Package: SamplerCompare (via r-universe)

November 13, 2026

Type Package

Title A Framework for Comparing the Performance of MCMC Samplers

Version 1.3.4.9001

Date 2023-04-23

Author Madeleine Thompson, except dchud.f and dchdd.f, which were written by G. W. Stewart.

Maintainer Madeleine Thompson <madeleineth@gmail.com>

Description A framework for running sets of MCMC samplers on sets of distributions with a variety of tuning parameters, along with plotting functions to visualize the results of those simulations. Compatibility updates by Renova.

License GPL-2

Copyright (c) 2010-2021 Madeleine Thompson, except dchud.f and dchdd.f, which are from LINPACK and are in the public domain in the United States.

LazyLoad yes

Depends R (>= 3.0)

Imports methods, mvtnorm

Suggests ggplot2 (>= 0.9.2), lintr

NeedsCompilation yes

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

Date/Publication 2026-11-13 04:56:48 UTC

RemoteUrl https://github.com/renova-r/SamplerCompare

RemoteRef renova-1.3.4.9001

RemoteSha b5835942cf03e6dfde4211232129c6168bedb4ed

Contents

adaptive.metropolis.sample 2
ar.act 3

arms.sample . . . 5
check.dist.gradient . . . 6
chud . . . 6
compare.samplers . . . 7
comparison.plot . . . 9
compounded.sampler . . . 11
cov.match.sample . . . 12
funnel.dist . . . 13
hyperrectangle.sample . . . 13
make.c.dist . . . 15
make.cone.dist . . . 16
make.dist . . . 17
make.gaussian . . . 19
make.multimodal.dist . . . 20
make.mv.gamma.dist . . . 20
multivariate.metropolis.sample . . . 21
nonadaptive.crumb.sample . . . 22
oblique.hyperrect.sample . . . 23
raw.symbol . . . 24
scdist-class . . . 25
schools.dist . . . 25
shrinking.rank.sample . . . 26
simulation.result . . . 27
stepout.slice.sample . . . 29
twonorm . . . 30
univar.eigen.sample . . . 31
wrap.c.sampler . . . 32

Index **33**

`adaptive.metropolis.sample` *Adaptive Metropolis*

Description

Generate a sample from a probability distribution with the Adaptive Metropolis algorithm

Usage

```
adaptive.metropolis.sample(target.dist, x0, sample.size,
                           tuning=0.1, beta=0.05, burn.in=0.2)
```

Arguments

target.dist	Target distribution; see make.dist.
x0	Numeric vector containing initial state.
sample.size	Requested sample size.
tuning	Standard deviation of first component of proposal distribution
beta	Weight of first component of proposal distribution
burn.in	Stop adaptation after this fraction of the chain. Set this to 1.0 to obtain the behavior described by Roberts and Rosenthal (2009).

Details

This function implements the Adaptive Metropolis algorithm as described by Roberts and Rosenthal (2009). Proposals are a mixture of a spherical Gaussian with standard deviation equal to tuning/sqrt(target.dist$ndim) (with weight beta) and a Gaussian with covariance equal to the sample covariance of the already-computed observations scaled by 2.38^2/target.dist$ndim (with weight 1-beta). The resulting Markov chain is not strictly stationary with the target distribution for the burn-in period of the chain, but is ergodic.

Value

A list containing the elements X, evals, reject.rate, and sample.cov. This sampler follows the calling convention of compare.samplers. reject.rate contains the fraction of proposals that were rejected. sample.cov is the most recent sample covariance used to update the proposal distribution.

References

Roberts, G. O. and Rosenthal, J. S. (2009), "Examples of Adaptive MCMC," Journal of Computational and Graphical Statistics 18(2):349-367.

See Also

compare.samplers, multivariate.metropolis.sample

---

ar.act *Compute the autocorrelation time of a chain*

Description

Computes the autocorrelation time of an MCMC chain using an AR model with order chosen by AIC.

Usage

```
ar.act(Y, true.mean=NULL)
```

Arguments

`Y`	A matrix or vector containing the states of a stationary Markov chain. If a matrix, each row is a single state.
`true.mean`	A vector containing the true mean of `Y`. It should be either NULL or have as many elements as `Y` has columns. If NULL, the sample mean of `Y` is used.

Details

This function fits an AR(p) model to each component of the chain with states Y using the Yule-Walker method to estimate the coefficients and AIC to estimate p. Let π be the vector of estimated AR coefficients for column i, and let ρ be the sample autocorrelation function to lag p. Then, the autocorrelation time of the component is estimated as:

$$\tau_i = \frac{1 - \pi^T \rho}{(1 - \sum \pi)^2}$$

For more discussion of this formula and its associated confidence intervals, see Thompson (2010).

The returned autocorrelation time (and associated confidence interval) are the maxima over the columns of Y.

Callers may want to remove a burn-in period from a sample before passing it to `ar.act`.

Value

A list with elements:

- *act*: the estimated autocorrelation time of the slowest-mixing column of Y.
- *se*: the standard error of *act*.
- *act.025*, *act.975*: a nominal 95% confidence interval for *act*. Since the interval is asymmetric about *act*, the standard error is not sufficient to generate these.
- *order*: The order of the AR model selected (p).

References

Thompson, M. B. (2010). Graphical comparison of MCMC performance. https://arxiv.org/abs/1011.4457.

See Also

`compare.samplers`, `ar.yw`, `CODA::effectiveSize`, `CODA::spectrum0.ar`

`arms.sample` *Adaptive Rejection Metropolis Sampler*

Description

Generate a sample from a probability distribution with Adaptive Rejection Metropolis Sampling

Usage

```
arms.sample(target.dist, x0, sample.size, tuning=1)
```

Arguments

`target.dist`	Target distribution; see `make.dist`.
`x0`	Numeric vector containing initial state.
`sample.size`	Sample size requested.
`tuning`	Scale for initial envelope; see details.

