Package 'samplr'

August 1, 2024

Type Package

Title Compare Human Performance to Sampling Algorithms

Version 1.0.0

Maintainer Lucas Castillo <lucas.castillo-marti@warwick.ac.uk>

Description Understand human performance from the perspective of sampling, both looking at how people generate samples and how people use the samples they have generated. A longer overview and other resources can be found at <https://sampling.warwick.ac.uk>.

License CC BY 4.0

Imports Rcpp (>= 1.0.6), ggplot2, latex2exp, pracma, stats, lme4, Rdpack, R6, graphics

Suggests knitr, rmarkdown, testthat (>= 3.0.0), vdiffr, bench, dplyr, tidyr, magrittr, mvtnorm, xml2, samplrData

LinkingTo Rcpp, RcppArmadillo, RcppDist, testthat,

RdMacros Rdpack

RoxygenNote 7.3.1

Encoding UTF-8

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/lucas-castillo/samplr

BugReports https://github.com/lucas-castillo/samplr/issues

Depends R (>= 2.10)

NeedsCompilation yes

Author Lucas Castillo [aut, cre, cph] (<https://orcid.org/0000-0003-0274-0777>), Yun-Xiao Li [aut, cph] (<https://orcid.org/0000-0002-3509-6618>), Adam N Sanborn [aut, cph] (<https://orcid.org/0000-0003-0442-4372>), European Research Council (ERC) [fnd]

Repository CRAN

Date/Publication 2024-08-01 09:00:02 UTC

Contents

`Bayesian_Sampler` *Bayesian Sampler Model*

Description

As described in (Zhu et al. 2020). Vectors can be provided for each parameter, allowing multiple estimates at once.

Usage

```
Bayesian_Sampler(
  a_and_b,
  b_and_not_a,
  a_and_not_b,
  not_a_and_not_b,
  beta,
  N,
  N2 = NULL
)
```

Arguments

`a_and_b, b_and_not_a, a_and_not_b, not_a_and_not_b`	True probabilites for the conjuctions and disjunctions of A and B. Must add to 1.
`beta`	Prior parameter.
`N`	Number of samples drawn
`N2`	Optional. Number of samples drawn for conjunctions and disjunctions. (called N' in the paper). If not given, it will default to N2=N. Must be equal or smaller than N.

Value

Named list with predicted probabilities for every possible combination of A and B.

Examples

```
Bayesian_Sampler(
    a_and_b = c(.4, .25),
    b_and_not_a = c(.4,  .25),
    a_and_not_b = c(.1, .25),
    not_a_and_not_b = c(.1, .25),
    beta = 1,
    N <- c(10, 12),
    N2 <- c(10, 10)
)
```

`calc_all` *Diagnostics Wrapper*

Description

Calculates all diagnostic functions in the samplr package for a given chain. Optionally, plots them.

Usage

```
calc_all(chain, plot = TRUE, acf.alpha = 0.05, acf.lag.max = 100)
```

Arguments

`chain`	Vector of n length, where n is the number of trials or sampler iterations
`plot`	Boolean. Whether to additionally plot the diagnostics.
`acf.alpha, acf.lag.max`	Additional parameters to calc_autocorr.

Value

A list with all diagnostic calculations (a list of lists); and optionally a grid of plots.

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
diagnostics <- calc_all(chain1[[1]])
names(diagnostics)
```

---

calc_autocorr *Autocorrelation Calculator*

Description

Calculates the autocorrelation of a given sequence, or of the size of the steps (returns).

Usage

```
calc_autocorr(chain, change = TRUE, alpha = 0.05, lag.max = 100, plot = FALSE)
```

Arguments

`chain`	Vector of n length, where n is the number of trials or sampler iterations
`change`	Boolean. If true, plot the autocorrelation of the change series. If false, plot the autocorrelation of the given chain.
`alpha`	Measure of Type I error - defaults to .05
`lag.max`	Length of the x axis. How far to examine the lags.
`plot`	Boolean. Whether to additionally plot the result.

Details

Markets display no significant autocorrelations in the returns of a given asset.

Value

A vector with the standard deviations at each lag

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
calc_autocorr(chain1[[1]], plot=TRUE)
```

`calc_levy` *Levy Flights Calculator*

Description

This function analyses if the length of the jumps the sampler is making (l) belongs to a Levy probability density distribution, $P(l) \approx l^{-\mu}$.

Usage

```
calc_levy(chain, plot = FALSE)
```

Arguments

`chain`	Matrix of n x d dimensions, n = iterations, d = dimensions.
`plot`	Boolean. plot Boolean. Whether to also plot the distance-frequency relationship.

Details

Values of $\mu \approx 2$ have been used to describe foraging in animals, and produce the most effective foraging (Viswanathan et al. 1999). See Zhu et al. (2018) for a comparison of Levy Flight and PSD measures for different samplers in multimodal representations.

