabet, 7
  - cpal, 11
  - seqdim, 64
  - seqeid, 87
  - seqeweight, 90
  - stlab, 164
- \* **State sequences**
  - is.stslist, 32
  - plot.stslist, 36
  - print.stslist, 48
  - seqdef, 58
  - seqfind, 91
  - seqgen, 98
  - seqistatd, 110
  - seqlogp, 118
  - seqnum, 123
  - seqpm, 135
  - seqstatf, 152
- \* **Transversal characteristics**
  - seqmodst, 121
  - seqstatd, 151
  - seqtab, 156
- \* **package**
  - TraMineR-package, 4
  - TraMineR.checkupdates, 166
- actcal, 5, 6
- actcal.tse, 6
- alphabet, 7, 11, 54, 97, 124, 159, 160, 165
- alphabet<- (alphabet), 7
- array, 54, 159
- barplot, 46, 48
- bfpdata20 (bfspell), 8
- bfspell, 8
- bfspell20 (bfspell), 8
- biofam, 8, 9, 9
- colors, 11, 59
- cpal, 11
- cpal<- (cpal), 11
- daisy, 54
- dissassoc, 12, 15, 19, 20, 24, 30, 62, 63
- disscenter, 14, 14, 20, 22, 24, 30, 147
- dissdomassoc, 16
- dissmfacw, 12–15, 18, 24, 30
- dissrep, 20, 147
- disstree, 14, 15, 20, 23, 28–30, 161–164
- disstree2dot, 23, 25, 164
- disstree2dotp (disstree2dot), 25
- disstreedisplay, 24, 28
- disstreedisplay (seqtreedisplay), 162
- disstreeleaf, 28
- dissvar, 14, 15, 20, 24, 29
- dist, 12, 14, 20, 29, 66, 71
- ex1, 30
- ex2, 31
- famform, 32
- getwd, 28
- gower\_matrix (dissmfacw), 18
- hist.dissassoc (dissassoc), 12
- is.seqelist, 86
- is.stslist, 32, 60
- is.subseqelist (seqefsub), 84
- layout, 131

- legend, [35](#), [114](#), [115](#), [131](#)
- lines, [35](#)
- mvad, [33](#)
- order, [83](#)
- par, [37](#), [39](#), [40](#), [42](#), [43](#), [45](#), [46](#), [50](#), [115](#), [125](#), [126](#), [130](#), [131](#), [163](#)
- pdf, [37](#), [132](#)
- plot.eseq (seqpcplot), [124](#)
- plot.seqalign (seqalign), [50](#)
- plot.seqdiff, [35](#)
- plot.seqelist (seqpcplot), [124](#)
- plot.stslist, [36](#), [48](#), [60](#), [131–133](#)
- plot.stslist.freq, [38](#), [131–133](#), [157](#)
- plot.stslist.meant, [40](#), [120](#), [132](#), [133](#)
- plot.stslist.modst, [41](#), [121](#), [132](#), [133](#)
- plot.stslist.rep, [43](#), [132](#), [133](#), [145](#), [147](#)
- plot.stslist.statd, [44](#), [131–133](#), [152](#)
- plot.subseqelist, [46](#), [86](#)
- plot.subseqelistchisq, [47](#), [78](#)
- png, [37](#), [132](#)
- postscript, [37](#), [132](#)
- print, [111](#)
- print.dissassoc (dissassoc), [12](#)
- print.dissmultifactor (dissmfacw), [18](#)
- print.disstree (disstree), [23](#)
- print.seqalign (seqalign), [50](#)
- print.seqdiff (seqdiff), [62](#)
- print.seqeconstraint (seqeconstraint), [79](#)
- print.seqidegrad (seqprecarity), [136](#)
- print.seqivardur (seqivardur), [111](#)
- print.seqprec (seqprecarity), [136](#)
- print.stslist, [48](#), [60](#)
- print.stslist.freq (seqtab), [156](#)
- print.subseqelist (seqefsub), [84](#)
- rbind.stslist (seqdef), [58](#)
- read.tda.mdist, [49](#)
- recodef (seqrecode), [143](#)
- rgb, [11](#)
- rownames, [37](#)
- runif, [98](#)
- seqalign, [50](#)
- seqcomp, [51](#)
- seqconc, [52](#), [57](#), [92](#)
- seqcost, [53](#), [65](#), [68](#), [69](#), [71](#), [72](#)
- seqdecomp, [53](#), [57](#), [92](#), [148](#)
- seqdef, [4](#), [7](#), [8](#), [11](#), [27](#), [33](#), [35–37](#), [40](#), [48](#), [50](#), [51](#), [53](#), [54](#), [56](#), [58](#), [62](#), [65](#), [66](#), [68](#), [69](#), [71–75](#), [82](#), [83](#), [89](#), [91](#), [94](#), [95](#), [97](#), [99](#), [100](#), [102](#), [104](#), [106](#), [108](#), [110](#), [111](#), [113](#), [114](#), [116](#), [118](#), [120](#), [121](#), [123–125](#), [127](#), [130](#), [131](#), [136](#), [137](#), [141](#), [143–145](#), [149](#), [151](#), [152](#), [155–157](#), [159](#), [161](#), [165](#)
- seqdiff, [35](#), [36](#), [62](#)
