Package 'rcicr'

September 20, 2026

Type Package

Title Reverse-Correlation Image-Classification Toolbox

Version 1.4.1

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 matlab, png, jpeg, dplyr, scales, viridis, utils, parallel, doSNOW, foreach, spatstat.explore, spatstat.geom, tibble, yesno

Suggests testthat (>= 3.0.0), 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 2026-09-20 00:30:02 UTC

Contents

`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 cis, each of which are a list containing the pixel matrices of at least the noise pattern (`$ci`) and if the noise patterns need to be written to PNGs, also the base image (`$base`).
`save_as_pngs`	Boolean, when set to true, the autoscaled noise patterns will be combined with their respective base images and saved as PNGs (using the key of the list as name).
`targetpath`	String specifying the directory to save PNGs to. Required when `save_as_pngs = TRUE`; there is no default path. It is created if it does not exist. Use `tempdir()` if you only want to try the function out.

Value

The input `cis` list, with the `$scaled` pixel matrix of each element replaced by its autoscaled version. The scaling constant that was determined is printed to the console, not returned.

Look at `$scaled`, not `$combined`. `$combined` is left exactly as it was passed in: autoscaling deliberately does not disturb it, so a combination made before this call survives unchanged. Only

`$scaled` reflects the autoscaled result. If you want the autoscaled noise shown over the base image, build it yourself with `(ci$scaled + ci$base) / 2` — that is exactly what `save_as_pngs = TRUE` writes to disk.

This matters most after `batchGenerateCI` or `batchGenerateCI2IFC`, which scale with `'none'` before handing over to this function: their `$combined` is therefore an overlay of the *unscaled* noise and will look almost blank, while `$scaled` is the image you want.

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)
```

---

batchGenerateCI *Generates multiple classification images by participant or condition*

Description

Generate classification image for any reverse correlation task that displays independently generated alternatives.

Usage

```
batchGenerateCI(
  data,
  by,
  stimuli,
  responses,
  baseimage,
  rdata,
  save_as_png = TRUE,
  targetpath,
  label = "",
  antiCI = FALSE,
  scaling = "autoscale",
  constant = 0.1
)
```

Arguments

`data`	Data frame
`by`	String specifying column name that specifies the smallest unit (participant, condition) to subset the data on and calculate CIs for.
`stimuli`	String specifying column name in data frame that contains the stimulus numbers of the presented stimuli.

responses	String specifying column name in data frame that contains the responses coded 1 for original stimulus selected and -1 for inverted stimulus selected.
baseimage	String specifying which base image was used. Not the file name, but the key used in the list of base images at time of generating the stimuli.
rdata	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli.
save_as_png	Boolean stating whether to additionally save the CI as PNG image.
targetpath	String specifying the directory to save PNGs to. Required when save_as_png = TRUE; there is no default path. It is created if it does not exist. Use tempdir() if you only want to try the function out.
label	Optional string to insert in file names of PNGs to make them easier to identify.
antiCI	Optional boolean specifying whether antiCI instead of CI should be computed.
scaling	Optional string specifying scaling method: none, constant, independent or autoscale (default).
constant	Optional number specifying the value used as constant scaling factor for the noise (only works for scaling='constant').

Details

This function saves the classification images by participant or condition as PNG to a folder and returns the CIs.

Value

List of classification image data structures (which are themselves lists of pixel matrix of classification noise only, scaled classification noise only, base image only and combined).

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 <- tempdir()
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 classification image for 2 images 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 = ""
)
```

Arguments

`data`	Data frame
`by`	String specifying column name that specifies the smallest unit (participant, condition) to subset the data on and calculate CIs for.
`stimuli`	String specifying column name in data frame that contains the stimulus numbers of the presented stimuli.
`responses`	String specifying column name in data frame that contains the responses coded 1 for original stimulus selected and -1 for inverted stimulus selected.
`baseimage`	String specifying which base image was used. Not the file name, but the key used in the list of base images at time of generating the stimuli.

rdata	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli.
save_as_png	Boolean stating whether to additionally save the CI as PNG image.
targetpath	String specifying the directory to save PNGs to. Required when save_as_png = TRUE; there is no default path. It is created if it does not exist. Use tempdir() if you only want to try the function out.
antiCI	Optional boolean specifying whether antiCI instead of CI should be computed.
scaling	Optional string specifying scaling method: none, constant, matched, independent, or autoscale (default).
constant	Optional number specifying the value used as constant scaling factor for the noise (only works for scaling='constant').
label	Optional string to insert in file names of PNGs to make them easier to identify.

