Package ‘lexsync’

November 22, 2026

Type Package

Title Lexical Optimisation and Hardware-Timed Experiment Generation

Version 0.2.0

Description A cross-platform toolkit for lexical-corpus access (bundled English, Spanish and Mandarin lexica and a registry of citable corpora), the matching of stimuli on several dimensions at once, deterministic pseudoword generation, counterbalancing and the automated generation of experiments from a declarative trial-event model, for 'PsychoPy', 'OpenSesame' and the browser ('jsPsych'). The laboratory targets send electroencephalography onset triggers on the screen refresh that shows the stimulus. Several paradigms are supported (factorial word contrasts, lexical decision, priming, self-paced reading and cued categorisation), and each design is accompanied by a materials datasheet, readable by machines and people, for reproducibility. A 'Python' package of the same name has the same structure and functions.

License MIT + file LICENSE

Copyright The MIT licence covers the source code only. The example lexica in inst/extdata are derived from third-party data and are distributed under CC BY-SA 4.0, https://creativecommons.org/licenses/by-sa/4.0/. The terms and the attribution they require are in the COPYRIGHTS file installed with the package (inst/COPYRIGHTS in the sources).

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports readr (>= 2.0.0), yaml, stringdist, stringi, jsonlite (>= 1.8.7), digest, stats, tools, utils

Suggests testthat (>= 3.1.8), knitr, rmarkdown, spelling, clue, shiny, bslib, DT, zip

VignetteBuilder knitr

URL https://github.com/pablobernabeu/lexsync, https://pablobernabeu.github.io/lexsync/r/

BugReports https://github.com/pablobernabeu/lexsync/issues

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Pablo Bernabeu [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1083-2460>)

Maintainer Pablo Bernabeu <pcbernabeu@gmail.com>

Repository CRAN

Date/Publication 2026-11-22 00:30:10 UTC

Contents

add_bigram_frequency . . . 3
add_neighbourhood . . . 4
add_pair_overlap . . . 4
assign_triggers . . . 5
balance_check . . . 6
balance_lists . . . 7
build_datasheet . . . 8
build_lexdec_stimuli . . . 11
build_pool . . . 12
cohens_d . . . 13
cohens_d_ci . . . 13
counterbalance . . . 14
count_syllables . . . 15
describe_stimuli . . . 16
design_stem . . . 17
export_experiments . . . 17
export_jspsych . . . 18
export_opensesame . . . 20
export_psychopy . . . 21
fetch_corpus . . . 22
generate_pseudowords . . . 24
lexsync_cache_clear . . . 24
lexsync_cache_dir . . . 25
list_corpora . . . 26
load_items . . . 27
load_lexicon . . . 27
load_pool . . . 28
log_artefact . . . 29
log_step . . . 30
make_pseudoword . . . 31
match_report . . . 32
match_report_continuous . . . 32
match_stimuli . . . 33

merge_norms . . . 35
methods_paragraph . . . 36
new_run_log . . . 36
normalise_lexicon . . . 37
PARADIGMS . . . 38
participant_table . . . 39
required_fields . . . 40
resample_stimuli . . . 41
resolve_events . . . 42
resolve_trial_timing . . . 42
run_all . . . 43
run_pipeline . . . 44
select_continuous_stimuli . . . 45
tost_equiv . . . 47
validate_design . . . 48
variance_ratio . . . 49
write_datasheet . . . 50
write_run_log . . . 50

Index **51**

`add_bigram_frequency` *Mean bigram probability (type-based, non-positional), a phonotactic-probability proxy*

Description

For each word, the mean over its adjacent letter bigrams of the corpus bigram probability (count divided by the total bigram count). It is computed from integer counts and then rounded.

Usage

```
add_bigram_frequency(df, reference = NULL)
```

Arguments

`df`	A data frame with a `word` column.
`reference`	A character vector of reference words (defaults to `df$word`).

Value

`df` with an added numeric `bigram_freq` column.

add_neighbourhood *Compute orthographic-neighbourhood dimensions (Coltheart's N and OLD20)*

Description

`n_density` is Coltheart's N: the number of reference words of the same length differing by a single letter substitution (Hamming distance 1). `old20` is the mean Levenshtein distance to the `n_old` (default 20) nearest other reference words (Yarkoni et al., 2008). Both are computed against `reference`, which should be a large word list (typically the whole lexicon), not just the experimental pool.

Usage

```
add_neighbourhood(df, reference = df$word, n_old = 20L)
```

Arguments

`df`	A data frame with a `word` column.
`reference`	A character vector of reference words, `df$word` by default. Duplicates count once.
`n_old`	Neighbourhood size for OLD (default 20). The column is named `old20` whatever its value.

Value

`df` with added integer `n_density` and numeric `old20` columns. A word with fewer than `n_old` other words in the reference has the mean over those it has, and `old20` is `NA` for a word with none.

References

Yarkoni, T., Balota, D., & Yap, M. (2008). Moving beyond Coltheart's N: A new measure of orthographic similarity. *Psychonomic Bulletin & Review, 15*(5), 971-979. doi:10.3758/PBR.15.5.971

add_pair_overlap *Orthographic overlap between the two members of each pair*

Description

Adds two columns. `pair.lev` is the Levenshtein distance between the pair's two orthographic forms, and `pair.overlap` is `1 - lev / max(nchar)`, the proportion of the longer form the two share. Overlap is the standard confound control in a priming design: a related pair that also shares letters confounds semantic relatedness with orthographic similarity.

Usage

```
add_pair_overlap(df, prime = "prime", target = "target")
```

Arguments

`df`	A pair table.
`prime`, `target`	Column names holding the two orthographic forms.

Details

The forms are trimmed, lower-cased and normalised to Unicode NFC before they are compared, as `load_lexicon()` does to a lexicon's words, so `Dog` and `dog` are at distance 0. Length is counted in code points. `pair.overlap` is rounded to nine decimal places, and a pair of two empty forms gives 0.

Value

`df` with `pair.lev` and `pair.overlap` added.

`assign_triggers` *Assign EEG trigger codes to stimuli*

Description

Adds `condition_trigger` (101, 102 and so on, one per condition) and `item_trigger` (40 to 239, one per item `set`). Events refer to these by the tokens `condition` and `item`, or carry their own codes, whole numbers from 1 to 255. Item codes follow the code-point order of the set labels, so with sets 1 to 20, set 10 is 41 and set 2 is 51. Decode an item code through the loop table, not from the set's label. The item range holds 200 codes (a limit of an 8-bit port), so past 200 sets the codes wrap and repeat, and a message says so.

Usage

```
assign_triggers(stimuli, conditions = NULL, events = NULL, schema = NULL)
```

Arguments

`stimuli`	A stimuli data frame.
`conditions`	Optional character vector of condition labels, in the order that assigns their codes from 101 upwards. `NULL` (the default) numbers the conditions in order of first appearance.
`events`	Optional list of the design's events, as `resolve_events()` returns them, which says which codes the experiment sends. `NULL` (the default) allocates the item codes as for a design that sends no other code beside them.
`schema`	Optional parsed global schema, read for `triggers.allow_code_overlap`, which must be `true` or `false` when given.

Details

`events`, the design's event list, says which codes the experiment sends. When it sends item codes together with condition codes, or with a code of its own from 40 to 239, the item codes skip 101 to 100 + k, the codes of the k conditions, and the design's own codes, so that no item takes a code that marks something else. Without the skip, sets 65 and 66 of a 70-set design would take 101 and 102, the condition codes. With it, a design with more item sets than free codes is refused, unless `schema` sets `triggers.allow_code_overlap: true`, which lets the item codes wrap and repeat within the free codes. Every other design, and a call without `events`, takes the item codes described above, from 40 upwards in code-point order of the set labels, wrapping and repeating past 200 sets. Each trigger in `events` is checked as `render_events()` checks it.

A notice is given for a code of the design's own that the stream already uses for something else: 254 or 255 (which the PsychoPy and OpenSesame experiments send when the session starts and ends), a condition code the design sends or a code within the range of the item codes it sends.

The condition codes follow `conditions` when it is given: its first entry is 101, its second 102 and so on, whether or not every entry occurs in `stimuli`, so the same condition keeps the same code in a subset such as a practice block. A condition present in `stimuli` but missing from `conditions` takes the next free codes, in code-point order of its label. Either way, the mapping depends only on the labels, never on the row order. Without `conditions`, the codes follow the order in which the conditions first appear in `stimuli`, as in lexsync 0.1.0. On a shuffled trial list, that order comes from the shuffle, so a different seed can swap two conditions' codes. `export_experiments()` passes the design's order, and the pipeline passes the order of the design's `conditions`, followed by any other condition in the order the item source lists it.

Value

`stimuli` with trigger columns added.

Examples

```
stim <- data.frame(condition = c("low", "high", "low", "high"), set = c(1, 1, 2, 2))
# First appearance: low = 101, high = 102.
unique(assign_triggers(stim)[, c("condition", "condition_trigger")])
# The design's order: high = 101, low = 102, however the rows are shuffled.
unique(assign_triggers(stim, conditions = c("high", "low"))[
  , c("condition", "condition_trigger")])
# Item codes follow the code-point order of the set labels: set 2 is 51 here.
twenty <- assign_triggers(data.frame(condition = "a", set = 1:20))
stats::setNames(twenty$item_trigger, twenty$set)
```

`balance_check` *Check that the levels of given columns occur equally often*

Description

Check that the levels of given columns occur equally often

Usage

```
balance_check(stimuli, columns)
```

Arguments

stimuli	A stimuli data frame.
columns	Columns whose level counts should be equal. A name the stimuli do not carry is skipped without a warning, so check the names first.

Value

A character vector of human-readable balance warnings (empty if none).

---

balance_lists *Assign item sets to counterbalancing lists so the lists match on the item dimensions*

Description

The factorial recipe's default deal is by set rank, which balances nothing. This searches instead for an assignment whose lists have near-equal totals on each declared dimension, by steepest-descent pairwise swaps between lists. List sizes are preserved, because a swap exchanges one set for another.

Usage

```
balance_lists(stimuli, design, schema)
```

Arguments

stimuli	A stimuli data frame with a `set` column and the balance dimensions.
design	A parsed design configuration. Reads `counterbalance.lists`, `counterbalance.balance_on` (defaulting to `match_on`, then to the continuous predictor and controls), `counterbalance.max_passes` and `counterbalance.scale` (`mean` or `mad`, defaulting to `mean`).
schema	The parsed global schema, which provides the seed.

Details

The search is deterministic. The objective is all-integer, the descent takes the single best swap in each pass and ties are broken by the seeded keyed hash, not by position, so no list is favoured by being numbered first. Because the cost is a non-negative integer that strictly decreases, the search terminates. `max_passes` bounds it all the same, and the report says whether the bound was reached.