Details

`arms.sample` implements Adaptive Rejection Metropolis Sampling (Gilks, Best, and Tan, 1995). As described by Gilks et al, a user of ARMS must specify an initial envelope roughly approximating the target density. This implementation attempts to provide a simpler interface for users by generating an envelope automatically.

To form an initial envelope for coordinate (i), four abscissae are needed. One is `x0`. The sampler tries points with abscissae `x0[i]-2^k*tuning` and `x0[i]+2^k*tuning` for whole-numbers `k` until points with log densities smaller than that at `x0` are found, then chooses a fourth point from the interior of the two found points. (Specifically, the interval between `x0` and the lowest density found point is binary-searched until a point with log-density larger than the found point is located.)

This scheme for defining an envelope does not depend on the current state in the dimension being sampled. For discussion of why this must be the case, see see Gilks, Neal, Best and Tan (1997).

Value

A list with elements `X`, `evals`, and `rejections`, following the calling convention of `compare.samplers`. `rejections` indicates how many Metropolis-Hastings proposals were rejected.

References

Gilks, W. R., Best, N. G., and Tan, K. K. C. (1995) "Adaptive Rejection Metropolis Sampling within Gibbs Sampling," Applied Statistics 44(4):455-472.

Gilks, W. R., Neal, R. M., Best, N. G., and Tan, K. K. C. (1997) "Corrigendum: Adaptive Rejection Metropolis Sampling," Applied Statistics 46(2):541-542.

See Also

`compare.samplers`

check.dist.gradient *Test a gradient function*

Description

Evaluates the gradient function of a distribution object and compares it to a numeric gradient computed from the log density function.

Usage

```
check.dist.gradient(ds, x, h=1e-7)
```

Arguments

ds	A distribution object with defined log density and gradient functions.
x	A point inside the support of ds.
h	An offset from x at which to evaluate the log density when computing numeric derivatives.

Details

check.dist.gradient computes the numeric derivative of ds$log.density at x in each of its coordinates and compares this to the value returned by ds$grad.log.density. If the relative error is greater than 0.001, an error is reported.

This function can be used when defining a distribution to ensure that the gradient function is implemented correctly.

See Also

make.dist

Examples

```
check.dist.gradient(N2weakcor.dist, runif(2))
```

chud *Cholesky Update/Downdate*

Description

Rank-one updates of Cholesky factors

Usage

```
chud(R,x)
chdd(R,x)
```

Arguments

R	an upper-triangular matrix
x	a vector

Details

chud computes Q such that:

$$Q^T Q = R^T R + xx^T$$

chdd computes Q such that:

$$Q^T Q = R^T R - xx^T$$

chdd reports an error if $R^T R - xx^T$ is not positive definite. The two functions use LINPACK's dchud and dchdd routines respectively, two of the few routines from LINPACK without analogues in LAPACK.

Value

An updated version of R.

References

Dongarra, J. J., Moler, C. B., Bunch, J. R., Stewart, G. W. (1979) LINPACK User's Guide.

See Also

chol

---

compare.samplers *Compare MCMC samplers on distributions*

Description

Simulate a set of distributions with a set of samplers and tuning parameters

Usage

```
compare.samplers(sample.size, dists, samplers, tuning = 1,
                 trace = TRUE, seed = 17, burn.in = 0.2)
```

Arguments

sample.size	An integer specifying how long a chain to simulate.
dists	A list of scdist objects (often generated by make.dist) specifying the probability distributions to simulate.
samplers	A list of sampler functions. See the section "Sampler calling convention".
tuning	A numeric vector of tuning parameters
trace	A logical indicating whether a message should be printed when a chain completes (useful for large simulations).
seed	If not null, the random seed is set to this with set.seed before each chain and restored afterwards. This makes each chain individually replicable, useful when debugging.
burn.in	Fraction of chain to discard before computing autocorrelation time.

Details

compare.samplers runs a single Markov chain simulation of length sampler.size size for each combination of the elements of dists, samplers, and tuning. Each chain starts at a point generated by the initial member of the distribution object, or a point uniformly drawn from the unit hypercube if initial is not defined. It returns a data frame with one row per simulation so that performance of the methods can be compared on the various distributions. The simplest way to visualize the results is with the comparison.plot function.

For an example of the use of this method, see the "Introduction to SamplerCompare" vignette. For discussion of the ideas behind it, see Thompson (2010).

Value

A data frame with columns dist, dist.expr, ndim, sampler, sampler.expr, tuning, act, act.025, act.975, act.y, act.y.025, act.y.975, evals, grads, cpu, err, and aborted. Each row represents a single simulation.

- sampler and dist are the names of the sampler and distribution taken from the lists passed to compare.samplers.
- sampler.expr and dist.expr are plotmath versions of sampler and dist. If not specified by the distribution object and sampler function, they are constructed from dist and sampler.
- ndim is the dimension of the state space of the target distribution.
- tuning is the tuning parameter for the chain.
- act is the estimated autocorrelation time, taken over all parameters of the simulation; see ar.act. This is more accurate if target.dist$mean is defined.
- act.025 and act.975 bound a nominal 95% confidence interval for act. Since the interval is asymmetric, a standard error is not sufficient.
- act.y, act.y.025, and act.y.975 are an estimate and endpoints for a nominal 95% confidence interval for the autocorrelation time of the log density. These are more accurate if target.dist$mean.log.dens is defined.
- evals and grads are the mean log-density and gradient evaluations per observation.

- cpu is the number of processor seconds used per observation.
- err is the two-norm of the difference between the estimated mean and the true mean. Set to NA if the distribution does not specify a true mean.
- aborted is a logical indicating whether the simulation returned fewer rows than requested.

Sampler calling convention

Sampler functions passed to compare.samplers should be of the form:

```
sampler(target.dist, x0, sample.size, tuning)
```

target.dist is a scdist object representing the distribution to sample from; see make.dist for more information on these. x0 is the initial state of the chain; it must be a numeric vector of length target.dist$ndim. sample.size is the desired length of the chain, passed down from compare.samplers. tuning is a scalar tuning parameter from the vector passed to compare.samplers.