Value

If plot is true, it returns a simple plot with the log absolute difference in estimates and their frequency, as well as an estimate for the μ parameter. If false it returns a list with what's required to make the plot.

References

Viswanathan GM, Buldyrev SV, Havlin S, Da Luz MGE, Raposo EP, Stanley HE (1999). "Optimizing the Success of Random Searches." *Nature*, **401**(6756), 911–914. doi:10.1038/44831.

Zhu J, Sanborn AN, Chater N (2018). "Mental Sampling in Multimodal Representations." *Advances in Neural Information Processing Systems*, **31**, 5748–5759.

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
calc_levy(chain1[[1]], plot=TRUE)
```

`calc_PSD` *Power Spectral Density Calculator*

Description

This function estimates the log power spectral density against the log frequency, and calculates a slope α.

Usage

```
calc_PSD(chain, plot = FALSE)
```

Arguments

`chain`	Matrix of n x d dimensions, n = iterations, d = dimensions sequence
`plot`	Boolean. Whether to return a plot or the elements used to make it.

Details

A number of studies have reported that cognitive activities contain a long-range slowly decaying autocorrelation. In the frequency domain, this is expressed as $S(f) \sim 1/f^{-\alpha}$, with f being frequency, $S(f)$ being spectral power, and $\alpha \epsilon [0.5, 1.5]$ is considered $1/f$ scaling. See See Zhu et al. (2018) for a comparison of Levy Flight and PSD measures for different samplers in multimodal representations.

Value

Returns a list with log frequencies, log PSDs, and slope and intercept estimates.

References

Zhu J, Sanborn AN, Chater N (2018). “Mental Sampling in Multimodal Representations.” *Advances in Neural Information Processing Systems*, **31**, 5748–5759.

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
calc_PSD(chain1[[1]], plot= TRUE)
```

`calc_qqplot` *QQ-Plot Calculator*

Description

Estimates values for a QQ plot of Empirical values against Theoretical values from a normal distribution, for either the chain points or the distances between successive points. Optionally, returns a plot as well as the values.

Usage

```
calc_qqplot(chain, change = TRUE, plot = FALSE)
```

Arguments

`chain`	Vector of n length, where n is the number of trials or sampler iterations
`change`	Boolean. If false, it calculates a qqplot of the given chain. If true, it creates a chain of step sizes.
`plot`	Boolean. Whether to plot the QQ plot or just return the values.

Value

A list with the theoretical and empirical quantiles, and the intercept and slope of the line connecting the points

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
calc_qqplot(chain1[[1]], plot = TRUE)
```

`calc_sigma_scaling` *Sigma Scaling Calculator*

Description

Calculates the sigma scaling of the chain, and optionally plots the result.

Usage

```
calc_sigma_scaling(chain, plot = FALSE)
```

Arguments

`chain`	Vector of n length, where n is the number of trials or sampler iterations
`plot`	Boolean. Whether to additionally plot the result.

Details

Sigma scaling is defined as the slope of the regression connecting log time lags and the standard deviation of value changes across time lags. Markets show values of 0.5.

Value

A list containing the vector of possible lags, the sd of the distances at each lag, their log10 counterparts, and the calculated intercept and slope.

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
calc_sigma_scaling(chain1[[1]], plot = TRUE)
```

`Mean_Variance` *Mean Variance Estimates*

Description

Estimates number of samples and prior parameters of the Bayesian Sampler using the Mean/Variance relationship as shown by (Sundh et al. 2023). For consistency with the Bayesian Sampler function we call beta the prior parameter, and b0 and b1 slope and intercept respectively.

Usage

```
Mean_Variance(rawData, idCol)
```

Arguments

`rawData`	Dataframe with the following column variables for N repetitions of each unique query: participant ID ('id'), response query 1, response query 2, ... , response query N
`idCol`	Name of the 'ID' column.

Value

A dataframe with values for the intercept (b0) and slope (b1) of the estimated regression, as well as estimates for N, d, and beta (termed b in the paper) for each participant.

