Package: gpumetropolis (via r-universe)

November 10, 2026

Title GPU-Portable Vendor-Agnostic Metropolis-Hastings Sampler

Version 0.7.0

Description A generic random-walk Metropolis-Hastings sampler for Markov chain Monte Carlo. The user declares a model by writing its log-likelihood and log-prior as one-sided formulas; the package compiles them to a stack-machine bytecode that a single portable kernel interprets, so the same model runs on the CPU and on the GPU from one source. The sampler advances many independent chains in one batched pass. A distributional equivalence harness based on the two-sample Kolmogorov-Smirnov test and the split R-hat statistic of Gelman and Rubin (1992) <doi:10.1214/ss/1177011136> checks the GPU output against the CPU output. Dispatching the kernel to a GPU helps when many chains are run or when the log-density is expensive to evaluate over a large data set, and does not help for small models run with few chains.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.85.0, xz

Depends R (>= 4.2)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, evd, diptest, loo, coda, MCMCpack, mcmc, nimble, BayesianTools, copula

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/pcbrom/gpumetropolis

BugReports https://github.com/pcbrom/gpumetropolis/issues

Config/pak/sysreqs xz-utils libclang-dev

Repository https://pcbrom.r-universe.dev

Date/Publication 2026-08-12 13:27:07 UTC

RemoteUrl https://github.com/pcbrom/gpumetropolis

RemoteRef HEAD

RemoteSha 261aae098590d267daff7c66e804c344ac072906

Contents

ess . . . 3
gaussian_mean_posterior . . . 3
gpu_metropolis . . . 4
gpum_bayes_factor . . . 7
gpum_copula . . . 7
gpum_crlb . . . 9
gpum_density_compare . . . 10
gpum_diagnose . . . 11
gpum_evidence . . . 12
gpum_fit_catalog . . . 13
gpum_hypothesis . . . 14
gpum_lfo . . . 15
gpum_lm . . . 16
gpum_loo . . . 17
gpum_model . . . 18
gpum_pairs . . . 19
gpum_ppc . . . 19
gpum_region . . . 20
gpum_rope . . . 21
gpum_surface . . . 21
gpum_ts_model . . . 22
gpum_waic . . . 24
hdi . . . 25
kendall_tau . . . 25
ks_equivalence . . . 26
marginal_cdf . . . 27
metropolis_gaussian_mean . . . 28
plot.gpum_catalog . . . 29
plot.gpum_copula . . . 30
rhat . . . 30
tail_dependence . . . 31

Index **32**

ess *Effective sample size*

Description

Estimates the effective sample size of a set of chains: the number of independent draws that would carry the same information as the autocorrelated Markov chain Monte Carlo output. Each chain is reduced by Geyer's initial positive sequence estimator and the per-chain values are summed.

Usage

```
ess(x, warmup = NULL)
```

Arguments

x	A gpumetropolis_fit object, an n_iter by n_chains numeric matrix, or a numeric vector of draws.
warmup	Number of initial iterations discarded before the estimate. When NULL, half of the iterations are discarded for matrix and fit inputs, and none for vector inputs.

Value

A single numeric value, the summed effective sample size.

References

Geyer, C. J. (1992). Practical Markov chain Monte Carlo. Statistical Science 7(4), 473-483. doi:10.1214/ss/1177011137.

Examples

```
set.seed(1)
x <- rnorm(800, mean = 0, sd = 1)
fit <- metropolis_gaussian_mean(x, sigma = 1, n_iter = 2000)
ess(fit)
```

gaussian_mean_posterior *Closed-form posterior of the Gaussian mean*

Description

Returns the analytic posterior of the mean mu for observations following Normal(mu, sigma^2) with known sigma and a flat prior on mu. The posterior is Normal(mean(data), sigma^2 / length(data)). It provides the ground truth used to check that metropolis_gaussian_mean() recovers known parameters.

Usage

```
gaussian_mean_posterior(data, sigma)
```

Arguments

data	Numeric vector of observations.
sigma	Positive finite scalar, the known standard deviation.

Value

A list with the posterior mean and standard deviation sd.

Examples

```
set.seed(1)
gaussian_mean_posterior(rnorm(500, mean = 3, sd = 2), sigma = 2)
```

---

gpu_metropolis *Run the GPU-portable batched Metropolis sampler*

Description

Advances many independent random-walk Metropolis chains over a model declared with gpum_model(). The log-density kernel runs on the chosen backend; the data is uploaded once and stays resident across the run.

Usage

```
gpu_metropolis(
  model,
  data = NULL,
  init = NULL,
  proposal_sd = 0.1,
  n_iter = 2000L,
  n_chains = 4L,
  warmup = NULL,
  adapt = TRUE,
  method = c("rwm", "pt", "de", "mala"),
  temperatures = NULL,
  swap_every = NULL,
  gamma = NULL,
  de_noise = 0.001,
  de_every = NULL,
  de_sync = FALSE,
  seed = 1L,
  backend = c("auto", "cpu", "cuda", "vulkan")
)
```