- seqdim, [64](#)
- seqdist, [14](#), [50](#), [51](#), [53](#), [56](#), [62](#), [64](#), [71](#), [72](#), [117](#), [145](#), [146](#), [159–161](#)
- seqdistmc, [69](#), [71](#)
- seqdplot, [152](#)
- seqdplot (seqplot), [129](#)
- seqdss, [73](#), [75](#), [149](#), [150](#), [155](#), [158](#)
- seqdur, [74](#), [75](#), [112](#), [150](#)
- seqeappliesub, [76](#), [79](#), [80](#), [83](#), [85](#), [86](#)
- seqecmpgroup, [47](#), [48](#), [77](#), [83](#), [90](#)
- seqeconstraint, [76](#), [78](#), [79](#), [85](#)
- seqecontain, [81](#)
- seqecreate, [4](#), [7](#), [8](#), [60](#), [77](#), [81](#), [82](#), [86](#), [87](#), [89](#), [125](#), [127](#)
- seqefsub, [47](#), [76](#), [78–81](#), [83](#), [84](#), [90](#)
- seqeid, [87](#)
- seqelength, [83](#), [87](#)
- seqelength<- (seqelength), [87](#)
- seqetm, [82](#), [83](#), [88](#), [92](#)
- seqeweight, [83](#), [85](#), [90](#)
- seqeweight<- (seqeweight), [90](#)
- seqfcheck, [58](#), [92](#)
- seqfind, [51](#), [91](#)
- seqformat, [6](#), [8](#), [58](#), [60](#), [83](#), [89](#), [92](#), [154](#)
- seqfplot (seqplot), [129](#)
- seqfpos, [51](#), [97](#)
- seqgen, [98](#)
- seqHtplot, [152](#)
- seqHtplot (seqplot), [129](#)
- seqibad, [99](#), [104](#), [105](#), [140](#)
- seqici, [100](#), [103–105](#), [113](#), [138](#), [140](#), [150](#)
- seqidegrad, [99](#), [104](#), [105](#)
- seqidegrad (seqprecarity), [136](#)
- seqient, [101](#), [102](#), [104](#), [105](#)
- seqindic, [101](#), [103](#), [103](#), [107](#), [109](#), [112](#), [113](#), [140](#), [150](#)
- seqinsecurity, [104](#), [105](#)

- seqinsecurity (seqprecarity), 136
- seqintegr, 99, 106, 108, 109, 113
- seqintegration, 99
- seqintegration (seqintegr), 106
- seqIplot (seqplot), 129
- seqiplot (seqplot), 129
- seqipos, 101, 104, 105, 107, 108, 113
- seqistatd, 110, 153
- seqivardur, 104, 105, 111, 149
- seqivolatility, 101, 104, 105, 107–109, 112, 150
- seqlegend, 114, 165
- seqlength, 104, 105, 115
- seqLLCP, 116, 122
- seqLLCS, 117, 122
- seqlogp, 118
- seqmeant, 40, 41, 119, 132
- seqmodst, 42, 121, 132
- seqmpos, 122
- seqmsplot (seqplot), 129
- seqmtpplot, 41, 120
- seqmtpplot (seqplot), 129
- seqnum, 123
- seqpcfilter (seqpcplot), 124
- seqpcplot, 124, 132, 133
- seqplot, 27, 28, 37–39, 41, 42, 44, 45, 60, 120, 121, 124, 127, 129, 145, 147, 157, 163, 164
- seqpm, 51, 135
- seqprecarity, 99, 104, 105, 136, 142
- seqprecorr (seqprecarity), 136
- seqprestart, 99, 137, 139, 140, 141
- seqrecode, 143
- seqrep, 22, 43, 44, 132, 133, 145
- seqrplot, 27, 44, 147
- seqrplot (seqplot), 129
- seqsep, 148
- seqsplot, 133
- seqST, 101, 103–105, 113, 149
- seqstatd, 44, 45, 103, 131, 132, 151, 153
- seqstatf, 152
- seqstatl, 58, 60, 153
- seqsubm, 68, 69, 160
- seqsubm (seqcost), 53
- seqsubs, 150, 154
- seqtab, 39, 131, 132, 156
- seqtransn, 104, 105, 157
- seqtrate, 54, 56, 159
- seqtree, 23, 24, 28, 160, 163, 164
- seqtree2dot (disstree2dot), 25
- seqtreedisplay, 24, 28, 161, 162, 162
- setwd, 28
- stlab, 164
- stlab<- (stlab), 164
- str.seqelist, 85
- summary.ddomassoc (disssdomassoc), 16
- summary.stslist (seqdef), 58
- title, 26, 45
- TraMineR (TraMineR-package), 4
- TraMineR-package, 4
- TraMineR.check.depr.args, 165
- TraMineR.checkupdates, 166
- TraMineRInternal, 167
- TraMineRInternalLayout (TraMineRInternal), 167
- TraMineRInternalLegend (TraMineRInternal), 167
- TraMineRInternalNodeInit (TraMineRInternal), 167
- TraMineRInternalSeage (TraMineRInternal), 167
- TraMineRInternalSeqgbar (TraMineRInternal), 167
- TraMineRInternalSplitInit (TraMineRInternal), 167
- TraMineRInternalWeightedInertiaDist (TraMineRInternal), 167
- weighted.rank, 17