Package: rust (via r-universe)

September 19, 2024

Type Package

Title Ratio-of-Uniforms Simulation with Transformation

Version 1.4.3

Date 2024-08-14

Description Uses the generalized ratio-of-uniforms (RU) method to simulate from univariate and (low-dimensional) multivariate continuous distributions. The user specifies the log-density, up to an additive constant. The RU algorithm is applied after relocation of mode of the density to zero, and the user can choose a tuning parameter r. For details see Wakefield, Gelfand and Smith (1991) <DOI:10.1007/BF01889987>, Efficient generation of random variates via the ratio-of-uniforms method, Statistics and Computing (1991) 1, 129-133. A Box-Cox variable transformation can be used to make the input density suitable for the RU method and to improve efficiency. In the multivariate case rotation of axes can also be used to improve efficiency. From version 1.2.0 the 'Rcpp' package <https://cran.r-project.org/package=Rcpp> can be used to improve efficiency.

Imports graphics, Rcpp (>= 0.12.10), stats

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 3.3.0)

RoxygenNote 7.2.3

Suggests bang, knitr, microbenchmark, revdbayes, rmarkdown, testthat

VignetteBuilder knitr

URL https://paulnorthrop.github.io/rust/,
https://github.com/paulnorthrop/rust

BugReports https://github.com/paulnorthrop/rust/issues

LinkingTo Rcpp (>= 0.12.10), RcppArmadillo

Config/testthat/edition 3

NeedsCompilation yes

Author Paul J. Northrop [aut, cre, cph]

Maintainer Paul J. Northrop <p.northrop@ucl.ac.uk>

Repository CRAN

Date/Publication 2024-08-17 06:30:02 UTC

Contents

`rust-package` *rust: Ratio-of-Uniforms Simulation with Transformation*

Description

Uses the multivariate generalized ratio-of-uniforms method to simulate from a distribution with log-density `logf` (up to an additive constant). `logf` must be bounded, perhaps after a transformation of variable.

Details

The main functions in the rust package are `ru` and `ru_rcpp`, which implement the generalized ratio-of-uniforms algorithm. The latter uses the Rcpp package to improve efficiency. Also provided are two functions, `find_lambda` and `find_lambda_one_d`, that may be used to set a suitable value for the parameter `lambda` if Box-Cox transformation is used prior to simulation. If `ru_rcpp` is used the equivalent functions are `find_lambda_rcpp` and `find_lambda_one_d_rcpp` Basic `plot` and `summary` methods are also provided.

See the following package vignettes for information:

- Introducing rust or `vignette("rust-a-vignette", package = "rust")`.
- When can rust be used? or `vignette("rust-b-when-to-use-vignette", package = "rust")`.
- Rusting faster: Simulation using Rcpp or `vignette("rust-c-using-rcpp-vignette", package = "rust")`.

Author(s)

Maintainer: Paul J. Northrop <p.northrop@ucl.ac.uk> [copyright holder]

References

Wakefield, J. C., Gelfand, A. E. and Smith, A. F. M. Efficient generation of random variates via the ratio-of-uniforms method. Statistics and Computing (1991) 1, 129-133. doi:10.1007/BF01889987.

Box, G. and Cox, D. R. (1964) An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211-252.

Eddelbuettel, D. and Francois, R. (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18. doi:10.18637/jss.v040.i08.

Eddelbuettel, D. (2013) Seamless R and C++ Integration with Rcpp. Springer, New York. ISBN 978-1-4614-6867-7.

See Also

`ru` and `ru_rcpp` to perform ratio-of-uniforms sampling.

`summary.ru` for summaries of the simulated values and properties of the ratio-of-uniforms algorithm.

`plot.ru` for a diagnostic plot.

`find_lambda_one_d` and `find_lambda_one_d_rcpp` to produce (somewhat) automatically a list for the argument `lambda` of `ru` for the `d` = 1 case.

`find_lambda` and `find_lambda_rcpp` to produce (somewhat) automatically a list for the argument `lambda` of `ru` for any value of `d`.

`create_log_j_xptr` *Create external pointer to a C++ function for* `log_j`

Description

Create external pointer to a C++ function for `log_j`

Usage

```
create_log_j_xptr(fstr)
```

Arguments

`fstr`	A string indicating the C++ function required.

Details

See the Rusting faster: Simulation using Rcpp vignette.

Examples

See the examples in `ru_rcpp`.