Sampler functions should return a list with elements X, evals, and (optionally) grads. X should be a matrix with target.dist$ndim columns and sample.size rows. If for some reason it is necessary to abort the chain, returning fewer rows is acceptable. evals and grads indicate the number of calls to target.dist$log.density and target.dist$grad.log.density respectively.

Sampler functions must have a name attribute with a human-readable name for the MCMC method. If desired, they may also have a name.expression attribute containing a more nicely-formatted version of the name in plotmath format.

See the vignette "Introduction to SamplerCompare" for an example of a function that implements this interface.

References

Thompson, M. B. (2010), Graphical comparison of MCMC performance, University of Toronto Dept. of Statistics technical report no. 1010.

Thompson, M. B. (2011), "Introduction to SamplerCompare," Journal of Statistical Software 43(12):1-10, doi:10.18637/jss.v043.i12.

See Also

make.dist, comparison.plot, ar.act, "Introduction to SamplerCompare" (vignette)

---

comparison.plot *Plot the results of compare.samplers*

Description

Generates a plot of representing results from compare.samplers.

Usage

```
comparison.plot(RS, xlab=NULL, ylab=NULL, base_size=10, ...)
```

Arguments

RS	A data frame in the form returned by compare.samplers and simulation.result.
xlab, ylab, ...	Options to be passed to ggplot2::qplot.
base_size	The text base size passed to ggplot2::theme_bw.

Details

This function generates a grid of subplots, where each column of plots represents a sampler and each row represents a distribution. The horizontal axis in each subplot represents the tuning parameter passed as tuning to compare.samplers, and the vertical axis represents the product RS$evals * RS$act, the number of log density evaluations per independent sample required for that distribution when simulated by that sampler with that tuning parameter. 95% confidence intervals, covering the range [RS$evals * RS$act.025, RS$evals * RS$act.975], are represented by vertical bars. If RS$evals or RS$act is missing or infinite, a question mark is plotted instead of the default plot character.

comparison.plot returns a **ggplot2** plot object. If it is called non-interactively, one must call print on the returned object for a plot to be displayed. To superimpose other figures of merit on the plot, one can add geom_* objects to the returned plot object before calling print.

For more discussion of this type of plot, see Thompson (2010).

Value

A **ggplot2** plot object.

Note

This is the only function in SamplerCompare that uses the **ggplot2** package, so it is loaded explicitly by comparison.plot instead of being listed as a package dependency. This way, compute servers calling compare.samplers do not need to have **ggplot2** installed on them.

References

Thompson, M. B. (2010), Graphical comparison of MCMC performance, University of Toronto Dept. of Statistics technical report no. 1010.

Thompson, M. B. (2011), "Introduction to SamplerCompare," Journal of Statistical Software 43(12):1-10, doi:10.18637/jss.v043.i12.

See Also

compare.samplers, simulation.result, ggplot2::qplot, "Introduction to SamplerCompare" (vignette)

`compounded.sampler` *Build a sampler from transition functions*

Description

Defines a probability distribution object for use with `compare.samplers`.

Usage

```
compounded.sampler(step.functions, name, name.expr=NULL)
```

Arguments

`step.functions`	A list of transition functions; see details.
`name`	A character string naming the sampler.
`name.expr`	A character string naming the sampler in `plotmath` notation.

Details

`compounded.sampler` builds an MCMC sampler following the conventions of `compare.samplers` from a list of transition functions. The returned sampler has four arguments: *target.dist*, *x0*, *sample.size*, and *limit*. Further arguments, including the standard argument *tuning*, are passed to every transition function. The first three arguments transition functions are passed are *target.dist*, a vector state x to transition from, and the log density at that state, y. They should return a list containing four elements: x, y, *evals*, and *grads*. x is the state transitioned to, y is the log density at that state, and *evals* and *grads* are the number of log density and gradient calls made in that transition.

Each MCMC iteration, the first transition function is called with the current state of the chain. The state it returns is passed to the second transition function, whose returned state is passed to the third, and so on. The state returned by the final transition function is taken to be the state of the chain as a whole at the end of the iteration.

This way, transition functions that provide complementary features, such as fast mixing in different coordinates, can be combined without modifying their internal structure. The `transition_fn` interface provides a similar mechanism for samplers implemented in C. It is documented in the vignette, "R/C Glue in SamplerCompare".

Value

A sampler function.

See Also

`compare.samplers`

`cov.match.sample` *Sample with covariance-matching slice sampling*

Description

Generate a sample from a probability distribution with the covariance-matching slice sampling method.

Usage

```
cov.match.sample(target.dist, x0, sample.size, tuning=1,
                 theta=1, limit=length(x0)*100)
```

Arguments

`target.dist`	Target distribution; see `make.dist`.
`x0`	Initial coordinates.
`sample.size`	Sample size to draw.
`tuning`	A tuning parameter; corresponds to σ_c in sec. 4 of Thompson and Neal (2010).
`theta`	A factor to scale the crumb standard deviation in every direction after a proposal is rejected. So, after k proposals, crumbs have standard deviation `theta^k*tuning` in directions orthogonal to all the proposal gradients.
`limit`	A limit on the number of log-density evaluations per observation before sampling is aborted.

Details

This function implements the covariance-matching method of slice sampling, as described by Thompson and Neal (2010). It can be passed to `compare.samplers` in the `samplers` list argument.

Value

A list with elements `X`, `evals`, `grads`, and `adapt.rate`. `adapt.rate` indicates the fraction of crumb draws that resulted in adaptation. This sampler follows the calling convention of `compare.samplers`.

References

Thompson, M. B. and Neal, R. M. (2010). Covariance-adaptive slice sampling. Technical Report TR-1002, Dept. of Statistics, University of Toronto.