Examples

```
library(dplyr)
library(tidyr)
library(magrittr)
library(samplrData)
data <- sundh2023.meanvariance.e3 %>%
  group_by(ID, querydetail) %>%
  mutate(iteration = LETTERS[1:n()]) %>%
  pivot_wider(id_cols = c(ID, querydetail),
      values_from = estimate, names_from = iteration) %>%
  mutate(across(where(is.numeric), \(x){x/100})) %>%
  ungroup %>%
  select(-querydetail)
head(data)
head(Mean_Variance(data, "ID"))
```

plot_2d_density *Density Plotter*

Description

Plots a 2D map of the density of a distribution. If plot = FALSE, returns a dataframe with the density for each cell in the grid

Usage

```
plot_2d_density(
  start,
  size,
  cellsPerRow = 50,
  names = NULL,
  params = NULL,
  weights = NULL,
  customDensity = NULL,
  plot = TRUE
)
```

Arguments

`start`	Vector c(x, y) with the coordinates of the bottom-left corner of the map.
`size`	Distance covered by the map. In other words, the top-right corner of the map has coordinates c(x + size, y + size)
`cellsPerRow`	Number of cells to plot in every row. The higher, the more resolution
`names`	Name of the distribution from which to sample from.
`params`	Distribution parameters.
`weights`	Distribution weights (if it's a mix of distributions)

`customDensity`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.
`plot`	Whether to return a plot or a dataframe with the density in each coordinate

Value

Density Plot or dataframe

Examples

```
# plot supported distribution
plot_2d_density(
c(-5, -5), 10, cellsPerRow = 100, names = c("mvnorm", "mvnorm"),
params = list(list(c(-2,1), diag(2)), list(c(2,1), diag(2)))
)

# plot custom distribution
customDensity_r <- function(x){
    if (x[1] > 0 && x[1] < 3 && x[2] < 0 && x[2] > -3){
        return (1)
    } else {
        return (0)
    }
}
plot_2d_density(start = c(0,-4), size = 5, customDensity = customDensity_r)
```

`plot_series` *Series Plotter*

Description

Plots the value of a one-dimensional series against the iteration where it occurred. Useful to see the general pattern of the chain (white noise, random walk, volatility clustering)

Usage

```
plot_series(chain, change = FALSE)
```

Arguments

`chain`	Vector of n length, where n is the number of trials or sampler iterations
`change`	Boolean. Whether to plot the series of values or the series of changes between values.

Value

A series plot

Examples

```
set.seed(1)
chain1 <- sampler_mh(1, "norm", c(0,1), diag(1))
plot_series(chain1[[1]])
```

sampler_hmc *Hamiltonian Monte-Carlo Sampler (HMC)*

Description

Hamiltonian Monte-Carlo, also called Hybrid Monte Carlo, is a sampling algorithm that uses Hamiltonian Dynamics to approximate a posterior distribution. Unlike MH and MC3, HMC uses not only the current position, but also a sense of momentum, to draw future samples. An introduction to HMC can be read in Betancourt (2018).

Usage

```
sampler_hmc(
  start,
  distr_name = NULL,
  distr_params = NULL,
  epsilon = 0.5,
  L = 10,
  iterations = 1024,
  weights = NULL,
  custom_density = NULL
)
```

Arguments

`start`	Vector. Starting position of the sampler.
`distr_name`	Name of the distribution from which to sample from.
`distr_params`	Distribution parameters.
`epsilon`	Size of the leapfrog step
`L`	Number of leapfrog steps per iteration
`iterations`	Number of iterations of the sampler.
`weights`	If using a mixture distribution, the weights given to each constituent distribution. If none given, it defaults to equal weights for all distributions.
`custom_density`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.

Details

This implementations assumes that the momentum is drawn from a normal distribution with mean 0 and identity covariance matrix (p ~ N (0, I)). Hamiltonian Monte Carlo does not support discrete distributions.

This algorithm has been used to model human data in Aitchison and Lengyel (2016), Castillo et al. (2024) and Zhu et al. (2022) among others.

Value

A named list containing

1. Samples: the history of visited places (an n x d matrix, n = iterations; d = dimensions)
2. Momentums: the history of momentum values (an n x d matrix, n = iterations; d = dimensions). Nothing is proposed in the first iteration (the first iteration is the start value) and so the first row is NA
3. Acceptance Ratio: The proportion of proposals that were accepted.

References

Aitchison L, Lengyel M (2016). "The Hamiltonian Brain: Efficient Probabilistic Inference with Excitatory-Inhibitory Neural Circuit Dynamics." *PLOS Computational Biology*, **12**(12), e1005186. doi:10.1371/journal.pcbi.1005186.

Betancourt M (2018). "A Conceptual Introduction to Hamiltonian Monte Carlo." `http://arxiv.org/abs/1701.02434`.

Castillo L, León-Villagrá P, Chater N, Sanborn A (2024). "Explaining the Flaws in Human Random Generation as Local Sampling with Momentum." *PLOS Computational Biology*, **20**(1), 1–24. doi:10.1371/journal.pcbi.1011739.

Zhu J, León-Villagrá P, Chater N, Sanborn AN (2022). "Understanding the Structure of Cognitive Noise." *PLoS Computational Biology*, **18**(8), e1010312. doi:10.1371/journal.pcbi.1010312.

