Package: YEAB (via r-universe)

March 3, 2025

Title Analyze Data from Analysis of Behavior Experiments

Version 1.0.6

Description Analyze data from behavioral experiments conducted using 'MED-PC' software developed by Med Associates Inc. Includes functions to fit exponential and hyperbolic models for delay discounting tasks, exponential mixtures for inter-response times, and Gaussian plus ramp models for peak procedure data, among others. For more details, refer to Alcala et al. (2023) <doi:10.31234/osf.io/8aq2j>.

Encoding UTF-8

RoxygenNote 7.3.2

Imports cluster, doParallel, data.table, dplyr, FNN, foreach, ggplot2, boot, grid, gridExtra, infotheo, ks, magrittr, minpack.lm, Polychrome, scales, sfsmisc, MASS, KernSmooth, zoo, usethis, stats, utils

Depends R (>= 3.5.0)

License GPL (>= 3)

Author Emmanuel Alcala [aut, cre], Rodrigo Sosa [aut], Victor Reyes [aut]

Maintainer Emmanuel Alcala <jealcalat@gmail.com>

Date 2025-01-23

Suggests knitr, rmarkdown

VignetteBuilder knitr

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2025-01-31 11:00:14 UTC

Config/pak/sysreqs git make libgit2-dev libssl-dev libx11-dev

Contents

ab_range_normalization *Normalization (or rescaling) between arbitrary a and b*

Description

Normalization (or rescaling) between arbitrary a and b

Usage

```
ab_range_normalization(x, a, b)
```

Arguments

x	numeric
a	numeric
b	numeric

Value

A numeric vector rescaled in the range $x' \in [a, b]$

Examples

```
x <- 5:100
a <- 0
b <- 1
x_scaled <- ab_range_normalization(x, a, b)
x_scaled
a <- 100
b <- 1000
x_scaled <- ab_range_normalization(x, a, b)
x_scaled
```

balci2019 *Peak individual trial analysis using moving average*

Description

Peak individual trial analysis using moving average

Usage

```
balci2019(tasa_norm, bins)
```

Arguments

tasa_norm	numeric, normalized response rate
bins	numeric

Details

Based on Balci et al 2010

Value

a list with params: a numeric vector with start, stop, spread and argmax (the bin at which response rate is max) mov_av: the moving average

Examples

```
data("r_times")
# binarize r_times to create response rate at 2 sec bins
bins <- get_bins(r_times, 0, 180, 2)
bin_res <- 6
tasa <- f_table(bins, 0, 180, bin_res)
tasa_norm <- tasa$prop / max(tasa$prop)
bins <- tasa$bins
balci_ind <- balci2019(tasa_norm, bins)

plot(bins, tasa_norm, xlab = "6 sec bins", )
lines(bins, balci_ind$mov_av, col = "blue", lwd = 2)
abline(v = balci_ind$params[c(1, 2, 4)], lwd = c(1, 1, 2), col = c(1, 1, "red4"))
```

---

berm *Biexponential Refractory Model (BERM)*

Description

Implements the biexponential refractory model (BERM) using maximum likelihood estimation to fit parameters for inter-response times (IRTs) within and between bouts.

The model is defined as:

$$p(IRT = \tau | \tau \geq \delta) = pwe^{-w(\tau-\delta)} + (1-p)be^{-b(\tau-\delta)}$$

where w and b are the rates for within and between bouts, p is the proportion of responses in bouts, and δ is the refractory period.

Calculates the negative log-likelihood for the BERM model.

Maps an unconstrained `d_hat` onto the observed minimum inter-response time (`d`), ensuring that it aligns with model constraints.

Converts raw parameters into their constrained forms to enforce model constraints on parameters such as `w`, `l0`, `l1`, and `d`.

Optimizes the log-likelihood function to estimate BERM model parameters based on observed inter-response times.

Usage

```
berm(irt, delta)

berm_log_likelihood(params, irt)

map_onto(d, d_hat)

param_conver(params, min_irt, parnames = c("w", "l0", "l1", "d"))

optimize_berm(irt)
```

Arguments

irt	A numeric vector of inter-response times.
delta	A numeric value for the refractory period.
params	A numeric vector of raw, unconstrained parameters.
d	Minimum inter-response time.
d_hat	Transformed parameter to be mapped onto d.
min_irt	Minimum inter-response time for mapping d.
parnames	Optional vector of parameter names for labeling.

Details

This function computes the negative log-likelihood based on biexponential functions for the BERM model, adjusting parameters using param_conver to meet constraints.

Value

A data frame with estimated parameters w (within-bout rate), b (between-bout rate), p (proportion of responses in bouts), and δ (adjusted refractory period).

Negative log-likelihood value used for parameter estimation.

Adjusted refractory period used in likelihood estimation.

A named numeric vector of transformed parameters with constraints applied.

A named vector of optimized parameters for the BERM model.

Examples

```
set.seed(43)
l1 <- 1 / 0.5
l2 <- 1 / 0.1
p <- 0.4
n <- 200
delta <- 0.03
irt <- c(rexp(round(n * p), l1), rexp(round(n * (1 - p)), l2)) + delta
optimize_berm(irt)
```

```
set.seed(43)
l1 <- 1 / 0.5
l2 <- 1 / 0.1
p <- 0.4
n <- 200
delta <- 0.03
irt <- c(rexp(round(n * p), l1), rexp(round(n * (1 - p)), l2)) + delta
optimize_berm(irt)
```

`biexponential` *Biexponential Model*

Description

Implements a simpler biexponential model without the refractory period parameter, δ.

The simpler model is defined as:

$$p(IRT = \tau) = pwe^{-w\tau} + (1 - p)be^{-b\tau}$$

where w and b represent the within- and between-bout response rates, and p is the proportion of responses in bouts.

Usage

```
biexponential(irt)
```

Arguments

`irt`	A numeric vector representing inter-response times.