Details

This function saves the classification images by participant or condition as PNG to a folder and returns the CIs.

Value

List of classification image data structures (which are themselves lists of pixel matrix of classification noise only, scaled classification noise only, base image only and combined).

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 <- tempdir()
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
))
```

`computeCumulativeCICorrelation` *Computes cumulative trial CIs correlations with final/target CI*

Description

Computes cumulative trial CIs correlations with final/target CI.

Usage

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

Arguments

`stimuli`	Numeric vector of stimulus numbers in response order, with one finite, positive whole number per response, within the trials saved for the selected base image. Repeated and nonconsecutive numbers are allowed. Factors, characters and logicals are rejected; verify imported labels against the generated stimulus filenames before converting them to numeric IDs.
`responses`	Vector specifying the responses in the same order of the stimuli vector, coded 1 for original stimulus selected and -1 for inverted stimulus selected.
`baseimage`	String specifying which base image was used. Not the file name, but the key used in the list of base images at time of generating the stimuli.
`rdata`	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli.
`targetci`	List Target CI object generated with rcicr functions to correlate cumulative CIs with.
`step`	Step size in sequence of trials to compute correlations with.

Details

Use for instance for plotting curves of trial-final/target CI correlations to estimate how many trials are necessary in your task

Value

Vector containing correlation between cumulative CI and final/target CI.

Repeated presentations of the same stimulus

This function walks trials in the order they were presented and does not aggregate repeated presentations of a stimulus, unlike `generateCI`, which averages the responses to each unique stimulus before building its classification image. That is deliberate: collapsing repeats would discard the presentation order a cumulative curve is entirely about.

One consequence is worth knowing. With no `targetci`, the final CI computed here is built from the same un-aggregated trials as the curve. Where the evaluated trials reach the last one – always so at the default `step = 1` – the curve's final point compares that CI with itself and is exactly 1: self-consistency, not evidence of 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 one evaluated is the fifth, and the curve ends at whatever that partial CI correlates to – 0.97 in one such set, not 1.

Both statements assume the CI being compared against varies at all. Responses that cancel exactly – every presentation of a stimulus answered both ways – average to a uniformly zero CI, and a correlation against a constant is undefined, so **every** point on the curve is `NA` rather than the last one being 1. Such a curve means the responses carry no net signal, not that the call failed.

A `targetci` carrying masked pixels – `generateCI` stores `NA` in every pixel a `mask` excludes – is handled by correlating over the unmasked pixels only. If the mask covers *every* pixel, there are no complete pairs and the curve is all-`NA`, same as the zero-variance case above.

Where every stimulus was presented the same number of times, that final CI is identical to the one `generateCI` returns. Where repeat counts differ, 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 actually report, pass it as `targetci = generateCI(...)` rather than relying on the self-computed default – built without a `mask`, per the note above.

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 <- tempdir()
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 Informational Value for a single CI in a 2IFC task.

Usage