`counterbalance.scale` sets each dimension's unit. `mean`, the default, divides by the dimension's mean, which suits ratio scales only. `mad` balances deviations from the mean in units of the mean absolute deviation, so adding a constant to a dimension, such as writing Zipf frequency as Zipf - 3,

leaves the lists unchanged. The report then gives `scale` too, and `cost_unit` names the unit under either scale.

Five situations are refused rather than answered with an assignment that would mislead. A Latin-square design is refused because every item already appears in every list there, so there is nothing left to equate, and fewer than two lists leaves no pair of lists to exchange sets between. A design with no resolvable balance dimension is refused, as is one naming a dimension the stimuli do not carry, and the message names the columns. The last refusal is arithmetic: the search stops if the integer objective would leave the range that a double represents exactly, beyond which its comparisons would no longer be exact.

Value

A list with `list_of_set` (a named integer vector mapping each set to a list) and `report` (the dimensions, the integer cost before and after, the number of swaps taken, whether the pass bound was reached, the unit the cost is counted in and, under the `mad` scale, the scale).

`build_datasheet` *Assemble the materials datasheet for one design*

Description

Assemble the materials datasheet for one design

Usage

```
build_datasheet(
  design,
  schema,
  report,
  stimuli,
  source_path,
  artifacts,
  seed,
  engine = "R",
  candidate_pool = NULL,
  norms = NULL,
  balance = NULL,
  blocks = NULL,
  design_path = NULL,
  schema_path = NULL,
  selection_audit = NULL,
  neighbourhood_reference = NULL,
  counterbalance_check = NULL,
  input_normalisation = NULL,
  generation_exclude = NULL,
  exclusions = NULL,
  zipf_derivation = NULL,
```

```
  trial_order = NULL
)
```

Arguments

design	A parsed design list.
schema	The parsed global schema (schema.yaml).
report	The match report, the list of descriptives and comparisons data frames that match_report() or match_report_continuous() returns, or NULL when the design has none, as for an item table without a continuous selection.
stimuli	The selected stimulus data frame.
source_path	Path of the lexicon or item table the stimuli came from.
artifacts	Named list of the artefact paths written for the design: stimuli, descriptives, comparisons and experiments, the result of export_experiments(). Each file is recorded with its sha256: the three tables, then every file in experiments$files, the loop and text tables included, or the three experiment paths of a result that has no files.
seed	The integer seed recorded for the counterbalanced trial order.
engine	Engine label for the record (default "R").
candidate_pool	Optional list of per-condition candidate-pool sizes (list(condition, n_candidates)): how many items satisfied each condition's window before matching, that is, the size of the pool the selection drew from. It is recorded so that item-selection bias can be audited (Forster, 2000; Simmons et al., 2011).
norms	Optional list of norm-table provenance records, from the design's norms: block. Each names a file, its sha256, the join key and the per-column coverage. Recorded because a norm table can supply the very variable a design manipulates, so a record that omitted it would describe a selection over columns of unstated origin.
balance	Optional balance-optimiser report, from balance_lists(). Recorded because it decides which items each participant sees.
blocks	Optional report of the practice and filler blocks. Recorded because those trials are presented but not analysed, so the presented and analysed counts differ and the record must say why.
design_path, schema_path	Optional paths of the design and schema files the run read. When given, their sha256 checksums complete the reproducibility record, because those two files decide everything the seed does not. The schema's path is recorded too, in POSIX form, since an unnamed schema is chosen from several candidates.
selection_audit	Optional matcher audit record. Its window_relaxations entries are recorded because a relaxed window changes what 'matched' means for that condition. Its incomplete_dropped entries (list(condition, n)) count the candidates excluded for missing a matched dimension, which candidate_pool still includes. They are recorded because the selection drew on a smaller pool than the window holds.

`neighbourhood_reference`
: Optional record of a supplied reference the derived dimensions were recomputed against (`list(source, n_words, sha256, applied_to)`, where `applied_to` lists the recomputed columns), recorded verbatim.

`counterbalance_check`
: Optional report of the pipeline's rotation check on the counterbalanced table. Without it, the check is run here on `stimuli`. The record gives the recipe that ran and the lists the table holds, or for a table without a `list` column the lists `counterbalance()` would deal. It says whether a Latin-square rotation is complete and counts the trials per list and condition. It also gives the presentation versions, one per (replicate, list) pair, and the rule that assigns each participant one of them.

`input_normalisation`
: Optional `list(changed, collapsed)`: how many text keys the run's inputs held that Unicode NFC normalisation rewrote, and how many duplicates the rewriting created and removed. Recorded only when given, which the pipeline does only when either count is above zero, because a merged spelling changes the candidate pool.

`generation_exclude`
: Optional list of the known-form lists of a generated design's `items.generation.exclude`, each `list(source, sha256, n_forms)`. A pseudoword was refused for equalling any form in them, so they decide which pseudowords a run produces, as the lexicon does.

`exclusions`
: Optional record of a corpus, pool or generated design's `exclude_words`, as `list(path, sha256, n_listed, n_removed)`. It gives the file, or `"inline"` for a list in the design, its checksum, the number of distinct words it lists and the number of candidates it removed. Recorded as `materials_source$exclusions`, because the words decide which candidates the selection could draw.

`zipf_derivation`
: Optional `list(n_values, near_boundary)` for a lexicon carrying `freq_permillion`, as `normalise_lexicon()` writes one: how many positive rates it holds, and how many put Zipf = log10(per million) + 3 within 1e-9 of a three-place rounding boundary, where a last-bit difference between `log10` implementations could change the rounded value. Recorded as `materials_source$zipf_derivation`.

`trial_order`
: Optional record of the trial-order settings on the presented trials, which the pipeline passes. Without it, the record is taken from `stimuli`. It gives `counterbalancing$max_run` (`list(k, run_on, longest)`, the longest run realised in each presentation version) when the design limits runs, and `counterbalancing$orders` (`list(m, rule)`) when it asks for more than one trial order.

Details

For a corpus or generated design, `materials_source$provenance` credits the lexicon that `source_path` names. A lexicon lexsync ships is credited by its source. A corpus fetched into the cache is credited by the registry's citation, and any other file by its file name and checksum.

`design$n_per_condition` is the request the selector applied, which is the selector's default when the design gives none. `items$n_per_condition_realised` counts the items each condition holds in `stimuli`, per replicate for resampled sets, so a selection that fell short of the request under

`shortfall: allow` shows it. The suggested analysis keys its by-item slope on the recipe that ran and adds a matched design's matched dimensions that vary among its items as covariates.

Value

The datasheet as a nested list, ready for `write_datasheet()`.

References

Forster, K. I. (2000). The potential for experimenter bias effects in word recognition experiments. *Memory & Cognition, 28*(7), 1109-1115. doi:10.3758/BF03211812

Simmons, J. P., Nelson, L. D., & Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science, 22*(11), 1359-1366. doi:10.1177/0956797611417632

---

build_lexdec_stimuli *Assemble a lexical-decision set of words and pseudowords from a candidate pool*

Description

Real words are drawn by an even spread across the byte-ordered pool, then a length-matched pseudoword is generated for each. The pool is first filtered to the words written wholly in letters the reference attests, which the generators substitute among, so the eligible pool can be smaller than the request. The pipeline's shortfall policy then decides whether that is an error. A pool word in a logographic script is refused. `reference_words` (the full lexicon) supplies the letters, the bigram statistics and the real-word list a pseudoword must avoid. The presented string is the `target` column. The conditions are `word` and `pseudoword`, and `set` pairs them.

Usage

```
build_lexdec_stimuli(
  pool,
  n,
  reference_words = NULL,
  method = "letter_substitution",
  exclude = NULL
)
```

Arguments

`pool`	Data frame of candidate words, such as the one `build_pool()` returns.
`n`	Number of real words to select.
`reference_words`	Character vector supplying the letters, the bigram statistics and the real word forms a pseudoword must avoid. It defaults to the pool's words.

`method`	Pseudoword generator: `"letter_substitution"` (default) or `"subsyllabic"`, which swaps subsyllabic constituents as Wuggy does (Keuleers & Brysbaert, 2010).
`exclude`	Optional character vector of further known word forms a pseudoword must not equal, which the reference may lack, as for `generate_pseudowords()`.

Value

A stimulus data frame with the columns `target`, `word` (the same form), `condition` (`word` or `pseudoword`), `length` (the base word's `length` from `pool`, which its pseudoword shares), `set` (pairing each word with its pseudoword), `base` (the word each pseudoword was made from, and a word's own form for a word) and `edit_distance` (the Levenshtein distance from `target` to `base`, 0 for a word). The word rows come first, then the pseudowords in the same order.

References

Keuleers, E., & Brysbaert, M. (2010). Wuggy: A multilingual pseudoword generator. *Behavior Research Methods, 42*(3), 627-633. doi:10.3758/BRM.42.3.627

`build_pool` *Build an experimental candidate pool by filtering a lexicon*

Description

A filter naming a column the frame does not have is silently skipped, because the same function filters lexica, supplied pools and pair tables, and those carry different columns. The cost is that a misspelt key silently widens a selection, so every caller that takes its filters from a design checks the names against the frame first: `run_pipeline()` for `pool_filters`, `match_stimuli()` for a condition's `define_by`, and the pair selector for both.

Usage

```
build_pool(lexicon, filters = NULL)
```

Arguments

`lexicon`	A lexicon data frame.
`filters`	A named list mapping columns to filters. A numeric vector of two values is an inclusive range, `c(min, max)`, and any other vector a set of permitted values, compared as text. To permit exactly two numbers, write them as text, as in `c("1", "3")`.

Value

The filtered lexicon, with row names dropped. A row missing the filtered column is dropped under either kind of filter, and a range with a reversed or non-finite bound is an error rather than an empty pool.

Examples

```
schema <- yaml::read_yaml(system.file("extdata", "schema.yaml", package = "lexsync"))
lex <- load_lexicon(system.file("extdata", "en_example.csv", package = "lexsync"),
                    schema)
nrow(build_pool(lex, list(length = c(4, 6), frequency = c(4, 6))))
```

`cohens_d` *Cohen's d (pooled-SD standardised mean difference)*

Description

Cohen's d (pooled-SD standardised mean difference)

Usage

```
cohens_d(x, y)
```

Arguments

`x`, `y`	Numeric vectors.

Value

The standardised mean difference. It is 0 when both samples share one constant, and `NA` when either sample has fewer than two values (one value has no spread, so the pooled SD is undefined) or when the pooled SD is zero but the means differ (the standardised difference is then unbounded, not zero).