See Also

`compare.samplers`, `shrinking.rank.sample`

funnel.dist *Funnel distribution object*

Description

A distribution object for Radford Neal's funnel distribution

Details

`funnel.dist` represents the funnel distribution described by Neal (2003, p. 732). It is a ten-dimensional distribution on the reals, with:

$$v \sim N(0, 3^2),\ x[k] \sim N(0, e^v) \text{ for } k = 1, \ldots, 9$$

The state space is $(v, x[1], x[2], ..., x[9])$. The name comes from the funnel-shaped two dimensional marginal distributions $(v, x[k])$.

This object is intended as a demonstration to be passed to `compare.samplers`.

References

Neal, Radford M. (2003), "Slice Sampling," The Annals of Statistics 31(3):705-767.

See Also

`make.dist`, `compare.samplers`

hyperrectangle.sample *Multivariate slice samplers*

Description

Generate a sample from a probability distribution with a slice sampler taking multivariate steps.

Usage

```
hyperrectangle.sample(target.dist, x0, sample.size, tuning=1,
                  use.gradient=TRUE, limit=length(x0)*100)
nograd.hyperrectangle.sample(...)
```

Arguments

target.dist	Target distribution; see make.dist.
x0	Numeric vector containing initial state.
sample.size	Sample size requested.
tuning	Initial edge length of hyperrectangle.
use.gradient	A logical indicating whether the sampler should use the gradient when shrinking the box.
limit	A limit on the number of log-density evaluations per observation before sampling is aborted.
...	nograd.hyperrectangle.sample takes the same arguments as hyperrectangle.sample, except use.gradient.

Details

hyperrectangle.sample implements multivariate slice sampling with hyperrectangles as described in Neal (sec. 5.1, 2003).

If use.gradient is set, when a proposal is rejected, the gradient at the rejected proposal is used to choose a direction to shrink the box. Neal suggested shrinking in the direction the gradient was largest, but this implementation shrinks in the direction that the gradient times the box length is largest to better handle poorly scaled distributions.

If use.gradient is not set, the gradient is not computed and the box is shrunk in all directions after every rejected proposal. Calling nograd.hyperrectangle.sample is equivalent to calling hyperrectangle.sample with use.gradient=FALSE; the extra name is provided for convenience when using either of these functions with compare.samplers.

Value

A list with elements X, evals, and grads. This sampler follows the calling convention of compare.samplers.

References

Neal, Radford M. (2003), "Slice Sampling," The Annals of Statistics 31(3):705-767.

See Also

compare.samplers, nonadaptive.crumb.sample, interval.slice.sample

make.c.dist *Define a probability distribution object with C log-density*

Description

Defines a probability distribution object for use with compare.samplers with log-density implemented in C.

Usage

```
make.c.dist(ndim, name, c.log.density, c.context = NULL,
            name.expression = NULL, mean = NULL, cov = NULL)
```

Arguments

ndim	The size of the distribution's state space.
name	A human-readable name for the distribution.
c.log.density	A C function returning the log-density and gradient of the target distribution.
c.context	An opaque object passed to c.log.density
name.expression	A name for the distribution in plotmath notation. Used in preference to name in plot functions when available.
mean	A vector specifying the true mean of the distribution.
cov	A matrix specifying the true covariance of the distribution.

Details

See make.dist for discussion of ndim, name, name.expression, mean, and cov.

c.log.density is a string containing the symbol name of a C function that computes the log density and log density gradient of the target distribution. It has the type log_density_t, defined in SamplerCompare.h as:

```
typedef double log_density_t(dist_t *ds, double *x,
                             int compute_grad, double *grad);
typedef struct {
  log_density_t *log_dens;
  SEXP context;
  int ndim;
} dist_t;
```

The ds structure defines the distribution, where the log_dens element is a pointer to the function named by c.log.density, the context element is a SEXP containing the c.context parameter, and ndim is the ndim parameter to make.c.dist.

The x parameter is an ndim-long array of doubles containing the location at which to evaluate the log-density, which the log_density_t should return.

If compute_grad is nonzero, the function should compute the gradient of the log density and store it in the double array pointed to by grad. If for some reason it cannot do this, it should call the R-internal error function to report an error to the user. If the implementor does not plan to sample from the distribution with a method that computes gradients, this can reduce implementation effort.

The details of this interface are described in greater detail in "R/C Glue in SamplerCompare".

Value

A scdist object.

See Also

compare.samplers, make.dist, "R/C Glue in SamplerCompare" (vignette)

---

make.cone.dist *Create a cone distribution object*

Description

Create a cone distribution object as defined by Roberts and Rosenthal

Usage

```
make.cone.dist(ndim)
```

Arguments

ndim	The dimension of the distribution's state space.

Details

Defines a distribution object with the following log density:

$$\pi(x) = e^{-\|x\|}$$

This is used in Roberts and Rosenthal (2002) to demonstrate the deterioration in slice sampler performance as dimensionality increases. It is intended to be passed to compare.samplers. Its implementation also serves as a simple demonstration of how to define a distribution in C.

Value

A scdist object.

References

Roberts, G. O. and Rosenthal, J. S. (2002) "The Polar Slice Sampler," Stochastic Models 18(2):257-280.

See Also

make.dist

make.dist *Define a probability distribution object*

Description

Defines a probability distribution object for use with compare.samplers.