Value

A data frame with estimated parameters w (proportion of responses in bouts), $l0$ (within-bout mean IRT), and $l1$ (between-bout mean IRT).

Examples

```
set.seed(43)
l1 <- 1 / 0.5
l2 <- 1 / 0.1
p <- 0.4
n <- 200
irt <- c(rexp(round(n * p), l1), rexp(round(n * (1 - p)), l2))
biexponential(irt)
```

bp_opt *Find the best fit for individual trials using* optim

Description

Find the best fit for individual trials by minimizing the negative sum of areas between the response rate and the target rate.

Usage

```
bp_opt(r_times, trial_duration, optim_method = "Brent")
```

Arguments

r_times	Vector of response times
trial_duration	Duration of the trial
optim_method	character, the optimization method to use

Value

A data frame with the following columns:

- bp: The breakpoint
- r1: The response rate before the breakpoint
- r2: The response rate after the breakpoint
- d1: The duration of the first state
- d2: The duration of the second state

Examples

```
data("r_times")
r_times <- r_times[r_times < 60]
bp_from_opt <- bp_opt(r_times, 60)
plot(r_times, seq_along(r_times),
  xlim = c(0, max(r_times)),
  main = "Cummulative Record",
  xlab = "Time (s)",
  ylab = "Cum Resp",
  col = 2, type = "s"
)
abline(v = bp_from_opt$bp)
```

ceiling_multiple *Find the nearest multiple*

Description

Find the nearest multiple

Usage

```
ceiling_multiple(x, multiple)
```

Arguments

x	numeric, the value for which we want to finde a multiple
multiple	numeric, the multiple

Value

the nearest multiple

Examples

```
ceiling_multiple(8, 10) # returns 10
ceiling_multiple(12, 10) # returns 20
ceiling_multiple(21, 11) # returns 22
```

curv_index_fry *Curvature index using Fry derivation*

Description

Curvature index using Fry derivation

Usage

```
curv_index_fry(cr, time_in, fi_val, n = 4)
```

Arguments

cr	A numeric vector of cumulative response
time_in	numeric, time (or the x axis in a cumulative response plot)
fi_val	the FI value
n	numeric, the number of subintervals

Value

The curvature index as exposed by Fry

Examples

```
data("r_times")
r_times <- r_times[r_times < 60]
cr <- seq_along(r_times)

plot(r_times, cr, type = "s", xlim = c(min(r_times), max(r_times)))
segments(
  x0 = min(r_times), y0 = min(cr),
  x1 = max(r_times), y1 = max(cr)
)
segments(
  x0 = min(r_times) + (max(r_times) - min(r_times)) / 2, y0 = min(cr),
  x1 = max(r_times), y1 = max(cr),
  col = "red"
)
curv_index_fry(cr, r_times, 60, 4)
```

---

curv_index_int *Curvature index by numerical integration*

Description

Curvature index by numerical integration

Usage

```
curv_index_int(cr, time_in)
```

Arguments

cr	numeric, cumulative response
time_in	numeric, time (or the x axis in a cumulative response plot)

Value

a numeric value that is the proportion of a rect triangle area minus the area under the curve

Examples

```
data("r_times")
r_times <- r_times[r_times < 60]
cr <- seq_along(r_times)

plot(r_times, cr, type = "s")
curv_index_int(cr, r_times)
```

```
segments(
  x0 = min(r_times), y0 = min(cr),
  x1 = max(r_times), y1 = max(cr)
)
segments(
  x0 = min(r_times) + (max(r_times) - min(r_times)) / 2, y0 = min(cr),
  x1 = max(r_times), y1 = max(cr),
  col = "red"
)
```

`DD_data` *Delay Discounting Data*

Description

Delay Discounting Data

Usage

```
DD_data
```

Format

A data frame with 6 rows and 2 columns:

norm_sv Normalized subjective values (numeric).

Delay Delays (in seconds) for rewards (numeric).

Details

A dataset containing normalized subjective values (SV) and delays used in a delay discounting task.

This dataset represents results from a delay discounting experiment. It demonstrates how subjective values decay with increasing delays.

Source

Generated for a delay discounting analysis.

dd_example *An example dataset of delays and normalized subjective values*

Description

A dataset containing delay durations and their respective normalized subjective values (norm_sv).

Usage

```
dd_example
```

Format

A data frame with 6 rows and 2 variables:

delay The delay duration (in seconds).

norm_sv The normalized subjective value corresponding to the delay.

Examples

```
data(dd_example)
head(dd_example)
```

entropy_kde2d *Shannon entropy in two dimensions*

Description

Shannon entropy in two dimensions

Usage

```
entropy_kde2d(x, y, n_grid = 150)
```

Arguments

x	numeric, random vector
y	numeric, random vector
n_grid	numeric, number of grid cells to evaluate density

Value

A numeric value of the entropy in 2D

Examples

```
set.seed(123)
# Generate a 2D normal distribution with a correlation of 0.6
n <- 1000
mean <- c(0, 0)
sd_x <- 1
sd_y <- 5
correlation <- 0.6
sigma <- matrix(
  c(
    sd_x^2,
    correlation * sd_x * sd_y,
    correlation * sd_x * sd_y,
    sd_y^2
  ),
  ncol = 2
)
library(MASS)
simulated_data <- mvrnorm(n, mu = mean, Sigma = sigma)
x <- simulated_data[, 1]
y <- simulated_data[, 2]

cov_matr <- cov(cbind(x, y))
sigmas <- diag(cov_matr)
det_sig <- prod(sigmas)
# According to https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Differential_entropy:

normal_entropy <- function(k, pi, det_sig) {
  # The left part is a constant;
  (k / 2) * (1 + log(2 * pi)) + (1 / 2) * log(det_sig)
}

entropia <- normal_entropy(k = 2, pi = pi, det_sig)
print(entropia) # Should return a value close to 4.3997

result <- entropy_kde2d(x, y, n_grid = 50)
print(result) # Should return a value close to 4.2177
```