`create_phi_to_theta_xptr`
Create external pointer to a C++ function for `phi_to_theta`

Description

Create external pointer to a C++ function for `phi_to_theta`

Usage

```
create_phi_to_theta_xptr(fstr)
```

Arguments

`fstr`	A string indicating the C++ function required.

Details

See the Rusting faster: Simulation using Rcpp vignette.

Examples

See the examples in `ru_rcpp`.

`create_xptr` *Create external pointer to a C++ function for* `logf`

Description

Create external pointer to a C++ function for `logf`

Usage

```
create_xptr(fstr)
```

Arguments

`fstr`	A string indicating the C++ function required.

Details

See the Rusting faster: Simulation using Rcpp vignette.

Examples

See the examples in `ru_rcpp`.

`find_lambda` *Selecting the Box-Cox parameter for general d*

Description

Finds a value of the Box-Cox transformation parameter lambda for which the (positive) random variable with log-density $\log f$ has a density closer to that of a Gaussian random variable. In the following we use `theta` (θ) to denote the argument of $\log f$ on the original scale and `phi` (ϕ) on the Box-Cox transformed scale.

Usage

```
find_lambda(
  logf,
  ...,
  d = 1,
  n_grid = NULL,
  ep_bc = 1e-04,
  min_phi = rep(ep_bc, d),
  max_phi = rep(10, d),
  which_lam = 1:d,
  lambda_range = c(-3, 3),
  init_lambda = NULL,
```

```
  phi_to_theta = NULL,
  log_j = NULL
)
```

Arguments

logf	A function returning the log of the target density f.
...	further arguments to be passed to logf and related functions.
d	A numeric scalar. Dimension of f.
n_grid	A numeric scalar. Number of ordinates for each variable in phi. If this is not supplied a default value of ceiling(2501 ^ (1 / d)) is used.
ep_bc	A (positive) numeric scalar. Smallest possible value of phi to consider. Used to avoid negative values of phi.
min_phi, max_phi	Numeric vectors. Smallest and largest values of phi at which to evaluate logf, i.e. the range of values of phi over which to evaluate logf. Any components in min_phi that are not positive are set to ep_bc.
which_lam	A numeric vector. Contains the indices of the components of phi that ARE to be Box-Cox transformed.
lambda_range	A numeric vector of length 2. Range of lambda over which to optimise.
init_lambda	A numeric vector of length 1 or d. Initial value of lambda used in the search for the best lambda. If init_lambda is a scalar then rep(init_lambda, d) is used.
phi_to_theta	A function returning (inverse) of the transformation from theta to phi used to ensure positivity of phi prior to Box-Cox transformation. The argument is phi and the returned value is theta.
log_j	A function returning the log of the Jacobian of the transformation from theta to phi, i.e. based on derivatives of phi with respect to theta. Takes theta as its argument.

Details

The general idea is to evaluate the density f on a d-dimensional grid, with n_grid ordinates for each of the d variables. We treat each combination of the variables in the grid as a data point and perform an estimation of the Box-Cox transformation parameter lambda, in which each data point is weighted by the density at that point. The vectors min_phi and max_phi define the limits of the grid and which_lam can be used to specify that only certain components of phi are to be transformed.

Value

A list containing the following components

lambda	A numeric vector. The value of lambda.
gm	A numeric vector. Box-Cox scaling parameter, estimated by the geometric mean of the values of phi used in the optimisation to find the value of lambda, weighted by the values of f evaluated at phi.

init_psi	A numeric vector. An initial estimate of the mode of the Box-Cox transformed density
sd_psi	A numeric vector. Estimates of the marginal standard deviations of the Box-Cox transformed variables.
phi_to_theta	as detailed above (only if phi_to_theta is supplied)
log_j	as detailed above (only if log_j is supplied)

References

Box, G. and Cox, D. R. (1964) An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211-252.

Andrews, D. F. and Gnanadesikan, R. and Warner, J. L. (1971) Transformations of Multivariate Data, Biometrics, 27(4).

See Also

ru and ru_rcpp to perform ratio-of-uniforms sampling.

find_lambda_one_d and find_lambda_one_d_rcpp to produce (somewhat) automatically a list for the argument lambda of ru/ru_rcpp for the d = 1 case.

find_lambda_rcpp for a version of find_lambda that uses the Rcpp package to improve efficiency.