```
computeInfoVal2IFC(
  target_ci,
  rdata,
  iter = 10000,
  force_gen_ref_dist = FALSE,
  response_seed = NULL,
  baseimage = NULL
)
```

Arguments

`target_ci`	A classification image object (list-type) as returned by generateCI
`rdata`	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli and possibly its corresponding reference distribution generated with generateReferenceDistribution().
`iter`	Number of iterations for the simulation of the reference distribution (only used if reference distribution is not already pre-generated and present in rdata file)
`force_gen_ref_dist`	Boolean specifying whether to override the default behavior to use pre-computed values for the reference distribution for specific task parameters and instead force to recompute the reference distribution (default: FALSE).
`response_seed`	Optional seed for the simulated random responses used to build the reference distribution. The default (`NULL`) uses the reference distribution stored in the `rdata` file, or generates the reproducible default one described under Reproducibility in `generateReferenceDistribution2IFC`. Supplying a number forces a fresh reference distribution to be simulated from an independent draw, which is how you check how much Monte Carlo error `iter` leaves in this Informational Value. The result is deliberately *not* written back to the `rdata` file, so a one-off check cannot change the number every later analysis of that stimulus set reports.

`baseimage` Saved base-image label used to generate `target_ci`. Required when saved base images have different noise parameters; use the same label passed to `generateCI()`. With a single base or identical parameter matrices, the default `NULL` retains the shared reference behavior. Independent-base caches are separate for each label; old unscoped reference norms are ignored for those files.

Details

The Informational Value metric can be considered as a z-score that quantifies the signal present in a classification image. The higher the Informational Value, the more signal. It is possible to use a cut-off such as z = 1.96 to select classification images with significant signal under alpha = 0.05.

Informational Value is computed by simulating random responding under identical task parameters to an empirical dataset (called the reference distribution). The metric quantifies how unlikely it is to observe these data under the null-hypothesis that there is no signal (i.e., that there is only random responding).

The simulation to compute the reference distribution takes a long time, and is only run locally when pre-computed values for the reference distribution matching the stimulus set in the .Rdata file have not been supplied by the rcicr package.

A reference the `rdata` file does not already carry is simulated and then stored there for reuse. Where that file cannot be written, the simulated reference is used for this call, a note names the file – and, for independent bases, the base – that could not be saved, and the next call simulates it again. An archive on read-only media is therefore still scoreable, at the cost of re-simulating each time.

For more information see Brinkman, Goffin, Aarts, van Haren, & Dotsch (in prep).

Value

Informational value (z-score)

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 <- tempdir()
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 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 in response order, with one finite, positive whole number per response, within the trials saved for the selected base image. Repeated and nonconsecutive numbers are allowed. Factors, characters and logicals are rejected; verify imported labels against the generated stimulus filenames before converting them to numeric IDs.
responses	Vector specifying the responses in the same order of the stimuli vector, coded 1 for original stimulus selected and -1 for inverted stimulus selected.
baseimage	String specifying which base image was used. Not the file name, but the key used in the list of base images at time of generating the stimuli.
rdata	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli.
participants	Optional vector specifying one participant ID per trial, with the same length as stimuli and responses. An all-NA input retains the no-grouping behavior. If specified, will compute the requested CIs in two steps: step 1, compute CI for each participant. Step 2, compute final CI by averaging participant CIs. If unspecified, the function defaults to averaging all data in the stimuli and responses vector.

`save_individual_cis`
: Optional boolean specifying whether individual CIs should be save as PNG images when the `participants` parameter is used.

`save_as_png`
: Optional boolean stating whether to additionally save the CI as PNG image.

`filename`
: Optional string to specify a file name for the PNG image.

`targetpath`
: String specifying the directory to save PNGs to. Required when `save_as_png = TRUE` or `save_individual_cis = TRUE`; there is no default path. It is created if it does not exist. Use `tempdir()` if you only want to try the function out.

`antiCI`
: Optional boolean specifying whether antiCI instead of CI should be computed.

`scaling`
