Package ‘codyna’

January 16, 2026

Title Complex Dynamic Systems

Version 0.1.0

Description Performs analysis of complex dynamic systems with a focus on the temporal unfolding of patterns, changes, and state transitions in behavioral data. Supports both time series and sequence data and provides tools for the analysis and visualization of complexity, pattern identification, trends, regimes, sequence typology as well as early warning signals.

License MIT + file LICENSE

URL `https://github.com/santikka/codyna/`

BugReports `https://github.com/santikka/codyna/issues/`

Depends R (>= 4.1.0)

Imports checkmate, cli, dplyr, ggplot2, patchwork, rlang, stats, tibble, tidyr, tidyselect

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

LazyData true

NeedsCompilation no

Author Santtu Tikka [aut, cre], Mohammed Saqr [aut]

Maintainer Santtu Tikka <`santtuth@gmail.com`>

Repository CRAN

Date/Publication 2026-01-16 11:00:13 UTC

Contents

`codyna-package` *The* `codyna` *Package.*

Description

Performs analysis of complex dynamic systems with a focus on the temporal unfolding of patterns, changes, and state transitions in behavioral data. The package supports both time series and sequence data and provides tools for the analysis and visualization of complexity, pattern identification, trends, regimes, sequence typology as well as early warning signals.

Author(s)

Santtu Tikka and Mohammed Saqr

See Also

Useful links:

- `https://github.com/santikka/codyna/`
- Report bugs at `https://github.com/santikka/codyna/issues/`

`complexity` *Calculate Dynamic Complexity Measures for Time-Series Data*

Description

Computes dynamic complexity and other rolling window measures for univariate time series data.

Usage

```
complexity(data, measures = "complexity", window = 7L, align = "center")
```

Arguments

`data`	`[ts, numeric()]` Univariate time series data.
`measures`	`[character()]` A vector of measures to calculate. See 'Details' for more information on the available measures.
`window`	`[integer(1)]` A positive `integer` specifying the rolling window size. Must be at least `2` (default: `7`).
`align`	`[character(1)]` Alignment of the window. The available options are: `"center"` (default), `"right"`, and `"left"`. The calculated measure is assigned to the center, rightmost, or leftmost point of the window, respectively.

Details

The following measures can be calculated:

- `"complexity"`: Product of fluctuation and distribution measures.
- `"fluctuation"`: Root mean square of successive differences.
- `"distribution"`: Deviation from uniform distribution.
- `"autocorrelation"`: Lag-1 autocorrelation coefficient.
- `"max"`: Rolling maximum.
- `"min"`: Rolling minimum.
- `"variance"`: Rolling variance.

The option `"all"` computes all of the above.

Value

A `tibble` with the time index, the original time-series data, and the calculated measures.

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)

# Single measure
comp_single <- complexity(ts_data, measures = "complexity")

# Multiple measures
comp_multi <- complexity(ts_data, measures = c("complexity", "variance"))
```

`convert` *Convert Sequence Data to Various Formats*

Description

Converts wide format sequence data into useful formats for analysis, such as frequency table, one-Hot encoding, or edge list (graph format).

Usage

```
convert(data, cols, format = "frequency")
```

Arguments

`data`	[`data.frame`, `matrix`, `stslist`] Sequence data in wide format (rows are sequences, columns are time points).
`cols`	[`expression`] A tidy selection of columns that should be considered as sequence data. By default, all columns are used.
`format`	[`character(1)`] The format to convert into: • `"frequency"`: Counts of each state per sequence. • `"onehot"`: Presence/absence (0/1) of each state per sequence. • `"edgelist"`: (state, next state) pairs. • `"reverse"`: Same as `"edgelist"` but in the reverse direction, i.e., (state, previous state) pairs.

Value

A `tibble` structured according to the requested format.

Examples