Examples

```
# Log-normal density ===================
# Note: the default value max_phi = 10 is OK here but this will not always
# be the case
lambda <- find_lambda(logf = dlnorm, log = TRUE)
lambda
x <- ru(logf = dlnorm, log = TRUE, d = 1, n = 1000, trans = "BC",
        lambda = lambda)

# Gamma density ===================
alpha <- 1
#  Choose a sensible value of max_phi
max_phi <- qgamma(0.999, shape = alpha)
# [Of course, typically the quantile function won't be available.  However,
# In practice the value of lambda chosen is quite insensitive to the choice
# of max_phi, provided that max_phi is not far too large or far too small.]

lambda <- find_lambda(logf = dgamma, shape = alpha, log = TRUE,
                      max_phi = max_phi)
lambda
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        trans = "BC", lambda = lambda)


# Generalized Pareto posterior distribution ===================

# Sample data from a GP(sigma, xi) distribution
```

```
gpd_data <- rgpd(m = 100, xi = -0.5, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
ss <- gpd_sum_stats(gpd_data)
# Calculate an initial estimate
init <- c(mean(gpd_data), 0)

n <- 1000
# Sample on original scale, with no rotation ----------------
x1 <- ru(logf = gpd_logpost, ss = ss, d = 2, n = n, init = init,
  lower = c(0, -Inf), rotate = FALSE)
plot(x1, xlab = "sigma", ylab = "xi")
# Parameter constraint line xi > -sigma/max(data)
# [This may not appear if the sample is far from the constraint.]
abline(a = 0, b = -1 / ss$xm)
summary(x1)

# Sample on original scale, with rotation ----------------
x2 <- ru(logf = gpd_logpost, ss = ss, d = 2, n = n, init = init,
  lower = c(0, -Inf))
plot(x2, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x2)

# Sample on Box-Cox transformed scale ----------------

# Find initial estimates for phi = (phi1, phi2),
# where phi1 = sigma
#   and phi2 = xi + sigma / max(x),
# and ranges of phi1 and phi2 over over which to evaluate
# the posterior to find a suitable value of lambda.
temp <- do.call(gpd_init, ss)
min_phi <- pmax(0, temp$init_phi - 2 * temp$se_phi)
max_phi <- pmax(0, temp$init_phi + 2 * temp$se_phi)

# Set phi_to_theta() that ensures positivity of phi
# We use phi1 = sigma and phi2 = xi + sigma / max(data)
phi_to_theta <- function(phi) c(phi[1], phi[2] - phi[1] / ss$xm)
log_j <- function(x) 0

lambda <- find_lambda(logf = gpd_logpost, ss = ss, d = 2, min_phi = min_phi,
  max_phi = max_phi, phi_to_theta = phi_to_theta, log_j = log_j)
lambda

# Sample on Box-Cox transformed, without rotation
x3 <- ru(logf = gpd_logpost, ss = ss, d = 2, n = n, trans = "BC",
  lambda = lambda, rotate = FALSE)
plot(x3, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x3)

# Sample on Box-Cox transformed, with rotation
x4 <- ru(logf = gpd_logpost, ss = ss, d = 2, n = n, trans = "BC",
  lambda = lambda)
```

```
plot(x4, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x4)

def_par <- graphics::par(no.readonly = TRUE)
par(mfrow = c(2,2), mar = c(4, 4, 1.5, 1))
plot(x1, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "mode relocation")
plot(x2, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "mode relocation and rotation")
plot(x3, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "Box-Cox and mode relocation")
plot(x4, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "Box-Cox, mode relocation and rotation")
graphics::par(def_par)
```

`find_lambda_one_d` *Selecting the Box-Cox parameter in the 1D case*

Description

Finds a value of the Box-Cox transformation parameter lambda (λ) for which the (positive univariate) random variable with log-density $\log f$ has a density closer to that of a Gaussian random variable. Works by estimating a set of quantiles of the distribution implied by $\log f$ and treating those quantiles as data in a standard Box-Cox analysis. In the following we use `theta` (θ) to denote the argument of $\log f$ on the original scale and `phi` (ϕ) on the Box-Cox transformed scale.

Usage

```
find_lambda_one_d(
  logf,
  ...,
  ep_bc = 1e-04,
  min_phi = ep_bc,
  max_phi = 10,
  num = 1001,
  xdiv = 100,
  probs = seq(0.01, 0.99, by = 0.01),
  lambda_range = c(-3, 3),
  phi_to_theta = NULL,
  log_j = NULL
)
```

Arguments

`logf`	A function returning the log of the target density f.
`...`	further arguments to be passed to `logf` and related functions.

