Package 'rcicr'

November 11, 2026

Type Package

Title Reverse-Correlation Image-Classification Toolbox

Version 1.6.1

URL https://rdotsch.github.io/rcicr/, https://github.com/rdotsch/rcicr

BugReports https://github.com/rdotsch/rcicr/issues

Description Generate stimuli and analyze data of reverse correlation image classification experiments (psychophysical tasks aimed at visualizing cognitive mental representations of faces). For the method see Dotsch and Todorov (2012) <doi:10.1177/1948550611430272>; for a practical primer see Brinkman, Todorov and Dotsch (2017) <doi:10.1080/10463283.2017.1381469>.

License GPL-2

Encoding UTF-8

Depends R (>= 4.1)

Imports png, jpeg, viridis, utils, parallel, doSNOW, foreach, spatstat.explore, spatstat.geom, Matrix

Suggests testthat (>= 3.0.0), tibble, covr, withr, knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation no

Author Ron Dotsch [aut, cre]

Maintainer Ron Dotsch <rdotsch@gmail.com>

Repository CRAN

Date/Publication 2026-11-11 07:30:10 UTC

Contents

autoscale 2
batchComputeInfoVal2IFC 3

batchGenerateCI 5
batchGenerateCI2IFC 7
checkStimulusPNGs2IFC 10
computeCumulativeCICorrelation 11
computeInfoVal2IFC 13
deg2rad 16
generateCI 17
generateCI2IFC 20
generateCINoise 23
generateGabor 24
generateNoiseImage 24
generateNoisePattern 25
generateReferenceDistribution2IFC 26
generateSinusoid 29
generateStimuli2IFC 30
plotZmap 32
simulateNoiseIntensities 34

Index **36**

`autoscale` *Determines optimal scaling constant for a list of CIs*

Description

Determines optimal scaling constant for a list of CIs

Usage

```
autoscale(cis, save_as_pngs = TRUE, targetpath)
```

Arguments

`cis`	List of classification images, each a list of pixel matrices holding at least the noise (`$ci`), plus the base image (`$base`) if PNGs are to be written.
`save_as_pngs`	Boolean: combine each autoscaled noise pattern with its base image and save it as a PNG, named after its key in `cis`. Every element then needs a name of its own; with `FALSE`, names are not needed.
`targetpath`	Directory to save PNGs to. Required when `save_as_pngs = TRUE`; there is no default. The directory is created if it does not exist; to just try the function out, use `tempdir()`.

Value

The input `cis` list, with each element's `$scaled` matrix replaced by its autoscaled version. The scaling constant is printed to the console, and recorded in each element's `scaling` attribute, which keeps the earlier record for `$combined`; `vignette("stored-data", package = "rcicr")` lists its fields.

Look at `$scaled`, not `$combined`. `$combined` is returned exactly as it was passed in, on purpose, so that existing scripts that plot it keep producing the same image. Only `$scaled` holds the autoscaled result. To show the autoscaled noise over the base image, compute `(ci$scaled + ci$base) / 2`: that is exactly what `save_as_pngs = TRUE` writes to disk.

This catches people out after `batchGenerateCI` or `batchGenerateCI2IFC`. Both scale with `'none'` before calling this function, so their `$combined` overlays the *unscaled* noise and looks almost blank, while `$scaled` is the image you want.

See Also

`vignette("reverse-correlation-walkthrough", package = "rcicr")`, sections "Scaling" and "Several participants at once".

Examples

```
cis <- list(
  participant1 = list(ci = matrix(runif(64, -0.2, 0.2), 8, 8), base = matrix(0.5, 8, 8)),
  participant2 = list(ci = matrix(runif(64, -0.3, 0.3), 8, 8), base = matrix(0.5, 8, 8))
)
scaled_cis <- autoscale(cis, save_as_pngs = FALSE)
```

---

batchComputeInfoVal2IFC

Computes Informational Values for several classification images

---

Description

Computes the Informational Value of every classification image in a list, from one 2IFC stimulus file, as `computeInfoVal2IFC` would for each in turn.

Usage

```
batchComputeInfoVal2IFC(
  target_cis,
  rdata,
  iter = 10000,
  force_gen_ref_dist = FALSE,
  response_seed = NULL,
  baseimage = NULL,
  reference_stimuli = NULL,
  reference_method = c("gram", "images")
)
```

Arguments

target_cis	A list of classification images, each as returned by generateCI; for example the result of batchGenerateCI2IFC.
rdata	Path to the .Rdata file written when the stimuli were generated. Every classification image in target_cis must come from it.
iter, force_gen_ref_dist, response_seed, reference_method	As in computeInfoVal2IFC, applied to every classification image.
baseimage	As in computeInfoVal2IFC: the one base-image label every classification image in target_cis was computed for. Score CIs for different base images in separate calls.
reference_stimuli	The stimulus numbers each classification image was built from, when that is not every saved stimulus. NULL (the default) scores every image against the full set; a vector scores every image against that subset; a list with one element per image (NULL for the full set) gives each its own. To use what generateCI recorded, pass lapply(target_cis, function(ci) attr(ci, "trial_design")$stimuli).

Details

Each distinct reference distribution is found or simulated once and used for every classification image scored against it. The stimulus file is read at most three times to find stored references, however many images there are, and again for each reference that has to be simulated. A loop over computeInfoVal2IFC() reads the file twice per image and, when the reference cannot be stored (a read-only file, a response_seed, or force_gen_ref_dist = TRUE), simulates it again for every image. The values are identical to that loop's.

A masked classification image is scored over its unmasked pixels, as described under "Masked classification images" in computeInfoVal2IFC. Images with the same stimuli and the same mask share one reference.

Messages about the trial design (see "Matching the reference to the classification image" in computeInfoVal2IFC) are collected into one message per kind, naming the classification images concerned, rather than one per image.

Value

A numeric vector of Informational Values, one per classification image, named as target_cis is.

See Also

vignette("recipes", package = "rcicr"), "Several classification images at once".

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
```

```
  base_face_files = list(face = base_face),
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

# two "participants", three trials each
data <- data.frame(
  participant = rep(c("p1", "p2"), each = 3),
  stimulus = 1:6,
  response = sample(c(1, -1), 6, replace = TRUE)
)
cis <- suppressWarnings(batchGenerateCI2IFC(
  data = data, by = "participant", stimuli = "stimulus", responses = "response",
  baseimage = "face", rdata = rdata_file, save_as_png = FALSE, scaling = "none"
))

# iter is kept tiny here for a fast example, in practice use iter >= 10000.
# Each participant saw three of the six stimuli, so each is scored over those.
suppressWarnings(batchComputeInfoVal2IFC(
  cis, rdata_file, iter = 20,
  reference_stimuli = list(1:3, 4:6)
))
```

---

batchGenerateCI *Generates multiple classification images by participant or condition*

Description

Generate one classification image per participant or condition, for any reverse correlation task.

Usage

```
batchGenerateCI(
  data,
  by,
  stimuli,
  responses,
  baseimage,
  rdata,
  save_as_png = TRUE,
  targetpath,
  label = "",
  antiCI = FALSE,
```

