Package: codyna (via r-universe)

May 31, 2026

Title Complex Dynamic Systems

Version 0.2.0

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, scales, stats, tibble, tidyr, tidyselect

Suggests knitr, lme4, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LazyData true

Config/pak/sysreqs libicu-dev

Repository https://santikka.r-universe.dev

Date/Publication 2026-03-02 13:12:52 UTC

RemoteUrl https://github.com/santikka/codyna

RemoteRef HEAD

RemoteSha 1b1ce172a1df847d0a0ceddb201972a6f7d395a3

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/

`analyze_outcome` *Analyze Pattern-Outcome Relationships*

Description

Fit a (mixed) logistic regression model to the data using sequence patterns as predictors. The patterns are first determined using `discover_patterns()` and used as predictors as specified by the user (either frequency or presence). The user can further select the maximum number of patterns to use and how to prioritize the patterns (frequency, support, lift, etc.). Additional covariates and random effects can also be included in the model. The logistic model is fitted using `stats::glm()` or by `lme4::glmer()` in the case of a mixed model.

Usage

```
analyze_outcome(
  data,
  cols = tidyselect::everything(),
  group = NULL,
  outcome = "last_obs",
  reference,
  n = 10,
  freq = FALSE,
  priority = "chisq",
  desc = TRUE,
  formula = ~1,
  re_formula,
  mixed = TRUE,
  type = "ngram",
  len = 1:2,
  gap = 1,
  min_support = 0.01,
  min_freq = 5,
  start,
  end,
  contain,
  ...
)
```

Arguments

`data`	[`data.frame`] Sequence data in wide format (rows are sequences, columns are time points). The input should be coercible to a `data.frame` object.
`cols`	[`tidy-select`] A tidy selection of columns that should be considered as sequence data. By default, all columns are used.

group [tidy-select]
An optional tidy selection of columns that define the grouping factors. If provided, group-specific random effects can be specified via re_formula. The default value NULL disables grouping and a fixed-effects model is fitted instead.

outcome [character(1), vector()]
Outcome variable specification. The option "last_obs" assumes that the last non-missing observation of each sequence specifies the outcome. Alternatively, a column name of data or a vector with the same length as the number of rows of data.

reference [character(1)]
The name of the outcome class to be taken as the reference. The probability of the other class is modeled. If not provided, uses the first class in lexicographic order.

n [integer(1)]
Maximum number of patterns to include in the model as covariates. The default is 10.

freq [logical()]
Should the pattern frequency be used as a predictor?. If FALSE (the default), instead defines a binary predictor that attains the value 1 if the pattern is present in the sequence and 0 otherwise.

priority [character(): "chisq"]
Criteria giving the priority of pattern inclusion in the model. Multiple criteria can be selected simultaneously. The available options are the column names of the return object of discover_patterns(), excluding pattern.

desc [logical(), TRUE]
A logical vector of the same length as priority that defines which criteria should be evaluated in descending order of magnitude. Automatically recycled if the lengths do not match.

formula [formula]
Formula specification for the fixed effects of the non-pattern covariates. The default is ~ 1, adding no covariates.

re_formula [formula]
Formula specification of the random effects when group is provided. By default, a random intercept is added for each grouping variable in group.

mixed [logical(1)]
Should a mixed model be fitted when group is provided? (default: TRUE)

type [character(1): "ngram"]
The pattern type to analyze:

- "ngram": Extract contiguous n-grams.
- "gapped": Discover patterns with gaps/wildcards.
- "repeated": Detect repeated occurrences of the same state.

len [integer(): 1:2]
Pattern lengths to consider for n-grams and repeated patterns.

gap [integer(): 1]
Gap sizes to consider for gapped patterns.

`min_support`	[`integer(1)`: `0.01`] Minimum support threshold, i.e., the proportion of sequences that must contain a specific pattern for the pattern to be included.
`min_freq`	[`integer(1)`: `2L`] Minimum pattern frequency threshold, i.e., the number of times a pattern must occur across all sequences for it to be included.
`start`	[`character()`] Filter patterns starting with these states.
`end`	[`character()`] Filter patterns ending with these states.
`contain`	[`character()`] Filter patterns containing these states.
`...`	Additional arguments passed to `discover_patterns()` and the model-fitting function (`stats::glm()` or `lme4::glmer()`).

Value

Either a `glm` or a `glmerMod` object depending on whether random effects were included.

Examples

```
fit <- analyze_outcome(engagement, outcome = rep(1:2, each = 500))
summary(fit)
```

`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)`: `7L`] A positive `integer` specifying the rolling window size. Must be at least `2`.

align [character(1): "center"]
Alignment of the window. The available options are: "center", "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 = tidyselect::everything(), format = "frequency")
```

Arguments

data	[data.frame] Sequence data in wide format (rows are sequences, columns are time points). The input should be coercible to a data.frame object.
cols	[tidy-select] A tidy selection of columns that should be considered as sequence data. By default, all columns are used.
format	[character(1): "frequency"] The data format to convert into: • "frequency": Counts of each state per sequence. • "onehot": Presence/absence (1/0) 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 = 10L,
  peak = 2,
  cumulative = 0.6
)
```

Arguments

data	[ts, numeric()] Univariate time series data.
method	[character(1): "smart"] Regime 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): "medium"] Detection sensitivity level. The available options are: "low", "medium", and "high". Controls thresholds and window sizes within the detection methods.
min_change	[integer(1)] Minimum number of observations between changes. If not provided, the value is determined automatically (typically 10% of observations, minimum of 10).
window	[integer(1): 10L] Base window size for rolling calculations. This is further adjusted by sensitivity
peak	[numeric(1): 2.0] Base z-score threshold for individual peak detection with the "cumulative_peaks" method. Adjusted by sensitivity.
cumulative	[numeric(1): 0.6] A value between 0 and 1 that defines the base proportion threshold for identifying cumulative peak regions. Adjusted by sensitivity.