Examples

```
cohens_d(c(5, 6, 7, 8), c(5, 6, 7, 9))
```

`cohens_d_ci` *Cohen's d with a confidence interval, complementing the TOST verdict*

Description

The interval is the `(1 - 2 * alpha)` confidence interval for the standardised mean difference, so `alpha = 0.05` gives the 90% interval that goes with a TOST decision at the .05 level (Lakens, 2017). Reporting the interval, rather than only a binary verdict, makes the realised imbalance and its sampling uncertainty explicit, and keeps the dependence on the number of items visible. With few items, the interval is wide, whatever the point estimate. Both the interval and the TOST treat the selected items as a sample, although a test on stimuli concerns a closed population. Sassenhagen and Alday (2016) recommend describing the stimuli and entering potential confounds in the analysis as covariates.

Usage

```
cohens_d_ci(x, y, alpha = 0.05)
```

Arguments

x, y	Numeric vectors.
alpha	Significance level matching the TOST (default 0.05).

Details

The limits are `d +/- t * SE(d)`, where `t` is the Student t quantile on `nx + ny - 2` degrees of freedom and `SE(d)` is the large-sample standard error of d, `sqrt(1/nx + 1/ny + d^2 / (2 * (nx + ny)))` (Hedges & Olkin, 1985; Borenstein et al., 2009, eq. 4.20). The `d^2` term carries the sampling error of the pooled standard deviation. It is negligible for a well-matched control, where d is near zero, but it dominates for a manipulated dimension and can more than double the width of that interval. TOST tests the raw mean difference, so the interval is slightly wider than the one its decision implies, and when a control's limit lies close to the bound, `tost_p` gives the verdict.

Value

A list with `d`, `ci_low` and `ci_high`, all three `NA` when either sample has fewer than two values, as for `cohens_d()`.

References

Borenstein, M., Hedges, L. V., Higgins, J. P. T., & Rothstein, H. R. (2009). *Introduction to meta-analysis*. Wiley. doi:10.1002/9780470743386

Hedges, L. V., & Olkin, I. (1985). *Statistical methods for meta-analysis*. Academic Press. doi:10.1016/C20090033960

Lakens, D. (2017). Equivalence tests: A practical primer for *t* tests, correlations, and meta-analyses. *Social Psychological and Personality Science, 8*(4), 355-362. doi:10.1177/1948550617697177

Sassenhagen, J., & Alday, P. M. (2016). A common misapplication of statistical inference: Nuisance control with null-hypothesis significance tests. *Brain and Language, 162*, 42-45. doi:10.1016/j.bandl.2016.08.001

`counterbalance` *Assign stimuli to lists and a randomised, reproducible trial order*

Description

The recipe is `counterbalance.recipe` when the design names one, and otherwise the recipe of the paradigm it names. That is the factorial recipe for matched word lists, or a Latin square over conditions for the paired and sentence paradigms. A design's own `events` list has no say in the recipe. The factorial recipe refuses more lists than item sets, since the surplus lists would be empty.

Usage

```
counterbalance(stimuli, design, schema, list_of_set = NULL)
```

Arguments

stimuli	A stimuli data frame (matched set or loaded item table).
design	A parsed design configuration. Reads paradigm, counterbalance.recipe, counterbalance.lists, counterbalance.max_run, counterbalance.run_on, counterbalance.orders and fillers.
schema	The parsed global schema, which provides the seed.
list_of_set	Optional named integer vector mapping each set to a list, from balance_lists(). The pipeline supplies it when counterbalance.optimise is on. When it is NULL, the factorial recipe deals sets to lists by rank, the first set to list 1, the second to list 2 and so on, cycling.

Details

Each list's trials are ordered by the seeded keyed hash. counterbalance.max_run repairs that order so that no run of one counterbalance.run_on value (by default the condition) is longer than the limit, and refuses a limit no order can meet. A design with fillers has the limit applied when the fillers are interleaved, to that order. counterbalance.orders: m adds order_1 to order_m, each trial's position in each of m orders of its list, order_1 being the order trial gives.

Value

The trials, sorted by replicate, list and trial, with list and trial columns added, and order_1 to order_m when there is more than one order. Under the factorial recipe, every row of stimuli appears once, in the list its set is dealt to. Under the Latin square, each list holds one row per item set, in the condition the rotation gives that set in that list, so the result has one row per set per list and not every row of stimuli.

---

count_syllables *Orthographic syllable estimate: the number of maximal vowel runs*

Description

Counts the maximal runs of Latin vowel letters after decomposing the word and setting its diacritics aside, so a vowel counts whatever its accent. The vowels of Polish, Czech, Hungarian, Romanian, Turkish, Danish, Norwegian and Vietnamese count as those of English, Spanish, French and German do. Hiatus (idea counts 2) and syllabic consonants such as the Czech l in vlk and r in krk stay undercounted.

Usage

```
count_syllables(word)
```

Arguments

word	Character vector of word forms.

Value

Integer vector: the estimated syllable count of each word, or NA for a missing word and for a word holding a character outside the Latin script (Chinese, Cyrillic or Greek, for example), where the estimate is undefined.

Examples

```
count_syllables(c("cat", "table", "beautiful"))
```

---

describe_stimuli *Per-group descriptive statistics for several dimensions*

Description

A grouping column or a dimension the stimuli do not carry is refused.

Usage

```
describe_stimuli(stimuli, dims, by = "condition")
```

Arguments

stimuli	A stimuli data frame.
dims	Character vector of dimension columns.
by	Grouping column (default "condition").

Value

A long data frame with one row per group and dimension, the groups in order of first appearance. Its columns are group, dimension, n (the non-missing values), mean, sd, min, median and max, the last five rounded to three decimal places. A value that is not a number counts as missing.

design_stem *The stem every file a design writes is named after*

Description

The slug of the design's `name` and `language`, which keeps letters and digits only. Two designs whose names differ only outside ASCII (two Chinese names, both with the language `chinese`) would share a slug and overwrite each other's files, and an all-Chinese name and language would give no slug at all. Whenever the name or the language holds a non-ASCII character, the stem therefore gains _ and the first eight hexadecimal digits of the SHA-256 of the NFC-normalised UTF-8 text `name|language`. An empty slug is replaced by those eight digits. NFC keeps a precomposed and a decomposed accented letter on one stem. A design whose name and language hold no letter or digit at all is refused.

Usage

```
design_stem(design)
```

Arguments

`design`	A parsed design configuration.

Value

The file stem, such as `"en_lexdec_english"` or `"chinese_14aadfa7"`.

Examples

```
design_stem(list(name = "en_lexdec", language = "english"))
# Two Chinese characters, written as code points to keep this file ASCII.
design_stem(list(name = intToUtf8(c(0x8bcd, 0x9891)), language = "chinese"))
```

export_experiments *Export all presentation targets (PsychoPy, OpenSesame and jsPsych)*

Description

Assigns the EEG trigger codes with `assign_triggers()` and then writes each target. The condition codes follow `conditions`, which defaults to the order of the design's `conditions` entries, so in a design that declares its conditions the first is 101, the second 102 and so on, whatever order the counterbalanced trials put them in. A design with no `conditions` block (such as a generated lexical decision or an item table) falls back to the order of first appearance, unless `conditions` is given. `assign_triggers()` is given the design's events and the schema, so the item codes of a design that sends them alongside other codes are kept apart from those codes, and a code of the design's own that the stream already uses is reported.

Usage

```
export_experiments(
  stimuli,
  design,
  schema,
  outdir,
  base = NULL,
  conditions = NULL
)
```

Arguments

stimuli	Stimuli with trigger columns (see assign_triggers()).
design	A parsed design configuration.
schema	The parsed global schema, which gives the trigger and presentation settings.
outdir	Output directory.
base	Optional file-name stem, by default the design's design_stem().
conditions	Optional character vector of condition labels in the order that assigns their trigger codes (see assign_triggers()). NULL (the default) takes the order of the design's conditions.

Details

Each trial that names no answer of its own takes the key the design's response mapping gives its condition in its list, so the loop tables can score every trial. A lexical decision maps word to its response event's first key and pseudoword to its second, unless the design gives response_mapping, and rotate_mapping_by_list: true rotates the keys across lists. A practice or filler trial keeps the answer its table gives. The stimuli passed in are not changed. Every target shows the design's instructions, or lexsync's English instructions for its task, and waits for the space bar before the first trial.

Value

A named list of generated file paths: psychopy, opensesame (version 1's experiment when a design has several, see export_opensesame()) and jspsych. files lists every file written, in a fixed order: the PsychoPy script, each OpenSesame experiment in version order, the jsPsych page, the PsychoPy conditions file and then each OpenSesame experiment's loop table and text table.

---

export_jspsych *Export a browser-runnable jsPsych experiment*

Description

The rendered events and the trial data are embedded in one HTML file, so anyone can reproduce the procedure online from the same materials. The jsPsych library, its keyboard-response plugin and its stylesheet are loaded from a CDN, so the machine running the file needs an internet connection. The trial data are embedded and the responses are saved locally, so no server is required either to run it or to collect them. Each of the three files carries its SHA-384 as Subresource Integrity, so a browser refuses a copy that differs from the published release. Stimuli are drawn `presentation.text_height_px` pixels high, 48 by default. Onset triggers are recorded in each trial's data (a browser cannot drive a parallel port). A stimulus that collects its response is one trial, recorded as `target`. It accepts the keys from its onset, shows the stimulus for its duration and stays open for the timeout after it. A key ends it, and `rt` runs from the onset.

Usage

```
export_jspsych(stimuli, design, schema, outdir, base = NULL)
```

Arguments

`stimuli`	Stimuli with trigger columns (see `assign_triggers()`).
`design`	A parsed design configuration.
`schema`	The parsed global schema, which gives the trigger and presentation settings.
`outdir`	Output directory.
`base`	Optional file-name stem, by default the design's `design_stem()`.

Details

The page embeds every presentation version of the design, one per (replicate, list) pair, and runs one, named in its address: `?version=K`, `?list=K` when there are no replicates, or `?participant=N`, whose version is `((N - 1) mod V) + 1`. A page with more than one version, opened with none of these, shows a screen naming them and runs nothing. With `counterbalance.orders: m`, the page presents the version in order `(((N - 1) div V) mod m) + 1`, or in order 1 without a participant number, and every data row carries the order. Its welcome screen shows the version's instructions, the design's `instructions` or lexsync's for its task, and waits for the space bar. The data rows carry the participant, the version, the list and the replicate, and the data file is named after the participant and the version. The page names the keys as the browser does (`arrowleft` for `left`) in its events and in each trial's `answer`, so a recorded response can be compared with the answer as it stands.

Value

The path to the generated `.html`, invisibly.

`export_opensesame` *Export a complete plain-text OpenSesame experiment*

Description

Writes `<base>.osexp` and the two tables it reads. Its loop runs over `<base>_opensesame.csv`, which holds numbers only: `lexsync_row` (the trial's position in loop order, from 1), `lexsync_trial`, `lexsync_list`, `replicate` in a resampled design, `lexsync_set`, the trigger codes, `critical_region` and any per-trial duration. The presented text, `condition`, `block`, `answer` and, with `counterbalance.orders`, `order_1` to `order_m` are in `<base>_opensesame_text.csv`, which the inline scripts read with Python's `csv` module, so OpenSesame never evaluates them. Copy both tables with the experiment.