Arguments

model	A gpum_model object.
data	A named list or data frame with one entry per data column named in the model. Ignored for a model with no data term.
init	Optional numeric matrix of starting values, n_chains rows by one column per parameter. When supplied its row count sets the number of chains. When NULL, chains start from independent standard normal draws.
proposal_sd	Initial scale of the Gaussian random-walk proposal. Accepts a scalar (recycled to all parameters and chains), a length- n_params vector (broadcast across chains) or an n_chains by n_params matrix (one row per chain). When adapt = TRUE, this is only the warmup seed and adaptation refines it per chain.
n_iter	Iterations the sampler runs per chain, counting warmup. Default 2000.
n_chains	Number of chains. Used only when init is NULL. Default 4.
warmup	Warmup iterations to discard before returning, following the convention of Stan and nimble. Must lie in [0, n_iter). Default floor(n_iter / 2), so fit$draws is post-warmup and is suitable for direct plotting. Set warmup = 0 to keep every iteration, useful for inspecting the burn-in trajectory in a trace plot. The string "auto" (adaptive "rwm" path) treats floor(n_iter / 2) as a budget and stops the warmup early once the per-chain acceptance has sat at the asymptotic target for two consecutive batches, spending the remaining iterations on sampling instead of discarding them.
adapt	Whether to adapt the per-chain proposal scale during warmup. When TRUE (default) and warmup > 0, the warmup runs in batches; between batches, Welford updates the per-chain running variance and Robbins-Monro updates the per-chain scalar toward the asymptotic optimum acceptance (0.234 in d >= 2, 0.44 in d = 1; Roberts-Gelman-Gilks 1997, Roberts-Rosenthal 2009). Adaptation stops cleanly at the end of warmup, so the sampling phase is stationary. Set FALSE to keep the trim-only warmup of 0.1.x.
method	Sampling method: "rwm" (default) is the random-walk Metropolis with optional per-chain adaptation; "pt" is parallel tempering with one chain per temperature on the same target and adjacent-pair swap proposals between batches, where the cold chain (T = 1) provides the posterior samples; "de" is Differential Evolution MCMC, whose proposal for each chain is the scaled difference of two other chains plus a small jitter, so the proposal aligns with the correlation of the target through the ensemble geometry without an explicit covariance (ter Braak 2006). The "de" path needs at least 4 chains. "mala" is the Metropolis-adjusted Langevin algorithm: the proposal drifts along the gradient of the log-posterior, obtained by reverse-mode automatic differentiation of the compiled model (JIT-compiled to native code alongside the density), and the warmup preconditions both drift and noise with the pooled posterior covariance. The gradient buys mixing that scales as d^{-1/3} against the random walk's d^{-1/2}, so "mala" is the method of choice as the dimension grows; it targets acceptance 0.574 (Roberts and Rosenthal 1998) and runs on backend = "cpu" in this release.
temperatures	Numeric vector of length n_chains, the temperature of each chain in method = "pt". Ignored for the other methods. When NULL (default), a geometric ladder from 1 to 10 is used. The first entry must be 1 for the cold chain.

swap_every — Integer, the iteration count between swap proposals in `method = "pt"`. Ignored for the other methods. When `NULL` (default), uses `max(n_chains, 10)`.

gamma — Scale of the difference vector in `method = "de"`. Ignored for the other methods. When `NULL` (default), uses `2.38 / sqrt(2 * n_params)`, the value optimal for an approximately Gaussian target (ter Braak 2006). With probability 0.1 each iteration the scale collapses to 1 for an occasional mode-crossing jump.

de_noise — Non-negative factor for the per-dimension jitter added to the `method = "de"` proposal, in units of the current per-chain per-dimension scale. A small positive value keeps the chain irreducible when the ensemble collapses onto a subspace. Default `1e-3`. Ignored for the other methods.

de_every — Integer, the iteration count between refreshes of the frozen population snapshot used for the difference vectors in `method = "de"`. Ignored for the other methods. When `NULL` (default), uses `max(n_chains, 10)`.

de_sync — Logical. When `TRUE`, `method = "de"` runs the synchronous per-generation Differential Evolution (path B): the population advances one generation at a time behind a barrier with a double buffer, so every proposal reads the previous generation, the canonical DE-MC mixing of ter Braak (2006). Better per-iteration mixing on curved or strongly correlated targets, at the cost of a barrier per generation. Currently CPU only (`backend = "cpu"`); the warmup is trim-only on this path and `proposal_sd` is the fixed jitter base. Default `FALSE`, the batched snapshot path A.

seed — Integer seed. Each chain advances its own counter-based stream from a triple32 hash; the seed is itself hashed, so runs with consecutive integer seeds get independent streams.

backend — Compute backend: `"cpu"`, `"cuda"` (NVIDIA-native), `"vulkan"` (vendor-agnostic, through wgpu), or `"auto"`. `"auto"` selects `"cuda"` when its feature was compiled into the build, then `"vulkan"`, and falls back to `"cpu"` with a one-shot per-session message stating that no GPU backend is available. Default `"auto"`.

Value

An object of class `gpum_fit`: a list with `draws` (an `n_iter - warmup` by `n_chains` by `n_params` array of post-warmup samples), `accept_rate`, `n_iter` (kept count), `n_iter_total` (raw count), `warmup` and the rest of the run metadata.

See Also

`gpum_model()`, `rhat()`, `ess()`

Examples

```
set.seed(1)
y <- rnorm(2000, mean = 3, sd = 2)
m <- gpum_model(~ -((y - mu)^2) / 8, params = "mu", data = "y")
fit <- gpu_metropolis(m, data = list(y = y), proposal_sd = 0.05,
                      n_iter = 1000, n_chains = 4)
rhat(fit$draws[, , 1])
```

gpum_bayes_factor *Bayes factor between two models by thermodynamic integration*

Description

Computes the Bayes factor `B_10 = p(y | model1) / p(y | model0)` from the log marginal likelihoods of the two models, each by `gpum_evidence()`. The Bayes factor is the formal Bayesian weight of evidence for one model over another; both models must carry a proper prior.

Usage

```
gpum_bayes_factor(model1, model0, data, ...)
```

Arguments

`model1`, `model0`	Two `gpum_model` objects with proper priors, the alternative and the null.
`data`	The data, shared by both models.
`...`	Passed to `gpum_evidence()` for both models.

Details

The Bayes factor is sensitive to the priors, sharply so for a point null under a diffuse prior, the Jeffreys-Lindley effect: a more diffuse prior on the larger model penalises it regardless of the data. Report the priors with the factor, and prefer `gpum_waic()` or `gpum_loo()` when the goal is predictive comparison rather than a weight of evidence.

Value

An object of class `gpum_bayes_factor` with `bf10`, `log_bf10`, the two log evidences and a verbal `interpretation` on the Jeffreys scale.

See Also

`gpum_evidence()`, `gpum_waic()`, `gpum_loo()`

gpum_copula *Fit a bivariate copula to two-column data*

Description

Infers the dependence structure of two variables by fitting a bivariate copula to their pseudo-observations (average ranks rescaled to the open unit square), so the fit is invariant to the marginal distributions and isolates the dependence. The copula parameter is sampled with `gpu_metropolis()`; the copula log-density is a normalised density on the unit square, so WAIC and PSIS-LOO compare families directly.