---

eq_hyp *Hyperbolic function*

Description

An hyperbolic function to simulate delay discounting data

Usage

```
eq_hyp(k, delay)
```

Arguments

k	numeric constant, the delay discounting parameter
delay	vector of delays

Value

A numeric vector of subjective values between 0 and 1

Examples

```
delay <- seq(0, 10, len = 100)
k <- 0.2
sv <- eq_hyp(k, delay)
plot(delay, sv,
  xlab = "delay",
  ylab = "Sv",
  type = "l"
)
```

---

event_extractor *Event extractor*

Description

A function to slice data based on start and stop events. This function should be used after read_med.r, which outputs a csv of 2 columns: time and events (in that order). Its use is exemplified at the end of the function.

Usage

```
event_extractor(data_df, ev0, ev1, evname)
```

Arguments

data_df	data frame with events ev0 and ev1 (e.g., start of trial and reinforcement delivery)
ev0	event ID start (where the event we want to extract begins)
ev1	event ID stop. This event won't be returned, so keep in mind that
evname	a string for the event name, for identification purposes. For example if the event we want to extract is component 1 in a multiple-2 schedule, this can be eventname = "c1", so when we extract the second component we can row-combine both in a unique dataframe.

Details

Works by trials

Value

data frame with nrows x 4 columns of time, events, cum_id and evname

Examples

```
# If we have a component starting with 5 and ending with 3,
# say a Fixed Interval 15s and a dataframe of events from the read_med() function,
# we can extract the data of component "FI15" following the next steps:
# 0 - From the output of read_med.R function, load the extracted data and assign it to df
# 1 - source the event_extractor.R function
# 2 - use it with the appropiate arguments as follows

# read raw data from MED
data("fi60_raw_from_med")
# see first 10 lines
head(fi60_raw_from_med, 10)
# create a temporary file to avoid non-staged installation warning
temp_file <- tempfile(fileext = ".txt")
# write the data to the temporary file
writeLines(fi60_raw_from_med, temp_file)
# Process the file using read_med
example_processed <- read_med(
  fname = temp_file, save_file = FALSE,
  col_r = "C:", out = TRUE,
  col_names = c("time", "event"), num_col = 6, time_dot_event = TRUE
)

# Extract specific events (FI15 in this case)
extracted_FI15 <- event_extractor(
  data_df = example_processed,
  ev0 = 5, ev1 = 3,
  evname = "FI15"
)

# Display the first rows of the extracted data
head(extracted_FI15, 30)
```

---

exhaustive_lhl *Individual trial analysis for peak procedure data*

Description

Individual trial analysis for peak procedure data

Usage

```
exhaustive_lhl(r_times, trial_duration)
```

Arguments

r_times	numeric, the times that a response was emitted in a trial
trial_duration	numeric, the peak trial duration

Value

a data.frame of start, stop, spread, middle time (mid) and the response rate at each state (r1 for low, r2 for high and r3 for the second low rate state)

Examples

```
data("r_times")
trial_duration <- max(r_times) |> ceiling() # 180
bps <- exhaustive_lhl(r_times, trial_duration)
plot(
  density(
    r_times,
    adjust = 0.8,
    from = 0,
    to = 180
  ),
  main = "",
  ylab = expression(italic(p(t[R]))),
  xlab = "time in peak trial"
)
abline(v = 60, lty = 2)
bps <- exhaustive_lhl(r_times, 180)
abline(v = c(bps$start, bps$stop), col = 2, lty = 2, lwd = 2)
# compare it with fwhm
den <- density(r_times, from = 0, to = trial_duration)
fval <- fwhm(den$x, den$y)
x1 <- fval$peak - fval$fwhm / 2
x2 <- fval$peak + fval$fwhm / 2
plot(den)
abline(v = c(x1, fval$peak, x2), col = c("blue", 1, "blue"))
```

---

exhaustive_sbp *Single breakpoint algorithm, the exhaustive version as the one used in Guilhardi & Church 2004*

Description

Single breakpoint algorithm, the exhaustive version as the one used in Guilhardi & Church 2004

Usage

```
exhaustive_sbp(r_times, trial_duration)
```

Arguments

r_times	numeric, the times at which a response was emitted in a trial
trial_duration	numeric, the duration of the IF interval

Details

This algorithm performs an extensive search of every combination (t1, t2) where t1 starts in the first response through (length(r_times) - 1)

Value

A data frame of 3 columns bp a numeric value which corresponds to the time at which a break point was detected r1 a numeric value of the response rate *before* the breakpoint r2 a numeric value of the responser rate *after* the breakpoint d1 a numeric value of the duration of the first state d2 a numeric value of the duration of the second state

Examples

```
data("r_times")
r_times <- r_times[r_times < 60]
single_bp <- exhaustive_sbp(r_times, 60)

plot(r_times, seq_along(r_times),
  xlim = c(0, max(r_times)),
  main = "Cummulative Record",
  xlab = "Time (s)",
  ylab = "Cum Resp",
  col = 2, type = "s"
)
abline(v = single_bp$bp)

bp_from_opt <- bp_opt(r_times, 60)
abline(v = bp_from_opt$bp, col = 3)
```

---

exp_fit *Exponential fit with nls*

Description

This function performs an exponential fit using non-linear least squares (nls).

Usage

```
exp_fit(value, delay, initial_guess, max_iter = 1e+05, scale_offset = 0)
```

Arguments

value	A numeric vector of the subjective values (indifference points).
delay	A numeric vector of the delays used in the experiment.
initial_guess	A numeric value providing an initial estimate for the parameter k.
max_iter	An integer specifying the maximum number of iterations for the nls fitting algorithm. Default is 1e5.
scale_offset	A numeric value for the scaling offset used in nls fitting control. Default is 0.