- `ep_bc` — A (positive) numeric scalar. Smallest possible value of phi to consider. Used to avoid negative values of phi.
- `min_phi, max_phi` — Numeric scalars. Smallest and largest values of phi at which to evaluate logf, i.e., the range of values of phi over which to evaluate logf. Any components in min_phi that are not positive are set to ep_bc.
- `num` — A numeric scalar. Number of values at which to evaluate logf.
- `xdiv` — A numeric scalar. Only values of phi at which the density f is greater than the (maximum of f) / xdiv are used.
- `probs` — A numeric scalar. Probabilities at which to estimate the quantiles of that will be used as data to find lambda.
- `lambda_range` — A numeric vector of length 2. Range of lambda over which to optimise.
- `phi_to_theta` — A function returning (inverse) of the transformation from theta to phi used to ensure positivity of phi prior to Box-Cox transformation. The argument is phi and the returned value is theta.
- `log_j` — A function returning the log of the Jacobian of the transformation from theta to phi, i.e. based on derivatives of ϕ with respect to θ. Takes theta as its argument. If this is not supplied then a constant Jacobian is used.

Details

The general idea is to estimate quantiles of f corresponding to a set of equally-spaced probabilities in probs and to use these estimated quantiles as data in a standard estimation of the Box-Cox transformation parameter lambda.

The density f is first evaluated at num points equally spaced over the interval (min_phi, max_phi). The continuous density f is approximated by attaching trapezium-rule estimates of probabilities to the midpoints of the intervals between the points. After standardizing to account for the fact that f may not be normalized, (min_phi, max_phi) is reset so that values with small estimated probability (determined by xdiv) are excluded and the procedure is repeated on this new range. Then the required quantiles are estimated by inferring them from a weighted empirical distribution function based on treating the midpoints as data and the estimated probabilities at the midpoints as weights.

Value

A list containing the following components

- `lambda` — A numeric scalar. The value of lambda.
- `gm` — A numeric scalar. Box-Cox scaling parameter, estimated by the geometric mean of the quantiles used in the optimisation to find the value of lambda.
- `init_psi` — A numeric scalar. An initial estimate of the mode of the Box-Cox transformed density
- `sd_psi` — A numeric scalar. Estimates of the marginal standard deviations of the Box-Cox transformed variables.
- `phi_to_theta` — as detailed above (only if phi_to_theta is supplied)
- `log_j` — as detailed above (only if log_j is supplied)

References

Box, G. and Cox, D. R. (1964) An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211-252.

Andrews, D. F. and Gnanadesikan, R. and Warner, J. L. (1971) Transformations of Multivariate Data, Biometrics, 27(4).

See Also

ru and ru_rcpp to perform ratio-of-uniforms sampling.

find_lambda and find_lambda_rcpp to produce (somewhat) automatically a list for the argument lambda of ru/ru_rcpp for any value of d.

find_lambda_one_d_rcpp for a version of find_lambda_one_d that uses the Rcpp package to improve efficiency.

Examples

```
# Log-normal density ===================

# Note: the default value of max_phi = 10 is OK here but this will not
# always be the case.

lambda <- find_lambda_one_d(logf = dlnorm, log = TRUE)
lambda
x <- ru(logf = dlnorm, log = TRUE, d = 1, n = 1000, trans = "BC",
        lambda = lambda)

# Gamma density ===================

alpha <- 1
# Choose a sensible value of max_phi
max_phi <- qgamma(0.999, shape = alpha)
# [I appreciate that typically the quantile function won't be available.
# In practice the value of lambda chosen is quite insensitive to the choice
# of max_phi, provided that max_phi is not far too large or far too small.]

lambda <- find_lambda_one_d(logf = dgamma, shape = alpha, log = TRUE,
                            max_phi = max_phi)
lambda
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        trans = "BC", lambda = lambda)

alpha <- 0.1
# NB. for alpha < 1 the gamma(alpha, beta) density is not bounded
# So the ratio-of-uniforms emthod can't be used but it may work after a
# Box-Cox transformation.
# find_lambda_one_d() works much better than find_lambda() here.

max_phi <- qgamma(0.999, shape = alpha)
lambda <- find_lambda_one_d(logf = dgamma, shape = alpha, log = TRUE,
                            max_phi = max_phi)
```

```
lambda
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        trans = "BC", lambda = lambda)


plot(x)
plot(x, ru_scale = TRUE)
```

---

find_lambda_one_d_rcpp *Selecting the Box-Cox parameter in the 1D case using Rcpp*

Description

Finds a value of the Box-Cox transformation parameter lambda for which the (positive univariate) random variable with log-density $\log f$ has a density closer to that of a Gaussian random variable. Works by estimating a set of quantiles of the distribution implied by $\log f$ and treating those quantiles as data in a standard Box-Cox analysis. In the following we use `theta` (θ) to denote the argument of $\log f$ on the original scale and `phi` (ϕ) on the Box-Cox transformed scale.