Examples

```
result <- sampler_hmc(
    distr_name = "norm", distr_params = c(0,1),
    start = 1, epsilon = .01, L = 100
    )
cold_chain <- result$Samples
```

sampler_mc3 *Metropolis-coupled MCMC sampler (MC3)*

Description

This sampler is a variant of MH in which multiple parallel chains are run at different temperatures. The chains stochastically swap positions which allows the coldest chain to visit regions far from its starting point (unlike in MH). Because of this, an MC3 sampler can explore far-off regions, whereas an MH sampler may become stuck in a particular point of high density.

Usage

```
sampler_mc3(
  start,
  distr_name = NULL,
  distr_params = NULL,
  sigma_prop = NULL,
  nChains = 6,
  delta_T = 4,
  swap_all = TRUE,
  iterations = 1024L,
  weights = NULL,
  custom_density = NULL,
  alpha = 0
)
```

Arguments

`start`	Either a vector or a matrix. If it is a vector, it will be the starting point of all the chains (with length = number of dimensions). If it's a matrix, every row will be the starting point of one chain (and so it must have as many rows as nChains, and as many columns as number of dimensions in the space).
`distr_name`	Name of the distribution from which to sample from.
`distr_params`	Distribution parameters.
`sigma_prop`	Covariance matrix of the proposal distribution. If sampling in 1D space, it can be instead a number.
`nChains`	Number of chains to run.
`delta_T`	numeric, >1. Temperature increment parameter. The bigger this number, the steeper the increase in temperature between the cold chain and the next chain
`swap_all`	Boolean. If true, every iteration attempts floor(nChains / 2) swaps. If false, only one swap per iteration.
`iterations`	Number of iterations of the sampler.
`weights`	If using a mixture distribution, the weights given to each constituent distribution. If none given, it defaults to equal weights for all distributions.

`custom_density`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.
`alpha`	autocorrelation of proposals parameter, from -1 to 1, with 0 being independent proposals

Details

This algorithm has been used to model human data in Castillo et al. (2024), Zhu et al. (2022) and Zhu et al. (2018) among others.

Value

A named list containing

1. Samples: the history of visited places (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain')
2. Proposals: the history of proposed places (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain'). Nothing is proposed in the first iteration (the first iteration is the start value) and so the first row is NA
3. Acceptance Ratio: The proportion of proposals that were accepted (for each chain).
4. Beta Values: The set of temperatures used in each chain
5. Swap History: the history of chain swaps
6. Swap Acceptance Ratio: The ratio of swap acceptances

References

Castillo L, León-Villagrá P, Chater N, Sanborn A (2024). "Explaining the Flaws in Human Random Generation as Local Sampling with Momentum." *PLOS Computational Biology*, **20**(1), 1–24. doi:10.1371/journal.pcbi.1011739.

Zhu J, León-Villagrá P, Chater N, Sanborn AN (2022). "Understanding the Structure of Cognitive Noise." *PLoS Computational Biology*, **18**(8), e1010312. doi:10.1371/journal.pcbi.1010312.

Zhu J, Sanborn AN, Chater N (2018). "Mental Sampling in Multimodal Representations." *Advances in Neural Information Processing Systems*, **31**, 5748–5759.

Examples

```
# Sample from a normal distribution
result <- sampler_mc3(
    distr_name = "norm", distr_params = c(0,1),
    start = 1, sigma_prop = diag(1)
)
cold_chain <- result$Samples[,,1]
```

sampler_mchmc *Metropolis-Coupled Hamiltonian Monte Carlo (MCHMC)*

Description

Metropolis-Coupled version of HMC, i.e. running multiple chains at different temperatures which stochastically swap positions.

Usage

```
sampler_mchmc(
  start,
  distr_name = NULL,
  distr_params = NULL,
  epsilon = 0.5,
  L = 10,
  nChains = 6,
  delta_T = 4,
  swap_all = TRUE,
  iterations = 1024L,
  weights = NULL,
  custom_density = NULL
)
```

Arguments

`start`	Vector. Starting position of the sampler.
`distr_name`	Name of the distribution from which to sample from.
`distr_params`	Distribution parameters.
`epsilon`	Size of the leapfrog step
`L`	Number of leapfrog steps per iteration
`nChains`	Number of chains to run.
`delta_T`	numeric, >1. Temperature increment parameter. The bigger this number, the steeper the increase in temperature between the cold chain and the next chain
`swap_all`	Boolean. If true, every iteration attempts floor(nChains / 2) swaps. If false, only one swap per iteration.
`iterations`	Number of iterations of the sampler.
`weights`	If using a mixture distribution, the weights given to each constituent distribution. If none given, it defaults to equal weights for all distributions.
`custom_density`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.

Details

Metropolis-Coupled HMC does not support discrete distributions.

This algorithm has been used to model human data in Castillo et al. (2024).