Usage

```
make.dist(ndim, name, name.expression=NULL,
          log.density=NULL, grad.log.density=NULL,
          log.density.and.grad=NULL, initial=NULL,
          mean=NULL, cov=NULL, mean.log.dens=NULL)
```

Arguments

ndim	The size of the distribution's state space.
name	A human-readable name for the distribution.
name.expression	A name for the distribution in plotmath notation. Used in preference to name in plot functions when available.
log.density	A function taking a vector argument that returns the log density of the distribution evaluated at that point.
grad.log.density	A function taking a vector argument that returns the gradient of the log density of the distribution evaluated at that point.
log.density.and.grad	A function taking a vector argument and a logical that returns a list with two elements, log.density and grad.log.density. The logical indicates whether the caller wants the gradient; if not, this function may omit the grad.log.density element in the return value.
initial	A function that returns an overdispersed initial state for an MCMC simulation of this distribution, used by compare.samplers. If unset, uniform draws on a unit hypercube are assumed to be acceptable.
mean	A vector specifying the true mean of the distribution.
cov	A matrix specifying the true covariance of the distribution.
mean.log.dens	A scalar specifying the true mean of the log density of the distribution. This will depend on the normalization of the log density function.

Details

Every distribution must have a name and a dimension. The log density and its gradient are optional; they are used by samplers implemented in R. Samplers implemented in other languages could specifically recognize the name of the distribution instead of calling back into R, though

there is a mechanism for C functions to call back. The mean and covariance do not affect sampling, only post-sample diagnostics like autocorrelation time.

For many distributions, it is easier to compute the log density and its gradient at the same time than separately; these will generally specify `log.density.and.grad` and leave `log.density` and `log.density.and.grad` as `NULL`. The returned object will fill those in with calls to `log.density.and.grad`. Similarly, if it is simpler to compute them separately, `log.density.and.grad` will be synthesized from `log.density` and `grad.log.density` if necessary.

`mean`, `cov`, and `mean.log.dens` values are intended to be used by diagnostic routines. `mean` and `mean.log.dens` are currently used by `compare.samplers` when estimating autocorrelation times.

See `make.c.dist` for a way to define distributions whose densities are implemented in C instead of R.

Value

A `scdist` object. It has elements with the same names as the arguments to `make.dist`.

See Also

`compare.samplers`, `make.c.dist`, `check.dist.gradient`, "R/C Glue in SamplerCompare" (vignette)

Examples

```
# A one dimensional Gamma(3,2) distribution.

# So that the density does not return NaN outside the support.
inflog <- function(x) ifelse(x<=0, -Inf, log(x))

# Define density; unnormalized densities are fine.

gamma32.log.density <- function(x) (3-1)*inflog(x) - x/2
gamma32.grad <- function(x) (3-1)/x - 1/2

# Use make.dist to define the distribution object.

gamma32.dist <- make.dist(1, 'Gamma32', 'plain("Gamma")(3,2)',
                          log.density=gamma32.log.density,
                          grad.log.density=gamma32.grad,
                          mean=3*2, cov=as.matrix(3*2^2))

# Make sure the log density and gradient agree at an arbitrary point.

check.dist.gradient(gamma32.dist, 17)
```

make.gaussian *Gaussian distribution objects*

Description

Gaussian distribution objects

Usage

```
make.gaussian(mean, sigma=NULL, rho=NULL)
N2weakcor.dist
N4poscor.dist
N4negcor.dist
```

Arguments

mean	The mean of the distribution as a numeric vector; implicitly specifies the dimension.
sigma	The covariance of the distribution.
rho	The marginal correlations between parameters.

Details

make.gaussian returns a distribution object representing a multivariate normal distribution. If sigma is specified, that is taken to be its covariance. Otherwise, if rho is specified, the covariance is taken to be a matrix with ones on the diagonal and rho on the off-diagonal elements. To preserve positive definiteness, rho must be between -1/(length(mean)-1) and 1.

N2weakcor.dist, N4poscor.dist, and N4negcor.dist are predefined distributions generated with make.gaussian. They are intended to be used as test cases with compare.samplers. The examples below show how they are defined. N2weakcor.dist is a weakly positively correlated two-dimensional Gaussian. N4poscor.dist is a highly positively correlated four-dimensional Gaussian. N4negcor.dist is a highly negatively correlated four-dimensional Gaussian. N4poscor.dist and N4negcor.dist are similarly conditioned, but N4poscor.dist has one large eigenvalue and three small ones, while N4negcor.dist has one small eigenvalue and three large ones.

See Also

compare.samplers, make.dist

Examples

```
N2weakcor.dist <- make.gaussian(c(0,0), rho=0.8)
N4poscor.dist <- make.gaussian(c(1,2,3,4), rho=0.999)
N4negcor.dist <- make.gaussian(c(1,2,3,4), rho=-0.3329)
```

make.multimodal.dist *Create a distribution object for a random mixture of Gaussians*

Description

Create a distribution object for a random mixture of Gaussians

Usage

```
make.multimodal.dist(nmodes, ndim, cube.size)
```

Arguments

nmodes	The number of components in the mixture model.
ndim	The dimension of the model.
cube.size	The edge length of the hypercube in which the modes are distributed.

Details

Defines a distribution object for a mixture of random Gaussians. The means of the the nmodes Gaussians are randomly distributed over an ndim-dimensional hypercube with one corner at the origin and the opposite cube.size away in each positive direction. The same random seed is temporarily set when drawing modes, so every time this function is called with the same parameters, the resulting distribution is the same.

This is included as a test case for comparing how MCMC methods perform on multimodal distributions.

Value

A scdist object. For convenience, the modes element is a matrix containing the modes as rows.

See Also

make.dist, compare.samplers

make.mv.gamma.dist *Create a distribution object for a set of uncorrelated Gamma distributions*

Description

Create a distribution object for a set of uncorrelated Gamma distributions

Usage

```
make.mv.gamma.dist(shape, scale=rep(1, length(shape)))
```

Arguments

shape	A vector of shape parameters.
scale	A vector of scale parameters. Must have the same length as shape

Details

Defines a distribution object for a multivariate distribution where each marginal density is Gamma and uncorrelated with the other coordinates. The log density is therefore equivalent to `sum(dgamma(x, shape, scale=scale, log=TRUE))`.

This is included as a test case for comparing how MCMC methods perform on asymmetric distributions.