Usage

```
find_lambda_one_d_rcpp(
  logf,
  ...,
  ep_bc = 1e-04,
  min_phi = ep_bc,
  max_phi = 10,
  num = 1001L,
  xdiv = 100,
  probs = seq(0.01, 0.99, by = 0.01),
  lambda_range = c(-3, 3),
  phi_to_theta = NULL,
  log_j = NULL,
  user_args = list()
)
```

Arguments

`logf` A pointer to a compiled C++ function returning the log of the target density f.

`...` further arguments to be passed to `logf` and related functions.

`ep_bc` A (positive) numeric scalar. Smallest possible value of `phi` to consider. Used to avoid negative values of `phi`.

`min_phi, max_phi` Numeric scalars. Smallest and largest values of `phi` at which to evaluate `logf`, i.e., the range of values of `phi` over which to evaluate `logf`. Any components in `min_phi` that are not positive are set to `ep_bc`.

num	A numeric scalar. Number of values at which to evaluate logf.
xdiv	A numeric scalar. Only values of phi at which the density f is greater than the (maximum of f) / xdiv are used.
probs	A numeric scalar. Probabilities at which to estimate the quantiles of that will be used as data to find lambda.
lambda_range	A numeric vector of length 2. Range of lambda over which to optimise.
phi_to_theta	A pointer to a compiled C++ function returning (the inverse) of the transformation from theta to phi used to ensure positivity of phi prior to Box-Cox transformation. The argument is phi and the returned value is theta. If phi_to_theta is undefined at the input value then the function should return NA.
log_j	A pointer to a compiled C++ function returning the log of the Jacobian of the transformation from theta to phi, i.e., based on derivatives of ϕ with respect to θ. Takes theta as its argument. If this is not supplied then a constant Jacobian is used.
user_args	A list of numeric components providing arguments to the user-supplied functions phi_to_theta and log_j.

Details

The general idea is to estimate quantiles of f corresponding to a set of equally-spaced probabilities in probs and to use these estimated quantiles as data in a standard estimation of the Box-Cox transformation parameter lambda.

The density f is first evaluated at num points equally spaced over the interval (min_phi, max_phi). The continuous density f is approximated by attaching trapezium-rule estimates of probabilities to the midpoints of the intervals between the points. After standardizing to account for the fact that f may not be normalized, (min_phi, max_phi) is reset so that values with small estimated probability (determined by xdiv) are excluded and the procedure is repeated on this new range. Then the required quantiles are estimated by inferring them from a weighted empirical distribution function based on treating the midpoints as data and the estimated probabilities at the midpoints as weights.

Value

A list containing the following components

lambda	A numeric scalar. The value of lambda.
gm	A numeric scalar. Box-Cox scaling parameter, estimated by the geometric mean of the quantiles used in the optimisation to find the value of lambda.
init_psi	A numeric scalar. An initial estimate of the mode of the Box-Cox transformed density
sd_psi	A numeric scalar. Estimates of the marginal standard deviations of the Box-Cox transformed variables.
phi_to_theta	as detailed above (only if phi_to_theta is supplied)
log_j	as detailed above (only if log_j is supplied)
user_args	as detailed above (only if user_args is supplied)

References

Box, G. and Cox, D. R. (1964) An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211-252.

Andrews, D. F. and Gnanadesikan, R. and Warner, J. L. (1971) Transformations of Multivariate Data, Biometrics, 27(4).

Eddelbuettel, D. and Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, **40**(8), 1-18. doi:10.18637/jss.v040.i08

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*, Springer, New York. ISBN 978-1-4614-6867-7.

See Also

`ru_rcpp` to perform ratio-of-uniforms sampling.

`find_lambda_rcpp` to produce (somewhat) automatically a list for the argument `lambda` of `ru` for any value of d.