Usage

```
export_opensesame(stimuli, design, schema, outdir, base = NULL)
```

Arguments

`stimuli`	Stimuli with trigger columns (see `assign_triggers()`).
`design`	A parsed design configuration.
`schema`	The parsed global schema, which gives the trigger and presentation settings.
`outdir`	Output directory.
`base`	Optional file-name stem, by default the design's `design_stem()`.

Details

Before the loop, an inline script, `lexsync_instructions`, shows the design's `instructions`, or lexsync's for its task, and waits for the space bar. When the trials hold practice trials and trials of another block, the experiment pauses where the practice trials end, as the PsychoPy and jsPsych experiments do. An inline script, `lexsync_block_break`, opens every trial, and on the first trial after the practice trials, it shows 'End of the practice trials.' and waits for the space bar. Escape there stops the session. Every trial is logged. A response after a text event is collected by that event's script from the stimulus's onset, which sets `response`, `response_time`, `timed_out` and `correct`. A response kept after the stimulus with `from: offset` is a keyboard_response, and a script after it sets `timed_out` and `correct`. Such a response cannot be restricted to `blocks`: a design that restricts one is refused here, although the PsychoPy and jsPsych targets accept it. Self-paced reading sets `lexsync_region_rts` and a question `question_response` and `question_rt`. A script clears these before each trial's first event, another copies the trial's labels into `lexsync_condition`, `lexsync_block` and `lexsync_answer`, and a logger item ends the trial.

Every canvas a trial shows is built before the trial runs, in the prepare phase of its script, and each event is timed from the onset its display returns. An event ends half a frame before its duration is up, so that the next display is shown on the refresh that ends it. The frame period is measured at start-up and logged as `lexsync_frame_ms`, with `presentation.assumed_refresh_hz` as the fallback when the measurement fails. Each trigger code is held for `triggers.trigger_hold_ms`, at least 10 ms, through the events after it when its own is shorter, unless the next onset's code replaces it

first. The session markers, 254 and 255, are sent just before the first trial and just after the last, held the same way.

The setup writes the codes to the parallel port at `triggers.parallel_address` or, with `triggers.backend: serial`, as single bytes to `triggers.serial_port` at `triggers.baudrate`. When that device cannot be opened, `triggers.mode: auto` prints the codes instead. An inline script, `lexsync_trigger_notice`, then says so before any other screen and waits for the C key. `hardware` stops the experiment there, and `mock` prints the codes without trying the device, after the same notice. Every trial logs `lexsync_trigger_mode`, `hardware` or `mock`. Text is drawn `presentation.text_height_px` pixels high, OpenSesame's `font_size`, and OpenSesame takes full screen from the mode it is launched in.

A design with more than one presentation version, one per (replicate, list) pair of the trials, gets one experiment per version instead, each with its own two tables holding only that version's trials: `<base>_list<k>.osexp`, or `<base>_rep<r>_list<k>.osexp` when the versions span several replicates. Each one checks OpenSesame's subject number at start-up and shows a warning when that participant would normally run another version. A design with `counterbalance.orders: m` gets one experiment per version and order, `<base>_list<k>_order<j>.osexp` (`<base>_order<j>.osexp` for a single version), each with tables sorted by its order, a `lexsync_order` variable that every trial logs, and a check that compares the participant's order too, order `(((p - 1) div V) mod m) + 1`. `export_experiments()` lists every file written.

OpenSesame evaluates the cells of a loop table: in 4.x, a cell holding `{` runs as a Python f-string, and in 3.3, a cell starting with `=` or holding `[=...]` runs as Python. A value in a numeric column that is not a plain number is therefore refused, naming the column. An empty cell passes. These rules were checked against the sources of OpenSesame 3.3.14 and 4.1.20.

Value

The path to the generated `.osexp`, version 1's when there are several, invisibly.

`export_psychopy` *Export a runnable PsychoPy script that interprets the event sequence*

Description

The conditions file, `<base>_psychopy.csv`, holds every presentation version of the design, one per (replicate, list) pair, and the script presents one per run: the version given by `--version` (or `--list` when there are no replicates), otherwise the participant's, by the rule `((participant - 1) mod V) + 1` from `--participant`, and otherwise the one a start-up dialogue asks for. A version the script does not have stops the run with a message. With `counterbalance.orders: m`, the conditions file holds each trial's position in each of m orders, and the script presents the version in order `(((participant - 1) div V) mod m) + 1`, or in order 1 without a participant number. The data file then records the order.

Usage

```
export_psychopy(stimuli, design, schema, outdir, base = NULL)
```

Arguments

stimuli	Stimuli with trigger columns (see assign_triggers()).
design	A parsed design configuration.
schema	The parsed global schema, which gives the trigger and presentation settings.
outdir	Output directory.
base	Optional file-name stem, by default the design's design_stem().

Details

Before the first trial, the script shows the version's instructions, the design's instructions or lexsync's for its task, and waits for the space bar, which also lets the experimenter start the run. Escape there stops the run before anything is recorded. The text is embedded as JSON data, one text per version.

Each run writes data/<base>_p<participant>_v<version>_<date>-<time>.csv beside the script, without the _p part when no participant number is given, one row per trial, each written as its trial ends. A row holds the run, the trial's fields and trigger codes, the response and its reaction time from the onset of the stimulus it answers, whether the window closed without one, whether the key was the trial's answer, the self-paced reading times, the question's response and reaction time, the measured refresh rate, the trigger mode the run used and each event's realised flips and onset.

Each trigger code goes out on its stimulus's first flip and is held for triggers.trigger_hold_ms, in whole flips at the measured refresh and never under 10 ms, through the events after it when its own is shorter, unless the next onset's code replaces it first. The session markers, 254 before the first trial and 255 after the last, are held the same way.

The script runs full screen unless presentation.fullscreen is false, when it opens a 1280 x 1024 window, and draws its stimuli presentation.text_height_px pixels high, 48 by default. It writes the codes to the parallel port at triggers.parallel_address or, with triggers.backend: serial, as single bytes to triggers.serial_port at triggers.baudrate. triggers.mode decides what happens when that device cannot be opened. Under auto, the script prints the codes instead and, before the instructions, shows a notice that the experimenter dismisses with the C key. Under hardware, it stops before its window opens, and under mock, it prints the codes without trying the device, after the same notice. The data file's trigger_mode column records hardware or mock.

Value

The path to the generated .py, invisibly.

---

fetch_corpus *Download a registered corpus into the cache*

Description

Downloads the file a registry entry names and caches it as a lexicon that load_lexicon() reads. name must be a single corpus name: one holding a slash, a backslash or a colon is refused before the registry is read, since the name becomes the cached file's name.

Usage

```
fetch_corpus(name, registry_path = NULL, dest = NULL)
```

Arguments

name	A corpus name present in the registry.
registry_path	Optional path to registry.yaml. A path that does not exist is refused. NULL (the default) finds the registry as list_corpora() describes.
dest	Optional destination file. It defaults to <name>.csv in lexsync_cache_dir().

Details

An entry's derived file, a lexicon already normalised, is fetched when it has one and cached as it arrives. Otherwise, its openlexicon file is fetched and converted by normalise_lexicon() with the entry's columns and duplicates, so fetch_corpus("subtlex_us") caches SUBTLEX-US as a lexicon. The URL's scheme is checked first. The download is then written to a .part file beside the destination, which is renamed to the destination only after the file has passed the size limit, the check that it is not an HTML or XML page, any sha256 the registry entry gives and the normalisation. A refusal of the registry entry, a failed transfer and a checksum mismatch name the registry file read. On success, a message gives the corpus's citation, which list_corpora() also lists, and a run that reads the file from the cache credits that citation in its datasheet (see build_datasheet()). The transfer uses utils::download.file(), so it is subject to getOption("timeout"), 60 seconds by default, which is often too short for a large file. Raise it, for example with options(timeout = 600), before fetching a large corpus.

The lexica of the wordfreq connector's languages are built by the Python package (lexsync fetch fr), and this package reads the result with load_lexicon(). A wordfreq language code given here is refused with that explanation.

The file is written to lexsync_cache_dir() unless dest names another path. The cache directory is created only once the registry entry and its URL have been accepted, so a refused call writes nothing. A download to the default destination begins by deleting any .part file in the cache that an interrupted transfer left there more than a day earlier. The cache persists between sessions, and one corpus may reach the 200 MB download cap, so several of them add up. lexsync_cache_clear() removes one corpus or the whole cache, and the next call downloads the corpus again. A dest the caller names, inside the cache or not, is used as given. Its directory must already exist, no .part file is deleted, and the only file beside it that lexsync writes or deletes is the <dest>.part file of the download itself.

Value

The path to the downloaded file, invisibly.

generate_pseudowords *A length-matched pseudoword for each base word*

Description

The base words are processed in byte order, so the pseudoword each one receives does not depend on the order in which they are given.

Usage

```
generate_pseudowords(base_words, reference_words, exclude = NULL)
```

Arguments

base_words	Character vector of words to derive pseudowords from.
reference_words	Character vector (typically the full lexicon) that supplies the bigram statistics and the real word forms to avoid.
exclude	Optional character vector of further word forms no pseudoword may equal, such as low-frequency words the reference lacks. They are trimmed, lower-cased and put in Unicode NFC, as a lexicon's words are, and play no part in the bigram statistics.

Details

The letters substituted in are those the reference words attest, so a Spanish reference lends the generator its accented vowels. A base word in a logographic script, such as Chinese, is refused.

Value

A data frame with columns base_word and pseudoword, in the order of base_words.

lexsync_cache_clear *Remove fetched corpora from the cache*

Description

Deletes what fetch_corpus() put in lexsync_cache_dir(). Given a name, that is the corpus's <name>.csv and any <name>.csv.part that an interrupted download left behind. With name left NULL, it is the whole cache directory. A corpus needed again is downloaded afresh by the next fetch_corpus() call. A file that fetch_corpus() wrote to a dest of the caller's choosing is not in the cache, and is never touched.

Usage

```
lexsync_cache_clear(name = NULL)
```

Arguments

name	A corpus name as given to fetch_corpus(), or NULL (the default) to remove the whole cache.

Value

The paths of the files removed, invisibly. The vector is empty when there was nothing to remove.

See Also

lexsync_cache_dir() for the location of the cache.