Value

An object of class nls containing the fitted model.

Examples

```
# See the examples of hyp_fit
```

---

fi60_raw_from_med *Raw Fixed Interval Data*

Description

Raw Fixed Interval Data

Usage

```
fi60_raw_from_med
```

Format

A character vector containing lines of data from the file.

Details

A dataset containing raw data from a fixed interval (FI) experiment.

This dataset is obtained from a raw file generated during an FI experiment. It provides raw, unprocessed behavioral data.

Source

The raw data was read from the file: inst/extdata/fi60_raw.txt.

`fleshler_hoffman` *Fleshler & Hoffman (1962) progression*

Description

This function calculates the values of intervals approximately for an exponential distribution, but avoiding extremely large values.

Usage

```
fleshler_hoffman(N, VI)
```

Arguments

`N`	The total number of intervals.
`VI`	The value of the Variable Interval

Details

This function calculates the values of intervals approximately for a exponential distribution, but avoiding extremely large values which can produce extinction. It uses the formula derived from the Fleshler & Hoffman article, where the first factor of the equation is -log(1 - p)^(-1), representing the expected value or mean of the intervals. This value is also the inverse of a Poisson process, 1/lambda. Since we want the expected value or mean to be the value of the IV, we replace that constant with VI. The function handles the case when n = N, where the value becomes undefined (log(0)), by using L'Hopital's rule to find the limit of the function as n approaches N. The resulting values are then multiplied by the IV and the logarithm of N to obtain the final calculated values.

Value

A vector of calculated values for the intervals.

References

Fleshler, M., & Hoffman, H. S. (1962). A progression for generating variable-interval schedules. Journal of the Experimental Analysis of Behavior, 5(4), 529-530.

Examples

```
# Calculate intervals for N = 10, and IV = 30
N <- 15
iv <- 90
intervals <- round(fleshler_hoffman(N,iv), 3)
# Plot the intervals and the exponential distribution corresponding to the
# same mean (IV)
hist(intervals, freq = FALSE)
curve(dexp(x, rate = 1/iv), add = TRUE, col = 'red')
legend('topright', legend = c('F&H', 'Exponential'), lty = 1, col = c('black', 'red'))
```

`fwhm` *Full Width at Half Maximum*

Description

Full Width at Half Maximum

Usage

```
fwhm(x, y)
```

Arguments

`x`	numeric, a vector of values from a distribution (density)
`y`	numeric, a vector of probabilities

Details

The function allows to compute the spread of a symmetric function even when it is not normally distributed. It first finds the x at which y is max, then x1 and x2 can be recovered using x1=peak-fwhm/2, x2=peak+fwhm/2

Value

a list with the fwhm and the x at which the max ocurred

Examples

```
set.seed(170)
rx <- rnorm(100)
den <- density(rx)
fval <- fwhm(den$x, den$y)
x1 <- fval$peak - fval$fwhm / 2
x2 <- fval$peak + fval$fwhm / 2
plot(den)
abline(v = c(x1, fval$peak, x2), col = c(1, 2, 1))
```

f_table *Frequency table for binned data*

Description

Creates a frequency table from a vector of bins from, for example, get_bins(). It includes zero-frequency bins. If the bins came from the responding times, this creates a data.frame of response rate.

Usage

```
f_table(x, min_x, max_x, bin_res)
```

Arguments

x	numeric, a vector of binned data
min_x	numeric, the minimal value of x
max_x	numeric, the maximal value of x
bin_res	numeric, the bin resolution

Value

A data frame

Examples

```
data("r_times")
bin_res <- 2
min_x <- 0
max_x <- 180
x <- get_bins(r_times, min_x, max_x, bin_res)
xt <- f_table(x, min_x, max_x, bin_res)
plot(xt$bins, xt$prop)
```

gaussian_fit *Gaussian + ramp fit with LM algorithm*

Description

Gaussian + ramp fit with LM algorithm

Usage

```
gaussian_fit(
  responses,
  time_vec,
  par = list(a = 0.1, d = 0.1, t0 = 18, b = 10, c = 1),
  max.iter = 500
)
```

Arguments

responses	numeric, vector of response or response rate
time_vec	numeric, time bins
par	a list of parameters for the gaussian + linear; see Buhusi, C. V., Perera, D., & Meck, W. H. (2005) for an explanation
max.iter	numeric, max number of iterations

Details

Ver Buhusi, C. V., Perera, D., & Meck, W. H. (2005). Memory for timing visual and auditory signals in albino and pigmented rats.

This algorithm uses the nonlinear least squares nls.lm (Levenberg–Marquardt) from the minpack.lm package

Value

a numeric vector of coefficients, the same as the par argument

Examples

```
# Function to create synthetic data
g_plus_lin <- function(par, time) {
  par$a * exp(-0.5 * ((time - par$t0) / par$b)**2) + par$c * (time - par$t0) + par$d
}
# real params
pars <- list(a = 20, t0 = 20, b = 10, c = 0.2, d = 1)
# time vector for simulation
ti <- seq(0, 60, 0.1)
# time vector for sampling with 2 sec of resolution
ti_data <- seq(0, 60, 2)
# r(t) real
y_curva <- g_plus_lin(par = pars, ti)
# r(t) sampled with noise
y_data <- g_plus_lin(par = pars, ti_data) + rnorm(length(ti_data), 0, sd = 2)
# param estimation
par_est <- gaussian_fit(responses = y_data, t = ti_data, par = pars, max.iter = 10500)
par_est
# fitted curve
y_hat <- g_plus_lin(par_est |> as.list(), ti)
# plot results
```

```
plot(ti,
  y_curva,
  type = "l",
  col = "blue",
  lwd = 2,
  ylim = c(0, max(y_curva, y_data)),
  xlab = "Time in trial",
  ylab = "R(t)",
)
points(ti_data, y_data, pch = 21, bg = "red", cex = 1.2)
lines(
  ti,
  y_hat,
  col = "green2",
  lwd = 2
)
legend(
  "topright",
  legend = c("real", "real + noise", "ajuste nls.lm"),
  lty = c(1, 0, 1),
  pch = c(NA, 21),
  pt.bg = c(NA, "red"),
  col = c("blue", 1, "green2"),
  pt.cex = 0.9,
  cex = 0.6
)
```

`gauss_example` *Gaussian Example Data*

Description

This dataset contains a series of bins and corresponding response averages from an experimental task.