Value

A named list containing

1. Samples: the history of visited places (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain')
2. Momentums: the history of momentum values (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain'). Nothing is proposed in the first iteration (the first iteration is the start value) and so the first row is NA
3. Acceptance Ratio: The proportion of proposals that were accepted (for each chain).
4. Beta Values: The set of temperatures used in each chain
5. Swap History: the history of chain swaps
6. Swap Acceptance Ratio: The ratio of swap acceptances

References

Castillo L, León-Villagrá P, Chater N, Sanborn A (2024). "Explaining the Flaws in Human Random Generation as Local Sampling with Momentum." *PLOS Computational Biology*, **20**(1), 1–24. doi:10.1371/journal.pcbi.1011739.

Examples

```
result <- sampler_mchmc(
    distr_name = "norm", distr_params = c(0,1),
    start = 1, epsilon = .01, L = 100
)
cold_chain <- result$Samples[,,1]
```

`sampler_mcrec` *Metropolis-Coupled Recycled-Momentum HMC Sampler (MCREC)*

Description

Metropolis-Coupled version of Recycled-Momentum HMC, i.e. running multiple chains at different temperatures which stochastically swap positions.

Usage

```
sampler_mcrec(
  start,
  distr_name = NULL,
  distr_params = NULL,
  epsilon = 0.5,
  L = 10,
  alpha = 0.1,
  nChains = 6,
  delta_T = 4,
  swap_all = TRUE,
  iterations = 1024L,
  weights = NULL,
  custom_density = NULL
)
```

Arguments

`start`	Vector. Starting position of the sampler.
`distr_name`	Name of the distribution from which to sample from.
`distr_params`	Distribution parameters.
`epsilon`	Size of the leapfrog step
`L`	Number of leapfrog steps per iteration
`alpha`	Recycling factor, from -1 to 1 (see Details).
`nChains`	Number of chains to run.
`delta_T`	numeric, >1. Temperature increment parameter. The bigger this number, the steeper the increase in temperature between the cold chain and the next chain
`swap_all`	Boolean. If true, every iteration attempts floor(nChains / 2) swaps. If false, only one swap per iteration.
`iterations`	Number of iterations of the sampler.
`weights`	If using a mixture distribution, the weights given to each constituent distribution. If none given, it defaults to equal weights for all distributions.
`custom_density`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.

Details

Metropolis-Coupled Recycled-Momentum HMC does not support discrete distributions.

This algorithm has been used to model human data in Castillo et al. (2024).

Value

A named list containing

1. Samples: the history of visited places (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain')

2. Momentums: the history of momentum values (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain'). Nothing is proposed in the first iteration (the first iteration is the start value) and so the first row is NA
3. Acceptance Ratio: The proportion of proposals that were accepted (for each chain).
4. Beta Values: The set of temperatures used in each chain
5. Swap History: the history of chain swaps
6. Swap Acceptance Ratio: The ratio of swap acceptances

References

Castillo L, León-Villagrá P, Chater N, Sanborn A (2024). "Explaining the Flaws in Human Random Generation as Local Sampling with Momentum." *PLOS Computational Biology*, **20**(1), 1–24. doi:10.1371/journal.pcbi.1011739.

Examples

```
result <- sampler_mcrec(
    distr_name = "norm", distr_params = c(0,1),
    start = 1, epsilon = .01, L = 100
)
cold_chain <- result$Samples[,,1]
```

sampler_mh *Metropolis-Hastings (MH) Sampler*

Description

This sampler navigates the proposal distribution following a random walk. At each step, it generates a new proposal from a proposal distribution (in this case a Gaussian centered at the current position) and chooses to accept it or reject it following the Metropolis-Hastings rule: it accepts it if the density of the posterior distribution at the proposed point is higher than at the current point. If the current position is denser, it still may accept the proposal with probability `proposal_density / current_density`.

Usage

```
sampler_mh(
  start,
  distr_name = NULL,
  distr_params = NULL,
  sigma_prop = NULL,
  iterations = 1024L,
  weights = NULL,
  custom_density = NULL,
  alpha = 0
)
```

Arguments

`start`	Vector. Starting position of the sampler.
`distr_name`	Name of the distribution from which to sample from.
`distr_params`	Distribution parameters.
`sigma_prop`	Covariance matrix of the proposal distribution. If sampling in 1D space, it can be instead a number.
`iterations`	Number of iterations of the sampler.
`weights`	If using a mixture distribution, the weights given to each constituent distribution. If none given, it defaults to equal weights for all distributions.
`custom_density`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.
`alpha`	autocorrelation of proposals parameter, from -1 to 1, with 0 being independent proposals

Details

As mentioned, the proposal distribution is a Normal distribution. Its mean is the current position, and its variance is equal to the `sigma_prop` parameter, which defaults to the identity matrix if not specified.