Examples

```
# A throwaway cache stands in for yours, so running this leaves your own alone.
local({
  tmp <- tempfile("lexsync-")
  old <- Sys.getenv("R_USER_CACHE_DIR", unset = NA)
  on.exit({
    if (is.na(old)) Sys.unsetenv("R_USER_CACHE_DIR") else
      Sys.setenv(R_USER_CACHE_DIR = old)
    unlink(tmp, recursive = TRUE)
  })
  Sys.setenv(R_USER_CACHE_DIR = tmp)
  # Two corpora and an interrupted download
  dir.create(lexsync_cache_dir(), recursive = TRUE)
  invisible(file.create(file.path(lexsync_cache_dir(),
    c("subtlex_nl.csv", "subtlex_nl.csv.part", "lexique_fr.csv"))))
  # One corpus, with its partial download
  print(basename(lexsync_cache_clear("subtlex_nl")))
  # Everything else, and the directory itself
  print(basename(lexsync_cache_clear()))
  dir.exists(lexsync_cache_dir())
})
```

---

lexsync_cache_dir *Per-user cache directory for fetched corpora*

Description

The directory tools::R_user_dir("lexsync", "cache") names for this package. It is where fetch_corpus() puts a download unless told otherwise, and it is the only place the package writes to without being handed a path. This function only reports the path. The directory is created by the first download into it, so until then it does not exist.

Usage

```
lexsync_cache_dir()
```

Details

The cache persists between sessions. A registered corpus is a delimited word list, and a download is refused above 200 MB, so a cache holding several large corpora can reach a few hundred megabytes. Each download into the cache first deletes any partial download that an interrupted transfer left there more than a day earlier, and fetching a corpus again replaces the earlier copy. Downloaded corpora otherwise stay until `lexsync_cache_clear()` removes them, one corpus at a time or all together. Nothing kept there is irreplaceable, so the next call downloads afresh.

Value

The cache directory path, which need not exist yet.

See Also

`lexsync_cache_clear()` to empty the cache.

Examples

```
lexsync_cache_dir()
```

`list_corpora` *List the corpora known to the registry*

Description

List the corpora known to the registry

Usage

```
list_corpora(registry_path = NULL)
```

Arguments

`registry_path`	Optional path to `registry.yaml`. A path that does not exist is refused. `NULL` (the default) takes the file `getOption("lexsync.registry")` names, which must exist. With the option unset or empty, it takes the first that exists of `corpora/registry.yaml` and `../corpora/registry.yaml` relative to the working directory, and last the copy bundled with the package, `system.file("extdata", "registry.yaml", package = "lexsync")`.

Value

A data frame with one row per entry under `corpora` in the registry and the columns `name`, `language` (the language's name), `iso` (its code), `status` (`validated`, `supported`, `manual` or `listed`, as the registry's header defines them), `connector` (`"openlexicon"` when the entry names none) and `citation`. The wordfreq connector and its languages have no row, and each entry's licence is in the registry file, not in the data frame.

load_items *Load a paradigm item table (such as prime-target pairs or sentences)*

Description

The table must carry an `item` identifier, a `condition` label and the paradigm's presented fields. Field values are validated (no control characters and a bounded length), so a crafted item cannot corrupt the generated loop table or scripts. Items are mapped to deterministic integer `set` ids in byte order, so counterbalancing works as it does for a corpus design. The item id, the condition and the presented fields are normalised to Unicode NFC. When that changed any cell, the attribute `input_normalisation` holds `list(changed, collapsed)`, with `collapsed` always 0 since no row is removed.

Usage

```
load_items(path, required_fields)
```

Arguments

`path`	Path to a UTF-8 CSV item table. A path with a `..` component is refused. Give it from the working directory, or as an absolute path.
`required_fields`	Character vector of presented fields the paradigm needs.

Value

A data frame with `set`, `condition` and the item fields.

load_lexicon *Load a lexicon from a CSV file*

Description

Reads a derived lexicon, validates the column contract, lower-cases the orthographic form and normalises it to Unicode NFC, removes duplicates (keeping the first row in file order) and attaches a stable integer `id` plus the inexpensive dimensions `length` and `frequency`. The orthographic neighbourhood dimensions are added later, on the experimental pool, by `add_neighbourhood()`, because they are quadratic in the size of the reference set.

Usage

```
load_lexicon(path, schema, language = NULL)
```

Arguments

path	Path to a derived lexicon CSV.
schema	The parsed schema, such as `yaml::read_yaml(system.file("extdata", "schema.yaml", package = "lexsync"))`. Its `dimensions$frequency$column` names the frequency column.
language	Optional language label to record in a `language` column.

Details

A `freq_zipf` value outside 0 to 9 is refused: it is almost always a frequency per million in the wrong column (convert it with `normalise_lexicon()`). A log count such as SUBTLEX-US's `Lg10WF`, log10(count + 1) and at most 6.33, passes the check and must still be converted, since it reads about 1.3 below Zipf.

Value

A data frame with at least `word`, `length`, `n_syllables`, `frequency` and `id`, plus the frequency column the schema names (by default `freq_zipf`) and every other column the file carried. Rows are in byte order of `word`, and `id` numbers them from 1. When normalisation changed or merged any word, the attribute `input_normalisation` holds `list(changed, collapsed)`: the words it rewrote and the duplicate rows it removed.

Examples

```
# Both inputs are bundled with the package, so this runs offline and touches
# nothing outside the installation.
schema <- yaml::read_yaml(system.file("extdata", "schema.yaml", package = "lexsync"))
lex <- load_lexicon(system.file("extdata", "en_example.csv", package = "lexsync"),
                    schema)
head(lex[, c("word", "frequency", "length", "n_syllables")])
```

---

load_pool *Load a supplied candidate pool of words and give it the matcher's dimensions*

Description

A curated word list, such as one from a previous study, a norming session or a colleague, can be matched, validated and documented without first being made into a corpus lexicon. This function reads such a list and returns a pool the matcher accepts.

Usage

```
load_pool(path, schema, lexicon = NULL, language = NULL)
```

Arguments

path	Path to a UTF-8 CSV with at least a word column. A path with a .. component is refused. Give it from the working directory, or as an absolute path.
schema	The parsed schema (used when a lexicon is loaded).
lexicon	Optional path to a derived lexicon to draw dimensions from.
language	Optional language label recorded in a language column.

Details

The list needs only a word column. Length and the syllable estimate are derived from the form. Every other dimension is either supplied on the list itself or looked up. With lexicon given, the corpus dimensions (frequency above all) are joined for those words, and a word the lexicon does not have is an error rather than an NA, because a word missing a value would be left out of the candidates and the pool would then be smaller than the user believes it is.

The returned reference matters as much as the pool. n_density and old20 describe a word within its language, not within the few words a study uses, so computing them against a 200-word supplied list would give meaningless numbers. When a lexicon is given, the reference is the lexicon's words. Without one, it is the pool itself. A reference_member column on whichever of the two supplies the reference keeps only the rows it marks 1.

Value

A list with pool (the data frame, carrying word, id, length, n_syllables and any joined or supplied dimensions) and reference (the word vector the neighbourhood dimensions should be computed against). Words are trimmed, lower-cased and normalised to Unicode NFC as in load_lexicon(), and pool carries the same input_normalisation attribute when that changed or merged any word of the pool or the lexicon.

---

log_artefact *Record a written artefact (path, rows, fingerprint) in the log*

Description

Record a written artefact (path, rows, fingerprint) in the log

Usage

```
log_artefact(log, path, rows = NA_integer_)
```

Arguments

log	A run-log object.
path	A file path that has just been written.
rows	Optional row count.

Details

The step is added with log_step(), so it is printed to the console as [lexsync] wrote '<file>' while the option lexsync.verbose is TRUE.

Value

The updated run-log object.

---

log_step *Append a step to a run log*

Description

Append a step to a run log

Usage

```
log_step(log, message, data = NULL)
```

Arguments

log	A run-log object.
message	A short description of the step.
data	An optional named list of step details.

Details

While the option lexsync.verbose is TRUE, its default, the message is also printed to the console as [lexsync] <message> with cat(), so suppressMessages() does not silence it. Set options(lexsync.verbose = FALSE) to keep it quiet. run_pipeline() sets the option from its verbose argument for the length of the run and restores it afterwards.

Value

The updated run-log object.

make_pseudoword *The most bigram-plausible legal non-word at the smallest edit distance*

Description

Searches single-letter substitutions first, then two-letter substitutions. Candidates are ranked by summed bigram frequency with a byte-order tie-break, so the choice is deterministic.

Usage

```
make_pseudoword(word, bigrams, lexset, usedset, alphabet = NULL)
```

Arguments

word	The base word to derive the pseudoword from.
bigrams	Named integer vector of bigram counts: one element for each pair of adjacent characters in the reference words, named by the pair and holding the number of times it occurs. A candidate is legal only when every one of its bigrams is among the names.
lexset	Environment used as a set of the real word forms a pseudoword must avoid: each form is the name of a variable in it, whatever its value.
usedset	Environment used as a set of the pseudowords already taken, in the same way. The function only reads it, so a caller generating several pseudowords assigns each result into it.
alphabet	Character vector of the letters substituted in, in UTF-8 byte order. The default, NULL, takes the letters of the bigram table's names, which hold every letter a legal candidate of two letters or more can use. generate_pseudowords() passes the letters of the whole reference.

Details

generate_pseudowords() builds bigrams and lexset from a reference word list and usedset from the pseudowords it generates, and is the function for ordinary use.

Value

A single pseudoword string, or NULL if no legal candidate exists.

match_report *Build the full match-quality report*

Description

Build the full match-quality report

Usage

```
match_report(stimuli, dims, schema, roles = NULL)
```

Arguments

stimuli	A matched-stimuli data frame (must contain condition).
dims	Dimensions to summarise and compare.
schema	The parsed global schema, which gives the equivalence settings.
roles	Optional function of condition, reference and dimension that returns the row's role: "manipulated" when the design sets the two conditions apart on the dimension, "controlled" when it matches them on it and "uncontrolled" otherwise. run_pipeline() derives it from the design. Without it, role is NA.

Value

A list with descriptives and comparisons data frames. Every pair of conditions is compared on every dimension. The rows against the first condition in order of appearance, the anchor, come first, then every further pair in condition order, so a design with a single condition has nothing to compare and comparisons comes back with its columns and no rows. A dimension with no numeric value among the stimuli, such as n_syllables for Chinese or a text label, is left out of both frames.

Each row of comparisons names the earlier condition of its pair in reference and the later one in condition. The signed statistics run from the reference to the other condition: cohens_d, d_ci_low and d_ci_high are the reference mean minus the condition mean, in pooled standard deviations, so a positive d means the reference scores higher. mean_diff is the same difference in the dimension's own unit, to three places. var_ratio is the other way up, the condition variance over the reference variance.

match_report_continuous *Realised-control report for a continuous design*

Description

Returns the same list shape as match_report() (descriptives and comparisons), but the comparisons describe a continuous predictor: its realised span and, for each control, the Pearson correlation with the predictor (near zero when the control is held constant).