Usage

```
gauss_example
```

Format

A data frame with 91 rows and 2 columns:

Bin Numeric. The bin number.

Response_Average Numeric. The average response in each bin.

Source

Generated as part of a synthetic example for the task.

gauss_example_1 *Gaussian Example 1 Data*

Description

This dataset contains a series of bins and corresponding response averages from another experimental task, similar to the first example.

Usage

```
gauss_example_1
```

Format

A data frame with 91 rows and 2 columns:

Bin Numeric. The bin number.

Response_Average Numeric. The average response in each bin.

Source

Generated as part of a synthetic example for the task.

gauss_example_2 *Gaussian Example 2 Data*

Description

This dataset contains a series of bins and corresponding response averages, this time with a slightly different distribution and experimental task design.

Usage

```
gauss_example_2
```

Format

A data frame with 91 rows and 2 columns:

Bin Numeric. The bin number.

Response_Average Numeric. The average response in each bin.

Source

Generated as part of a synthetic example for the task.

gell_like *Gellerman-like series*

Description

Gellerman-like series

Usage

```
gell_like(n)
```

Arguments

n	numeric, a vector of 0 and 1 (see Details)

Details

The algorithm implements a Gellerman-like series based on Herrera, D., & Treviño, M. (2015). http://doi.org/10.1371/journal.pone.0136084 The algorithm samples from a binomial distribution and imposes two restrictions

1. no more than 3 consecutive values of 0s or 1s.
2. the number of trials 0 or 1 must be the same for a given n.

Value

a numeric vector of randomly distributed 0s and 1s

Examples

```
set.seed(165)
gell_like(8) # 0 0 1 1 1 0 1 0
```

get_bins *A function to binarize a numeric vector with a given resolution*

Description

A function to binarize a numeric vector with a given resolution

Usage

```
get_bins(x, x_min, x_max, res)
```

Arguments

x	numeric, the vector to be binarized
x_min	numeric, the min value of a vector to create the bins (e.g., 0)
x_max	numeric, the maximum value of the vector x to binarize
res	numeric, the resolution; if x is time, res can be 1 s

Value

the vector of bins for which x is in

Examples

```
x <- 1:20
get_bins(x, 0, 20, 5)
# Returns
# [1]  5  5  5  5  5 10 10 10 10 10 15 15 15 15 15 20 20 20 20 20
# set.seed(10)
x <- runif(20, 0, 10)
get_bins(x, 0, 10, 0.5)
# Returns
# 1] 5.5 3.5 4.5 7.0 1.0 2.5 3.0 3.0 6.5 4.5 7.0 6.0 1.5 6.0 4.0 4.5 1.0 3.0 4.0 8.5
```

---

hyperbolic_fit *Hyperbolic fit with nls*

Description

Hyperbolic fit with nls

Usage

```
hyperbolic_fit(value, delay, initial_guess, max_iter = 1e+05, scale_offset = 0)
```

Arguments

value	A numeric vector of the subjective values (indifference points)
delay	A numeric vector of the delays used
initial_guess	A numeric value providing an initial start for k
max_iter	Positive integer with maximum number of iterations
scale_offset	A constant to be added if the residuals are close to 0. This is to avoid division by 0, which is know to cause problems of convergence.

Value

An object of class nls

Examples

```
# Simulated data with k = 0.5
data("hyp_data")
delay <- hyp_data$delay
sv <- hyp_data$sv
real_k <- hyp_data$real_k
model_hyp <- hyperbolic_fit(sv, delay, initial_guess = 0.01)
summary(model_hyp)
k_est <- coef(model_hyp)
k_est
# plot real and estimated sv
delay_real <- seq(0, max(delay), len = 100)
# first, simulate how the data should look with the real k
real_sv <- eq_hyp(real_k, delay_real)
# simulate estimated fitting line
est_sv <- eq_hyp(k_est, delay_real)

plot(
  delay, sv,
  pch = 21,
  col = 1,
  bg = 8,
  xlab = "Delay",
  ylab = "Subjective value"
)
lines(
  delay_real,
  est_sv,
  col = "red",
  lwd = 2
)
# real data
lines(
  delay_real,
  real_sv,
  type = "l",
  col = "blue",
  lwd = 2
)
legend(
  "topright",
  legend = c("data", "real", "fit"),
  text.col = "white",
  pch = c(21, NA, NA),
  col = c(1, NA, NA),
  pt.bg = c(8, NA, NA),
  bty = "n"
)
legend(
  "topright",
  legend = c("data", "real", "fit"),
  pch = c(NA, NA, NA),
```

```
  lty = c(NA, 1, 1),
  col = c(NA, "blue", "red"),
  bty = "n"
)

# Now an example with real data
data("DD_data")
# first, fit a linear model
lineal_m <- lm(norm_sv ~ Delay, data = DD_data)
# hyperbolic model
hyp_m <- hyperbolic_fit(DD_data$norm_sv, delay = DD_data$Delay, 0.1)
# exponential model
exp_m <- exp_fit(DD_data$norm_sv, delay = DD_data$Delay, 0.1)
AIC(lineal_m, hyp_m, exp_m)
# compare visually
k_hyp <- coef(hyp_m)
k_exp <- coef(exp_m)
k_lin <- coef(lineal_m)
delay_vec <- seq(0, max(DD_data$Delay), len = 200)
plot(
  DD_data$Delay,
  DD_data$norm_sv,
  ylim = c(0, 1),
  pch = 21,
  ylab = "SV",
  xlab = "Delay",
  bg = "orange",
  col = "black"
)
lines(
  delay_vec,
  eq_hyp(k = k_hyp, delay_vec),
  col = "green4",
  lwd = 2
)
lines(
  delay_vec,
  exp(-k_exp * delay_vec),
  col = "steelblue",
  lwd = 2
)
abline(lineal_m, lty = 2, lwd = 2)

legend(
  "topright",
  legend = c("data", "exp fit", "hyp fit", "linear fit"),
  text.col = "white",
  pch = c(21, NA, NA, NA),
  col = c(1, NA, NA, NA),
  pt.bg = c("orange", NA, NA, NA),
  bty = "n"
)
legend(
```

```
  "topright",
  legend = c("data", "exp fit", "hyp fit", "linear fit"),
  pch = c(NA, NA, NA, NA),
  lty = c(NA, 1, 1, 2),
  col = c(NA, "steelblue", "green4", 1),
  bty = "n"
)
# plot AIC values
aic_val <- AIC(lineal_m, hyp_m, exp_m) |> round(2)
leg <- sprintf(paste(rownames(aic_val), "= %s", sep = " "), aic_val$AIC)
legend(
  "bottomleft",
  title = "AIC\n(the smaller, the better)",
  legend = leg,
  bty = "n"
)
```