Usage

```
gpum_copula(
  data,
  family = "auto",
  criterion = c("waic", "loo"),
  method = "mala",
  warmup = "auto",
  n_iter = 8000L,
  n_chains = 4L,
  seed = 1L,
  ...
)
```

Arguments

`data`	A two-column data frame or matrix; the columns are the two variables whose dependence is modelled.
`family`	One of `"gumbel"`, `"clayton"`, `"frank"`, `"gaussian"`, or `"auto"` to fit all four and select by predictive comparison. Default `"auto"`.
`criterion`	Selection criterion for `family = "auto"`: `"waic"` (default) or `"loo"`. Lower expected log predictive density loss wins.
`method`	Sampler passed to `gpu_metropolis()`. Default `"mala"`: the copula log-density is smooth and one-dimensional in the raw parameter, and the Frank family in particular has a flat far tail in which a random walk can strand a chain, which the gradient path avoids.
`warmup`	Warmup passed to `gpu_metropolis()`. Default `"auto"`.
`n_iter`, `n_chains`, `seed`, `...`	Passed to `gpu_metropolis()`.

Details

With `family = "auto"` every candidate family is fit and the workflow returns the one preferred by predictive comparison together with the full ranking, since the choice of family is a hypothesis about tail behaviour. Gumbel carries upper-tail dependence, Clayton lower-tail, Frank symmetric and tail-independent, Gaussian elliptical and tail-independent; a selected Gumbel is itself the statement that co-movement concentrates in the upper tail.

The three Archimedean families (Gumbel, Clayton, Frank) are parametrised for positive association in this release; the Gaussian family carries the sign of the dependence through its correlation. For data with negative rank correlation, the Gaussian family applies directly and the others do not.

Value

An object of class `gpum_copula`: a list with the selected `family`, the `gpum_fit`, the posterior summaries of the dependence parameter and of Kendall's tau and the tail-dependence coefficients, the pseudo-observations, and (for `"auto"`) the `comparison` table.

See Also

gpu_metropolis(), kendall_tau(), tail_dependence()

Examples

```
set.seed(1)
u <- runif(300)
v <- pnorm(0.8 * qnorm(u) + 0.6 * rnorm(300))
fit <- gpum_copula(data.frame(x = qexp(u), y = qgamma(v, 2)),
                   family = "auto", n_iter = 4000, n_chains = 4)
fit
```

---

gpum_crlb *Observed Fisher information and Cramer-Rao bound for a fit*

Description

Computes the observed Fisher information of the model log-likelihood at the posterior mean by a central-difference Hessian, inverts it to the Cramer-Rao lower bound on the covariance of an unbiased estimator, and returns it beside the empirical posterior covariance for comparison. This is an optional reference, not the headline of the fit: the bound is a frequentist asymptotic object and is meaningful only where the model is regular. The function applies four guards and reports `applicable = FALSE` with a reason when any fails.

Usage

```
gpum_crlb(fit, data = NULL, at = NULL, rhat_max = 1.05)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
data	The same data passed to gpu_metropolis(). Required when the model has a data term; ignored otherwise.
at	Optional numeric vector, the point at which to evaluate the information. Defaults to the posterior mean (the cold chain for a parallel-tempering fit, the pooled chains otherwise).
rhat_max	Convergence guard: when the largest split R-hat over the parameters exceeds this, the comparison is flagged not applicable, since the asymptotic-normal reference assumes a unimodal converged posterior. Default 1.05.

Details

Under the Bernstein-von Mises theorem, for a regular model with a flat or weak prior the posterior covariance approaches the inverse Fisher information; an agreement is then a check that the sampler recovered the information-bound geometry. The identification is asymptotic and assumes an interior true parameter, identifiability, a finite non-singular information and a smooth correctly specified likelihood. It does not transfer to multimodal posteriors or to parameters on a boundary, and an informative prior legitimately makes the posterior tighter than the bound.

The comparison rests on estimator theory: under the same regularity the posterior mean is a consistent and asymptotically efficient estimator, that is it concentrates on the true parameter and attains the Cramer-Rao bound as the data grow, so the bound is the right yardstick for the posterior spread. Consistency is the weaker and more robust property and can hold where asymptotic efficiency does not; this function reports the bound only where the stronger normal-approximation conditions also hold.

Value

An object of class `gpum_crlb`: a list with `information` (the observed Fisher information matrix), `crlb` (its inverse, the bound on the covariance), `crlb_sd`, `crlb_cor`, the empirical `posterior_cov`, `posterior_sd` and `posterior_cor`, the evaluation point `at`, a logical `applicable` and a character `note`. When not applicable the matrix entries are `NA` and `note` states why.

Sampling regimes

The reference is not restricted to i.i.d. data. For independent, non-identically distributed rows (regression-type models, every row with its own covariates) the observed information is the sum of the per-row contributions and the asymptotic normality that backs the comparison holds under the Lindeberg-Feller conditions and local asymptotic normality (Le Cam; van der Vaart 1998, chapter 7). For dependent data declared through the conditional factorisation of `gpum_ts_model()`, the per-row terms are conditional log-densities, their score is a martingale difference sequence, and the summed conditional curvature is again the right information: the martingale central limit theorem (Hall and Heyde 1980) replaces Lindeberg-Feller, with the Bernstein-von Mises theorem for ergodic Markov processes (Borwanker, Kallianpur and Prakasa Rao 1971) backing the posterior side. The stationarity and ergodicity of the process are model assumptions the guards below cannot check.

See Also

`gpu_metropolis()`, `gpum_diagnose()`, `gpum_ts_model()`

`gpum_density_compare` *Compare an observed distribution with generated ones on one plot*

Description

Overlays the kernel density of the observed data with the kernel density of one or more generated samples on a single set of axes, the visual check that a fitted model reproduces the data it was fit to. For competing models the better fit, especially in the relevant tail, is read off directly.

Usage