```
  scaling = "autoscale",
  constant = 0.1,
  participants = NULL
)
```

Arguments

data	Data frame with one row per trial.
by	Name of the column that splits the data into units, such as participants or conditions. One CI is computed per unit.
stimuli	Name of the column holding the stimulus numbers of the presented stimuli.
responses	Name of the column holding the responses: 1 where the original stimulus was chosen, -1 where the inverted one was. The column must be numeric with no missing or infinite values; the call stops before computing anything otherwise.
baseimage	String naming the base image: not its file name, but its key in the base_face_files list passed to generateStimuli2IFC.
rdata	Path to the .Rdata file written when the stimuli were generated. It holds the contrast parameters of every stimulus.
save_as_png	Boolean: also save the CI as a PNG image.
targetpath	Directory to save PNGs to. Required when save_as_png = TRUE; there is no default. The directory is created if it does not exist; to just try the function out, use tempdir().
label	Optional string added to the PNG file names to make them easier to identify.
antiCI	Boolean: compute the anti-CI, the classification image with its sign flipped, instead of the CI.
scaling	Scaling method: none, constant, matched, independent or autoscale (default). autoscale computes the CIs unscaled and then puts them on one scale with autoscale.
constant	Scaling constant for the noise. Used only when scaling = 'constant'.
participants	Optional name of a column holding a participant ID per trial. When given, each unit's CI is computed as generateCI does with participants: one CI per participant in the unit, then their average. With by naming a condition, that gives one CI per condition with participants nested in it, and each participant counts equally however many trials they contributed. Every row needs an ID; the call stops before computing anything if any is missing. Default: NULL, all of a unit's trials pooled into one CI.

Details

Splits data by the by column, calls generateCI for each part, and returns the CIs. By default each CI is also saved as a PNG.

Value

Named list with one classification image per unit. Each is itself a list of pixel matrices: the raw noise (ci), the scaled noise (scaled), the base image (base) and the two combined (combined).

See Also

vignette("reverse-correlation-walkthrough", package = "rcicr"), section "Several participants at once".

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

# two "participants", three trials each
data <- data.frame(
  participant = rep(c("p1", "p2"), each = 3),
  stimulus = 1:6,
  response = sample(c(1, -1), 6, replace = TRUE)
)

cis <- suppressWarnings(batchGenerateCI(
  data = data, by = "participant", stimuli = "stimulus", responses = "response",
  baseimage = "face", rdata = rdata_file, save_as_png = FALSE
))
```

---

batchGenerateCI2IFC *Generates multiple 2IFC classification images by participant or condition*

Description

Generate one classification image per participant or condition, for a two-image forced-choice reverse correlation task.

Usage

```
batchGenerateCI2IFC(
  data,
  by,
```

```
  stimuli,
  responses,
  baseimage,
  rdata,
  save_as_png = TRUE,
  targetpath,
  antiCI = FALSE,
  scaling = "autoscale",
  constant = 0.1,
  label = "",
  participants = NULL
)
```

Arguments

`data`	Data frame with one row per trial.
`by`	Name of the column that splits the data into units, such as participants or conditions. One CI is computed per unit.
`stimuli`	Name of the column holding the stimulus numbers of the presented stimuli.
`responses`	Name of the column holding the responses: 1 where the original stimulus was chosen, -1 where the inverted one was. The column must be numeric with no missing or infinite values; the call stops before computing anything otherwise.
`baseimage`	String naming the base image: not its file name, but its key in the `base_face_files` list passed to `generateStimuli2IFC`.
`rdata`	Path to the `.Rdata` file written when the stimuli were generated. It holds the contrast parameters of every stimulus.
`save_as_png`	Boolean: also save the CI as a PNG image.
`targetpath`	Directory to save PNGs to. Required when `save_as_png = TRUE`; there is no default. The directory is created if it does not exist; to just try the function out, use `tempdir()`.
`antiCI`	Boolean: compute the anti-CI, the classification image with its sign flipped, instead of the CI.
`scaling`	Scaling method: `none`, `constant`, `matched`, `independent` or `autoscale` (default). `autoscale` computes the CIs unscaled and then puts them on one scale with `autoscale`.
`constant`	Scaling constant for the noise. Used only when `scaling = 'constant'`.
`label`	Optional string added to the PNG file names to make them easier to identify.
`participants`	Optional name of a column holding a participant ID per trial. When given, each unit's CI is computed as `generateCI` does with `participants`: one CI per participant in the unit, then their average. With `by` naming a condition, that gives one CI per condition with participants nested in it, and each participant counts equally however many trials they contributed. Every row needs an ID; the call stops before computing anything if any is missing. Default: `NULL`, all of a unit's trials pooled into one CI.

Details

Splits `data` by the `by` column, calls `generateCI2IFC` for each part, and returns the CIs. By default each CI is also saved as a PNG.

Value

Named list with one classification image per unit. Each is itself a list of pixel matrices: the raw noise (`ci`), the scaled noise (`scaled`), the base image (`base`) and the two combined (`combined`).

See Also

`vignette("reverse-correlation-walkthrough", package = "rcicr")`, section "Several participants at once"; `vignette("recipes", package = "rcicr")`, "Several classification images at once".

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

# two "participants", three trials each
data <- data.frame(
  participant = rep(c("p1", "p2"), each = 3),
  stimulus = 1:6,
  response = sample(c(1, -1), 6, replace = TRUE)
)

cis <- suppressWarnings(batchGenerateCI2IFC(
  data = data, by = "participant", stimuli = "stimulus", responses = "response",
  baseimage = "face", rdata = rdata_file, save_as_png = FALSE
))
```

`checkStimulusPNGs2IFC` *Check a stimulus file against stimulus PNGs*

Description

Compares the noise a stimulus `.Rdata` file records with the stimulus PNGs in a folder, trial by trial. An original and an inverted stimulus hold the same base image, so their difference is the trial's noise alone: `ori - inv` equals the noise divided by 0.6 for ordinary interior pixels. Older PNGs can wrap a value above white into the interior; their difference is compared with the corresponding 8-bit byte-wrap offset too. The base images are therefore not needed. A wrapped pixel encoded as an endpoint is excluded.

Usage

```
checkStimulusPNGs2IFC(rdata, png_dir, label = NULL, seed = NULL)
```

Arguments

`rdata`	Path to the `.Rdata` file to check, as written by `generateStimuli2IFC`.
`png_dir`	Directory holding the stimulus PNGs. Required; there is no default.
`label`, `seed`	The `label` and `seed` the PNGs were generated with, which their names carry. When omitted: the values stored in `rdata`. Pass the archive's own when `rdata` is a candidate regenerated with other settings, so the PNGs can be found; `seed = NULL` for a set generated with `seed = NULL`.

Details

Use it to confirm that a regenerated `.Rdata` file matches an archive of stimulus PNGs whose original file was lost; `vignette("recipes", package = "rcicr")`, "When the `.Rdata` file is lost", shows how.

Value

A data frame with one row per base label and trial: `base`, `trial`, `share` (the share of compared pixels whose `ori - inv` agrees with the file's noise divided by 0.6, directly or after a feasible historical 8-bit byte wrap, to within 1/255), `compared` (the number of pixels with both decoded PNG values strictly between 0 and 1) and `missing` (`TRUE` when either PNG is absent, with `share` `NA`). Its `unchecked` attribute lists PNGs in `png_dir` that carry `label` and `seed` but belong to no row, such as trials beyond the file's `n_trials`.

No verdict is returned. In the configurations measured in https://github.com/rdotsch/rcicr/blob/main/analyses/stimulus-png-residuals.md, a wrong `nscales`, seed, noise type, base order or `use_same_parameters` agreed on 2.1% to 7.4% of pixels per trial, but a Gabor `sigma` of 24 where the PNGs used 25 on 95% to 99.6%: compare candidates against the same PNGs. Check every base label, since with several bases a wrong `use_same_parameters` shows only after the first.

Warns about missing and unchecked PNGs, and stops if no PNG named for `label` and `seed` exists.

See Also

`vignette("recipes", package = "rcicr")`, "When the `.Rdata` file is lost".

Examples