Value

A `scdist` object. For convenience, the `shape` and `scale` elements are filled in with the parameters passed to `make.mv.gamma.dist`.

See Also

`make.gaussian`, `compare.samplers`

`multivariate.metropolis.sample` *Metropolis samplers*

Description

Generate a sample from a probability distribution with the Metropolis algorithm.

Usage

```
multivariate.metropolis.sample(target.dist, x0, sample.size, tuning=1)
univar.metropolis.sample(target.dist, x0, sample.size, tuning=1)
```

Arguments

target.dist	Target distribution; see `make.dist`.
x0	Numeric vector containing initial state.
sample.size	Sample size requested.
tuning	Proposal standard deviation

Details

These two functions implement variants of the Metropolis algorithm for sampling a target distribution, following the interface used by `compare.samplers`. `multivariate.metropolis.sample` uses spherically symmetric Gaussian proposals with marginal standard deviation equal to the `tuning` parameter. `univar.metropolis.sample` updates each coordinate in sequence using univariate Gaussian proposals with standard deviation equal to the `tuning` parameter.

So that these two functions are roughly comparable, with a p-dimensional target distribution, `multivariate.metropolis.sa` performs p accept-reject steps each time between observations, so that both functions evaluate the log density a number of times roughly equal to p times the sample size. While there are often efficiency optimizations possible when only one coordinate is updated, `univar.metropolis.sample` does not support these; these two samplers are included for comparison rather than for practical use.

Value

A list with elements `X`, `evals`, and `reject.rate`. See `compare.samplers` for more information on `X` and `evals`. `reject.rate` is the fraction of proposals not accepted.

See Also

`compare.samplers`, `adaptive.metropolis.sample`

`nonadaptive.crumb.sample`

Sample with nonadaptive-crumb slice sampling

Description

Generate a sample from a probability distribution with the nonadaptive-crumb slice sampling method.

Usage

```
nonadaptive.crumb.sample(target.dist, x0, sample.size,
                         tuning=1, downscale=0.95)
```

Arguments

`target.dist`	Target distribution; see `make.dist`.
`x0`	Numeric vector containing initial state.
`sample.size`	Requested sample size.
`tuning`	Initial crumb standard deviation.
`downscale`	Factor to reduce crumb standard deviation by when a proposal is rejected.

Details

This function implements slice sampling with nonadaptive crumbs. Crumbs are Gaussian with spherical covariance starting at `tuning`, decreasing by `downscale` each time a proposal is rejected. More information can be found in sec. 5.2 of Neal (2003). This function can be passed to `compare.samplers` in the `samplers` list argument.

Value

A list with elements `X`, `evals`, and `grads`, following the calling convention of `compare.samplers`.

References

Neal, Radford M. (2003), "Slice Sampling," The Annals of Statistics 31(3):705-767.

See Also

`shrinking.rank.sample`, `compare.samplers`

`oblique.hyperrect.sample`

Eigendecomposition-based hyperrectangle method

Description

Generate a sample from a probability distribution with the hyperrectangle method with slice approximation axes oriented along eigenvectors.

Usage

```
oblique.hyperrect.sample(target.dist, x0, sample.size, tuning=1,
                    edge.scale=5, cheat=FALSE)
cheat.oblique.hyperrect.sample(target.dist, x0, sample.size, tuning=1)
```

Arguments

`target.dist`	Target distribution; see `make.dist`.
`x0`	Numeric vector containing initial state.
`sample.size`	Sample size requested.
`tuning`	Scale of initial/fallback hypercube edge; *w* in Thompson (2011, ch. 3).
`edge.scale`	The initial slice approximation has edges of length equal to the square root of the corresponding eigenvalue times this factor.
`cheat`	Set to true to use the covariance from *target.dist* instead of estimating it. This is not possible on real problems but can be useful for debugging.

Details

These two functions implement the hyperrectangle method (Neal, 2003, sec. 5.1) with the hyperrectangle oriented along estimates of the eigenvectors of the target distribution's covariance, as described by Thompson (2011, ch. 3). The functions follow the interface used by `compare.samplers`. Calling `cheat.oblique.hyperrect.sample` is equivalent to calling `oblique.hyperrect.sample` with `cheat=TRUE`; it is provided as a convenience so that it can be passed directly to `compare.samplers`.

Value

A list with elements `X`, `evals`, and `grads`. See `compare.samplers` for more information.

References

Neal, Radford M. (2003), "Slice Sampling," The Annals of Statistics 31(3):705-767.

Thompson, M. B. (2011), Slice Sampling with Multivariate Steps. http://hdl.handle.net/1807/31955.

See Also

`compare.samplers`, `univar.eigen.sample`

`raw.symbol` *Locate a symbol*

Description

Call R_FindSymbol and return function pointer in a raw vector

Usage

```
raw.symbol(symbol)
```

Arguments

`symbol`	a length one character vector containing a C symbol

Details

This function calls `R_FindSymbol(symbol, "", NULL)` in C. If the symbol is found, the function pointer is returned as a raw vector. If not, an error is thrown.

This is intended to be used to fill in context objects for samplers and distributions implemented in C. Exposing this interface in R prevents the need for extra C glue that does nothing except call `R_FindSymbol`.

Value

A raw vector containing a function pointer.

See Also

wrap.c.sampler, make.c.dist, "R/C Glue in SamplerCompare" (vignette)

---

scdist-class *A class representing a probability distribution*

Description

This class represents a probability distribution. See make.dist for more information.

---

schools.dist *Eight schools distribution object*

Description

A distribution object for the eight-schools distribution

Details

This object represents the distribution of "eight schools," a ten-dimensional multilevel model from Gelman et al (2004). The first and second parameters are mean and log-variance hyperparameters, and the third through tenth are group-level means.

This object is intended as a demonstration to be passed to compare.samplers.

References

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). Bayesian Data Analysis, Second Edition. Chapman and Hall/CRC. pp. 138-145.