```
gpum_density_compare(
  observed,
  generated,
  labels = NULL,
  main = "Observed against generated densities",
  xlab = "value",
  colors = NULL,
  ...
)
```

Arguments

observed	Numeric vector of the observed data.
generated	A numeric vector, or a named list of numeric vectors, of generated or posterior-predictive samples (see gpum_ppc()).
labels	Optional character labels for the generated samples; taken from the names of generated when it is a named list.
main, xlab	Plot title and x-axis label.
colors	Optional colours for the generated densities.
...	Passed to plot().

Value

Invisibly, a list with the observed and generated density objects.

See Also

gpum_ppc()

---

gpum_diagnose *One-call diagnostic: convergence stats, plots and verdict.*

Description

Produces a per-parameter table (mean, standard deviation, 2.5%, 50% and 97.5% quantiles, split R-hat, effective sample size and Monte Carlo standard error) and a convergence verdict from the asymptotic canonical thresholds (R-hat below 1.01 in every parameter and ESS at or above 400). When plot = TRUE, opens a multi-panel plot per parameter showing the trace, the pooled density, the running mean per chain and the pooled autocorrelation; when the fit carries adaptation book-keeping, an extra panel shows the per-chain acceptance rate by warmup batch with the asymptotic optimum drawn as a horizontal reference.

Usage

```
gpum_diagnose(fit, plot = TRUE, return_data = FALSE, crlb = NULL)
```

Arguments

fit	A gpum_fit object from gpu_metropolis().
plot	Whether to open the diagnostic plot. Default TRUE.
return_data	Whether to return the structured stats invisibly. Default FALSE; the function then returns NULL invisibly.
crlb	Optional gpum_crlb object from gpum_crlb(). When supplied and applicable, the printout adds a posterior-sd against Cramer-Rao-bound comparison and the density panels overlay the asymptotic-normal reference from the inverse Fisher information. Default NULL.

Value

When return_data = TRUE, a list with summary (one row per parameter), verdict (the convergence diagnosis and a suggested next step), adaptation (passed through from fit$adaptation) and adaptation_hint (a character string when the last warmup batch closed below 80% of the asymptotic target acceptance, NULL otherwise). Otherwise NULL invisibly.

See Also

gpu_metropolis(), rhat(), ess()

---

gpum_evidence *Log marginal likelihood by thermodynamic integration*

Description

Estimates the log marginal likelihood (the model evidence) by thermodynamic integration along a path of power posteriors p_beta(theta) proportional to p(y | theta)^beta p(theta) from the prior (beta = 0) to the posterior (beta = 1). The log evidence is the integral over beta of the expected log-likelihood under p_beta (Gelman and Meng 1998; Friel and Pettitt 2008), evaluated on a power-law grid that is dense near the prior, where the integrand is most curved. Each rung is sampled by the package's own sampler, so no kernel change is needed.

Usage

```
gpum_evidence(
  model,
  data,
  n_rungs = 24L,
  power = 5,
  n_iter = 4000L,
  n_chains = 8L,
  warmup = NULL,
  method = "de",
  seed = 1L,
  backend = "cpu"
)
```

Arguments

model	A gpum_model with a proper prior.
data	The data for the model.
n_rungs	Number of intervals in the beta grid. Default 24.
power	Exponent of the power-law grid beta = (k / n_rungs)^power. Default 5, the value that controls the discretisation bias of trapezoidal thermodynamic integration.
n_iter, n_chains, warmup, method, seed, backend	Passed to gpu_metropolis() for the per-rung sampling. Defaults are 4000 iterations, 8 chains, method = "de".

Details

The evidence requires a proper prior: the marginal likelihood is an integral against the prior, undefined for a flat or absent prior. The function stops when the model has no prior term. The estimate is sensitive to the prior, a property of the marginal likelihood itself, not of the sampler.

Value

An object of class gpum_evidence with log_evidence, the betas grid and the expected_loglik per rung.

See Also

gpum_bayes_factor(), gpum_waic()

---

gpum_fit_catalog *Automatic marginal distribution selection for one column*

Description

Detects the support and the modality of a numeric column, fits every eligible parametric family with the sampler, and ranks them by predictive comparison. The winner's fitted cumulative distribution is the probability-integral transform used to feed gpum_copula(), so the explicit marginals and the copula together model the full joint distribution (Sklar's theorem).

Usage

```
gpum_fit_catalog(
  data,
  catalog = "auto",
  ranking = c("waic", "loo"),
  n_iter = 8000L,
  n_chains = 4L,
  seed = 1L,
  backend = "cpu",
  ...
)
```

Arguments

data	A numeric vector, or a one-column data frame or matrix.
catalog	Either "auto" (default, eligible families chosen from the detected support and modality) or a character vector of family names to fit. Available families: normal, student_t, logistic, laplace, gamma, lognormal, weibull, exponential, beta, gmm2.
ranking	Selection criterion, "waic" (default) or "loo".
n_iter, n_chains, seed, backend, ...	Passed to gpu_metropolis().

Details

Every family is fit in an unconstrained parametrisation, so the sampler never proposes an invalid parameter; the natural parameters are recovered for reporting. A column detected multimodal brings the two-component Gaussian mixture into the eligible set, which is fit with parallel tempering (method = "pt") and relabeled so the lower-location component is first, since a random walk cannot cross the modes.

Value

An object of class gpum_catalog: a list with the best family name and its fit, the comparison ranking table, the support and modality verdict, and the fitted CDF of the winner.

See Also

gpu_metropolis(), gpum_copula(), marginal_cdf()

Examples

```
set.seed(1)
y <- rgamma(400, shape = 3, rate = 2)
result <- gpum_fit_catalog(y, n_iter = 4000, n_chains = 4, backend = "cpu")
result
```

---

gpum_hypothesis *Posterior probability of a hypothesis on one parameter*

Description

Reports the posterior probability that a parameter lies in an interval, below it and above it, directly from the draws. This is an estimation decision: it needs no marginal likelihood and inherits the robustness of the posterior the sampler returned.

Usage

```
gpum_hypothesis(fit, parameter, lower = -Inf, upper = Inf)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
parameter	Name of the parameter.
lower, upper	Bounds of the hypothesis interval. Defaults span the real line, so lower = 0 gives the one-sided P(theta > 0 | y) as prob, the mass in the interval (0, Inf); prob_below is then the complementary P(theta <= 0 | y).

Value

An object of class gpum_hypothesis with prob (mass in the interval), prob_below, prob_above and the inputs.