Examples

```
# Log-normal density ===================

# Note: the default value of max_phi = 10 is OK here but this will not
# always be the case.

ptr_lnorm <- create_xptr("logdlnorm")
mu <- 0
sigma <- 1
lambda <- find_lambda_one_d_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma)
lambda
x <- ru_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma, log = TRUE, d = 1,
             n = 1000, trans = "BC", lambda = lambda)

# Gamma density ===================

alpha <- 1
# Choose a sensible value of max_phi
max_phi <- qgamma(0.999, shape = alpha)
# [I appreciate that typically the quantile function won't be available.
# In practice the value of lambda chosen is quite insensitive to the choice
# of max_phi, provided that max_phi is not far too large or far too small.]

ptr_gam <- create_xptr("logdgamma")
lambda <- find_lambda_one_d_rcpp(logf = ptr_gam, alpha = alpha,
                                 max_phi = max_phi)
lambda
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = 1000, trans = "BC",
             lambda = lambda)

alpha <- 0.1
# NB. for alpha < 1 the gamma(alpha, beta) density is not bounded
# So the ratio-of-uniforms emthod can't be used but it may work after a
```

```
# Box-Cox transformation.
# find_lambda_one_d() works much better than find_lambda() here.

max_phi <- qgamma(0.999, shape = alpha)
lambda <- find_lambda_one_d_rcpp(logf = ptr_gam, alpha = alpha,
                                 max_phi = max_phi)
lambda
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = 1000, trans = "BC",
             lambda = lambda)

plot(x)
plot(x, ru_scale = TRUE)
```

`find_lambda_rcpp` *Selecting the Box-Cox parameter for general d using Rcpp*

Description

Finds a value of the Box-Cox transformation parameter lambda for which the (positive) random variable with log-density $\log f$ has a density closer to that of a Gaussian random variable. In the following we use `theta` (θ) to denote the argument of `logf` on the original scale and `phi` (ϕ) on the Box-Cox transformed scale.

Usage

```
find_lambda_rcpp(
  logf,
  ...,
  d = 1,
  n_grid = NULL,
  ep_bc = 1e-04,
  min_phi = rep(ep_bc, d),
  max_phi = rep(10, d),
  which_lam = 1:d,
  lambda_range = c(-3, 3),
  init_lambda = NULL,
  phi_to_theta = NULL,
  log_j = NULL,
  user_args = list()
)
```

Arguments

`logf`	A pointer to a compiled C++ function returning the log of the target density f.
`...`	further arguments to be passed to `logf` and related functions.
`d`	A numeric scalar. Dimension of f.

n_grid — A numeric scalar. Number of ordinates for each variable in phi. If this is not supplied a default value of ceiling(2501 ^ (1 / d)) is used.

ep_bc — A (positive) numeric scalar. Smallest possible value of phi to consider. Used to avoid negative values of phi.

min_phi, max_phi — Numeric vectors. Smallest and largest values of phi at which to evaluate logf, i.e., the range of values of phi over which to evaluate logf. Any components in min_phi that are not positive are set to ep_bc.

which_lam — A numeric vector. Contains the indices of the components of phi that ARE to be Box-Cox transformed.

lambda_range — A numeric vector of length 2. Range of lambda over which to optimise.

init_lambda — A numeric vector of length 1 or d. Initial value of lambda used in the search for the best lambda. If init_lambda is a scalar then rep(init_lambda, d) is used.

phi_to_theta — A pointer to a compiled C++ function returning (the inverse) of the transformation from theta to phi used to ensure positivity of phi prior to Box-Cox transformation. The argument is phi and the returned value is theta. If phi_to_theta is undefined at the input value then the function should return NA.

log_j — A pointer to a compiled C++ function returning the log of the Jacobian of the transformation from theta to phi, i.e., based on derivatives of phi with respect to $theta$. Takes theta as its argument.

user_args — A list of numeric components providing arguments to the user-supplied functions phi_to_theta and log_j.

Details

The general idea is to evaluate the density f on a d-dimensional grid, with n_grid ordinates for each of the d variables. We treat each combination of the variables in the grid as a data point and perform an estimation of the Box-Cox transformation parameter lambda, in which each data point is weighted by the density at that point. The vectors min_phi and max_phi define the limits of the grid and which_lam can be used to specify that only certain components of phi are to be transformed.