This algorithm has been used to model human data in many places (e.g. Castillo et al. 2024; Dasgupta et al. 2017; Lieder et al. 2018; Zhu et al. 2022).

Value

A named list containing

1. Samples: the history of visited places (an n x d matrix, n = iterations; d = dimensions)
2. Proposals: the history of proposed places (an n x d matrix, n = iterations; d = dimensions). Nothing is proposed in the first iteration (the first iteration is the start value) and so the first row is NA
3. Acceptance Ratio: The proportion of proposals that were accepted.

References

Castillo L, León-Villagrá P, Chater N, Sanborn A (2024). "Explaining the Flaws in Human Random Generation as Local Sampling with Momentum." *PLOS Computational Biology*, **20**(1), 1–24. doi:10.1371/journal.pcbi.1011739.

Dasgupta I, Schulz E, Gershman SJ (2017). "Where Do Hypotheses Come From?" *Cognitive Psychology*, **96**, 1–25. doi:10.1016/j.cogpsych.2017.05.001.

Lieder F, Griffiths TL, M. Huys QJ, Goodman ND (2018). "The Anchoring Bias Reflects Rational Use of Cognitive Resources." *Psychonomic Bulletin & Review*, **25**(1), 322–349. doi:10.3758/s1342301712868.

Zhu J, León-Villagrá P, Chater N, Sanborn AN (2022). "Understanding the Structure of Cognitive Noise." *PLoS Computational Biology*, **18**(8), e1010312. doi:10.1371/journal.pcbi.1010312.

Examples

```
# Sample from a normal distribution
result <- sampler_mh(
        distr_name = "norm", distr_params = c(0,1),
        start = 1, sigma_prop = diag(1)
        )
cold_chain <- result$Samples
```

sampler_rec *Recycled-Momentum HMC Sampler (REC)*

Description

Recycled-Momentum HMC is a sampling algorithm that uses Hamiltonian Dynamics to approximate a posterior distribution. Unlike in standard HMC, proposals are autocorrelated, as the momentum of the current trajectory is not independent of the last trajectory, but is instead updated by a parameter alpha (see Details).

Usage

```
sampler_rec(
  start,
  distr_name = NULL,
  distr_params = NULL,
  epsilon = 0.5,
  L = 10,
  alpha = 0.1,
  iterations = 1024L,
  weights = NULL,
  custom_density = NULL
)
```

Arguments

`start`	Vector. Starting position of the sampler.
`distr_name`	Name of the distribution from which to sample from.
`distr_params`	Distribution parameters.
`epsilon`	Size of the leapfrog step
`L`	Number of leapfrog steps per iteration
`alpha`	Recycling factor, from -1 to 1 (see Details).
`iterations`	Number of iterations of the sampler.
`weights`	If using a mixture distribution, the weights given to each constituent distribution. If none given, it defaults to equal weights for all distributions.
`custom_density`	Instead of providing names, params and weights, the user may prefer to provide a custom density function.

Details

While in HMC the momentum in each iteration is an independent draw,, here the momentum of the last utterance p^{n-1} is also involved. In each iteration, the momentum p is obtained as follows

$$p \leftarrow \alpha \times p^{n-1} + (1-\alpha^2)^{\frac{1}{2}} \times v$$

; where $v \sim N(0, I)$.

Recycled-Momentum HMC does not support discrete distributions.

This algorithm has been used to model human data in Castillo et al. (2024)

Value

A named list containing

1. Samples: the history of visited places (an n x d x c array, n = iterations; d = dimensions; c = chain index, with c==1 being the 'cold chain')
2. Momentums: the history of momentum values (an n x d matrix, n = iterations; d = dimensions). Nothing is proposed in the first iteration (the first iteration is the start value) and so the first row is NA
3. Acceptance Ratio: The proportion of proposals that were accepted (for each chain).

References

Castillo L, León-Villagrá P, Chater N, Sanborn A (2024). "Explaining the Flaws in Human Random Generation as Local Sampling with Momentum." *PLOS Computational Biology*, **20**(1), 1–24. doi:10.1371/journal.pcbi.1011739.

Examples

```
result <- sampler_rec(
    distr_name = "norm", distr_params = c(0,1),
    start = 1, epsilon = .01, L = 100
)
cold_chain <- result$Samples
```

`Zhu23ABS` *Auto-correlated Bayesian Sampler by Zhu (2023)*

Description

This Auto-correlated Bayesian Sampler model (ABS, Zhu et al. 2024) is developed by Zhu.

Super class

`samplr::CoreABS` -> `Zhu23ABS`

Public fields

`width` the standard deviation of the proposal distribution for MC3.

`lambda` the rate parameter of the Erlang distribution for decision time.