```
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)
stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(list(face = base_face), n_trials = 3, img_size = 32,
                    stimulus_path = stimulus_path, seed = 1, ncores = 1, nscales = 2)
rdata <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)
checkStimulusPNGs2IFC(rdata, stimulus_path)
```

---

computeCumulativeCICorrelation

Computes cumulative trial CIs correlations with final/target CI

Description

Correlates the CI built from the first trials with the final or target CI, adding trials one step at a time.

Usage

```
computeCumulativeCICorrelation(
  stimuli,
  responses,
  baseimage,
  rdata,
  targetci = list(),
  step = 1
)
```

Arguments

`stimuli`	Numeric vector of stimulus numbers, one per response and in the same order. Each must be a positive whole number no larger than the number of trials saved for the selected base image. Numbers may repeat and need not be consecutive. Factors, characters and logicals are rejected: if your data hold stimulus labels, check them against the generated stimulus filenames before converting them to numbers.
`responses`	Vector of responses in the same order as `stimuli`: 1 where the original stimulus was chosen, -1 where the inverted one was. Must be numeric and finite; other finite numbers, such as ratings, weight the trials accordingly. Factors, characters and logicals are rejected, as are `NA`, `NaN` and infinite values: remove trials without a response from every argument alike.
`baseimage`	String naming the base image: not its file name, but its key in the `base_face_files` list passed to `generateStimuli2IFC`.

rdata	Path to the .Rdata file written when the stimuli were generated. It holds the contrast parameters of every stimulus.
targetci	Optional target CI to correlate the cumulative CIs with, as returned by generateCI. Without it, the final CI of these trials is used.
step	Number of trials added between successive correlations.

Details

Plot the resulting curve to estimate how many trials your task needs.

Value

Vector of correlations between each cumulative CI and the final or target CI.

Repeated presentations of the same stimulus

This function takes the trials in the order they were presented and does not average repeated presentations of a stimulus. generateCI does average them, per unique stimulus, before building its classification image. Averaging here would discard the presentation order that a cumulative curve is about.

Without a targetci, the final CI is built here from the same trials as the curve. When the evaluated trials reach the last one, as they always do at the default step = 1, the curve's last point compares that CI with itself and is exactly 1. That shows self-consistency, not convergence. A larger step can stop short, because trials are taken at seq(1, length(responses), step): with six responses and step = 2, the last trial evaluated is the fifth, and the curve ends at whatever that partial CI correlates to (0.97 in one such set, not 1).

This assumes the CI compared against varies at all. Responses that cancel exactly, with every presentation of a stimulus answered both ways, average to a CI that is zero everywhere. A correlation with a constant is undefined, so then **every** point on the curve is NA. Such a curve means the responses carry no net signal, not that the call failed.

A targetci with masked pixels (generateCI stores NA in every pixel a mask excludes) is correlated over the unmasked pixels only. If the mask covers *every* pixel, no pairs remain and the whole curve is NA, as above.

If every stimulus was shown equally often, the final CI computed here is identical to the one generateCI returns. If not, the two weight the data differently (each trial equally here, each unique stimulus equally there) and they diverge: on an 8-trial set with counts 4/2/1/1 they correlate at 0.77.

So to see how the CI approaches the one you will report, pass that CI as targetci = generateCI(...) instead of relying on the default, which is always built without a mask.

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
```

```
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

responses <- sample(c(1, -1), 6, replace = TRUE)
correlations <- suppressWarnings(computeCumulativeCICorrelation(
  stimuli = 1:6, responses = responses, baseimage = "face", rdata = rdata_file
))
```

`computeInfoVal2IFC` *Computes Informational Value*

Description

Computes the Informational Value of a single CI from a 2IFC task.

Usage

```
computeInfoVal2IFC(
  target_ci,
  rdata,
  iter = 10000,
  force_gen_ref_dist = FALSE,
  response_seed = NULL,
  baseimage = NULL,
  reference_stimuli = NULL,
  reference_method = c("gram", "images")
)
```

Arguments

`target_ci`	A classification image, as the list returned by `generateCI`.
`rdata`	Path to the `.Rdata` file written when the stimuli were generated. It holds the contrast parameters of every stimulus and, once computed, the reference distribution (see `generateReferenceDistribution2IFC`).
`iter`	Number of simulated classification images in the reference distribution. Used only when the reference distribution has to be simulated.
`force_gen_ref_dist`	Boolean: simulate the reference distribution again even if the `rdata` file already holds one (default: `FALSE`).

`response_seed` Optional seed for the simulated random responses behind the reference distribution. The default, `NULL`, uses the reference distribution stored in the `rdata` file, or simulates the reproducible default one described under Reproducibility in `generateReferenceDistribution2IFC`, which needs the stimulus seed saved in the file. For a file without one, pass a number. A number forces a fresh reference distribution from an independent draw; use it to check how much Monte Carlo error `iter` leaves in the Informational Value. That result is *not* written back to the `rdata` file, so a one-off check cannot change the number every later analysis of the stimulus set reports.

`baseimage` The base-image label `target_ci` was computed for, the same one passed to `generateCI()`. Required when the base images have different noise parameters; each base then gets its own stored reference distribution, and a shared one left by an older version is ignored. With a single base image, or base images sharing one parameter set, leave it at `NULL`.

`reference_stimuli`
Optional stimulus numbers the classification image was built from, each once, when that is not every saved stimulus. The reference is then built over exactly those stimuli and stored in the `rdata` file apart from the default one, as described in `generateReferenceDistribution2IFC`. The default, `NULL`, uses every saved stimulus, as does passing all of them, with the same result in every case. A subset too small for random responses to give distinct norms (one stimulus, and usually two) is refused, since its MAD is 0 and the InfoVal would not be a number. See "Matching the reference to the classification image" below.

`reference_method`
`"gram"` (the default) or `"images"`: how the reference is computed, when it has to be. `"images"` reproduces rcicr 1.5.0 and earlier bit for bit, with the same BLAS and LAPACK; a reference already stored in `rdata` is reused whichever is given. See "Reference method" in `generateReferenceDistribution2IFC`.

Details

The Informational Value is a z-score for the signal in a classification image: the higher it is, the more signal. A cut-off such as z = 1.96 selects classification images with significant signal at alpha = 0.05.

It is computed against a reference distribution: classification images simulated from random responses under the same task parameters as the real data. The Informational Value expresses how unlikely the observed CI is under the null hypothesis that the responses were random.

Simulating the reference distribution takes a long time. It is simulated whenever the `rdata` file does not already hold one, and then stored there for reuse. If the file cannot be written, the simulated reference is used for this call only: a note names the file (and, for independent base images, the base) that could not be updated, and the next call simulates it again. An archive on read-only media can therefore still be scored, at the cost of simulating each time.

Value

The Informational Value, a z-score.