: Optional string specifying scaling method: `none`, `constant`, `matched`, or `independent` (default). This scaling applies to the group-level CIs if both individual-level and group-level CIs are being generated.

`scaling_constant`
: Optional number specifying the value used as constant scaling factor for the noise (only works for `scaling='constant'`). This scaling applies to the group-level CIs if both individual-level and group-level CIs are being generated.

`individual_scaling`
: Optional string specifying scaling method for individual CIs: `none`, `constant`, `independent` (default).

`individual_scaling_constant`
: Optional number specifying the value used as constant scaling factor for the noise of individual CIs (only works for `individual_scaling='constant'`).

`zmap`
: Boolean specifying whether a z-map should be created (default: FALSE).

`zmapmethod`
: String specifying the method to create the z-map. Can be: `quick` (default), `t.test`.

`zmapdecoration`
: Optional boolean specifying whether the Z-map should be plotted with margins, text (sigma, threshold) and a scale (default: TRUE).

`sigma`
: Integer specifying the amount of smoothing to apply when generating the z-maps (default: 3).

`threshold`
: Integer specifying the threshold z-score (default: 3). Z-scores below the threshold will not be plotted on the z-map.

`zmaptargetpath`
: String specifying the directory to save z-map PNGs to. Required when `zmap = TRUE`; there is no default path. It is created if it does not exist. Use `tempdir()` if you only want to try the function out.

`n_cores`
: Optional integer specifying the number of CPU cores to use to generate the z-map (default: `detectCores()-1`; 2 under `R CMD check`, per CRAN policy).

`mask`
: Optional 2D matrix that defines the mask to be applied to the CI (0 = masked, 1 = unmasked). May also be a string specifying the path to a grayscale PNG image (black = masked, white = unmasked). Default: NA. Note the matrix convention was documented the wrong way round (as 1 = masked) up to and including 1.1.0; the code has always masked where the matrix is 0, matching the PNG form, and that is what is described here.

`zmappointsize` Integer specifying the text size of the Z-map decoration, in points (default: 12). Passed to `plotZmap()`, which sizes the Z-map image to `img_size`. The decoration needs roughly `12.3 * zmappointsize` pixels on a 72 ppi device and `16.4 * zmappointsize` on a 96 ppi one, so a stimulus set below about 160-200px cannot carry it at the default and `generateCI()` stops with an error naming the minimum for the device in use. Lower this to fit the decoration onto a small Z-map, or set `zmapdecoration = FALSE`.

Details

This function saves the classification image as PNG to a folder and returns the CI. Your choice of scaling matters. The default, `'independent'`, picks the lowest scaling constant that avoids clipping this particular classification image (see `'constant'` scaling below for the formula), so it is not comparable across classification images with different noise ranges.

`'matched'` scaling will match the range of the intensity of the pixels to the range of the base image pixels. This scaling is nonlinear and depends on the range of both base image and noise pattern. It is truly suboptimal, because it shifts the 0 point of the noise (that is, pixels that would not have changed the base image at all before scaling may change the base image after scaling and vice versa). It is however the quick and dirty way to see how the CI noise affects the base image.

For more control, use `'constant'` scaling, where the scaling is independent of the base image and noise range, but where the choice of constant is arbitrary (provided by the user with the `constant` parameter). The noise is then scale as follows: `scaled <- (ci + constant) / (2*constant)`. Note that pixels can take intensity values between 0 and 1. If your scaled noise exceeds those values, a warning will be given. You should pick a higher constant (but do so consistently for different classification images that you want to compare). The higher the constant, the less visible the noise will be in the resulting image.

When creating multiple classification images a good strategy is to find the lowest constant that works for all classification images. This can be automatized using the `autoscale` function.

Value

List of pixel matrix of classification noise only, scaled classification noise only, base image only and combined.

Repeated presentations of the same stimulus

When `participants` is `NA` (the default), repeated presentations of the same stimulus are collapsed before building the CI: each unique stimulus gets equal weight, regardless of how many times it was presented. Where every stimulus was presented the same number of times, this is equivalent to weighting each trial equally and changes nothing. Where repeat counts differ, it changes the estimand: a stimulus presented three times counts the same as one presented once, rather than three times as much.

This is worth knowing for unbalanced designs. If a participant saw some stimuli more often than others – because of an adaptive procedure, a crashed session, or a design choice – the CI reflects the average response per unique stimulus, not per trial. The difference can be substantial: on an 8-trial set with counts 4/2/1/1 the two weightings correlate at 0.77.