See Also

make.dist, compare.samplers

shrinking.rank.sample *Sample with shrinking-rank slice sampling*

Description

Generate a sample from a probability distribution with the shrinking-rank slice sampling method.

Usage

```
shrinking.rank.sample(target.dist, x0, sample.size, tuning=1,
         downscale=0.95, min.dimension=1)
```

Arguments

target.dist	Target distribution; see make.dist.
x0	Numeric vector containing initial state.
sample.size	Requested sample size.
tuning	A tuning parameter; corresponds to σ_c in sec. 5 of Thompson and Neal (2010).
downscale	Factor to reduce crumb standard deviation by when a proposal is rejected.
min.dimension	The minimum dimension to sample crumbs from.

Details

shrinking.rank.slice.sample implements the shrinking-rank method of slice sampling, as described by Thompson and Neal (2010). It can be passed to compare.samplers in the samplers list argument.

Value

A list with elements X, evals, and grads, following the calling convention of compare.samplers.

References

Thompson, M. B. and Neal, R. M. (2010). Covariance-adaptive slice sampling. Technical Report TR-1002, Dept. of Statistics, University of Toronto.

See Also

compare.samplers cov.match.sample

`simulation.result` *Summarize one MCMC chain*

Description

Summarize one MCMC chain in the format used by compare.samplers

Usage

```
simulation.result(target.dist, sampler.name, X,
                  evals=NULL, grads=NULL, tuning=NULL, cpu=NULL,
                  burn.in=0.2, y=NULL,
                  sampler.expr=sprintf("plain('%s')", sampler.name),
                  aborted=NA)
```

Arguments

`target.dist`	A distribution object of the sort generated by `make.dist` representing the distribution sampled from. This is used to obtain the dimension and name of the distribution; the log density function does not need to be specified.
`sampler.name`	The name of the sampler that generated this simulation. If generated by **SamplerCompare**, this would usually be the *name* attribute of the sampler function.
`X`	A matrix (or object that can be coerced to a matrix) containing the simulation results. It should have one row per iteration and one column for each component of the state space. Corresponds to the X element of the list returned by a sampler.
`evals`	The total number of log density evaluations used in the simulation; corresponds to the *evals* element of the list returned by a sampler.
`grads`	The total number of log density gradient evaluations used in the simulation; corresponds to the *grads* element of the list returned by a sampler.
`tuning`	The scalar tuning parameter passed to the sampler.
`cpu`	The processor time used to generate the simulation in seconds.
`burn.in`	Initial fraction of X to discard before computing autocorrelation times.
`y`	A vector with the same number of elements as X has rows containing the log densities at the states represented by those rows.
`sampler.expr`	The name of the sampler that generated this simulation in `plotmath` format. If generated by **SamplerCompare**, this would usually be the *name.expression* attribute of the sampler function.
`aborted`	A logical scalar indicating whether the simulation was prematurely aborted.

Details

This function summarizes a simulation into a single-row data frame by computing the autocorrelation time of its slowest-mixing component and, if possible, the autocorrelation time of the log

density and the error in the sample mean. The autocorrelation time of the slowest-mixing component can always be estimated, but is more accurate if the true mean is specified in *target.dist*. The autocorrelation time of the log density can be estimated if either the log density function is specified in *target.dist* or an explicit vector of log densities is passed as *y*. The error in the sample mean can be computed if the mean is specified in *target.dist*.

This function is intended to be called once per simulation for a variety of simulations. The results are to be combined with `rbind` and can be visualized with `comparison.plot`. While the *evals* and *tuning* arguments are optional, the result cannot be used with `comparison.plot` if it is not set. `simulation.result` is normally called internally by `compare.samplers` but is exported so that simulations run in external systems such as JAGS can be analyzed with **SamplerCompare**. See the "Examples" section for an example of this usage.

Value

A single-row data frame of the format returned by `compare.samplers`.

References

Thompson, M. B. (2011), "Introduction to SamplerCompare," Journal of Statistical Software 43(12):1-10, doi:10.18637/jss.v043.i12.

See Also

`compare.samplers`, `comparison.plot`, "Introduction to SamplerCompare" (vignette)

Examples

```
## Not run:
# An example generated with the following JAGS model:
#
# model {
#   mu[1] <- 0
#   mu[2] <- 0
#   Sigma[1,1] <- 1
#   Sigma[2,2] <- 1
#   Sigma[1,2] <- 0.7
#   Sigma[2,1] <- 0.7
#   x ~ dmnorm(mu, inverse(Sigma))
# }
#
# and the following JAGS script:
#
# model in "mv.7.model"
# compile, nchains(1)
# initialize
# update 1000
# monitor x
# update 10000
# coda *

# Load data written by JAGS
```

```
library(coda)
X <- read.coda('CODAchain1.txt', 'CODAindex.txt')

# Dummy distribution object.

N2.dist <- make.dist(2, '2D Normal, cor=0.7', mean=c(0,0))

# Compute simulation result.  evals and tuning are hacks; they
# are undefined with Gibbs sampling.  JAGS can do its own burn-in,
# so set burn.in to zero.

sim.result <- simulation.result(N2.dist, 'JAGS', X,
                                evals=nrow(X)*ncol(X), tuning=1,
                                burn.in=0)

## End(Not run)
```

`stepout.slice.sample` *Univariate slice samplers*

Description

Generate a sample from a probability distribution with a slice sampler.

Usage

```
stepout.slice.sample(target.dist, x0, sample.size, tuning=1,
                     step.out=TRUE, limit=length(x0)*100)
interval.slice.sample(...)
```

Arguments

`target.dist`	Target distribution; see `make.dist`.
`x0`	Numeric vector containing initial state.
`sample.size`	Requested sample size.
`tuning`	Initial interval length for slice.
`step.out`	Flag indicating whether to expand the initial interval before proposing a new coordinate.
`limit`	A limit on the number of log-density evaluations per observation before sampling is aborted.
`...`	`interval.slice.sample` takes the same arguments as `stepout.slice.sample`, except `step.out`.