Matching the reference to the classification image

The reference must be built from the same stimuli as the classification image (Brinkman et al., 2019, Part I). By default it uses every saved stimulus, with one response each. Matching it to your design is your responsibility, because the default cannot know which trials you dropped. A CI built from fewer stimuli (after removing missed trials, say, or from part of the set) has a larger norm under random responding, and the default reference inflates its InfoVal. At 512 pixels, in the stimulus sets measured, pure-noise CIs had a median InfoVal of 0.09 to 0.11 with 1% of trials missing and 0.51 to 0.58 with 5% (https://github.com/rdotsch/rcicr/blob/main/analyses/infoval-design-mismatch.md).

Pass the stimuli the CI was built from as `reference_stimuli`. `generateCI` records them on its result, so `computeInfoVal2IFC(ci, rdata, reference_stimuli = attr(ci, "trial_design")$stimuli)` does it, and a message says when a CI is scored against a reference over different stimuli. The message never changes the number returned. To score many classification images, one per participant say, use `batchComputeInfoVal2IFC`, which takes `reference_stimuli` per image and computes each distinct reference once.

No reference is defined for a CI that averages repeated presentations of a stimulus or several participants: every reference here assumes one response per stimulus from one responder. For participants who each saw every stimulus once, compute the InfoVal of each participant's own CI instead.

Masked classification images

A classification image computed with `mask` (see `generateCI`) holds `NA` in its masked pixels. Its InfoVal is computed over the unmasked pixels only, against a reference built over the same pixels from the same stimuli: Brinkman et al.'s (2019) statistic for the region analysed. The mask is read from the classification image itself. Each mask gets its own reference, simulated once and stored in the `rdata` file apart from the default one; `generateReferenceDistribution2IFC(mask = )` stores one in advance. An InfoVal over part of the image is not comparable with one over the whole image.

For the method, see Brinkman, L., Goffin, S., van de Schoot, R., van Haren, N. E. M., Dotsch, R., & Aarts, H. (2019). Quantifying the informational value of classification images. *Behavior Research Methods*, *51*, 2059-2073. doi:10.3758/s13428019012322

See Also

`vignette("reverse-correlation-walkthrough", package = "rcicr")`, section "Is there actually signal?"; `vignette("recipes", package = "rcicr")`, "Matching the InfoVal reference to the design", for dropped trials, masks, batches and group averages, and "Reproducing a number from an earlier version".

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 6,
```

```
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

# compute (and cache in rdata_file) a reference distribution; iter is kept
# tiny here for a fast example, in practice use iter >= 10000.
suppressWarnings(generateReferenceDistribution2IFC(rdata_file, iter = 3, ncores = 1))

responses <- sample(c(1, -1), 6, replace = TRUE)
target_ci <- generateCI(
  stimuli = 1:6, responses = responses, baseimage = "face",
  rdata = rdata_file, save_as_png = FALSE
)

computeInfoVal2IFC(target_ci = target_ci, rdata = rdata_file)
```

`deg2rad` *Convert angle in degrees to radians*

Description

Convert angle in degrees to radians

Usage

```
deg2rad(deg)
```

Arguments

deg	Angle in degrees

Value

The angle in radians.

Examples

```
deg2rad(180)
```

`generateCI` *Generates classification image*

Description

Generate a classification image for any reverse correlation task.

Usage

```
generateCI(
  stimuli,
  responses,
  baseimage,
  rdata,
  participants = NA,
  save_individual_cis = FALSE,
  save_as_png = TRUE,
  filename = "",
  targetpath,
  antiCI = FALSE,
  scaling = "independent",
  scaling_constant = 0.1,
  individual_scaling = "independent",
  individual_scaling_constant = 0.1,
  zmap = FALSE,
  zmapmethod = "quick",
  zmapdecoration = TRUE,
  sigma = 3,
  threshold = 3,
  zmaptargetpath,
  n_cores = default_ncores(),
  mask = NA,
  zmappointsize = 12
)
```

Arguments

`stimuli`	Numeric vector of stimulus numbers, one per response and in the same order. Each must be a positive whole number no larger than the number of trials saved for the selected base image. Numbers may repeat and need not be consecutive. Factors, characters and logicals are rejected: if your data hold stimulus labels, check them against the generated stimulus filenames before converting them to numbers.
`responses`	Vector of responses in the same order as `stimuli`: 1 where the original stimulus was chosen, -1 where the inverted one was. Must be numeric and finite; other finite numbers, such as ratings, weight the trials accordingly. Factors, characters

and logicals are rejected, as are NA, NaN and infinite values: remove trials without a response from every argument alike.

`baseimage` String naming the base image: not its file name, but its key in the `base_face_files` list passed to `generateStimuli2IFC`.

`rdata` Path to the `.Rdata` file written when the stimuli were generated. It holds the contrast parameters of every stimulus.

`participants` Optional vector with one participant ID per trial, the same length as `stimuli` and `responses`. When given, the CI is computed in two steps: one CI per participant, then their average. When missing or all NA, one CI is computed from all trials together. Some, but not all, NA is an error: give every trial an ID, or remove the trials without one.

`save_individual_cis` Boolean: when `participants` is given, also save each participant's CI as a PNG image.

`save_as_png` Boolean: also save the CI as a PNG image.

`filename` Optional file name for the PNG image.

`targetpath` Directory to save PNGs to. Required when `save_as_png = TRUE` or `save_individual_cis = TRUE`; there is no default. The directory is created if it does not exist; to just try the function out, use `tempdir()`.

`antiCI` Boolean: compute the anti-CI, the classification image with its sign flipped, instead of the CI.

`scaling` Scaling method: `none`, `constant`, `matched` or `independent` (default). When both individual and group CIs are computed, this applies to the group CI.

`scaling_constant` Scaling constant for the noise, used only when `scaling = 'constant'`. When both individual and group CIs are computed, this applies to the group CI.

`individual_scaling` Scaling method for the individual CIs: `none`, `constant` or `independent` (default).

`individual_scaling_constant` Scaling constant for the individual CIs, used only when `individual_scaling = 'constant'`.

`zmap` Boolean: also create a z-map (default: FALSE).

`zmapmethod` Method for the z-map: `quick` (default) or `t.test`.

`zmapdecoration` Boolean: draw the z-map with margins, a caption (sigma, threshold) and a scale (default: TRUE).

`sigma` Amount of smoothing applied when creating the z-map (default: 3).

`threshold` Threshold z-score (default: 3). Z-scores below it are not drawn on the z-map.

`zmaptargetpath` Directory to save z-map PNGs to. Required when `zmap = TRUE`; there is no default. The directory is created if it does not exist; to just try the function out, use `tempdir()`.

`n_cores` Number of CPU cores for the per-participant CIs (with `participants`) and the `'t.test'` z-map (default: `detectCores() - 1`; 2 under `R CMD check`, per

CRAN policy). Each core runs a worker holding its own copy of the noise basis and a render's working memory: 0.8 to 1 GB per worker at 512 pixels in https://github.com/rdotsch/rcicr/blob/main/analyses/worker-memory.md. No more workers start than there are participants or trials.

`mask` Optional mask for the CI: a 2D matrix (0 = masked, 1 = kept) or the path to a greyscale PNG image (black = masked, white = kept). Default: `NA`, no mask. Documentation up to and including 1.1.0 described the matrix the wrong way round (1 = masked); the code has always masked where the matrix is 0.

`zmappointsize` Text size of the z-map decoration, in points (default: 12). Passed to `plotZmap`, which makes the z-map image `img_size` pixels wide. The decoration needs roughly `12.3 * zmappointsize` pixels on a 72 ppi device and `16.4 * zmappointsize` on a 96 ppi one. At the default, a stimulus set smaller than about 160 to 200 pixels is too small for it, and `generateCI()` stops with an error naming the minimum for your device. Lower this value to fit the decoration on a small z-map, or set `zmapdecoration = FALSE`.

Details

This function returns the classification image (CI) and, by default, saves it as a PNG. How the CI is scaled for display decides what the image looks like and whether two images can be compared. The default, `'independent'`, picks the lowest scaling constant that avoids clipping this particular CI (see `'constant'` below for the formula). Each CI therefore gets its own constant, and CIs with different noise ranges cannot be compared by eye.

`'matched'` scaling matches the range of the CI's pixel intensities to the range of the base image's. This is nonlinear and depends on the ranges of both. It also shifts the zero point of the noise: a pixel that would not have changed the base image before scaling may change it afterwards, and the other way round. Use it as a quick look at how the noise affects the base image, not for reporting.

`'constant'` scaling does not depend on the base image or the noise range, but the constant is yours to choose, with the `scaling_constant` argument. The noise is scaled as `scaled <- (ci + constant) / (2 * constant)`. Pixel intensities must lie between 0 and 1; if the scaled noise falls outside that range you get a warning and should pick a higher constant. The higher the constant, the fainter the noise in the resulting image. Use the same constant for every classification image you want to compare.

For several classification images, the lowest constant that works for all of them is a good choice. `autoscale` finds it for you.

Value

List of pixel matrices: the raw classification noise (`ci`), the scaled noise (`scaled`), the base image (`base`) and the two combined (`combined`), plus the z-map (`zmap`) when `zmap = TRUE`. Its `trial_design` attribute records which saved stimuli the CI was built from, so that `computeInfoVal2IFC` can check that its reference matches the CI; its `scaling` attribute records how `scaled` was made, for the group CI and, with `participants`, for the individual CIs. `vignette("stored-data", package = "rcicr")` lists the fields of both. To keep them in a file, save the whole result with `saveRDS()`.

Repeated presentations of the same stimulus

When `participants` is `NA` (the default), repeated presentations of the same stimulus are averaged before the CI is built, so each unique stimulus gets equal weight however often it was shown. If every stimulus was shown equally often, this is the same as weighting each trial equally. If not, it changes what the CI estimates: a stimulus shown three times counts the same as one shown once, not three times as much.

So in an unbalanced design, where a participant saw some stimuli more often than others (an adaptive procedure, a crashed session, or by design), the CI is the average response per unique stimulus, not per trial. The two can differ a lot: on an 8-trial set with counts 4/2/1/1 they correlate at 0.77.

`computeCumulativeCICorrelation` does *not* average repeats; it weights each trial equally. With unequal counts, the final CI it computes itself therefore differs from the one this function returns. To compare against the CI you will report, pass this function's output as its `targetci`.

See Also

`vignette("reverse-correlation-walkthrough", package = "rcicr")`, sections "Computing one classification image" and "Scaling"; `vignette("recipes", package = "rcicr")` for rating scales as responses and for masked classification images.

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

responses <- sample(c(1, -1), 6, replace = TRUE)
ci <- generateCI(
  stimuli = 1:6, responses = responses, baseimage = "face",
  rdata = rdata_file, save_as_png = FALSE
)
```