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

Eearly warning signal (EWS) detection for time series data. Both rolling window and expanding window approaches are supported. Includes various methods for detrending the data before analysis. For visualizing the results, see `plot.ews()`.

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)`: `"rolling"`] Name of the analysis method. Either `"rolling"` or `"expanding"` for rolling window and expanding window, respectively.
`metrics`	[`character(1)`: `"all"`] Names of the EWS metrics to compute. 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): 0.5]
Window size as a proportion of the total time series length.

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

demean [logical(1): TRUE]
Should the time series be demeaned before analysis? If TRUE, 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): "none"]
Name of the detrending method to apply to the time series data before computing the metrics. 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 (no detrending).

threshold [numeric(1): 2.0]
The z-score threshold value for the expanding window method.

consecutive [integer(1): 2L]
The number of times the threshold has to be crossed consecutively to be counted as a detection.

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

Discover various types of patterns in sequence data. Provides n-gram extraction, gapped pattern discovery, analysis of repeated patterns and targeted pattern search. Supports comparison of pattern presence between groups.

Usage

```
discover_patterns(
  data,
  cols = tidyselect::everything(),
  outcome,
  type = "ngram",
  pattern,
  len = 2:5,
  gap = 1:3,
  min_freq = 2,
  min_support = 0.01,
  start,
  end,
  contain
)
```

Arguments

`data`	[`data.frame`] Sequence data in wide format (rows are sequences, columns are time points). The input should be coercible to a `data.frame` object.
`cols`	[`tidy-select`] A tidy selection of columns that should be considered as sequence data. By default, all columns are used.
`outcome`	[`character(1)`, `vector()`] Optional grouping specification. The option `"last_obs"` assumes that the last non-missing observation of each sequence specifies the outcome group. Alternatively, a column name of `data` or a `vector` with the same length as the number of rows of `data`.
`type`	[`character(1)`: `"ngram"`] The pattern type to analyze: • `"ngram"`: Extract contiguous n-grams. • `"gapped"`: Discover patterns with gaps/wildcards. • `"repeated"`: Detect repeated occurrences of the same state.

`pattern` [character(1)]
A specific pattern to search for as a character string (e.g., "A->*->B"). If provided, type is ignored. Supports wildcards * to denote an arbitrary state.

`len` [integer(): 2:5]
Pattern lengths to consider for n-grams and repeated patterns.

`gap` [integer(): 1:3]
Gap sizes to consider for gapped patterns.

`min_freq` [integer(1): 2L]
Minimum pattern frequency threshold, i.e., the number of times a pattern must occur across all sequences for it to be included.

`min_support` [integer(1): 0.01]
Minimum support threshold, i.e., the proportion of sequences that must contain a specific pattern for the pattern to be included.

`start` [character()]
Filter patterns starting with these states.

`end` [character()]
Filter patterns ending with these states.

`contain` [character()]
Filter patterns containing these states.

Value

An object of class patterns which is a tibble with the following columns:

- pattern: The discovered patterns.
- length: The length of the pattern.
- frequency: The number of times the pattern occurs across all sequences.
- proportion: Frequency divided by the total frequency of patterns of the same length.
- count: The number of sequences that contain the pattern.
- support: The proportion of sequences that contain the pattern.
- lift: the support divided by the product of the supports of the individual states of the pattern. For wildcards, the support is always 1.

In addition, if outcome is provided, additional columns giving the counts in each outcome group, the chi-squared test statistic values (chisq), and p-values (p_value) are included.

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.patterns *Plot Discovered Patterns*

Description

Plot Discovered Patterns

Usage

```
## S3 method for class 'patterns'
plot(x, n = 10L, prop = TRUE, group, global = TRUE, ...)
```

Arguments

x	[patterns] Output of discover_patterns().
n	[integer(1): 10L] Maximum number of patterns to include in the plot.
prop	[logical(1): TRUE] Should outcome-specific count proportions be displayed in the plot? The default is TRUE. Ignored if outcome was not originally specified.
group	[character(1)] Name of the outcome class to draw the plot for. If not provided, shows the counts and proportions by outcome class for each pattern. If provided, only the proportions of the specific class are drawn by pattern. Ignored if outcome was not originally specified.
global	[logical(1): TRUE] Should a line be added showing the global proportion when group is provided? Also colors the patterns according to whether the proportion is above or below the global value.
...	Ignored.

Value

A ggplot object. ngrams <- discover_patterns(engagement, type = "ngram") plot(ngrams)

`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] 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.patterns *Print Discovered Patterns*

Description

Print Discovered Patterns

Usage

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

Arguments

x	[patterns] Pattern discovery result from discover_patterns().
...	Additional arguments passed to the generic print method.

Value

x (invisibly).

Examples

```
ngrams <- discover_patterns(engagement, type = "ngram")
print(ngrams)
```

`print.regimes` *Print Regime Detection Results*

Description

Print Regime Detection Results

Usage

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

Arguments

x	[regimes] 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 = tidyselect::everything(), favorable, omega = 1)
```

Arguments

data	[data.frame] Sequence data in wide format (rows are sequences, columns are time points). The input should be coercible to a data.frame object.
cols	[tidy-select] 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 favorable.
omega	[numeric(1): 1.0] Omega parameter value used to compute the integrative potential.

Value

A tibble containing the index values.

Examples

```
sequence_indices(engagement)
```

Index