computeCumulativeCICorrelation does *not* aggregate and weights each trial equally, so its self-computed final CI diverges from the one this function returns under unequal counts. Pass this function's output as `targetci` to compare against the CI you will actually report.

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 <- tempdir()
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 classification image for 2 images forced choice reverse correlation task. This function exists for backwards compatibility. You can also just use `generateCI()`, which this function wraps.

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 in response order, with one finite, positive whole number per response, within the trials saved for the selected base image. Repeated and nonconsecutive numbers are allowed. Factors, characters and logicals are rejected; verify imported labels against the generated stimulus filenames before converting them to numeric IDs.
responses	Vector specifying the responses in the same order of the stimuli vector, coded 1 for original stimulus selected and -1 for inverted stimulus selected.
baseimage	String specifying which base image was used. Not the file name, but the key used in the list of base images at time of generating the stimuli.
rdata	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli.
save_as_png	Boolean stating whether to additionally save the CI as PNG image.
filename	Optional string to specify a file name for the PNG image.
targetpath	String specifying the directory to save PNGs to. Required when `save_as_png = TRUE`; there is no default path. It is created if it does not exist. Use `tempdir()` if you only want to try the function out.
antiCI	Optional boolean specifying whether antiCI instead of CI should be computed.
scaling	Optional string specifying scaling method: `none`, `constant`, `matched`, or `independent` (default).
constant	Optional number specifying the value used as constant scaling factor for the noise (only works for `scaling='constant'`).

Details

This function saves the classification image as PNG to a folder and returns the CI. Your choice of scaling matters. The default, `'independent'`, picks the lowest scaling constant that avoids clipping this particular classification image (see `'constant'` scaling below for the formula), so it is not comparable across classification images with different noise ranges.

`'matched'` scaling will match the range of the intensity of the pixels to the range of the base image pixels. This scaling is nonlinear and depends on the range of both base image and noise pattern. It is truly suboptimal, because it shifts the 0 point of the noise (that is, pixels that would not have changed the base image at all before scaling may change the base image after scaling and vice versa). It is however the quick and dirty way to see how the CI noise affects the base image.

For more control, use `'constant'` scaling, where the scaling is independent of the base image and noise range, but where the choice of constant is arbitrary (provided by the user with the `constant` parameter). The noise is then scale as follows: `scaled <- (ci + constant) / (2*constant)`. Note that pixels can take intensity values between 0 and 1. If your scaled noise exceeds those values, a warning will be given. You should pick a higher constant (but do so consistently for different classification images that you want to compare). The higher the constant, the less visible the noise will be in the resulting image.

When creating multiple classification images a good strategy is to find the lowest constant that works for all classification images. This can be automatized using the `autoscale` function.

Value

List of pixel matrix of classification noise only, scaled classification noise only, base image only and combined.

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 <- tempdir()
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 containing the response to each trial (1 if participant selected original, -1 if participant selected inverted; this can be changed into a scale).
p	3D patch matrix (generated using `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 with each value specifying the contrast of each patch in noise.
p	3D patch matrix (generated using `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	Integer specifying size of the noise pattern in number of pixels.
nscales	Integer specifying the number of incremental spatial scales. Defaults to 5. Higher numbers will add higher spatial frequency scales.
noise_type	String specifying noise pattern type (defaults to sinusoid; other options: gabor).
sigma	Number specifying the sigma of the Gabor patch if noise_type is set to gabor (defaults to 25).
pre_0.3.0	Boolean specifying whether the noise pattern should be created in a way compatible with older versions of rcicr (< 0.3.0). If you are starting a new project, you should keep this at the default setting (FALSE). There is no reason to set this to TRUE, with the sole exception to recreate behavior of rcicr prior to version 0.3.0.

Value

List with two elements: the 3D noise matrix with size img_size, and an indexing matrix with the same size to easily change contrasts.

Examples