Usage

```
match_report_continuous(
  stimuli,
  predictor,
  controls,
  schema,
  audit = attr(stimuli, "audit")
)
```

Arguments

stimuli	A stimuli data frame (a single continuous group).
predictor	The spanned predictor dimension.
controls	Character vector of control dimensions.
schema	The parsed global schema.
audit	The selector's audit record, by default the "audit" attribute select_continuous_stimuli() attaches to its result. Its control_windows give each control's window and pool SD. Without them, a control's sd_ratio and window are NA.

Details

sd is each dimension's realised SD. For the predictor, sd_ratio sets it against the SD of as many values spaced evenly over the same range, (max - min) * sqrt(n * (n + 1) / 12) / (n - 1) with the n - 1 that sd uses, so an even spread has a ratio of 1 whatever its size and a selection that follows a skewed pool a ratio below 1. That benchmark tends to (max - min) / sqrt(12), the SD of a uniform distribution over the range, as n grows. For a control, sd_ratio sets the realised SD against the SD of the pool the selector drew from, and window_low and window_high give the window the selector banded it to, before any relaxation. Both come from audit, the selector's record.

Value

A list with descriptives and comparisons data frames.

---

match_stimuli *Match stimuli across conditions on several lexical dimensions*

Description

Under the default method, the first condition is the anchor. Its items are chosen by an even spread across the sorted candidate subpool. Every other condition is then matched to the anchor item by item, on the match_on dimensions, using standardised Euclidean distance under a tolerance window derived from the anchor.

Usage

```
match_stimuli(pool, design, schema, verbose = FALSE)
```

Arguments

pool	A lexicon or pool with every match_on dimension present (see add_neighbourhood()).
design	A parsed design configuration (conditions, match_on and n_per_condition or n_per_cell). match_on must name at least one dimension. The number of items must be a whole number of at least 1 and is 20 when neither key is given.
schema	The parsed global schema, which holds the tolerances.
verbose	Whether to report tolerance relaxations and an anchor set smaller than requested.

Details

method, read from the design's matching block with the schema as fallback, chooses the matcher. "standardised_euclidean" (the default) is the anchored procedure above. "mahalanobis" is the same procedure with distances weighted by the inverse of the pool's correlation matrix, which, with two or more matched dimensions, needs at least k + 2 pool rows complete on the k dimensions. "joint" and "optimal" need exactly two conditions. They drop the anchor and pair the two conditions' candidates directly: joint takes the cheapest disjoint pairs first, and optimal takes the n pairs of least summed distance (exactly so for disjoint conditions) and needs the 'clue' package. Neither consults tolerance_k or on_insufficient_tolerance, and each first keeps at most 1,200 candidates per condition, those nearest the other condition's centroid. The Python package selects the same stimuli under standardised_euclidean and joint. Under mahalanobis and optimal, the two agree closely but not byte for byte, and optimal may return a different set of pairs with the same summed distance.

Two policies, each read from the design's matching block with the schema as fallback, govern a selection that cannot be made in full. Under shortfall, "error", the default, refuses to return fewer sets than requested, and "allow" accepts fewer. Under on_insufficient_tolerance, "relax", the default, widens an undersupplied tolerance window to the full condition subpool and records the relaxation in an "audit" attribute, and "error" refuses instead.

A candidate missing any match_on dimension cannot be scored, so every method excludes it before selecting, and the number excluded per condition is recorded in the "audit" attribute's incomplete_dropped entry. The shortfall policy then applies to what remains, and a condition left with no complete candidate is refused.

tie_break, read in the same way, decides between candidates at exactly the same rounded distance, which integer dimensions make common: "hash", the default, prefers the smaller SHA-256 of lexsync-tie-v1| followed by the word, which favours no part of the alphabet, and "bytes" prefers the spelling that sorts first and reproduces lexsync 0.1. The rule also orders the tied pairs of joint and optimal and the tied rows at their candidate cap. The anchor's even spread keeps its byte-order tie-break under both rules.

Value

A data frame of selected stimuli with a condition label and a set index pairing matched items across conditions.

`merge_norms` *Left-join a norm table (such as concreteness, age of acquisition or valence)*

Description

Norm data, such as semantic dimensions, are fetched separately, since their licences vary, and joined here so that the matcher can equate on them. The join is deterministic.

Usage

```
merge_norms(lexicon, norms, on = "word", columns = NULL)
```

Arguments

`lexicon`	A lexicon data frame.
`norms`	A data frame, or the path to a UTF-8 CSV, holding the `on` column and the norm columns.
`on`	The join column (default `"word"`).
`columns`	Optional norm columns to keep.

Details

The result is the lexicon itself with the norm columns appended, so no lexicon column is moved or renamed and no row is reordered. A norm column whose name the lexicon already has is refused.

The key is trimmed, case-folded and normalised to NFC on both sides, so a lexicon holding `Dog` matches a norm row for `dog`. The lexicon's own spelling is preserved rather than folded in place: `word` is the byte-order tie-break behind every selection, so the join must not rewrite it. A missing key never matches, and when two rows of the norm table share a key once it is normalised, the first of them is used.

Value

`lexicon` with the norm columns appended, in the lexicon's own row and column order. Rows with no matching norm get `NA`.

methods_paragraph *A ready-to-adapt methods paragraph rendered from a datasheet*

Description

Every number in the paragraph is printed by lexsync's own formatter, rounded half away from zero from the value the datasheet stores, so a record read back from its JSON gives the same text.

Usage

```
methods_paragraph(ds)
```

Arguments

ds	A datasheet list, from build_datasheet().

Details

The paragraph states the items each condition holds, and the request beside them when they fell short of it. It describes the realised control before testing it: the largest difference on a matched dimension in that dimension's unit, with Cohen's d and its interval, then the two one-sided tests, which it says cannot pass when every condition holds fewer items than the bound needs, then, largest first, up to five comparisons of an uncontrolled dimension whose |d| exceeds the bound. Every interval is labelled by the schema's alpha. A continuous design's paragraph gives the predictor's spread, the control windows and the controls' realised spread. When a response mapping applies, the paragraph says which key the participants pressed for which condition.

Value

A single character string describing the materials procedure.

new_run_log *Start a new run log*

Description

Start a new run log

Usage

```
new_run_log(name, meta = list())
```

Arguments

name	A label for the run.
meta	A named list of run-level metadata, such as the seed and the versions.

Value

A run-log object (a list).

`normalise_lexicon` *Turn a corpus file in its publisher's layout into a lexicon*

Description

Reads a tab- or comma-separated corpus file, such as SUBTLEX-US or Lexique as they are published, and returns a lexicon that `load_lexicon()` reads, writing it to `out` when given.

Usage

```
normalise_lexicon(
  path,
  columns = NULL,
  entry = NULL,
  delimiter = NULL,
  out = NULL
)
```

Arguments

`path`	Path to the corpus file.
`columns`	Optional named list mapping lexsync's column names to the file's, as a registry entry's `columns` does: `word`, either `freq_zipf` or `freq_permillion` and, optionally, `pos`. Without a map, the entry's is used. Without either, the file's own columns of those names are read.
`entry`	Optional registry entry, as a list, or the name of one in the registry lexsync finds by default (see `list_corpora()`). It supplies the map when `columns` is `NULL`, and the `duplicates` rule.
`delimiter`	`","` or a tab character. Left `NULL`, it is taken from the extension (`.tsv` and `.tab` are tab-separated, `.csv` comma-separated) and otherwise from the header line, which is tab-separated when it holds a tab. A tab-separated file is read with no quoting.
`out`	Optional path to write the lexicon to, comma-separated with LF line endings.

Details

Words are trimmed, lower-cased and put in Unicode NFC, as `load_lexicon()` keys them, and rows missing a word or a frequency are dropped. A form that then appears more than once is resolved by the entry's `duplicates` rule: `"first"` (the default) keeps the first row in file order, and `"sum"` adds its rows' frequencies per million, as Lexique needs, since it splits a homograph's frequency between its rows. Frequencies per million are converted with Zipf = log10(per million) + 3, rounded to three places by lexsync's own rounding function. A value of 0 or below has no Zipf value and is refused. Zipf values read as they are must lie between 0 and 9.

A message gives the number of forms and, for derived Zipf values, the number that lay within 1e-9 of a rounding boundary after scaling. That is the only step at which a last-bit difference between `log10` implementations could change a rounded value, and the count is expected to be 0. A run over the lexicon records the count in its run log and datasheet.

Value

The lexicon, invisibly when `out` is given: a data frame with `word`, `freq_zipf`, then `freq_permillion` when Zipf was derived from it, and `pos` when the file has one, in the UTF-8 byte order of `word`. Its attribute `normalisation` holds `list(n_read, n_forms, n_merged, duplicates, zipf_derived, near_boundary)`.

See Also

`fetch_corpus()`, which downloads a registered corpus and normalises it the same way.

Examples

```
# A SUBTLEX-US-like table, written to a temporary file.
path <- tempfile(fileext = ".tsv")
writeLines(c("Word\tSUBTLWF\tDom_PoS_SUBTLEX", "the\t29449.18\tArticle",
             "Dog\t192.76\tNoun"), path)
normalise_lexicon(path, columns = list(word = "Word", freq_permillion = "SUBTLWF",
                                       pos = "Dom_PoS_SUBTLEX"))
unlink(path)
```

PARADIGMS *The paradigm registry: default event sequences and required fields*

Description

Each event is a list with a `type` (`fixation`, `text`, `mask`, `blank`, `region_by_region`, `response`, `question` or `feedback`) and the keys its type reads. `content` is a literal or a `{field}` reference. An event's length, where it has one, is `duration_ms`, or `duration_frames` converted at the schema's `presentation.assumed_refresh_hz`, and a fixation, text, mask or blank event may instead take a per-trial `duration` block (`from_column`, or `jitter` with an optional `as`, as `resolve_trial_timing()` describes). A fixation, text or mask event takes an optional `trigger` (a whole-number EEG code from 1 to 255 or the token `condition` or `item`). A response takes `keys`, `timeout_ms` and `from` (where its window opens: `onset`, with the text event before it, which is the default after a text event, or `offset`, once that has gone), and a question takes `keys` and `timeout_ms`. A text event takes `case` (`upper` or `lower`, which the experiments apply as they show the text), and a mask takes `length_of` (the fields whose longest text, in characters, sets how often its one-character content is repeated). A region-by-region event takes `sep` (the region separator in its content, `"|"` by default), `advance` (the key that shows the next region, `"space"` by default) and `critical_region_trigger` (a code as for `trigger`, sent at the region the item's `critical_region` names). A feedback event takes `answer` (the field holding the correct key, `"answer"` by default) and its texts `correct`, `incorrect` and `no_response`. Any event may take `blocks`, restricting it to named blocks. Every event's `trigger` is written on that event's onset flip, and `onset_locked`, which the built-in paradigms set, is accepted but does not change this.