`generateCI2IFC` *Generates 2IFC classification image*

Description

Generate a classification image for a two-image forced-choice reverse correlation task. This function wraps generateCI and is kept so that older scripts still run; new code can call generateCI() directly.

Usage

```
generateCI2IFC(
  stimuli,
  responses,
  baseimage,
  rdata,
  save_as_png = TRUE,
  filename = "",
  targetpath,
  antiCI = FALSE,
  scaling = "independent",
  constant = 0.1
)
```

Arguments

stimuli	Numeric vector of stimulus numbers, one per response and in the same order. Each must be a positive whole number no larger than the number of trials saved for the selected base image. Numbers may repeat and need not be consecutive. Factors, characters and logicals are rejected: if your data hold stimulus labels, check them against the generated stimulus filenames before converting them to numbers.
responses	Vector of responses in the same order as stimuli: 1 where the original stimulus was chosen, -1 where the inverted one was. Must be numeric and finite; other finite numbers, such as ratings, weight the trials accordingly. Factors, characters and logicals are rejected, as are NA, NaN and infinite values: remove trials without a response from every argument alike.
baseimage	String naming the base image: not its file name, but its key in the base_face_files list passed to generateStimuli2IFC.
rdata	Path to the .Rdata file written when the stimuli were generated. It holds the contrast parameters of every stimulus.
save_as_png	Boolean: also save the CI as a PNG image.
filename	Optional file name for the PNG image.
targetpath	Directory to save PNGs to. Required when save_as_png = TRUE; there is no default. The directory is created if it does not exist; to just try the function out, use tempdir().
antiCI	Boolean: compute the anti-CI, the classification image with its sign flipped, instead of the CI.
scaling	Scaling method: none, constant, matched or independent (default).
constant	Scaling constant for the noise. Used only when scaling = 'constant'.

Details

This function returns the classification image (CI) and, by default, saves it as a PNG. How the CI is scaled for display decides what the image looks like and whether two images can be compared. The default, `'independent'`, picks the lowest scaling constant that avoids clipping this particular CI (see `'constant'` below for the formula). Each CI therefore gets its own constant, and CIs with different noise ranges cannot be compared by eye.

`'matched'` scaling matches the range of the CI's pixel intensities to the range of the base image's. This is nonlinear and depends on the ranges of both. It also shifts the zero point of the noise: a pixel that would not have changed the base image before scaling may change it afterwards, and the other way round. Use it as a quick look at how the noise affects the base image, not for reporting.

`'constant'` scaling does not depend on the base image or the noise range, but the constant is yours to choose, with the `constant` argument. The noise is scaled as `scaled <- (ci + constant) / (2 * constant)`. Pixel intensities must lie between 0 and 1; if the scaled noise falls outside that range you get a warning and should pick a higher constant. The higher the constant, the fainter the noise in the resulting image. Use the same constant for every classification image you want to compare.

For several classification images, the lowest constant that works for all of them is a good choice. `autoscale` finds it for you.

Value

List of pixel matrices: the raw classification noise (`ci`), the scaled noise (`scaled`), the base image (`base`) and the two combined (`combined`).

See Also

`generateCI`, which new code should use; `vignette("reverse-correlation-walkthrough", package = "rcicr")`, section "Computing one classification image".