Methods

Public methods:

- `Zhu23ABS$new()`
- `Zhu23ABS$simulate()`
- `Zhu23ABS$confidence_interval()`
- `Zhu23ABS$reset_sim_results()`
- `Zhu23ABS$clone()`

Method `new()`**:** Create a new 'Zhu23ABS' object.

Usage:

```
Zhu23ABS$new(
  width,
  n_chains,
  nd_time,
  s_nd_time,
  lambda,
  distr_name = NULL,
  distr_params = NULL,
  custom_distr = NULL,
  custom_start = NULL
)
```

Arguments:

`width` a numeric value of the standard deviation of the proposal distribution for MC3.

`n_chains` an integer of the number of chains for the sampler.

`nd_time` a numeric value of the non-decision time (in seconds). When `s_nd_time` is not 0, `nd_time` represents the lower bound of the non-decision time.

`s_nd_time` a numeric value of the inter-trial-variability of the non-decision time (in seconds).

`lambda` a numeric value of the rate parameter of the Erlang distribution for decision time.

`distr_name` a character string indicating the type of the posterior hypothesis distribution.

`distr_params` a numeric vector of the additional parameters for the posterior hypothesis distribution.

`custom_distr` a list of functions that define the posterior hypothesis distribution.

`custom_start` a numeric value of the starting point if "custom_distr" is provided.

Returns: A new 'Zhu23ABS' object.

Examples:

```
zhuabs <- Zhu23ABS$new(
    width = 1, n_chains = 5, nd_time = 0.3, s_nd_time = 0.5,
    lambda = 10, distr_name = 'norm', distr_params = 1
)
```

Method `simulate()`**:** Simulate the ABS model.

Usage:

```
Zhu23ABS$simulate(stopping_rule, start_point = NA, ...)
```

Arguments:

`stopping_rule` a character string indicating the stopping rule of ABS to be applied. Possible values are `"fixed"` and `"relative"`. See also `Details`.

`start_point` a numeric vector setting the start point of each trial for the sampler. By default, it's set to `NA`, indicating that the starting point of the first trial is a random point from the posterior of hypotheses, and the starting points of subsequent trials are set to the last sample of the previous trial. For more detailed information, please refer to the vignette "Simulations of the Autocorrelated Bayesian Sampler".

`...` further arguments passed to the ABS model, see also `Details`.

Details: The ABS model has two types of stopping rules: fixed and relative. The fixed stopping rule means that a fixed number of samples are drawn to complete the tasks such as estimations and confidence intervals. This rule applies to tasks such as estimation tasks. On the other hand, the relative stopping rule means that the model counts the difference in evidence between the two hypotheses, and terminates the sampling process whenever the accumulated difference exceeds a threshold. This rule applies to tasks such as two-alternative force choice tasks.

When the `stopping rule` is `"fixed"`, the following arguments are required:

- `n_sample` an integer of the fixed number of samples for each trial.
- `trial_stim` a numeric vector of the stimulus of each trial.

When the `stopping rule` is `"relative"`, the following arguments are required:

- `delta` an integer of the relative difference between the number of samples supporting each hypothesis.
- `dec_bdry` a numeric value of the decision boundary that separates the posterior hypothesis distribution.
- `discrim` a numeric value of the stimuli discriminability.
- `trial_stim` a factor that indicates the stimuli of each trial. It only consists of either one level or two levels. By definition, level 1 represents the stimulus below the decision boundary, while level 2 represents the stimulus above the decision boundary.
- `prior_on_resp` a numeric vector for the Beta prior on responses. Defaults to `c(1,1)` representing the distribution `Beta(1,1)`.
- `prior_depend` a boolean variable that control whether the prior on responses changes regarding the last stimulus. Defaults to `TRUE`. Please refer to the vignette for more information.
- `max_iterations` an integer of the maximum length of the MC3 sampler. Defaults to 1000. The program will stop the sampling process after the length of the sampling sequence reaches to this limitation.

No values will be return after running this method, but the field `sim_results` will be updated instead. If the stopping rule is "fixed", `simulation_results` will be a data frame with five columns:

1. trial: The index of trials;
2. samples: The samples of ABS sampler for the trial;
3. stimulus: The stimuli of the experiment;
4. rt: The response time;

5. point_est: The response of point estimation;

On the other hand, if the stopping rule is "relative", `sim_results` will be a data frame with seven columns:

1. trial: The index of trials;
2. samples: The samples of ABS sampler for the trial;
3. response: The response predicted by ABS;
4. stimulus: The stimuli of the experiment;
5. accuracy: Whether the response is the same as the feedback. 0 represents error, and 1 represents correct;
6. rt: The response time, including both the non-decision and the decision time;
7. confidence: The confidence of the response.