`hyp_data` *Simulated Data for Hyperbolic Discounting*

Description

Simulated Data for Hyperbolic Discounting

Usage

```
hyp_data
```

Format

A list with 3 elements:

sv A numeric vector of normalized subjective values with added noise.

delay A numeric vector of delays (in seconds).

real_k The real value of the discounting parameter.

Details

A list of simulated data for fitting hyperbolic discounting models.

This dataset was generated to simulate the behavior of a hyperbolic discounting function. It is commonly used in behavioral economics and psychology to study delay discounting behaviors.

Source

Generated using a custom simulation function.

hyp_data_list *Hypothetical dataset list for testing purposes*

Description

This list contains three components: sv, delay, and real_k, representing subjective values, delay durations, and the real discount rate respectively.

Usage

```
hyp_data_list
```

Format

A list with three components:

sv A numeric vector of subjective values.

delay A numeric vector of delay durations (in arbitrary units).

real_k A numeric value representing the real discount rate.

Source

Hypothetical data generated for demonstration.

Examples

```
data(hyp_data_list)
str(hyp_data_list)
```

ind_trials_obj_fun *Objective function for finding the best fit for individual trials*

Description

This function is used by `optim` to find the best fit for individual trials by minimizing the sum of areas between the response rate and the target rate. Do not call this function directly.

Usage

```
ind_trials_obj_fun(params, r_times, trial_duration)
```

Arguments

`params`	A vector of parameters to be optimized.
`r_times`	Vector of response times
`trial_duration`	Duration of the trial

Value

a numeric value representing the sum of areas between the response rate and the target rate.

`ind_trials_opt` *Find the best fit for individual trials using* `optim`

Description

Find the best fit for individual trials by minimizing the sum of areas between the response rate and the target rate.

Usage

```
ind_trials_opt(r_times, trial_duration, optim_method = "Nelder-Mead")
```

Arguments

`r_times`	Vector of response times
`trial_duration`	Duration of the trial
`optim_method`	Optimization method. See `optim` for details.

Value

A data frame with the following columns:

- `start`: The start time of the peak
- `stop`: The stop time of the peak
- `spread`: The spread of the peak (stop - start)
- `middle`: The middle of the peak (mean of start and stop)

Examples

```
response_times <- c(
  28.1, 40.7, 44.2, 44.4, 44.7, 45, 45.4, 47.9, 48.1, 48.3,
  48.6, 48.8, 49.8, 50.2, 50.7, 51.2, 51.4, 51.7, 51.9, 52.7, 53, 53.5, 53.7,
  53.9, 54.1, 54.3, 54.9, 55.3, 55.5, 55.7, 55.8, 57.2, 57.4, 57.7, 58.3,
  58.5, 58.7, 60.4, 60.6, 60.7, 61.1, 61.6, 61.8, 62.6, 62.8, 63.1, 63.3,
  63.5, 63.8, 64.4, 64.8, 64.9, 65.1, 66.1, 66.4, 67, 68.7, 68.9, 69.5, 69.6,
  70.1, 70.9, 71, 71.3, 71.6, 71.8, 73.9, 74.1, 74.4, 74.6, 75.2, 76.4,
  76.6, 77.4, 77.6, 77.8, 78.2, 79.3, 79.9, 80.5, 80.7, 81.3, 82.2, 82.4,
  82.6, 82.9, 83, 83.1, 83.7, 84.4, 84.4, 84.8, 85, 85.6, 86.6, 87, 87.1,
  87.3, 87.4, 87.8, 88.1, 88.2, 89.4, 99.1, 99.3, 99.6, 99.8, 100.2,
  133.1, 133.1, 133.6, 134.9, 135.2, 135.3, 135.4, 135.7, 136.5, 173.8,
  174.1, 174.3, 174.7, 175.9, 176.3, 176.6, 177.4, 177.5, 177.7, 178.1,
  178.2, 178.4, 178.5, 178.8, 179.4
)
# Replace with your own initial guess
```

```
initial_guess <- c(min(response_times), mean(response_times))
trial_duration <- max(response_times)
result <- ind_trials_opt(response_times, trial_duration)
plot(
  density(
    response_times,
    adjust = 0.8,
    from = 0,
    to = trial_duration
  ),
  main = "Density plot of response times",
  xlab = "Response time (ms)",
  ylab = expression(italic(p(t[R]))),
)
abline(v = 60, lty = 2)
abline(v = result$start, col = "red")
abline(v = result$stop, col = "red")
abline(v = result$middle, col = "red")
```

`KL_div` *Computes the Kullback-Leibler divergence based on kernel density estimates*

Description

Computes the Kullback-Leibler divergence based on kernel density estimates of two samples.