Usage

```
PARADIGMS
```

Format

A named list with one entry per paradigm (`factorial`, `lexical_decision`, `priming`, `masked_priming`, `categorisation` and `self_paced_reading`), each holding `stimulus_fields`, a `counterbalance` recipe and an `events` list.

Details

`priming` is backward-masked: a 50 ms prime, a 33 ms ##### mask and the target, with no forward mask and the item table's letter case. `masked_priming` has the arrangement of Forster and Davis (1984): a forward mask of hash marks, a brief lower-case prime and an upper-case target. Its defaults show the mask for 500 ms, the prime for three frames and the target until the response or for 2,000 ms.

Value

A plain named list of the built-in paradigm specifications, with no class of its own. In each element, `stimulus_fields` is a character vector naming the required item-table fields, `counterbalance` is a character scalar naming the counterbalancing recipe and `events` is an ordered list of event specifications. A design that names a paradigm inherits its trial sequence, required item fields and counterbalancing rule from that entry.

References

Forster, K. I., & Davis, C. (1984). Repetition priming and frequency attenuation in lexical access. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 10*(4), 680-698. doi:10.1037/02787393.10.4.680

`participant_table` *Build a participant counterbalancing table*

Description

Crosses the factors with `expand.grid()`, whose first factor varies fastest, and gives participant p row ((p - 1) mod n) + 1 of the n crossed cells, cycling through the cells as often as `n_participants` needs. `participant_table(list(version = 1:V, order = 1:m), N)` therefore reproduces the version and order a generated experiment assigns participant p when none is chosen at launch.

Usage

```
participant_table(factors, n_participants)
```

Arguments

`factors`	A named list of factors, each a vector of levels.
`n_participants`	Number of participants to allocate.

Value

A data frame with one row per participant: one column per factor, then `participant`.

Examples

```
participant_table(list(list = 1:2, order = c("a", "b")), 6)
```

`required_fields` *The item fields a design needs, from its paradigm and its events*

Description

The item fields a design needs, from its paradigm and its events

Usage

```
required_fields(design)
```

Arguments

`design`	A parsed design list.

Value

Character vector of the item fields the design's trials reference.

Examples

```
required_fields(list(paradigm = "categorisation"))
```

`resample_stimuli` *Produce several disjoint matched item sets (items as a random factor)*

Description

Each replicate is an independent, fully matched set drawn from the pool with the items of earlier replicates removed, so no item is reused. This lets a study treat its items as a random factor (such as running different item samples across participant groups or showing that an effect holds across samples) instead of treating them as a fixed set (Clark, 1973; Yarkoni, 2022). The selection is deterministic.

Usage

```
resample_stimuli(pool, design, schema, n_sets, verbose = FALSE)
```

Arguments

`pool`	A candidate pool with the `match_on` dimensions present.
`design`	A parsed design configuration.
`schema`	The parsed global schema.
`n_sets`	Number of disjoint matched sets to draw, a whole number of at least 1.
`verbose`	Passed to `match_stimuli()`.

Value

A data frame of matched stimuli with an added `replicate` column. The first replicate draws on the whole pool, so its count of the candidates excluded for a missing matched value is the one recorded in the `"audit"` attribute's `incomplete_dropped` entry, as `match_stimuli()` records it. A relaxed tolerance window inside a replicate is not recorded: it reaches the console under `verbose` but not the run log or the datasheet.

References

Clark, H. H. (1973). The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. *Journal of Verbal Learning and Verbal Behavior, 12*(4), 335-359. doi:10.1016/S00225371(73)800143

Yarkoni, T. (2022). The generalizability crisis. *Behavioral and Brain Sciences, 45*, Article e1. doi:10.1017/S0140525X20001685

resolve_events *The trial events of a design: its own* events*, or else its paradigm's*

Description

The trial events of a design: its own events, or else its paradigm's

Usage

```
resolve_events(design)
```

Arguments

design	A parsed design list.

Value

The list of trial events the design presents.

Examples

```
vapply(resolve_events(list(paradigm = "lexical_decision")),
       function(e) e$type, character(1))
```

resolve_trial_timing *Realise per-trial event durations onto the stimuli table*

Description

An event may declare a duration that varies from trial to trial, either read from an item column or drawn from a range. A drawn value is determined by the keyed hash alone, so a run with the same seed and design gives the same milliseconds. It is written into the stimuli table as well as the generated script, because timing that varies is a variable the analysis needs.

Usage

```
resolve_trial_timing(stimuli, design, schema)
```

Arguments

stimuli	A counterbalanced stimuli data frame.
design	A parsed design configuration.
schema	The parsed global schema, which provides the seed.

Details

An event declares it as `duration: {from_column: <column>}`, to read each trial's milliseconds from an item column, or as `duration: {jitter: [lo, hi], as: <column>}`, to draw a whole number of milliseconds from `lo` to `hi`, both included, into the column `as` names (`event<i>_ms` by default, `i` counting the design's events from 1). The draw is keyed on the seed, the column, the list, the set and the condition.

Every value read from a column must be a finite number of milliseconds, not negative. The first that is not, a blank cell included, is refused, naming the column and the trial's set and condition. When a column holds fractional milliseconds, each of its values is rounded once, by lexsync's own rounding function, and written back, so the stimuli, the loop tables and every target carry the same whole milliseconds. A column of whole numbers is left as it is.

Value

`stimuli` with one integer column per jittered event, and every column of fractional milliseconds rounded. Each column rounded is recorded in the `"duration_rounding"` attribute, as one `list(event, column, n_rounded)` entry, which `run_pipeline()` writes into the run log.

---

run_all *Run the lexsync pipeline for every design configuration*

Description

Run the lexsync pipeline for every design configuration

Usage

```
run_all(
  config_dir = "config",
  schema_path = NULL,
  outdir = NULL,
  verbose = TRUE
)
```

Arguments

`config_dir`	Directory of `design_*.yaml` configurations.
`schema_path`	Path to the global schema, or `NULL` (the default) to resolve it: `schema.yaml` in `config_dir`, then `config/schema.yaml` in the working directory, then the copy bundled with the package, `system.file("extdata", "schema.yaml", package = "lexsync")`. A message names the file chosen. A named schema that does not exist is an error, raised before any design runs.
`outdir`	Output directory. It must be supplied, as lexsync does not choose a default output location.
`verbose`	Whether to print progress.

Value

A named list of per-design results, invisibly.

`run_pipeline` *Run the lexsync pipeline for one design*

Description

A design may refer to a derived dimension (`n_density`, `old20` or `bigram_freq`) in `match_on`, `pool_filters`, `counterbalance.balance_on`, a condition's `define_by` or a continuous predictor or control. When the lexicon lacks it, the pipeline computes it on the pool against the neighbourhood reference, after the filters the lexicon can answer and before those on the derived dimension.

Usage

```
run_pipeline(
  design_path,
  schema_path = NULL,
  outdir = NULL,
  reference_words = NULL,
  verbose = TRUE
)
```

Arguments

`design_path`	Path to a design configuration (YAML).
`schema_path`	Path to the global schema (YAML), or `NULL` (the default) to resolve it: `config/schema.yaml` in the working directory, then `schema.yaml` beside the design, then the copy bundled with the package, `system.file("extdata", "schema.yaml", package = "lexsync")`. A message names the file chosen. A named schema that does not exist is an error. The path used is recorded in the run log and in the datasheet's `reproducibility$schema_path`.
`outdir`	Output directory, in which the subdirectories `stimuli`, `reports` and `experiments` are created. It must be supplied, as lexsync does not choose a default output location.
`reference_words`	Optional character vector of words to compute the derived dimensions against. By default, they are computed against the lexicon, or against its rows marked 1 in a `reference_member` column. For a corpus or pool design, a supplied list recomputes `n_density` and `old20`, and `bigram_freq` when the lexicon carries it or the design refers to it, overriding the lexicon's columns. The datasheet then records the list's size and checksum and the columns it was applied to. A generate or table design does not use it, and the run log says so. A list that is empty or holds a missing value is refused. Its words are normalised to Unicode NFC before use, like every other text key the pipeline reads.
`verbose`	Whether to print progress.

Details

A design that refers to `n_syllables` in any of those places is refused when a word that passes the other filters the lexicon can answer has no syllable estimate, which is the case for every word outside the Latin script (see `count_syllables()`).

A corpus, pool or generate design may list words to leave out under `exclude_words`, or name a file of them with `exclude_words: {path: ...}`. They are removed from the candidates after any `norms:` join and before the pool filters, and the datasheet records the list with its checksum. The neighbourhood reference, and the words a generated pseudoword must not equal, keep them.

The run log notes a design in a language other than English that gives no `instructions`, since lexsync's own are in English, and a design whose response keys lexsync's instructions only name, such as a factorial or priming design. It also notes every practice or filler answer that the response mapping contradicts. A `response_mapping` that would leave a trial without an answer is refused before anything is written.

Value

Invisibly, a named list. `stimuli` is the path of `stimuli/<stem>_stimuli_R.csv` under `outdir`. `descriptives` and `comparisons` are the paths of `reports/<stem>_descriptives_R.csv` and `reports/<stem>_comparisons_R.csv`, both `NULL` for an item table without a continuous selection, which has no match report. `experiments` is the list `export_experiments()` returns, and `log` is the path of `reports/<stem>_run_log_R.md`. The run also writes the JSON Lines log `reports/<stem>_run_log_R.jsonl` and the datasheet `reports/<stem>_datasheet_R.json` and `reports/<stem>_datasheet_R.md`, which the list does not name. `<stem>` is `design_stem()` of the design.

---

select_continuous_stimuli

Select a set spread over a continuous predictor, with windows on the controls

---

Description

Instead of dichotomising the predictor into conditions and matching, items are spread evenly over the predictor's ranks in the pool, so the selection follows the pool's distribution of the predictor and does not cover its range evenly. The control dimensions are held within a tolerance band, which trims their tails without holding them constant, so read each control's correlation with the predictor in `match_report_continuous()` before relying on it. Tightening `tolerance_k` does not reliably lower those correlations. The set is meant for analysis by regression or mixed models, not by contrasts between conditions (Kuperman, 2015; Liben-Nowell et al., 2019). The selection is made by two deterministic even-spread passes.