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

responses <- sample(c(1, -1), 6, replace = TRUE)
ci <- generateCI2IFC(
```

```
  stimuli = 1:6, responses = responses, baseimage = "face",
  rdata = rdata_file, save_as_png = FALSE
)
```

`generateCINoise` *Generate classification image noise pattern based on set of stimuli (matrix: trials, parameters), responses (vector), and sinusoid*

Description

Generate classification image noise pattern based on set of stimuli (matrix: trials, parameters), responses (vector), and sinusoid

Usage

```
generateCINoise(stimuli, responses, p)
```

Arguments

`stimuli`	Matrix with one parameter row per response. A parameter vector is also accepted for a single trial with exactly one response.
`responses`	Vector with the response to each trial: 1 where the participant chose the original, -1 where they chose the inverted image. Other values, such as ratings on a scale, weight the trials accordingly.
`p`	Noise basis, as returned by `generateNoisePattern`.

Value

The classification image as pixel matrix.

Examples

```
p <- generateNoisePattern(img_size = 32, nscales = 1)
nparams <- max(p$patchIdx)

# two trials, one chosen original (1) and one inverted (-1)
stimuli <- matrix(runif(2 * nparams, -1, 1), nrow = 2)
responses <- c(1, -1)

ci <- generateCINoise(stimuli, responses, p)
```

generateGabor *Generate single gabor patch*

Description

Generate single gabor patch

Usage

```
generateGabor(img_size, cycles, angle, phase, sigma, contrast)
```

Arguments

img_size	Integer specifying size of gabor patch in number of pixels.
cycles	Integer specifying number of cycles the sinusoid should span.
angle	Value specifying the angle (rotation) of the sinusoid.
phase	Value specifying phase of sinusoid.
sigma	Value specifying the standard deviation, in pixels, of the Gaussian mask applied on top of the sinusoid.
contrast	Value between -1.0 and 1.0 specifying contrast of sinusoid.

Value

The gabor patch image with size img_size.

Examples

```
generateGabor(512, 2, 90, pi/2, 25, 1.0)
```

generateNoiseImage *Generate single noise image based on parameter vector*

Description

Generate single noise image based on parameter vector

Usage

```
generateNoiseImage(params, p)
```

Arguments

params	Vector of contrast weights, one per patch in the noise basis.
p	Noise basis, as returned by generateNoisePattern.

Value

The noise pattern as pixel matrix.

Examples

```
p <- generateNoisePattern(img_size = 32, nscales = 2)

# one contrast weight per patch index
params <- rnorm(max(p$patchIdx))

noise <- generateNoiseImage(params, p)
```

---

generateNoisePattern *Generate sinusoid noise pattern*

Description

Generate sinusoid noise pattern

Usage

```
generateNoisePattern(
  img_size = 512,
  nscales = 5,
  noise_type = "sinusoid",
  sigma = 25,
  pre_0.3.0 = FALSE
)
```

Arguments

`img_size`	Width and height of the noise pattern, in pixels. It must be divisible by `2^(nscales - 1)`, because the finest scale tiles the image with that many patches per side.
`nscales`	Number of spatial scales (default: 5). Each additional scale adds a higher spatial frequency. `img_size` must be divisible by `2^(nscales - 1)`.
`noise_type`	Noise pattern type: `sinusoid` (default) or `gabor`.
`sigma`	Sigma of the Gabor patches when `noise_type = 'gabor'` (default: 25).
`pre_0.3.0`	Boolean: build the noise pattern the way rcicr did before version 0.3.0. Leave it at `FALSE` unless you need to recreate that old behaviour.

Value

The noise basis: a list holding the 3D array of patches (`patches`), an index array of the same size mapping each pixel to its contrast parameter (`patchIdx`), and the `noise_type` and `generator_version`.

Examples

```
generateNoisePattern(256)
```

`generateReferenceDistribution2IFC` *Generates reference distribution*

Description

Generates the reference distribution of norms for a stimulus set.

Usage

```
generateReferenceDistribution2IFC(
  rdata,
  iter = 10000,
  ncores = default_ncores(),
  response_seed = NULL,
  save_rdata = TRUE,
  baseimage = NULL,
  reference_stimuli = NULL,
  reference_method = c("gram", "images"),
  mask = NA
)
```

Arguments

`rdata`	Path to the `.Rdata` file written when the stimuli were generated. It holds the contrast parameters of every stimulus.
`iter`	Number of simulated classification images, each built from random responses; the distribution holds one norm per image.
`ncores`	Number of CPU cores used to render the saved noise with `reference_method = "images"` (default: `detectCores() - 1`; 2 under `R CMD check`, per CRAN policy). `"gram"` does not use it. Each core runs a worker holding its own copy of the noise basis and a render's working memory: 0.8 to 1 GB per worker at 512 pixels in https://github.com/rdotsch/rcicr/blob/main/analyses/worker-memory.md. No more workers start than there are trials.
`response_seed`	Optional seed for the simulated random responses. The default, `NULL`, continues from the state the stimulus generator left behind, as described under Reproducibility; it needs the stimulus seed saved in the file. A number gives an independent draw of the null from the same stimuli.
`save_rdata`	Boolean: write the reference distribution into the `rdata` file (default `TRUE`). With `FALSE`, later calls to `computeInfoVal2IFC` keep using what the file already holds. Set it to `FALSE` whenever you set `response_seed`, so a one-off null does not become the file's permanent reference. The exception is a file without a stimulus seed: there the seeded reference is the one to keep, as the file's reproducible reference.

`baseimage` Base-image label, the same key passed to `generateCI()`. Required when the base images have different noise parameters. With a single base image, or base images sharing one parameter set, leave it at `NULL`.

`reference_stimuli` Optional stimulus numbers, each once, to build the reference over instead of every saved stimulus: the stimuli a classification image was built from, when that is not all of them. See "Matching the reference to the classification image" in `computeInfoVal2IFC`. The simulated responses continue the same stream as the default, so the reference is reproducible from the file. Passing every saved stimulus is the same as the default, `NULL`.

`reference_method` `"gram"` (the default) or `"images"`: how the reference norms are computed. `"images"` reproduces rcicr 1.5.0 and earlier bit for bit, with the same BLAS and LAPACK. See "Reference method" below.

`mask` Optional mask, in any form `generateCI` accepts: a 0/1 matrix or the path to a PNG, black (0) where masked. The reference is then built over the unmasked pixels only, as `computeInfoVal2IFC` needs for a classification image computed with that mask, and stored apart from the default one. The default, `NA`, uses every pixel. See "Masked classification images" in `computeInfoVal2IFC`.

Details

`computeInfoVal2IFC` scores a classification image against this distribution. By default the result is saved in the `rdata` file for later reuse.

Value

The reference distribution, invisibly, as a numeric vector of `iter` norms. Unless `save_rdata = FALSE`, it is also added to the `rdata` file as `reference_norms`, with `reference_norms_seed` recording the `response_seed` it was drawn with. A later `computeInfoVal2IFC` call on the same file then reuses it instead of simulating again. For independent base images it goes in `reference_norms_by_base` instead, as described above. A reference over `reference_stimuli` goes in `reference_norms_by_stimuli`, a list with one entry per stimulus set (and base image, where the bases have different noise), leaving the default reference untouched.

Reproducibility

With the default `response_seed = NULL`, the reference distribution depends only on the stimulus `.Rdata` file: not on the current random state, not on `ncores`, and, for files that record the RNG kind, not on the session's `RNGkind`. Two researchers computing InfoVal from the same stimulus file get the same reference distribution, and the same number, on any machine and in any session.

This needs the stimulus seed saved in the file. A file without one (made with `generateStimuli2IFC(seed = NULL)`, or with the field removed) has no stream to continue, so the default stops with an error; pass a `response_seed` instead.

Stimulus files record the `RNGkind` the stimuli were drawn under, as `rng_kind`, and the simulated responses are drawn under it, with or without a `response_seed`. When it differs from the session's kind, the caller's random stream is restored afterwards, kind and position, so the session is not left on the file's generator. In the session's own kind the stream is left where the draws end, as it always

was. Files written before the kind was recorded draw under the session's kind: a session with a different `RNGkind()` draws different responses, and so a different null. To reproduce an earlier reference from such a file, use the RNG kind that built it; see https://github.com/rdotsch/rcicr/issues/315.

The noise is rebuilt from the basis and parameters saved in the stimulus file, so the base images are never reopened and a moved or archived experiment can still be scored. The simulated responses continue the random stream from the saved stimulus seed, after the draws for one parameter matrix, which reproduces the historical default reference.

Pass a `response_seed` to draw a *different* null from the same stimuli, for instance to check how much Monte Carlo error a given `iter` leaves in your InfoVal. This changes only the simulated responses, not the stimuli or the noise basis the null is built on.

Reference method

`reference_method = "gram"`, the default, computes the norms from the stimulus Gram matrix, without calling `generateNoiseImage()` or multiplying the noise for every draw. For a small stimulus set the Gram matrix is built from the noise rendered through the sparse basis; otherwise from the basis's cross-product, kept sparse. Either way the basis is built a block of image columns at a time, so no full-size copy of it, or of the noise, is held. `"images"` is the calculation of rcicr 1.5.0 and earlier: every noise image rendered, in parallel over `ncores`, and multiplied for every draw. It reproduces references from those versions bit for bit with the same BLAS and LAPACK. A library such as MKL, which does not promise the same last bits on every run, reproduces them only to rounding.

The two agree to rounding. No configuration measured was bit-identical, and none differed by more than a relative 5e-14 in a single norm (reference BLAS; see https://github.com/rdotsch/rcicr/blob/main/analyses/gram-reference-accuracy.md).

Computing a reference with `"gram"` prints a message saying so, and naming `"images"` as the way to reproduce earlier references. Silence it with `suppressMessages()`. The method is never switched for you.

A reference already stored in the file is reused as stored, whichever method is asked for. To rebuild one with a particular method, add `force_gen_ref_dist = TRUE` in `computeInfoVal2IFC`, or call this function with `save_rdata = TRUE`. The method a reference was computed with is stored beside it, as `reference_norms_method` or as `method` in each `reference_norms_by_base` and `reference_norms_by_stimuli` entry.

Independent base images

When the base images have different parameter matrices, `baseimage` says whose noise to use. The distributions are then stored in `reference_norms_by_base`, one entry per base label, each holding `norms` and `response_seed`. A shared `reference_norms` in such a file is neither used nor overwritten. Files whose base images share one parameter matrix keep using `reference_norms`.

See Also

`vignette("reverse-correlation-walkthrough", package = "rcicr")`, section "Is there actually signal?"; `vignette("recipes", package = "rcicr")`, "Matching the InfoVal reference to the design" and "Reproducing a number from an earlier version".

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

stimulus_path <- tempfile("stimuli")
generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 6,
  img_size = 32,
  stimulus_path = stimulus_path,
  seed = 1,
  ncores = 1,
  nscales = 1,
  save_as_png = FALSE
)
rdata_file <- list.files(stimulus_path, pattern = "\\.Rdata$", full.names = TRUE)[1]

# iter is kept tiny here for a fast example; in practice use iter >= 10000.
suppressWarnings(generateReferenceDistribution2IFC(rdata_file, iter = 3, ncores = 1))
```