```
convert(engagement, format = "frequency")
convert(engagement, format = "onehot")
convert(engagement, format = "edgelist")
convert(engagement, format = "reverse")
```

`detect_regimes` *Regime Detection for Time Series Data*

Description

Detects regime changes in time series data using multiple methods including cumulative peaks, changepoint detection, variance shifts, threshold analysis, gradient changes, and entropy analysis.

Usage

```
detect_regimes(
  data,
  method = "smart",
  sensitivity = "medium",
  min_change,
  window = 10,
  peak = 2,
  cumulative = 0.6
)
```

Arguments

`data`
: `[ts, numeric()]`
Univariate time series data.

`method`
: `[character(1)]`
Detection method. The available options are:
  - `"cumulative_peaks"`: Detects cumulative complexity peaks using Z-tests.
  - `"changepoint"`: Change point detection (multi-window mean-shift test).
  - `"threshold"`: Adaptive quartile-based regime classification.
  - `"variance_shift"`: Detects changes in variance patterns.
  - `"slope"`: Detects changes in local slope (rolling linear models).
  - `"entropy"`: Detects changes in the Shannon entropy of the complexity series, calculated in rolling windows.
  - `"smart"` (default): Combines gradient, peaks, and changepoint methods.
  - `"all"`: Applies all individual methods listed above and uses ensemble voting.

`sensitivity`
: `[character(1)]`
Detection sensitivity level. The available options are: `"low"`, `"medium"`, `"high"`. The default is `"medium"`. This affects thresholds and window sizes within the detection methods.

`min_change`
: `[integer(1)]`
Minimum number of observations between changes. If missing (default), the value is determined automatically (typically 10% of observations, minimum of 10).

`window`	`[integer(1)]` base window size for rolling calculations. This is further adjusted by `sensitivity`. The default is `10`.
`peak`	`[numeric(1)]` Base z-score threshold for individual peak detection with the `"cumulative_peaks"` method. Adjusted by `sensitivity`. The default is `2.0`.
`cumulative`	`[numeric(1)]` A value between 0 and 1 that defines the base proportion threshold for identifying cumulative peak regions. Adjusted by `sensitivity`. The default is `0.6`.

Value

An object of class `regimes` which is a `tibble` containing the following columns:

`value`: Original time series data. `time`: Original time points. `change`: A `logical` vector indicating regime changes. `id`: An `integer` regime identifier. `type`: Type of change detected by the method. `magnitude`: Magnitude of the change (method-specific interpretation) `confidence`: Confidence in the detection (method-specific interpretation, typically between `0` and `1`, or `NA`) `stability`: Categorical stability: `"Stable"`, `"Transitional"`, and `"Unstable"`. `score`: A `numeric` stability score between `0` and `1`.

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)
regimes <- detect_regimes(
  data = ts_data,
  method = "threshold",
  sensitivity = "medium"
)
```

`detect_warnings` *Detect Early Warning Signals in a Time Series*

Description

Detect Early Warning Signals in a Time Series

Usage

```
detect_warnings(
  data,
  method = "rolling",
  metrics = "all",
  window = 0.5,
  burnin = 0.1,
  demean = TRUE,
  detrend = "none",
  threshold = 2,
  consecutive = 2L,
  bandwidth,
  span,
  degree
)
```

Arguments

`data`
: `[ts, numeric()]`
  Univariate time series data.

`method`
: `[character(1)]`
  Name of the analysis method. Either `"rolling"` or `"expanding"` for rolling window and expanding window, respectively.

`metrics`
: `[character(1)]`
  Names of the EWS metrics to compute. The default is `"all"` computing all metrics. The available options are:
  - `"ar1"`: The autoregressive coefficient of an AR1 model.
  - `"sd"`: Standard deviation.
  - `"skew"`: Skewness.
  - `"kurt"`: Kurtosis.
  - `"cv"`: Coefficient of variation.
  - `"rr"`: Return rate (`1 - ar1`).
  - `"all"`: All of the above.