See Also

gpum_rope()

---

gpum_lfo *Exact leave-future-out cross-validation for dependent data*

Description

Estimates the expected log predictive density of one-step-ahead forecasts by leave-future-out cross-validation with an expanding window: for each held-out time t in the evaluation range, the model is refit on rows 1:(t - 1) and scored on the conditional log predictive density of row t, integrated over the posterior by averaging the pointwise likelihood across the draws. This is the evaluation scheme that respects temporal ordering; gpum_waic() and gpum_loo() assume exchangeable pointwise contributions and are not valid for dependent rows (Burkner, Gabry and Vehtari 2020). The refits are exact rather than importance-sampling approximations: the sampler is fast enough that the approximation of Burkner, Gabry and Vehtari (2020) buys nothing here.

Usage

```
gpum_lfo(ts_model, L = NULL, n_iter = 4000L, n_chains = 4L, seed = 1L, ...)
```

Arguments

ts_model	A gpum_ts from gpum_ts_model().
L	Minimum training rows before the first evaluation: rows 1:L train the first refit and row L + 1 is the first scored point. Default keeps the last quarter of the rows for evaluation.
n_iter, n_chains, seed, ...	Passed to gpu_metropolis() for each refit.

Value

A list of class gpum_lfo with elpd_lfo (the summed expected log predictive density), se (its standard error over the evaluated points), and pointwise (one row per evaluated time).

See Also

gpum_ts_model(), gpum_waic(), gpum_loo()

---

gpum_lm *Declare a conjugate Gaussian linear model*

Description

Builds the conjugate specification of the Gaussian linear model `y = X beta + eps`, `eps ~ N(0, sigma^2 I)`, under the Normal-inverse-Gamma prior `beta | sigma^2 ~ N(b0, sigma^2 B0^{-1})`, `sigma^2 ~ InvGamma(c0 / 2, d0 / 2)`, with `B0` a prior precision matrix. Under this prior the joint posterior of `(beta, sigma^2)` is again Normal-inverse-Gamma in closed form, so gpu_metropolis() samples it exactly and independently, with no warmup, no proposal and no rejection; the effective sample size equals the number of draws. The marginal likelihood is also available in closed form and is returned by the fit as `log_marginal`.

Usage

```
gpum_lm(formula, data, b0 = 0, B0 = 1e-06, c0 = 0.002, d0 = 0.002)
```

Arguments

`formula`	A standard R model formula, e.g. `mpg ~ wt`. Interactions and transformations follow the usual `model.matrix` rules.
`data`	A data frame holding the variables of `formula`.
`b0`	Prior mean of `beta`; scalar recycled or a vector of length equal to the number of columns of the design matrix. Default 0.
`B0`	Prior precision of `beta` (inverse covariance, scaled by `sigma^2`); a scalar for `B0 * I` or a full matrix. Default `1e-6`, near-flat but proper.
`c0`	Prior shape times two of `sigma^2`: `sigma^2 ~ InvGamma(c0 / 2, d0 / 2)`. Default 0.002.
`d0`	Prior rate times two of `sigma^2`. Default 0.002.

Details

The prior must be proper (`B0` positive definite, `c0 > 0`, `d0 > 0`) for the marginal likelihood to exist; the sampling itself only needs the posterior precision `X'X + B0` to be positive definite. The defaults are weakly informative and proper.

Value

An object of class `gpum_conjugate` holding the design matrix, the response and the prior, ready for gpu_metropolis().

See Also

gpu_metropolis(), gpum_model()

Examples

```
m <- gpum_lm(mpg ~ wt, data = mtcars)
fit <- gpu_metropolis(m, n_iter = 4000, n_chains = 4, seed = 1)
fit$log_marginal
```

---

gpum_loo *Pareto-smoothed importance-sampling leave-one-out (PSIS-LOO)*

Description

Computes PSIS-LOO cross-validation by delegating to the loo package, which implements the Pareto-smoothed importance-sampling estimator and its diagnostics (Vehtari, Gelman and Gabry 2017). Like WAIC it estimates out-of-sample predictive accuracy without a marginal likelihood, and the Pareto k diagnostic flags observations where the estimate is unreliable. As with gpum_waic(), the leave-one-out question is only meaningful for independent rows; for the dependent rows of a gpum_ts_model() use gpum_lfo().

Usage

```
gpum_loo(fit, data, max_draws = 4000L)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
data	The same data passed to gpu_metropolis().
max_draws	Cap on the number of posterior draws used. Default 4000.

Value

The loo object returned by loo::loo().

See Also

gpum_waic()

gpum_model *Declare a model for the GPU-portable Metropolis sampler*

Description

Compiles a log-likelihood and an optional log-prior, declared as one-sided formulas, into the bytecode the sampler runs. The log-likelihood is a per-observation expression: the sampler sums it over the data. The formulas may use `+`, `-`, `*`, `/`, `^`, unary `-`, and `exp`, `log`, `sqrt`; any other symbol or function is rejected with a clear error.

Usage

```
gpum_model(loglik, params, data = character(0), prior = NULL)
```

Arguments

`loglik`	One-sided formula, the per-observation log-likelihood, up to an additive constant. It may reference the parameter names and the data column names.
`params`	Character vector of parameter names.
`data`	Character vector of data column names. Empty for a model with no data term.
`prior`	One-sided formula, the joint log-prior over the parameters, up to an additive constant. It may reference only the parameter names. `NULL` is a flat prior.

Value

An object of class `gpum_model`.

See Also

`gpu_metropolis()`

Examples

```
# Gaussian mean with known sd = 2 and a flat prior on mu.
m <- gpum_model(
  loglik = ~ -((y - mu)^2) / 8,
  params = "mu",
  data = "y"
)
```

gpum_pairs *Pairs plot of the joint posterior*

Description

A matrix of panels over the parameters: the marginal posterior density with its highest density interval on the diagonal, and the bivariate posterior with its credible-region contour off the diagonal. When a gpum_crlb object is supplied and applicable, the Cramer-Rao ellipse is overlaid on the off-diagonal panels, so the recovered convergence region can be read against the information-bound reference.

Usage

```
gpum_pairs(fit, crlb = NULL, level = 0.95, max_points = 3000L, n = 60L, ...)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
crlb	Optional gpum_crlb object for the ellipse overlay.
level	Mass of the highest density interval and credible region. Default 0.95.
max_points	Cap on plotted draws per panel. Default 3000.
n	Grid resolution of the bivariate density. Default 60.
...	Unused.

Value

The fit, invisibly.