Value

A list containing the following components

lambda — A numeric vector. The value of lambda.

gm — A numeric vector. Box-Cox scaling parameter, estimated by the geometric mean of the values of phi used in the optimisation to find the value of lambda, weighted by the values of f evaluated at phi.

init_psi — A numeric vector. An initial estimate of the mode of the Box-Cox transformed density

sd_psi — A numeric vector. Estimates of the marginal standard deviations of the Box-Cox transformed variables.

phi_to_theta — as detailed above (only if phi_to_theta is supplied)

log_j — as detailed above (only if log_j is supplied)

user_args — as detailed above (only if user_args is supplied)

References

Box, G. and Cox, D. R. (1964) An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211-252.

Andrews, D. F. and Gnanadesikan, R. and Warner, J. L. (1971) Transformations of Multivariate Data, Biometrics, 27(4).

Eddelbuettel, D. and Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, **40**(8), 1-18. doi:10.18637/jss.v040.i08

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*, Springer, New York. ISBN 978-1-4614-6867-7.

See Also

`ru_rcpp` to perform ratio-of-uniforms sampling.

`find_lambda_one_d_rcpp` to produce (somewhat) automatically a list for the argument `lambda` of `ru` for the d = 1 case.

Examples

```
# Log-normal density ===================
# Note: the default value max_phi = 10 is OK here but this will not always
# be the case
ptr_lnorm <- create_xptr("logdlnorm")
mu <- 0
sigma <- 1
lambda <- find_lambda_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma)
lambda
x <- ru_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma, d = 1, n = 1000,
             trans = "BC", lambda = lambda)

# Gamma density ===================
alpha <- 1
#  Choose a sensible value of max_phi
max_phi <- qgamma(0.999, shape = alpha)
# [Of course, typically the quantile function won't be available.  However,
# In practice the value of lambda chosen is quite insensitive to the choice
# of max_phi, provided that max_phi is not far too large or far too small.]

ptr_gam <- create_xptr("logdgamma")
lambda <- find_lambda_rcpp(logf = ptr_gam, alpha = alpha, max_phi = max_phi)
lambda
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = 1000, trans = "BC",
             lambda = lambda)


# Generalized Pareto posterior distribution ===================

n <- 1000
# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = -0.5, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
```

```
ss <- gpd_sum_stats(gpd_data)
# Calculate an initial estimate
init <- c(mean(gpd_data), 0)

n <- 1000
# Sample on original scale, with no rotation ----------------
ptr_gp <- create_xptr("loggp")
for_ru_rcpp <- c(list(logf = ptr_gp, init = init, d = 2, n = n,
                     lower = c(0, -Inf)), ss, rotate = FALSE)
x1 <- do.call(ru_rcpp, for_ru_rcpp)
plot(x1, xlab = "sigma", ylab = "xi")
# Parameter constraint line xi > -sigma/max(data)
# [This may not appear if the sample is far from the constraint.]
abline(a = 0, b = -1 / ss$xm)
summary(x1)

# Sample on original scale, with rotation ----------------
for_ru_rcpp <- c(list(logf = ptr_gp, init = init, d = 2, n = n,
                      lower = c(0, -Inf)), ss)
x2 <- do.call(ru_rcpp, for_ru_rcpp)
plot(x2, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x2)

# Sample on Box-Cox transformed scale ----------------

# Find initial estimates for phi = (phi1, phi2),
# where phi1 = sigma
#   and phi2 = xi + sigma / max(x),
# and ranges of phi1 and phi2 over over which to evaluate
# the posterior to find a suitable value of lambda.
temp <- do.call(gpd_init, ss)
min_phi <- pmax(0, temp$init_phi - 2 * temp$se_phi)
max_phi <- pmax(0, temp$init_phi + 2 * temp$se_phi)

# Set phi_to_theta() that ensures positivity of phi
# We use phi1 = sigma and phi2 = xi + sigma / max(data)

# Create an external pointer to this C++ function
ptr_phi_to_theta_gp <- create_phi_to_theta_xptr("gp")
# Note: log_j is set to zero by default inside find_lambda_rcpp()
lambda <- find_lambda_rcpp(logf = ptr_gp, ss = ss, d = 2, min_phi = min_phi,
                           max_phi = max_phi, user_args = list(xm = ss$xm),
                           phi_to_theta = ptr_phi_to_theta_gp)
lambda

# Sample on Box-Cox transformed, without rotation
x3 <- ru_rcpp(logf = ptr_gp, ss = ss, d = 2, n = n, trans = "BC",
              lambda = lambda, rotate = FALSE)
plot(x3, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x3)
```

```
# Sample on Box-Cox transformed, with rotation
x4 <- ru_rcpp(logf = ptr_gp, ss = ss, d = 2, n = n, trans = "BC",
              lambda = lambda)
plot(x4, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x4)

def_par <- graphics::par(no.readonly = TRUE)
par(mfrow = c(2,2), mar = c(4, 4, 1.5, 1))
plot(x1, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "mode relocation")
plot(x2, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "mode relocation and rotation")
plot(x3, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "Box-Cox and mode relocation")
plot(x4, xlab = "sigma", ylab = "xi", ru_scale = TRUE,
  main = "Box-Cox, mode relocation and rotation")
graphics::par(def_par)
```