Usage

```
KL_div(x, y, from_a, to_b)
```

Arguments

`x`	numeric, the values from a sample p
`y`	numeric, the values from a sample q
`from_a`	numeric, the lower limit of the integration
`to_b`	numeric, the upper limit of the integration

Details

The Kullback-Leibler divergence is defined as

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx$$

Value

a numeric value that is the kl divergence

Examples

```
set.seed(123)
p <- rnorm(100)
q <- rnorm(100)
KL_div(p, q, -Inf, Inf) # 0.07579204
q <- rnorm(100, 10, 4)
KL_div(p, q, -Inf, Inf) # 7.769912
```

`mut_info_discrete` *Mutual information of continuous variables using discretization*

Description

Mutual information of continuous variables using discretization

Usage

```
mut_info_discrete(x, y, method = "emp")
```

Arguments

`x`	A numeric vector
`y`	A numeric vector or equal or unequal size as x
`method`	The method to estimate entropy; available methods are "emp", "mm", "shrink", "sg" (default:"emp"). See details

Details

This function is based on the infotheo package. It uses equalfreq discretization by default. x and y need not be of equal size.

Value

A numeric value representing the mutual information between x and y

References

Meyer, P. E. (2008). Information-Theoretic Variable Selection and Network Inference from Microarray Data. PhD thesis of the Universite Libre de Bruxelles.

Examples

```
set.seed(123)
x <- rnorm(1000)
y <- rnorm(1000)
plot(x, y)
# close to 0 if they are independent
mut_info_discrete(x, y)
y <- 100 * x + rnorm(length(x), 0, 12)
plot(x, y)
# far from 0 if they are not independent
mut_info_discrete(x, y)
# simulate a sine function with noise
set.seed(123)
x <- seq(0, 5, 0.1)
y <- 5 * sin(x * pi)
y_with_noise <- y + rnorm(length(x), 0, 1)
plot(x, y_with_noise)
lines(x, y, col = 2)
# add a regression line
abline(lm(y ~ x))
# compute correlation coefficient; for nonlinear functions is close to 0
cor(x, y_with_noise)
# mutual information can detect nonlinear dependencies
mut_info_discrete(x, y_with_noise)
```

---

mut_info_knn *Mutual Information for Continuous Variables using kNN*

Description

Mutual Information for Continuous Variables using kNN

Usage

```
mut_info_knn(x, y, k = 5, direct = TRUE)
```

Arguments

x	Numeric vector.
y	Numeric vector.
k	Number of nearest neighbors to use; default is 5.
direct	Logical; if TRUE, mutual information is calculated using the k-nearest neighbors (kNN) estimator; if FALSE, it is computed via entropy estimates as $I(X;Y) = H(X) + H(Y) - H(X,Y)$. Default is TRUE.

Value

Numeric; an estimate of the mutual information.

Examples

```
set.seed(123)
x <- rnorm(1000)
y <- rnorm(1000)
# Close to 0 if they are independent
mut_info_knn(x, y, k = 5)
y <- 100 * x + rnorm(length(x), 0, 12)
# Far from 0 if they are not independent
mut_info_knn(x, y, k = 5)
```

---

n_between_intervals *Find maximum value within intervals*

Description

This function searches for the maximum value within a distribution (represented by vector x) that falls within a series of intervals specified by the vector intervals.

Usage

```
n_between_intervals(x, intervals, time_in)
```

Arguments

x	A numeric vector representing the distribution from which to find the maximum value within intervals.
intervals	A numeric vector specifying the intervals within which to search for the maximum value.
time_in	A numeric vector representing the corresponding time points for the values in the vector x, which is used to determine whether the values fall within the specified intervals.

Value

A numeric vector containing the maximum value within each interval specified by 'intervals'. If no values fall within an interval, returns 0 for that interval.

Examples

```
# Create a vector of data with a logarithmically increasing distribution
log_data <- round(exp(seq(log(1), log(100), length.out = 100)))

# Define intervals to cover the range 1-100
intervals <- seq(1, 100, by = 20)

# Create a corresponding time vector
time_in <- seq(1, 100, length.out = 100)
```

```
# Find maximum value within intervals
n_between_intervals(log_data, intervals, time_in)
```

`objective_bp` *Objective function for the breakpoint optimization algorithm*

Description

Objective function for the breakpoint optimization algorithm for fixed interval trials. This function is used by `optim` to find the optimal breakpoint. Do not call this function directly.

Usage

```
objective_bp(param, r_times, trial_duration)
```

Arguments

`param`	A numeric value of the breakpoint
`r_times`	A numeric vector of response times
`trial_duration`	A numeric value of the trial duration

Value

A numeric value representing the sum of areas between the response rate and the target rate.

`optimize_biexponential`

Optimization Function for the Biexponential Model

Description

Optimizes the log-likelihood function to estimate biexponential model parameters based on observed inter-response times.

Calculates the negative log-likelihood for the simpler biexponential model, which does not include the refractory period parameter, δ.

Usage

```
optimize_biexponential(irt)

biexponential_log_likelihood(params, irt)
```

Arguments

`irt`	A numeric vector representing inter-response times.
`params`	A numeric vector of initial parameter estimates for optimization.

Details

This function computes the negative log-likelihood based on biexponential functions for the simpler biexponential model, adjusting parameters using transformations to meet constraints.

Value

A named vector of optimized parameters for the biexponential model.

Negative log-likelihood value used for parameter estimation.