See Also

gpum_region(), gpum_surface(), gpum_crlb()

gpum_ppc *Posterior predictive sample from a fit*

Description

Draws n parameter vectors from the posterior (the cold chain under parallel tempering) and, for each, calls generate() to simulate from the model, returning the pooled posterior-predictive sample. The package does not yet generate from an arbitrary likelihood on its own; the user supplies the family's one-line simulator, which they already know from the model they declared. An automatic generator from the model itself is the goal of the synthesis release on the roadmap.

Usage

```
gpum_ppc(fit, generate, n = 4000L)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
generate	A function of one named parameter vector (one posterior draw) returning one or more simulated observations. It may close over the data to simulate a covariate-dependent response.
n	Number of posterior draws to simulate from. Default 4000.

Value

A numeric vector, the pooled posterior-predictive sample.

See Also

gpum_density_compare()

---

gpum_region *Credible region of a parameter pair*

Description

Returns the highest posterior density region of two parameters at level level, as contour polygons of the bivariate posterior density. The region is the set the parameters concentrate in, the convergence region of the pair. The result composes onto any plot with lines().

Usage

```
gpum_region(fit, params, level = 0.95, n = 80L)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
params	Length-two character vector of parameter names.
level	Probability mass of the region. Default 0.95.
n	Grid resolution of the density estimate. Default 80.

Value

An object of class gpum_region with the contours (a list of x, y polygons), the params and the level. Plot it directly or add it to an existing plot with lines().

See Also

gpum_pairs(), gpum_surface()

gpum_rope *ROPE and HDI decision on one parameter*

Description

Applies the region-of-practical-equivalence rule (Kruschke 2018): the parameter is practically equivalent to the null when its highest density interval lies inside the ROPE, practically different when the interval lies entirely outside it, and undecided otherwise. The decision is from the draws and needs no marginal likelihood.

Usage

```
gpum_rope(fit, parameter, rope, null = 0, ci = 0.95)
```

Arguments

`fit`	A `gpum_fit` from `gpu_metropolis()`.
`parameter`	Name of the parameter.
`rope`	Either the half-width of a symmetric ROPE around `null`, or a length-two vector giving the ROPE interval directly.
`null`	Centre of a symmetric ROPE. Ignored when `rope` is an interval. Default 0.
`ci`	Mass of the highest density interval. Default 0.95.

Value

An object of class `gpum_rope` with the `hdi`, the `rope` interval, the share `pct_in_rope` of the posterior inside it, and a `decision`.

See Also

`hdi()`, `gpum_hypothesis()`

gpum_surface *Posterior density surface of a parameter pair*

Description

Plots the bivariate posterior density of two parameters as level curves in two dimensions and as a three-dimensional surface with the credible-region contours projected on the floor. The level curves are the highest posterior density regions at the requested masses.

Usage

```
gpum_surface(
  fit,
  params,
  type = c("both", "contour", "persp"),
  levels = c(0.5, 0.8, 0.95),
  n = 80L,
  theta = 30,
  phi = 25,
  ...
)
```

Arguments

fit	A gpum_fit from gpu_metropolis().
params	Length-two character vector of parameter names.
type	One of "both" (default), "contour" or "persp".
levels	Probability masses of the level curves. Default c(0.5, 0.8, 0.95).
n	Grid resolution. Default 80.
theta, phi	Viewing angles of the 3D surface.
...	Passed to persp().

Value

The fit, invisibly.

See Also

gpum_region(), gpum_pairs()

---

gpum_ts_model *Declare a Markov (time-series) model by conditional factorisation*

Description

Builds a gpum_model() for dependent data with a Markov dependence of order p. The user writes the conditional log-density of one observation given its p predecessors, referring to the current value by the series name and to the lagged values as <name>_lag1 through <name>_lagp; gpum_ts_model() assembles the lagged design so the engine's row sum computes the conditional log-likelihood sum_{t = p + 1}^{n} log f(y_t | y_{t-1}, ..., y_{t-p}).

Usage

```
gpum_ts_model(
  loglik,
  params,
  series,
  order = 1L,
  covariates = NULL,
  prior = NULL
)
```

Arguments

loglik	One-sided formula, the conditional log-density of the current observation. Refer to the series by its name in series and to lags as <name>_lag1, ..., <name>_lagp. Exogenous covariates from covariates enter by their own names (aligned to the current time index).
params	Character vector of parameter names.
series	Named list with one entry, the observed series (list(y = <numeric>)); the name is the symbol used in loglik.
order	Markov order p >= 1, the number of lags the conditional density uses.
covariates	Optional named list of exogenous covariate vectors, each the same length as the series; row t of the design carries the covariate at time t.
prior	Optional one-sided formula for the log-prior (plus, when wanted, the stationary log-density of the initial state), evaluated once per parameter vector.

Details

The likelihood conditions on the first p observations, the standard conditional-likelihood treatment (Billingsley 1961; Hamilton 1994, section 5.2); for n moderately large the information in the first p points is negligible. A stationary initial density, when available in closed form, can be added through prior, which is evaluated once per parameter vector.

Inference downstream is licensed by the dependent-data asymptotics: the conditional score is a martingale difference sequence, so the observed information of the conditional factorisation is the right curvature (martingale CLT, Hall and Heyde 1980) and the posterior satisfies a Bernstein-von Mises theorem for ergodic Markov processes (Borwanker, Kallianpur and Prakasa Rao 1971). Model comparison by gpum_waic() or gpum_loo() assumes exchangeable pointwise terms and does not transfer to dependent rows; use gpum_lfo() instead.

Value

A list of class gpum_ts with the compiled model (a gpum_model), the assembled data list ready for gpu_metropolis(), the order, and the effective sample rows n_rows = length(y) - order.

See Also

gpum_model(), gpu_metropolis(), gpum_lfo()

Examples

```
set.seed(1)
y <- as.numeric(arima.sim(list(ar = 0.6), n = 300))
ts_m <- gpum_ts_model(
  ~ -0.5 * ((y - mu - phi * (y_lag1 - mu)) / exp(ls))^2 - ls,
  params = c("mu", "phi", "ls"), series = list(y = y), order = 1,
  prior = ~ -0.5 * (mu / 10)^2 - 0.5 * (phi / 2)^2 - 0.5 * (ls / 2)^2
)
fit <- gpu_metropolis(ts_m$model, data = ts_m$data, n_iter = 2000,
                      n_chains = 4, seed = 1)
```

---

gpum_waic *Widely applicable information criterion (WAIC)*

Description

Computes WAIC from the pointwise log-likelihood of the fit (Watanabe 2010; Vehtari, Gelman and Gabry 2017). WAIC estimates out-of-sample predictive accuracy without a marginal likelihood, so it is robust where Bayes factors are prior-sensitive; lower WAIC is preferred when comparing models on the same data.