Details

`stepout.slice.sample` implements univariate slice sampling with stepping out as described in sec. 4 of Neal (2003). If `step.out=FALSE` or `interval.slice.sample` is called instead, no stepping out is performed; the wrapper function `interval.slice.sample` is provided for convenience when calling `compare.samplers`.

If `target.dist` is a multivariate distribution, each step of the Markov chain updates each coordinate once in sequence.

Value

A list with elements `X`, `evals`, and `grads`, following the calling convention of `compare.samplers`.

References

Neal, Radford M. (2003), "Slice Sampling," The Annals of Statistics 31(3):705-767.

See Also

`compare.samplers`, `hyperrectangle.sample`

---

twonorm *Euclidean norm of a vector*

Description

Computes the Euclidean norm of a vector.

Usage

```
twonorm(x)
```

Arguments

x	A vector.

Details

`twonorm` computes the Euclidean norm of a vector: `sqrt(sum(x^2))`.

Value

A numeric vector of length one containing the two-norm of x.

univar.eigen.sample *Eigendecomposition-based slice samplers*

Description

Generate a sample from a probability distribution with slice sampling with univariate steps along eigenvectors.

Usage

```
univar.eigen.sample(target.dist, x0, sample.size, tuning=1,
                    steps.out=100, cheat=FALSE)
cheat.univar.eigen.sample(target.dist, x0, sample.size, tuning=1,
                    steps.out=100)
```

Arguments

target.dist	Target distribution; see make.dist.
x0	Numeric vector containing initial state.
sample.size	Sample size requested.
tuning	Initial slice approximation length.
steps.out	Maximum number of iterations the stepping out algorithm should run when choosing an initial slice approximation. Set to NULL to refrain from stepping out.
cheat	Set to true to use the covariance from *target.dist* instead of estimating it. This is not possible on real problems but can be useful for debugging.

Details

These two functions implement slice sampling with univariate steps along estimated eigenvectors. Thompson (2011, ch. 3) has details on the algorithms. The functions follow the interface used by compare.samplers. Calling cheat.univar.eigen.sample is equivalent to calling univar.eigen.sample with cheat=TRUE; it is provided as a convenience so that it can be passed directly to compare.samplers.

Value

A list with elements X, evals, and grads. See compare.samplers for more information.

References

Thompson, M. B. (2011), Slice Sampling with Multivariate Steps. http://hdl.handle.net/1807/31955.

See Also

compare.samplers, oblique.hyperrect.sample

`wrap.c.sampler` *Create an R stub function for a sampler implemented in C*

Description

Create an R stub function for a sampler implemented in C.

Usage

```
wrap.c.sampler(sampler.symbol, sampler.context,
               name, name.expression=NULL)
```

Arguments

`sampler.symbol`	A one-element character vector containing the symbol of the C sampler function.
`sampler.context`	An arbitrary R object to be passed to the sampler function.
`name`	The `name` attribute for the sampler function.
`name.expression`	The `name.expression` attribute for the sampler function, in `plotmath` format.

Details

This function is intended to allow `compare.samplers` to be able to invoke sampler functions written in C. It wraps a C sampler function in an R function implementing the standard sampler interface. The function named by `sampler.symbol` is expected to have the function prototype:

```
sampler(SEXP sampler_context, dist_t *ds, double *x0,
        int sample_size, double tuning, double *X_out);
```

This is defined as a type `sampler_t` in `SamplerCompare.h`. The parameter `sampler_context` is the same as the R object `sampler.context` passed to `wrap.c.sampler`. `ds` describes the distribution to be sampled. `x0`, `sample_size`, and `tuning` are as described in `compare.samplers` and should be considered read-only. `X_out` is a column-major matrix to be filled in with the generated sample; it has dimension `ds->ndim * sample_size`.

The vignette "R/C Glue in SamplerCompare" covers this interface in greater detail.

Value

An R function implementing the interface described in `compare.samplers`.

See Also

"R/C Glue in SamplerCompare" (vignette)

Index

adaptive.metropolis.sample, 2, *22*
ar.act, 3, *8, 9*
arms.sample, 5

chdd *(*chud*)*, 6
cheat.oblique.hyperrect.sample *(*oblique.hyperrect.sample*)*, 23
cheat.univar.eigen.sample *(*univar.eigen.sample*)*, 31
check.dist.gradient, 6, *18*
chud, 6
compare.samplers, *3–5*, 7, *9–26*, *28*, *30–32*
comparison.plot, *8*, 9, *9*, *28*
compounded.sampler, 11
cov.match.sample, 12, *26*

funnel.dist, 13

hyperrectangle.sample, 13, *30*

interval.slice.sample, *14*
interval.slice.sample *(*stepout.slice.sample*)*, 29

make.c.dist, 15, *18*, *25*
make.cone.dist, 16
make.dist, *3*, *5, 6*, *8, 9*, *12–16*, 17, *19–23*, *25–27*, *29*, *31*
make.gaussian, 19, *21*
make.multimodal.dist, 20
make.mv.gamma.dist, 20
multivariate.metropolis.sample, *3*, 21

N2weakcor.dist *(*make.gaussian*)*, 19
N4negcor.dist *(*make.gaussian*)*, 19
N4poscor.dist *(*make.gaussian*)*, 19
nograd.hyperrectangle.sample *(*hyperrectangle.sample*)*, 13
nonadaptive.crumb.sample, *14*, 22

oblique.hyperrect.sample, 23, *31*

raw.symbol, 24

scdist-class, 25
schools.dist, 25
shrinking.rank.sample, *12*, *23*, 26
simulation.result, *10*, 27
stepout.slice.sample, 29

twonorm, 30

univar.eigen.sample, *24*, 31
univar.metropolis.sample *(*multivariate.metropolis.sample*)*, 21

wrap.c.sampler, *25*, 32