```
generateNoisePattern(256)
```

---

generateReferenceDistribution2IFC

Generates reference distribution

Description

Generates reference distribution of norms for a particular set of task parameters.

Usage

```
generateReferenceDistribution2IFC(
  rdata,
  iter = 10000,
  ncores = default_ncores(),
  response_seed = NULL,
  save_rdata = TRUE,
  baseimage = NULL
)
```

Arguments

rdata	String pointing to .RData file that was created when stimuli were generated. This file contains the contrast parameters of all generated stimuli.
iter	Number of iterations for the simulation (i.e., the number of norms generated with classification images based on random responding).
ncores	Number of CPU cores to use when rebuilding the saved noise (default: detectCores()-1; 2 under R CMD check, per CRAN policy).
response_seed	Optional seed for the simulated random responses. The default (NULL) draws them from the state the stimulus generator left behind, which is the reproducible behaviour described under Reproducibility. Supply a number to obtain an independent draw of the null from the same stimuli.
save_rdata	Boolean specifying whether the reference distribution should be written back into the rdata file (default TRUE). Set to FALSE to compute a distribution without changing what later calls to computeInfoVal2IFC will use – worth doing whenever response_seed is set, so a one-off null does not become the file's permanent reference.
baseimage	Saved base-image label, using the same key as generateCI(). Required when the saved base images have different noise parameters. With a single base or identical parameter matrices, NULL retains the shared reference behavior.

Details

In order to compute the Informational Value metric. Saves its results in the supplied 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 supplied rdata file as reference_norms (alongside reference_norms_seed, recording the response_seed it was generated with), so a later call to computeInfoVal2IFC using the same file can reuse it instead of re-simulating. Independent-base references instead use reference_norms_by_base, as described above.

Reproducibility

With the default response_seed = NULL, the reference distribution is determined by the stimulus .Rdata file and the session's RNGkind. It does not depend on the ambient random number state, and it does not depend on ncores. Two researchers who compute InfoVal from the same stimulus file therefore get the same number, and the same reference distribution, on different machines and in different sessions – provided both sessions use the same RNG kind.

set.seed() keeps whatever kind the session already has rather than restoring one, and no stimulus file records which was in force, so a session that has changed RNGkind() changes the simulated response draws and therefore the null. The saved noise basis and stimulus parameters remain unchanged. To reproduce an earlier reference, use the RNG kind that built it; see https://github.com/rdotsch/rcicr/issues/315.

The noise is reconstructed from the basis and parameters the stimulus file saved, so the base images themselves are never reopened and an archived or moved experiment can still be scored. Responses

are seeded from the state following one parameter matrix's draws at the saved stimulus seed, preserving the historical default response stream.

Pass an explicit `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; the stimuli themselves, and so the noise basis the null is built on, are unaffected.

Independent base images

When saved parameter matrices differ, supply `baseimage` explicitly to say which base's noise to use. Cached distributions are stored in `reference_norms_by_base`, keyed by base label, with `norms` and `response_seed` in each entry. Old unscoped `reference_norms` are neither reused nor overwritten for independent bases. Existing shared-parameter files continue using their unscoped cache and reconstruction.

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 <- tempdir()
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 2 images 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 face file names used as base images for stimuli, e.g. `list(aName = 'baseface.jpg')`. Accepts JPEG and PNG images, recognised by a `.png`, `.jpg` or `.jpeg` extension. Each name labels that base image's stimulus files and indexes the .Rdata file that `generateCI` reads back, so every element must be named, and named uniquely. Each image must be square and exactly `img_size` by `img_size` pixels: rcicr does not resize base images. All of this is checked before any stimuli are generated, and the message names the offending entry.

`n_trials`
: Number specifying how many trials the task will have (function will generate two images for each trial per base image: original and inverted/negative noise).

`img_size`
: Number specifying the number of pixels that the stimulus image will span horizontally and vertically (will be square, so only one integer needed).

`stimulus_path`
: String specifying the 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 path. It is created if it does not exist. Use `tempdir()` if you only want to try the function out.

`label`
: Label to prepend to each file for your convenience.

`use_same_parameters`
: Boolean specifying whether for each base image the same set of parameters is used (TRUE) or a unique set is created for each base image (FALSE).

`seed`
: Integer seeding the random number generator (for reproducibility).

`maximize_baseimage_contrast`
: Boolean specifying whether the pixel values of the base image should be rescaled to maximize its contrast. A base image with no contrast at all — every pixel the same value — has nothing to rescale, and is rejected with an error rather than silently turned into an all-`NaN` base image. Such an image is still usable with `maximize_baseimage_contrast = FALSE`.

`noise_type`
: String specifying noise pattern type (defaults to `sinusoid`; other options: `gabor`).

`nscales`
: Integer specifying the number of incremental spatial scales. Defaults to 5. Higher numbers will add higher spatial frequency scales.

`sigma`
: Number specifying the sigma of the Gabor patch if noise_type is set to `gabor` (defaults to 25).

`ncores`
: Number of CPU cores to use (default: `detectCores()-1`; 2 under `R CMD check`, per CRAN policy).

`return_as_dataframe`
: Boolean specifying whether to return a data frame with the raw noise of the stimuli that were generated (default: FALSE). Data frame columns represent pixel values, data frame rows represent stimuli. The frame holds one noise image per trial, so it is meaningful only under the default `use_same_parameters = TRUE`, where every base image shares a single parameter set and one noise image per trial is all there is. With `use_same_parameters = FALSE` and more than one base image, only the first base image's noise is returned; the frame's shape cannot represent trial x base image. Stimuli are still written to disk for every base image either way, and `save_rdata = TRUE` records the full parameter set, so nothing is lost from the files themselves.

save_as_png Boolean specifying whether to write the stimuli as images to disk (default: TRUE).

save_rdata Boolean specifying whether .RData file with stimulus parameters will be saved (default: TRUE). Note: you always need to save the .RData file so that you can retrieve the stimulus parameters to compute classification images. This function argument exists primarily for internal rcicr use.

Details

Will save the stimuli as PNGs to a folder, including .Rdata file needed for analysis of data after data collection. This .Rdata file contains the parameters that were used to generate each stimulus.

Value

Nothing, everything is saved to files, unless return_as_dataframe is set to TRUE.

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 = tempdir(),
  seed = 1,
  ncores = 1,
  nscales = 1
)
```