Usage

```
gpum_waic(fit, data, max_draws = 4000L)
```

Arguments

`fit`	A `gpum_fit` from `gpu_metropolis()`.
`data`	The same data passed to `gpu_metropolis()`.
`max_draws`	Cap on the number of posterior draws used, to bound the pointwise matrix. Default 4000.

Details

The comparison is valid only when the log-likelihood is normalised. The model DSL evaluates the log-density up to an additive constant, which is enough for sampling a fixed model but not for comparing models with different likelihoods, since the dropped constants differ. To compare models with WAIC or LOO, write the full normalised per-observation log-likelihood in the formula, including the normalising constant such as `-0.5 * log(2 * pi * sigma^2)` for a Gaussian.

The pointwise decomposition also assumes the rows are exchangeable contributions, which holds for independent data (identically distributed or not) but fails for the dependent rows of a `gpum_ts_model()` conditional factorisation: leaving out a past observation that later rows condition on is not a predictive question. For dependent data use `gpum_lfo()`, which respects the temporal ordering.

Value

An object of class `gpum_waic` with `waic`, `elpd_waic` (the expected log pointwise predictive density), `p_waic` (the effective number of parameters), the standard error `se` and the `pointwise` terms.

See Also

`gpum_loo()`, `gpum_bayes_factor()`

`hdi` *Highest posterior density interval*

Description

The shortest interval that contains a probability mass `ci` of the samples, found by scanning every window of the sorted draws.

Usage

```
hdi(x, ci = 0.95)
```

Arguments

`x`	Numeric samples.
`ci`	Probability mass in the interval. Default 0.95.

Value

A named numeric vector `c(lower, upper)`.

See Also

`gpum_rope()`

`kendall_tau` *Posterior Kendall's tau of a fitted copula*

Description

Returns the posterior summary of Kendall's tau, the rank-correlation functional of the copula parameter, on its natural `[-1, 1]` scale. Tau is comparable across families, unlike the family-specific dependence parameter.

Usage

```
kendall_tau(object)
```

Arguments

object	A gpum_copula from gpum_copula().

Value

A named numeric vector: mean, q2.5, median, q97.5.

See Also

gpum_copula(), tail_dependence()

---

ks_equivalence *Distributional equivalence by the two-sample Kolmogorov-Smirnov test*

Description

Pools the post-warmup draws of two samplers and applies the two-sample Kolmogorov-Smirnov test. It is the equivalence gate for the package: a CPU and a GPU sampler are treated as equivalent when the test does not reject the hypothesis that their draws come from the same distribution.

Usage

```
ks_equivalence(x, y, warmup = NULL, alpha = 0.05, thin = TRUE)
```

Arguments

x, y	Each a gpumetropolis_fit object, an n_iter by n_chains numeric matrix, or a numeric vector of draws.
warmup	Number of initial iterations discarded from x and y before pooling. When NULL, half of the iterations are discarded for matrix and fit inputs, and none for vector inputs.
alpha	Significance level of the test. Default 0.05.
thin	Controls thinning before the test. TRUE (default) thins each pooled sample to its effective sample size. A positive integer keeps every thin-th draw. FALSE applies no thinning.

Details

The Kolmogorov-Smirnov test assumes independent draws, while Markov chain Monte Carlo output is autocorrelated; feeding it raw draws makes the test reject far too often. By default the pooled draws are therefore thinned down to the effective sample size returned by ess() before the test is applied.

Not rejecting is evidence consistent with equivalence, not a proof that the two distributions are identical. Report the statistic and the p-value, and read equivalent as "no detected difference at level alpha".

Value

A list with the test `statistic`, the `p_value`, the chosen `alpha`, a logical `equivalent` (`TRUE` when `p_value > alpha`) and the post-thinning pooled sample sizes `n_x` and `n_y`.

Examples

```
set.seed(1)
x <- rnorm(800, mean = 1, sd = 1)
a <- metropolis_gaussian_mean(x, sigma = 1, n_iter = 2000, seed = 1)
b <- metropolis_gaussian_mean(x, sigma = 1, n_iter = 2000, seed = 2)
ks_equivalence(a, b)
```

---

marginal_cdf *Fitted cumulative distribution of the selected marginal*

Description

Returns the probability-integral transform of the winning family at the posterior-mean parameters, either as a function or evaluated at the data. This is the transform that carries a column into the unit interval for `gpum_copula()`, the explicit-marginal counterpart of the rank transform.

Usage

```
marginal_cdf(object, at = NULL)
```

Arguments

`object`	A `gpum_catalog` from `gpum_fit_catalog()`.
`at`	Optional numeric vector at which to evaluate the CDF; when omitted the function itself is returned.

Value

A function, or the CDF values at `at`.

See Also

`gpum_fit_catalog()`, `gpum_copula()`

`metropolis_gaussian_mean` *Batched random-walk Metropolis sampler for a Gaussian mean*

Description

Samples the posterior of the mean `mu` of observations assumed to follow `Normal(mu, sigma^2)` with known `sigma` and a flat prior on `mu`. Many independent chains are advanced together: at every iteration the candidate state of each chain is evaluated by a single batched call to the log-density kernel.

Usage

```
metropolis_gaussian_mean(
  data,
  sigma,
  n_iter = 2000L,
  n_chains = 4L,
  init = NULL,
  proposal_sd = NULL,
  seed = 1L
)
```

Arguments

`data`	Numeric vector of observations. Must be finite and non-empty.
`sigma`	Positive finite scalar, the known standard deviation of the observations.
`n_iter`	Number of iterations recorded per chain. Default 2000.
`n_chains`	Number of independent chains. Used only when `init` is `NULL`. Default 4.
`init`	Optional numeric vector of starting values, one per chain. When supplied, its length sets the number of chains. When `NULL`, chains start evenly spread over `mean(data) +/- 2 * sigma`, which overdisperses them so that `rhat()` is informative while staying bounded for many chains.
`proposal_sd`	Positive finite scalar, the standard deviation of the Gaussian random-walk proposal. When `NULL`, defaults to `2.4 * sigma / sqrt(length(data))`, scaled to the width of the posterior.
`seed`	Single integer seed. Each chain runs an independent PCG64 stream derived from `(seed, chain_index)`, so the result is reproducible and does not depend on chain scheduling.