Examples:

```
trial_stim <- round(runif(5, 10, 50))
zhuabs$simulate(stopping_rule='fixed', n_sample = 5, trial_stim = trial_stim)
zhuabs$sim_results

zhuabs$reset_sim_results()
trial_stim <- factor(sample(c('left', 'right'), 5, TRUE))
zhuabs$simulate(stopping_rule='relative',
   delta = 4, dec_bdry = 0,
   discrim = 1, trial_stim = trial_stim
)
zhuabs$sim_results
```

Method `confidence_interval()`**:** This function calculates the confidence interval of the `simulate` method's results when the "fixed" stopping rule was used.

Usage:

```
Zhu23ABS$confidence_interval(conf_level)
```

Arguments:

`conf_level` the required confidence level.

Details: No values will be returned by this method. Instead, two new columns will be added to the `sim_results`:

1. conf_interval_l: The lower bound of the confidence interval with the given level;
2. conf_interval_u: The upper bound of the confidence interval with the given level;

Examples:

```
zhuabs$confidence_interval(conf_level = 0.9)
```

Method `reset_sim_results()`**:** This function is for resetting the `sim_results` to run new simulations.

Usage:

```
Zhu23ABS$reset_sim_results()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Zhu23ABS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Zhu J, Sundh J, Spicer J, Chater N, Sanborn AN (2024). "The Autocorrelated Bayesian Sampler: A Rational Process for Probability Judgments, Estimates, Confidence Intervals, Choices, Confidence Judgments, and Response Times." *Psychological Review*, **131**(2), 456–493. doi:10.1037/rev0000427.

Examples

```
## ------------------------------------------------
## Method `Zhu23ABS$new`
## ------------------------------------------------

zhuabs <- Zhu23ABS$new(
    width = 1, n_chains = 5, nd_time = 0.3, s_nd_time = 0.5,
    lambda = 10, distr_name = 'norm', distr_params = 1
)


## ------------------------------------------------
## Method `Zhu23ABS$simulate`
## ------------------------------------------------


trial_stim <- round(runif(5, 10, 50))
zhuabs$simulate(stopping_rule='fixed', n_sample = 5, trial_stim = trial_stim)
zhuabs$sim_results

zhuabs$reset_sim_results()
trial_stim <- factor(sample(c('left', 'right'), 5, TRUE))
zhuabs$simulate(stopping_rule='relative',
   delta = 4, dec_bdry = 0,
   discrim = 1, trial_stim = trial_stim
)
zhuabs$sim_results


## ------------------------------------------------
## Method `Zhu23ABS$confidence_interval`
## ------------------------------------------------

zhuabs$confidence_interval(conf_level = 0.9)
```

Z_identities *Z Identities*

Description

Calculates identities Z1 to Z18 as defined in (Costello and Watts 2016; Zhu et al. 2020). Probability theory predicts that these will all equal 0.

Usage

```
Z_identities(
  a = NULL,
  b = NULL,
  a_and_b = NULL,
  a_or_b = NULL,
  a_given_b = NULL,
  b_given_a = NULL,
  a_given_not_b = NULL,
  b_given_not_a = NULL,
  a_and_not_b = NULL,
  b_and_not_a = NULL,
  not_a = NULL,
  not_b = NULL
)
```

Arguments

`a`, `b`, `a_and_b`, `a_or_b`, `a_given_b`, `b_given_a`, `a_given_not_b`, `b_given_not_a`, `a_and_not_b`, `b_and_not_a`	Probability estimates given by participants
`not_a`, `not_b`	Probability estimates given by participants. If not given, they'll default to 1-a and 1-b respectively

Details

If some of the probability estimates are not given, calculation will proceed and equalities that cannot be calculated will be coded as NA.

Value

Dataframe with identities Z1 to Z18

Examples

```
Z_identities(
 a=.5,
 b=.1,
 a_and_b=.05,
 a_or_b=.55,
 a_given_b=.5,
 b_given_a=.1,
 a_given_not_b=.5,
 b_given_not_a=.1,
 a_and_not_b=.45,
 b_and_not_a=.05,
 )
#Get identities for a set of participants
library(magrittr)
library(dplyr)
library(tidyr)
data.frame(
 ID = LETTERS[1:20],
 a=runif(20),
 b=runif(20),
 a_and_b=runif(20),
 a_or_b=runif(20),
 a_given_b=runif(20),
 b_given_a=runif(20),
 a_given_not_b=runif(20),
 b_given_not_a=runif(20),
 a_and_not_b=runif(20),
 b_and_not_a=runif(20),
 not_a=runif(20),
 not_b=runif(20)
) %>%
 group_by(ID) %>%
 do(
   Z_identities(
     .$a,
     .$b,
     .$a_and_b,
     .$a_or_b,
     .$a_given_b,
     .$b_given_a,
     .$a_given_not_b,
     .$b_given_not_a,
     .$a_and_not_b,
     .$b_and_not_a,
     .$not_a,
     .$not_b
   )
 )
```

Index