---

plotZmap *Plots a Z-map*

Description

Plots a Z-map given a matrix of z-scores that maps onto a specified base image.

Usage

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

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

Arguments

zmap	A matrix containing z-scores that map onto a given base image. zmap and baseimage must have the same dimensions.
bgimage	A matrix containing the grayscale image to use as a background. This should be either the base image or the final CI. If not this argument is not given, only the Z-map will be drawn.
sigma	The sigma of the smoothing that was applied to the CI to create the Z-map.
threshold	Integer specifying the threshold z-score (default: 3). Z-scores below the threshold will not be plotted on the z-map.
mask	Optional. A binary matrix with the same dimensions as zmap: cells that are 0 (or FALSE) are masked, cells that are 1 (or TRUE) are kept. Can also be the filename (as a string) of a black and white PNG image, in which case black (0) is masked and white (1) is kept. This is the same convention `generateCI()` uses for its own `mask` argument, so a mask can be used with both. Note that earlier versions of this documentation stated the opposite for the matrix form; the description here matches what the code does.
decoration	Optional boolean specifying whether the Z-map should be plotted with margins, text (sigma, threshold) and a scale (default: TRUE).
targetpath	String specifying the directory to save the Z-map PNG to. Required: this function exists to write a file, and has no default path. It is created if it does not exist. Use `tempdir()` if you only want to try the function out.
filename	Optional string to specify a file name for the Z-map PNG.
size	Integer specifying the width and height of the PNG image (default: 512).
...	Additional arguments to be passed to `graphics::image`. Only applied when decoration is TRUE.
pointsize	Integer specifying the 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 size is measured in inches and therefore 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 160px and 200px respectively at the default. Below that `plotZmap()` stops and names the minimum for the device in use. Lowering it is what makes a decorated z-map possible on a small device – at the cost of a smaller map, since the margins shrink but the labels still need room. Ignored when `decoration = FALSE`, which has no margins and works at any size.

Details

This function takes in a matrix of z-scores (as returned by generateCI) and an Rdata file containing a base image. It returns a Z-map image in PNG format. Unlisted additional arguments will be passed to `graphics::image`. For example, a different color palette can be specified using the `col` argument. See `graphics::image` for details. Versions up to and including 1.2.3 passed these to the raster package's plot method instead; `col` works the same way in both, but an argument specific to that method will no longer be understood.

Value

Nothing. It writes a Z-map image.

Reproducibility across platforms

The z-scores themselves are ordinary R arithmetic and do not depend on your operating system, and neither do classification images, scaling or informational value. The PNG this function writes is different: it is drawn through a graphics device, and graphics devices differ between platforms both in colour management and in whether they write an alpha channel. The same z-map rendered on Linux and on macOS gives visibly identical figures whose files are not byte-identical – macOS renders a mid-grey background at roughly 0.573 where the cairo device gives 0.502.

So when you are checking that an analysis reproduces, compare the numbers rather than 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()`, which is the only function in the package that opens a graphics device. Every other PNG written by `rcicr` – stimuli, classification images, autoscaled classification images – is written straight from the pixel array by `png::writePNG()` and carries no such dependence.

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)

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