Details

This is the CPU reference sampler. Its log-density kernel runs on the CPU. Later package versions dispatch the same kernel to a GPU and are validated for distributional equivalence against this function with `ks_equivalence()` and `rhat()`. The sequential dependence within a chain (`x_{t+1}` depends on `x_t`) is intrinsic to Markov chain Monte Carlo and is not parallelised; the parallel axes are the chains and the sum over the data.

Value

An object of class gpumetropolis_fit: a list with draws (an n_iter by n_chains numeric matrix), accept_rate (per-chain acceptance rate) and the run metadata.

See Also

rhat(), ess(), ks_equivalence(), gaussian_mean_posterior()

Examples

```
set.seed(1)
x <- rnorm(500, mean = 3, sd = 2)
fit <- metropolis_gaussian_mean(x, sigma = 2, n_iter = 1000)
rhat(fit)
```

---

plot.gpum_catalog *Visual inspection of a fitted marginal catalogue*

Description

Draws a three-panel diagnostic for the winning family: the fitted density over the data histogram (observed against generated), the QQ plot of the empirical against the fitted quantiles, and the WAIC ranking of the candidate families. Visual companion of the printed text diagnostic.

Usage

```
## S3 method for class 'gpum_catalog'
plot(x, ...)
```

Arguments

x	A gpum_catalog from gpum_fit_catalog().
...	Ignored.

Value

x, invisibly.

See Also

gpum_fit_catalog()

plot.gpum_copula *Visual inspection of a fitted bivariate copula*

Description

Draws a four-panel diagnostic: the pseudo-observations with the fitted copula density contours over them, the posterior of the dependence parameter, the posterior of Kendall's tau with the empirical value marked, and the observed-against-generated check on the unit square. This is the visual companion of the text diagnostic printed by the object.

Usage

```
## S3 method for class 'gpum_copula'
plot(x, n_grid = 40L, ...)
```

Arguments

x	A gpum_copula from gpum_copula().
n_grid	Grid resolution for the density contours. Default 40.
...	Ignored.

Value

x, invisibly. Called for the plot.

See Also

gpum_copula(), gpum_density_compare()

rhat *Split R-hat convergence diagnostic*

Description

Computes the split potential scale reduction factor for a set of chains. Each chain is split in half and the halves are treated as separate chains, which makes the statistic sensitive to non-stationarity within a chain. A value near 1 is consistent with the chains having reached the same distribution; values above roughly 1.01 to 1.1 indicate that they have not.

Usage

```
rhat(x, warmup = NULL)
```

Arguments

x	A gpumetropolis_fit object or an n_iter by n_chains numeric matrix of draws.
warmup	Number of initial iterations discarded before the split. When NULL, half of the iterations are discarded.

Value

A single numeric value, the split R-hat statistic.

References

Gelman, A. and Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. Statistical Science 7(4), 457-472. doi:10.1214/ss/1177011136.

Examples

```
set.seed(1)
x <- rnorm(500, mean = 0, sd = 1)
fit <- metropolis_gaussian_mean(x, sigma = 1, n_iter = 1000)
rhat(fit)
```

---

tail_dependence *Posterior tail-dependence coefficients of a fitted copula*

Description

Returns the posterior summaries of the lower and upper tail-dependence coefficients, the probabilities of joint extremes in each tail. Gumbel has upper-tail dependence and no lower, Clayton the reverse, Frank and Gaussian neither.

Usage

```
tail_dependence(object)
```

Arguments

object	A gpum_copula from gpum_copula().

Value

A list with lower and upper, each a named numeric vector.

See Also

gpum_copula(), kendall_tau()

Index

ess, 3
ess(), *6*, *12*, *26*, *29*

gaussian_mean_posterior, 3
gaussian_mean_posterior(), *29*
gpu_metropolis, 4
gpu_metropolis(), *7–10*, *12–24*
gpum_bayes_factor, 7
gpum_bayes_factor(), *13*, *25*
gpum_copula, 7
gpum_copula(), *13, 14*, *26, 27*, *30, 31*
gpum_crlb, 9
gpum_crlb(), *12*, *19*
gpum_density_compare, 10
gpum_density_compare(), *20*, *30*
gpum_diagnose, 11
gpum_diagnose(), *10*
gpum_evidence, 12
gpum_evidence(), *7*
gpum_fit_catalog, 13
gpum_fit_catalog(), *27*, *29*
gpum_hypothesis, 14
gpum_hypothesis(), *21*
gpum_lfo, 15
gpum_lfo(), *17*, *23, 24*
gpum_lm, 16
gpum_loo, 17
gpum_loo(), *7*, *15, 16*, *23*, *25*
gpum_model, 18
gpum_model(), *4*, *6*, *17*, *22, 23*
gpum_pairs, 19
gpum_pairs(), *20*, *22*
gpum_ppc, 19
gpum_ppc(), *11*
gpum_region, 20
gpum_region(), *19*, *22*
gpum_rope, 21
gpum_rope(), *15*, *25*
gpum_surface, 21
gpum_surface(), *19, 20*
gpum_ts_model, 22
gpum_ts_model(), *10*, *15–17*, *24*
gpum_waic, 24
gpum_waic(), *7*, *13*, *15–17*, *23*

hdi, 25
hdi(), *21*

kendall_tau, 25
kendall_tau(), *9*, *31*
ks_equivalence, 26
ks_equivalence(), *28, 29*

marginal_cdf, 27
marginal_cdf(), *14*
metropolis_gaussian_mean, 28
metropolis_gaussian_mean(), *3*

plot.gpum_catalog, 29
plot.gpum_copula, 30

rhat, 30
rhat(), *6*, *12*, *28, 29*

tail_dependence, 31
tail_dependence(), *9*, *26*