`window`
: `[numeric(1)]`
  Window size as a proportion of the total series length (default `0.5`).

`burnin`
: `[numeric(1)]`
  Burn-in period as a proportion of the total series length (default `0.1`).

`demean`
: `[logical(1)]`
  Should the time series be demeaned before analysis? If `TRUE` (the default), the `"ar1"` metric will be based on an AR1 model where the mean of the observations is first subtracted. See `stats::ar.ols()` for details.

`detrend`
: `[character(1)]`
  Name of the detrending method to apply to the time series data before computing the metrics. The default is `"none"` for no detrending. The available options are:
  - `"gaussian"`: Estimates a smooth curve via kernel-based regression using `stats::ksmooth()` with a Gaussian kernel which is then subtracted from the time series.
  - `"loess"`: Estimates a smooth curve via local polynomial regression using `stats::loess()` which is then subtracted from the time series.
  - `"linear"`: Fits a linear regression model via `stats::lm()` and uses the residuals for computing the metrics.
  - `"first-diff"`: Uses the differences between the time series and its first-order lagged values.

	• "none": Use the original time series data.
threshold	[numeric(1)] The z-score threshold value for the expanding window method. The default is 2.0.
consecutive	[integer(1)] The number of times the threshold has to be crossed consecutively to be counted as a detection. The default is 2.
bandwidth	See stats::ksmooth().
span	See stats::loess().
degree	See stats::loess().

Value

An object of class ews containing the EWS results as a tibble.

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)

# Rolling window (default)
ews_roll <- detect_warnings(ts_data)

# Expanding window
ews_exp <- detect_warnings(ts_data, method = "expanding")
```

---

discover_patterns *Discover Sequence Patterns*

Description

Discovering various types of patterns in sequence data. Provides n-gram extraction, gapped pattern discovery, analysis of repeated patterns and targeted pattern search.

Usage

```
discover_patterns(
  data,
  type = "ngram",
  pattern,
  len = 2:5,
  gap = 1:3,
  min_support = 0.01,
  min_count = 2,
  start,
  end,
  contains
)
```

Arguments

`data`	`[data.frame, matrix, stslist]` Sequence data in wide format (rows are sequences, columns are time points).
`type`	`[character(1)]` Type of pattern analysis: • `"ngram"`: Extract contiguous n-grams. • `"gapped"`: Discover patterns with gaps/wildcards. • `"repeated"`: Detect repeated occurrences of the same state.
`pattern`	`[character(1)]` Specific pattern to search for as a character string (e.g., `"A->*->B"`). If provided, `type` is ignored. Supports wildcards: `*` (single) and `**` (multi-wildcard).
`len`	`[integer()]` Pattern lengths to consider for n-grams and repeated patterns (default: `2:5`).
`gap`	`[integer()]` Gap sizes to consider for gapped patterns (default: `1:3`).
`min_support`	`[integer(1)]` Minimum support threshold, i.e., the proportion of sequences that must contain a specific pattern for it to be included (default: `0.01`).
`min_count`	`[integer(1)]` Minimum count threshold, i.e., the numbers of times a pattern must occur across all sequences for it to be included (default: `2`).
`start`	`[character(1)]` Filter patterns starting with these states.
`end`	`[character(1)]` Filter patterns ending with these states.
`contains`	`[character(1)]` Filter patterns containing these states.

Value

A `tibble` containing the discover patterns, counts, proportions, and support.

Examples

```
# N-grams
ngrams <- discover_patterns(engagement, type = "ngram")

# Gapped patterns
gapped <- discover_patterns(engagement, type = "gapped")

# Repeated patterns
repeated <- discover_patterns(engagement, type = "repeated")

# Custom pattern with a wildcard state
custom <- discover_patterns(engagement, pattern = "Active->*")
```

`ema` *Ecological Momentary Assessment (EMA) Data*

Description

Example data for complex adaptive systems perspective to behavior change research. The dataset consists of 20 individuals with 9 self-report variables (and time of response) each. For more information on the data, please see `https://heinonmatti.github.io/complexity-behchange/dataset-info.html`

Usage