`read_med` *Process MED to csv based on standard data structure event.time*

Description

Process MED to csv based on standard data structure event.time

Usage

```
read_med(
  fname,
  save_file = FALSE,
  path_save = NULL,
  col_r = "C:",
  col_names = c("time", "event"),
  out = TRUE,
  num_col = 6,
  time_dot_event = TRUE,
  ...
)
```

Arguments

`fname`	chr, the name of a MED file to read; can include the directory
`save_file`	logical, save csv on the disk? TRUE or FALSE (default)
`path_save`	chr, directory to save csv files if save_file is TRUE;
`col_r`	chr, MED array to read (may be an event.time variable; see Details)
`col_names`	chr, a vector of column names
`out`	logical, if true returns the data.frame of n x 2
`num_col`	int, corresponds to DISKCOLUMNS of MED
`time_dot_event`	logical, if true, assumes that array to process has a time.event format
`...`	other arguments passed to `read.table`

Details

The default behavior of this function has time_dot_event = TRUE, which means that the raw MED can be should be in time.event convention. For example, if a response is coded as 23, the time is in 1/100 seconds and a response occurred at 2 minutes, the event is saved in, say, column C as 6000.23. This will be processed as time event 6000 23

However, if time_dot_event = FALSE, the output will be a data.frame with one column values. For example values 6000.23

Value

if out is true, returns a data.frame; if save_file is TRUE, writes the data.frame in csv format at path_save

Examples

```
# read raw data from MED
data("fi60_raw_from_med")
# see first 10 lines
head(fi60_raw_from_med, 10)
# create a temporary file to avoid non-staged installation warning
temp_file <- tempfile(fileext = ".txt")
# write the data to the temporary file
writeLines(fi60_raw_from_med, temp_file)
# Use the temporary file for processing
fi60_processed <- read_med(fname = temp_file, save_file = FALSE,
  col_r = "C:", out = TRUE,
  col_names = c("time", "event"), num_col = 6, time_dot_event = TRUE)
head(fi60_processed)
# ____________________________________________________________________________
## To use in bulk
# 1) Generate a list of filenames of raw MED data
# 2) Loop over the list with the function, using each element
#    of the list as the fname argument.
# ____________________________________________________________________________
# Suppose all raw MED files start with 2020, and you are in the working directory
# If all the raw MED files are in the wd, we can directly get the filenames
# with unspecified path
# filenames <- list.files(pattern = "^2020")
# The above line will look in the wd for all the files starting with "2020"
# and it will save it as a vector of strings in "filenames".
# With that vector, make a for loop like the following:
# ____________________________________________________________________________
# If you want to work immediately with the processed data, first create an empty
# dataframe to store the data file per file
# df_working = data.frame()
# ____________________________________________________________________________
# for (f in filenames) {
#   df_tmp <- read_med(fname = f,
#                      path_save = "data/processed/", # put here your path to save the csv
#                       col_r = 'C:', # if the time.event vector is saved in variable C
#                       out = TRUE ) # If you want to store processed data in df_tmp,
```

```
# otherwise write out = FALSE
# now append at rows the new data.frame
#   df_working = rbind(df_working, df_tmp)
# }
# Thats all.
```

`r_times` *Reaction Times from Peak Procedure*

Description

Reaction Times from Peak Procedure

Usage

```
r_times
```

Format

A numeric vector with 132 elements representing reaction times.

Details

A dataset containing reaction times (in seconds) from an experiment using the peak procedure.

These times are derived from a peak procedure experiment, typically used in behavioral experiments to measure timing abilities in subjects.

Source

Generated during a behavioral analysis experiment.

`sample_from_density` *Sample from a density estimate*

Description

Sample from a density estimate

Usage

```
sample_from_density(x, n)
```

Arguments

`x`	A numeric variable from a (un)known distribution
`n`	Number of samples to return

Value

A sample with distribution close to x

Examples

```
x <- rnorm(1000)
y <- sample_from_density(x, 1000)

hist(x,
  breaks = 30,
  freq = FALSE,
  col = "grey",
  main = "Original data"
)
```

`trapezoid_auc` *Area under the curve (AUC)*

Description

Calculate the area under the curve (AUC) using the trapezoid method.

Usage

```
trapezoid_auc(x, y)
```

Arguments

x	A numeric vector of x values
y	A numeric vector of y values

Value

A numeric value of the area under the curve

Examples

```
x_values <- c(0, 1, 2, 3, 4) # Delay times
y_values <- c(1, 0.8, 0.6, 0.4, 0.2) # Discounted values
auc_result <- trapezoid_auc(x_values, y_values)
print(paste("Area Under Curve: ", auc_result))
```

unit_normalization *Min-max normalization (also feature rescaling)*

Description

Min-max normalization (also feature rescaling)

Usage

```
unit_normalization(x)
```

Arguments

x	numeric, vector of values to rescale

Value

A numeric vector rescaled in the range $x' \in [0, 1]$

Examples

```
x <- 5:100
x_scaled <- unit_normalization(x)
x_scaled
```

val_in_interval *True value in interval*

Description

True value in interval

Usage

```
val_in_interval(df, lowLim, upLim, true.val)
```

Arguments

df	A data frame containing the intervals to be evaluated. Each row should correspond to an interval with lower and upper limits.
lowLim	the column index or name in the data frame df corresponding to the lower limit of the interval.
upLim	the column index or name in the data frame df corresponding to the upper limit of the interval.
true.val	the true value to be checked if it falls within the interval defined by lowLim and upLim.

Value

A numeric vector of n elements with an integer value for each interval: 0 if the value is below the interval, 1 if it is inside the interval (with a rightmost open limit), and 2 if it is above the interval.

Examples

```
# Example data frame with intervals
df <- data.frame(lower = c(1, 5, 10), upper = c(3, 8, 15))

# Check if the value 6 is within any of the intervals
val_in_interval(df, "lower", "upper", 6)
```

Index