---

generateSinusoid *Generate single sinusoid patch*

Description

Generate single sinusoid patch

Usage

```
generateSinusoid(img_size, cycles, angle, phase, contrast)
```

Arguments

img_size	Integer specifying size of sinusoid patch in number of pixels.
cycles	Integer specifying number of cycles sinusoid should span.
angle	Value specifying the angle (rotation) of the sinusoid.
phase	Value specifying phase of sinusoid.
contrast	Value between -1.0 and 1.0 specifying contrast of sinusoid.

Value

The sinusoid image with size img_size.

Examples

```
generateSinusoid(512, 2, 90, pi/2, 1.0)
```

`generateStimuli2IFC` *Generates 2IFC stimuli*

Description

Generate stimuli for a two-image forced-choice reverse correlation task.

Usage

```
generateStimuli2IFC(
  base_face_files,
  n_trials = 770,
  img_size = 512,
  stimulus_path,
  label = "rcic",
  use_same_parameters = TRUE,
  seed = 1,
  maximize_baseimage_contrast = TRUE,
  noise_type = "sinusoid",
  nscales = 5,
  sigma = 25,
  ncores = default_ncores(),
  return_as_dataframe = FALSE,
  save_as_png = TRUE,
  save_rdata = TRUE
)
```

Arguments

`base_face_files`
: Named list of base image files, e.g. `list(aName = 'baseface.jpg')`. JPEG and PNG images are accepted, recognised by a `.png`, `.jpg` or `.jpeg` extension. Each name labels that base image's stimulus files and is the key `generateCI` uses to find it in the `.Rdata` file, so every element needs a unique name. Each image must be square and exactly `img_size` pixels wide: rcicr does not resize base images. All of this is checked before any stimuli are generated, and an error names the offending entry.

`n_trials`
: Number of trials. Each trial gets two images per base image: one with the noise added (original) and one with it subtracted (inverted).

`img_size`
: Width and height of the square stimulus images, in pixels. It must be divisible by `2^(nscales - 1)`, because the finest scale tiles the image with that many patches per side.

`stimulus_path`
: Directory to save the stimuli and the `.Rdata` file to. Required unless both `save_as_png` and `save_rdata` are FALSE; there is no default. The directory is created if it does not exist; to just try the function out, use `tempdir()`.

`label`
: Label put at the start of each file name.

use_same_parameters
: Boolean: all base images share one set of noise parameters (TRUE) or each gets its own (FALSE).

seed
: Seed for the random number generator, for reproducibility. It is saved in the .Rdata file with the session's RNGkind(), and the default InfoVal reference replays it under that kind. With seed = NULL there is nothing to replay, so InfoVal references for that file need an explicit response_seed. The caller's own random stream is left as it was: the next random number drawn after this call is the one that would have been drawn without it.

maximize_baseimage_contrast
: Boolean: rescale the base image's pixel values to maximize its contrast. A base image with no contrast at all, every pixel the same value, cannot be rescaled and is rejected with an error. It can still be used with maximize_baseimage_contrast = FALSE.

noise_type
: Noise pattern type: sinusoid (default) or gabor.

nscales
: Number of spatial scales (default: 5). Each additional scale adds a higher spatial frequency. img_size must be divisible by 2^(nscales - 1).

sigma
: Sigma of the Gabor patches when noise_type = 'gabor' (default: 25).

ncores
: Number of CPU cores to use (default: detectCores() - 1; 2 under R CMD check, per CRAN policy). Each core runs a worker holding its own copy of the noise basis and a render's working memory: 0.8 to 1 GB per worker at 512 pixels in https://github.com/rdotsch/rcicr/blob/main/analyses/worker-memory.md. No more workers start than there are trials.

return_as_dataframe
: Boolean: return a data frame with the raw noise of the generated stimuli (default: FALSE), one row per pixel and one column per trial. With the default use_same_parameters = TRUE every base image shares the same noise, so that is all of it. With use_same_parameters = FALSE and more than one base image, only the first base image's noise is returned, because one column per trial cannot hold several. The stimuli are still written for every base image, and save_rdata = TRUE records every parameter set, so nothing is missing from the files.

save_as_png
: Boolean: write the stimuli to disk as PNG images (default: TRUE). They are named <label>_<base label>_<seed>_<trial>_ori.png and _inv.png, with no time, so a later call into the same folder with the same label, base label and seed would write the same names. Existing PNGs are never overwritten: the call stops before generating or writing anything, and also stops when two base labels name the same file on this file system (for example, labels differing only in case). Use a different label or stimulus_path, or, to regenerate a stimulus set on purpose, delete its PNGs and its .Rdata file first. While PNGs are being written, a second call into the same folder with the same seed stops.

save_rdata
: Boolean: save the .Rdata file with the stimulus parameters (default: TRUE). Computing classification images needs that file, so keep this TRUE; the argument exists mainly for internal use. The file is named <label>_seed_<seed>_time_<month>_<day>_<year>_ for the minute the call started. An existing file of that name is never overwritten: the call stops before generating anything. So does a call into the same folder with the same seed, started in the same minute, while another is still running.