```
ema
```

Format

A `data.frame` object.

Source

`https://github.com/heinonmatti/complexity-behchange`

`engagement` *Example Data on Student Engagement*

Description

Students' engagement states (Active / Average / Disengaged) throughout a whole study program. The data was generated synthetically based on the article "The longitudinal association between engagement and achievement varies by time, students' profiles, and achievement state: A full program study". Used also in the `tna` package.

Usage

```
engagement
```

Format

An `stslist` object (sequence data).

Source

doi:10.1016/j.compedu.2023.104787

References

Tikka S, López-Pernas S, Saqr M (2025). "tna: An R Package for Transition Network Analysis." *Applied Psychological Measurement*. doi:10.1177/01466216251348840

`group_regulation` *Example Data on Group Regulation*

Description

Students' regulation during collaborative learning. Students' interactions were coded as: "adapt", "cohesion", "consensus", "coregulate", "discuss", "emotion", "monitor", "plan", "synthesis". Used also in the `tna` package.

Usage

```
group_regulation
```

Format

A `data.frame` object.

Source

The data was generated synthetically.

References

Tikka S, López-Pernas S, Saqr M (2025). "tna: An R Package for Transition Network Analysis." *Applied Psychological Measurement*. doi:10.1177/01466216251348840

`plot.ews` *Plot EWS Results*

Description

Plot EWS Results

Usage

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

Arguments

`x`	`[ews]` Output of `detect_warnings()`.
`...`	Ignored.

Value

A `ggplot` object.

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)
ews_roll <- detect_warnings(ts_data)
plot(ews_roll)
```

`plot.regimes` *Plot Time Series Data with Detected Regime Stability*

Description

Plot Time Series Data with Detected Regime Stability

Usage

```
## S3 method for class 'regimes'
plot(x, points = FALSE, ...)
```

Arguments

`x`	`[regimes]` Output of `detect_regimes()`.
`points`	`[logical(1)]` Should a point be added for each observation? The points are colored by regime stability (default: `FALSE`).
`...`	Ignored.

Value

A `ggplot` object.

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)
regimes <- detect_regimes(
  data = ts_data,
  method = "threshold",
  sensitivity = "medium"
)
plot(regimes)
```

`print.ews` *Print EWS Detection Results*

Description

Print EWS Detection Results

Usage

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

Arguments

`x`	`[ews]` An EWS detection result from `detect_warnings()`.
`...`	Additional arguments passed to the generic print method.

Value

`x` (invisibly).

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)
ews <- detect_warnings(ts_data)
print(ews)
```

`print.regimes` *Print Regime Detection Results*

Description

Print Regime Detection Results

Usage

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

Arguments

`x`	`[regimes]` A regime detection result from `detect_regimes()`.
`...`	Additional arguments passed to the generic print method.

Value

x (invisibly).

Examples

```
set.seed(123)
ts_data <- stats::arima.sim(list(order = c(1, 1, 0), ar = 0.6), n = 200)
regimes <- detect_regimes(
  data = ts_data,
  method = "threshold",
  sensitivity = "medium"
)
print(regimes)
```

`sequence_indices` *Compute Sequence Indices for Sequence Data*

Description

Compute Sequence Indices for Sequence Data

Usage

```
sequence_indices(data, cols, favorable, omega = 1)
```

Arguments

`data`	[`data.frame`, `matrix`, `stslist`] Sequence data in wide format (rows are sequences, columns are time points).
`cols`	[`expression`] A tidy selection of columns that should be considered as sequence data. By default, all columns are used.
`favorable`	[`character()`] Names of states that should be considered as favorable states.
`omega`	[`numeric(1)`] Omega parameter value used to compute the integrative potential.

Value

A `tibble` containing the index values.

Examples

```
sequence_indices(engagement)
```

Index