Usage

```
select_continuous_stimuli(
  pool,
  design,
```

```
  schema,
  verbose = FALSE,
  key = "word",
  label = "continuous",
  renumber_sets = TRUE
)
```

Arguments

pool	A candidate pool with the predictor and control dimensions present.
design	A parsed design configuration carrying a continuous block (predictor and controls). The number of items is n_per_condition, else n_per_cell, and either must be a whole number of at least 1. It is 60 when neither key is given. tolerance_k, shortfall and on_insufficient_tolerance are read from its matching block with the schema as fallback, as for match_stimuli(), and a relaxed window widens to the whole complete pool.
schema	The parsed global schema, which gives the tolerance windows.
verbose	Whether to report a window relaxation.
key	Column used as the selection unit and the byte-order tie-break, by default "word". Pair designs pass "set". Once a pair table is collapsed to one row per item set, there is no word column, and set is an integer that is unique per row and already derived deterministically.
label	Value written into the result's condition column, or NULL to leave the existing conditions alone. Pair designs pass NULL, because their rows already carry the design's own conditions, and overwriting them would remove the contrast the design tests.
renumber_sets	Whether to renumber the selected rows 1..n. Pair designs pass FALSE, because their set ids must be kept for the result to be expanded back to the full pair table.

Details

The design is checked before anything is selected, so a design that cannot be honoured is refused outright. continuous.controls must be non-empty and must not name the predictor, match_on must name exactly the same dimensions as continuous.controls, every dimension named and the key column must be present in the pool, no tolerance_k may be negative, and the pool must not be empty.

A row missing the predictor or a control is excluded before the first pass, and the number excluded is recorded in the "audit" attribute's incomplete_dropped entry under the condition continuous. A pool with no complete row is refused.

The "audit" attribute's control_windows entry records, for each control, the window the first pass set (low, high) and the SD of the complete pool (pool_sd), which match_report_continuous() sets the realised SD against. A relaxed window is recorded as computed, and the relaxation in window_relaxations.

Value

A data frame of the selected stimuli. Unless `label` is `NULL`, the `condition` column is set to it (`"continuous"` by default), and unless `renumber_sets` is `FALSE`, the `set` column is renumbered `1..n`.

References

Kuperman, V. (2015). Virtual experiments in megastudies: A case study of language and emotion. *Quarterly Journal of Experimental Psychology, 68*(8), 1693-1710. doi:10.1080/17470218.2014.989865

Liben-Nowell, D., Strand, J., Sharp, A., Wexler, T., & Woods, K. (2019). The danger of testing by selecting controlled subsets, with applications to spoken-word recognition. *Journal of Cognition, 2*(1), Article 2. doi:10.5334/joc.51

`tost_equiv` *Two one-sided tests (TOST) of equivalence on a Cohen's d bound*

Description

Reports the larger of the two one-sided p-values. A value below `alpha` supports equivalence within +/- `bound_d` standard deviations. A non-significant difference test is not itself evidence of equivalence, so TOST is reported alongside the standardised mean difference (Lakens, 2017).

Usage

```
tost_equiv(x, y, bound_d = 0.5, alpha = 0.05)
```

Arguments

`x, y`	Numeric vectors.
`bound_d`	Smallest effect size of interest (Cohen's d). It defaults to 0.5, the schema's value. That value is a convention and not a default of Lakens (2017), who asks for the bound to be justified for the study. At 0.5 and alpha .05, no dimension that varies can pass with fewer than 23 items per condition, even when the conditions are identical.
`alpha`	Significance level.

Value

A list with `p` and logical `equivalent`.

References

Lakens, D. (2017). Equivalence tests: A practical primer for *t* tests, correlations, and meta-analyses. *Social Psychological and Personality Science, 8*(4), 355-362. doi:10.1177/1948550617697177

validate_design *Check a design and its schema before anything is selected or written*

Description

run_pipeline() calls this right after reading the two files, so a design that cannot be honoured is refused before any output exists. It first refuses a key within one or two edits of a key lexsync reads at that level, naming the key it was probably meant to be (one edit for a known key of up to five characters, two for a longer one). It then refuses a missing, empty or non-text name or language, an n_per_condition, n_per_cell or resample.n_sets that is not a whole number of at least 1, a counterbalance.optimise that is not true or false, a counterbalance.scale other than mean or mad, a counterbalance.max_run or counterbalance.orders that is not a whole number of at least 1 and a counterbalance.run_on that is not a single non-empty text value. It does not see the trials, so a run_on that names no column of them is refused when counterbalance.max_run is applied to the trial order, by counterbalance() or, in a design with fillers, when run_pipeline() interleaves them. It refuses a name, language or font that cannot be written into the generated experiments, a trigger port that is not an address and an event list the renderer refuses, such as one with an unknown timing key, a bad response key or a response key that YAML reads as a boolean (an unquoted yes, no, on or off). It refuses a presentation.fullscreen that is not true or false and a presentation.text_height_px that is not a whole number from 8 to 400. It refuses a triggers.mode other than auto, hardware or mock, a triggers.backend other than parallel or serial and a triggers.baudrate that is not a positive whole number. A serial backend needs a triggers.serial_port, with a name the experiments can carry. Last, it refuses a condition without a name, with a name that is not text or with a repeated name, a corpus or pool design with conditions but no match_on and a name and language that leave no file stem. It also refuses an exclude_words that is neither a list of words nor a mapping whose path names a file, and one that a design with an item table names.

Usage

```
validate_design(design, schema)
```

Arguments

design	A parsed design configuration.
schema	The parsed global schema.

Details

Any other key lexsync does not read is not refused yet. It is returned as a note, which run_pipeline() writes into the run log, and a later release will refuse it. description and keys beginning x_ are annotations and are never noted. A schema that sets triggers.inter_trigger_ms, which lexsync ignores, gets a note too.

The experiments' instructions and response mapping are checked here too. It refuses instructions that are not text, are empty or hold a control character other than a line break. It refuses a response_mapping that is not a mapping of conditions to single keys, names a key the response event does not allow, or

comes with no response event. It also refuses a `rotate_mapping_by_list` that is not true or false or that has no mapping to rotate.

Value

A character vector of notes on unknown keys, in the order the keys were written, then the notes on `triggers.inter_trigger_ms` and on an event's `onset_locked: false`. It is empty when there are none.

Examples

```
validate_design(list(name = "demo", language = "english", x_note = "pilot"), list())
```

---

variance_ratio *Variance ratio: a distributional balance check*

Description

The ratio of a condition's variance to the reference's, complementing the mean-based Cohen's d and TOST. Two conditions can share a mean yet differ in spread and still confound, which a mean-based statistic misses (Armstrong et al., 2012; Austin, 2009). A ratio near 1 is balanced. A common heuristic flags ratios outside 0.5 to 2 (Rubin, 2001, as cited in Stuart, 2010).

Usage

```
variance_ratio(cond, ref)
```

Arguments

`cond`, `ref`	Numeric vectors (condition and reference).

Value

The variance ratio. Missing values are dropped first. The ratio is `NA` when either vector has fewer than two non-missing values, or when the reference's variance is zero and the condition's is not. It is 1 when both variances are zero.

References

Armstrong, B. C., Watson, C. E., & Plaut, D. C. (2012). SOS! An algorithm and software for the stochastic optimization of stimuli. *Behavior Research Methods, 44*(3), 675-705. doi:10.3758/s1342801101829

Austin, P. C. (2009). Balance diagnostics for comparing the distribution of baseline covariates between treatment groups in propensity-score matched samples. *Statistics in Medicine, 28*(25), 3083-3107. doi:10.1002/sim.3697

Stuart, E. A. (2010). Matching methods for causal inference: A review and a look forward. *Statistical Science, 25*(1), 1-21. doi:10.1214/09STS313

Examples

```
variance_ratio(c(1, 2, 3, 4), c(1, 2, 3, 8))
```

`write_datasheet` *Write a datasheet to a JSON record and a Markdown rendering*

Description

Write a datasheet to a JSON record and a Markdown rendering

Usage

```
write_datasheet(ds, json_path, md_path)
```

Arguments

`ds`	A datasheet list, from `build_datasheet()`.
`json_path`	Output path for the machine-readable JSON record.
`md_path`	Output path for the human-readable Markdown rendering.

Value

Invisibly, the two paths written.

`write_run_log` *Write the run log to Markdown (and optionally JSON Lines)*

Description

Write the run log to Markdown (and optionally JSON Lines)

Usage

```
write_run_log(log, md_path, jsonl_path = NULL)
```

Arguments

`log`	A run-log object.
`md_path`	Output path for the Markdown log.
`jsonl_path`	Optional output path for the JSON Lines log: one JSON object per step, with its `time`, `message` and `data` (`null` for a step without details). The run's name, engine, times and metadata are in the Markdown log only. Unlike `md_path`'s, its directory is not created.

Value

`md_path`, invisibly.

Index

add_bigram_frequency, 3
add_neighbourhood, 4
add_neighbourhood(), *27*, *34*
add_pair_overlap, 4
assign_triggers, 5
assign_triggers(), *17–20*, *22*

balance_check, 6
balance_lists, 7
balance_lists(), *9*, *15*
build_datasheet, 8
build_datasheet(), *23*, *36*, *50*
build_lexdec_stimuli, 11
build_pool, 12
build_pool(), *11*

cohens_d, 13
cohens_d(), *14*
cohens_d_ci, 13
count_syllables, 15
count_syllables(), *45*
counterbalance, 14
counterbalance(), *10*, *48*

describe_stimuli, 16
design_stem, 17
design_stem(), *18–20*, *22*, *45*

export_experiments, 17
export_experiments(), *6*, *9*, *21*, *45*
export_jspsych, 18
export_opensesame, 20
export_opensesame(), *18*
export_psychopy, 21

fetch_corpus, 22
fetch_corpus(), *24, 25*, *38*

generate_pseudowords, 24
generate_pseudowords(), *12*, *31*

lexsync_cache_clear, 24
lexsync_cache_clear(), *23*, *26*
lexsync_cache_dir, 25
lexsync_cache_dir(), *23–25*
list_corpora, 26
list_corpora(), *23*, *37*
load_items, 27
load_lexicon, 27
load_lexicon(), *5*, *22, 23*, *29*, *37*
load_pool, 28
log_artefact, 29
log_step, 30
log_step(), *30*

make_pseudoword, 31
match_report, 32
match_report(), *9*, *32*
match_report_continuous, 32
match_report_continuous(), *9*, *45, 46*
match_stimuli, 33
match_stimuli(), *12*, *41*, *46*
merge_norms, 35
methods_paragraph, 36

new_run_log, 36
normalise_lexicon, 37
normalise_lexicon(), *10*, *23*, *28*

PARADIGMS, 38
participant_table, 39

render_events(), *6*
required_fields, 40
resample_stimuli, 41
resolve_events, 42
resolve_events(), *5*
resolve_trial_timing, 42
resolve_trial_timing(), *38*
run_all, 43
run_pipeline, 44

run_pipeline(), *12*, *30*, *32*, *43*, *48*

select_continuous_stimuli, 45
select_continuous_stimuli(), *33*

tost_equiv, 47

utils::download.file(), *23*

validate_design, 48
variance_ratio, 49

write_datasheet, 50
write_datasheet(), *11*
write_run_log, 50