Details

Saves the stimuli as PNGs, together with an `.Rdata` file holding the parameters used to generate each stimulus. Analysing the responses later requires that file.

Value

Nothing: everything is saved to files. `vignette("stored-data", package = "rcicr")` lists what the `.Rdata` file holds. With `return_as_dataframe = TRUE`, the data frame described there.

See Also

`vignette("getting-started", package = "rcicr")` for the shortest working example; `vignette("reverse-correlatio` `package = "rcicr")`, section "Generating stimuli", for choosing the settings; `vignette("recipes", package = "rcicr")`, "Noise-only stimuli", for stimuli without a face.

Examples

```
# a synthetic square grayscale image stands in for a real base face photo
base_face <- tempfile(fileext = ".png")
png::writePNG(matrix(runif(32 * 32), 32, 32), base_face)

generateStimuli2IFC(
  base_face_files = list(face = base_face),
  n_trials = 4,
  img_size = 32,
  stimulus_path = tempfile("stimuli"),
  seed = 1,
  ncores = 1,
  nscales = 1
)
```

`plotZmap` *Plots a Z-map*

Description

Plots a z-map: a matrix of z-scores drawn over a background image.

Usage

```
plotZmap(
  zmap,
  bgimage = "",
  sigma,
  threshold = 3,
  mask = NULL,
  decoration = TRUE,
  targetpath,
```

```
  filename = "zmap",
  size = 512,
  ...,
  pointsize = 12
)
```

Arguments

zmap	Matrix of z-scores. Must have the same dimensions as bgimage.
bgimage	Greyscale image matrix to draw behind the z-map, normally the base image or the final CI. Without it, only the z-map is drawn.
sigma	Sigma of the smoothing applied to the CI to create the z-map, shown in the decoration.
threshold	Threshold z-score (default: 3). Z-scores below it are not drawn on the z-map.
mask	Optional mask: a binary matrix the size of zmap (0 or FALSE = masked, 1 or TRUE = kept), or the path to a black-and-white PNG image (black = masked, white = kept). This is the same convention as generateCI()'s mask, so one mask works for both. Earlier versions of this documentation described the matrix the wrong way round.
decoration	Boolean: draw the z-map with margins, a caption (sigma, threshold) and a scale (default: TRUE).
targetpath	Directory to save the z-map PNG to. Required, since writing that file is this function's purpose; there is no default. The directory is created if it does not exist; to just try the function out, use tempdir().
filename	Optional file name for the z-map PNG.
size	Width and height of the PNG image, in pixels (default: 512).
...	Further arguments passed to graphics::image. Used only when decoration = TRUE.
pointsize	Text size of the decoration, in points (default: 12, the graphics device's own default). Margins are measured in lines of text, so this also sets how much of the image the decoration takes up. The minimum image size is fixed in inches, so in pixels it depends on the device's resolution: roughly 12.3 * pointsize pixels at 72 ppi (Linux, macOS) and 16.4 * pointsize at 96 ppi (Windows), about 160 and 200 pixels at the default. Below that, plotZmap() stops and names the minimum for your device. A lower pointsize fits a decorated z-map onto a small image, at the cost of a smaller map: the margins shrink but the labels still need room. Ignored when decoration = FALSE, which has no margins and works at any size.

Details

Takes a matrix of z-scores, such as the zmap that generateCI returns, and optionally a background image matrix, and writes the z-map as a PNG. Further arguments are passed to graphics::image; for example, col sets a different colour palette. Versions up to and including 1.2.3 passed them to the raster package's plot method instead. col works the same in both, but arguments specific to that method are no longer understood.

Value

Nothing; the z-map is written as a PNG.

Reproducibility across platforms

The z-scores are ordinary R arithmetic and do not depend on your operating system; nor do classification images, scaling or informational value. The PNG this function writes does: it is drawn through a graphics device, and devices differ between platforms in colour management and in whether they write an alpha channel. The same z-map rendered on Linux and on macOS gives figures that look identical but are not byte-identical. macOS renders a mid-grey background at roughly 0.573, where the cairo device gives 0.502.

So to check that an analysis reproduces, compare the numbers, not the rendered figures. A z-map PNG that differs pixel for pixel on a colleague's machine is not a different result, and regenerating figures on another platform is safe.

This applies only to `plotZmap()`, the one function in the package that opens a graphics device. Every other PNG `rcicr` writes (stimuli, classification images, autoscaled classification images) comes straight from the pixel array via `png::writePNG()`.

See Also

`vignette("reverse-correlation-walkthrough", package = "rcicr")`, section "Is there actually signal?".

Examples

```
set.seed(1)
zmap <- matrix(rnorm(64, sd = 5), 8, 8)
plotZmap(zmap, sigma = 3, threshold = 3, decoration = FALSE,
         targetpath = tempdir(), size = 200)
```

---

simulateNoiseIntensities

Simulate pixel intensity range for noise

Description

Simulate pixel intensity range for noise

Usage

```
simulateNoiseIntensities(nrep = 1000, img_size = 512)
```

Arguments

`nrep`	Number of replications
`img_size`	Size of noise pattern in pixels (one value equal for width and height). The pattern has five scales, so it must be a multiple of 16.

Value

Matrix with range of noise intensities for each replication

Examples

```
# nrep and img_size are kept small here so the example is fast; the defaults
# (1000 replications at 512px) are what you want for a real estimate.
simulateNoiseIntensities(nrep = 10, img_size = 64)
```

Index

autoscale, 2, *6*, *8*, *19*, *22*

batchComputeInfoVal2IFC, 3, *15*
batchGenerateCI, *3*, 5
batchGenerateCI2IFC, *3, 4*, 7

checkStimulusPNGs2IFC, 10
computeCumulativeCICorrelation, 11, *20*
computeInfoVal2IFC, *3, 4*, 13, *19*, *26–28*

deg2rad, 16

generateCI, *4*, *6*, *8*, *12, 13*, *15*, 17, *21, 22*, *27*, *30*, *33*
generateCI2IFC, *9*, 20
generateCINoise, 23
generateGabor, 24
generateNoiseImage, 24
generateNoisePattern, *23, 24*, 25
generateReferenceDistribution2IFC, *13, 14*, 26
generateSinusoid, 29
generateStimuli2IFC, *6*, *8*, *10, 11*, *18*, *21*, 30

plotZmap, *19*, 32

RNGkind, *27*, *31*

simulateNoiseIntensities, 34