---

gpd_init *Initial estimates for Generalized Pareto parameters*

Description

Calculates initial estimates and estimated standard errors (SEs) for the generalized Pareto parameters σ and ξ based on an assumed random sample from this distribution. Also, calculates initial estimates and estimated standard errors for

$\phi_1 = \sigma$ and $\phi_1 = \xi + \sigma x_{(m)}$, where $x_{(m)}$ is the sample maximum threshold exceedance.

Usage

```
gpd_init(gpd_data, m, xm, sum_gp = NULL, xi_eq_zero = FALSE, init_ests = NULL)
```

Arguments

gpd_data	A numeric vector containing positive sample values.
m	A numeric scalar. The sample size, i.e., the length of gpd_data.
xm	A numeric scalar. The sample maximum.
sum_gp	A numeric scalar. The sum of the sample values.
xi_eq_zero	A logical scalar. If TRUE assume that the shape parameter $\xi = 0$.
init_ests	A numeric vector. Initial estimate of $\theta = (\sigma, \xi)$. If supplied gpd_init() returns the corresponding initial estimate of $\phi = (\phi_1, \phi_2)$.

Details

The main aim is to calculate an admissible estimate of θ, i.e., one at which the log-likelihood is finite (necessary for the posterior log-density to be finite) at the estimate, and associated estimated SEs. These are converted into estimates and SEs for ϕ. The latter can be used to set values of `min_phi` and `max_phi` for input to `find_lambda`.

In the default setting (`xi_eq_zero = FALSE` and `init_ests = NULL`) the methods tried are Maximum Likelihood Estimation (MLE) (Grimshaw, 1993), Probability-Weighted Moments (PWM) (Hosking and Wallis, 1987) and Linear Combinations of Ratios of Spacings (LRS) (Reiss and Thomas, 2007, page 134) in that order.

For $\xi < -1$ the likelihood is unbounded, MLE may fail when ξ is not greater than -0.5 and the observed Fisher information for (σ, ξ) has finite variance only if $\xi > -0.25$. We use the ML estimate provided that the estimate of ξ returned from `gpd_mle` is greater than -1. We only use the SE if the MLE of ξ is greater than -0.25.

If either the MLE or the SE are not OK then we try PWM. We use the PWM estimate only if is admissible, and the MLE was not OK. We use the PWM SE, but this will be `c(NA, NA)` if the PWM estimate of ξ is $> 1/2$. If the estimate is still not OK then we try LRS. As a last resort, which will tend to occur only when ξ is strongly negative, we set $\xi = -1$ and estimate sigma conditional on this.

Value

If `init_ests` is not supplied by the user, a list is returned with components

- `init` A numeric vector. Initial estimates of σ and ξ.
- `se` A numeric vector. Estimated standard errors of σ and ξ.
- `init_phi` A numeric vector. Initial estimates of $\phi_1 = \sigma$ and $\phi_1 = \xi + \sigma x_{(m)}$ where $x_{(m)}$ is the maximum of `gpd_data`.
- `se_phi` A numeric vector. Estimated standard errors of ϕ_1 and ϕ_2.

If `init_ests` is supplied then only the numeric vector `init_phi` is returned.

References

Grimshaw, S. D. (1993) Computing Maximum Likelihood Estimates for the Generalized Pareto Distribution. Technometrics, 35(2), 185-191. and Computing (1991) 1, 129-133. doi:10.1007/BF01889987.

Hosking, J. R. M. and Wallis, J. R. (1987) Parameter and Quantile Estimation for the Generalized Pareto Distribution. Technometrics, 29(3), 339-349. doi:10.2307/1269343.

Reiss, R.-D., Thomas, M. (2007) Statistical Analysis of Extreme Values with Applications to Insurance, Finance, Hydrology and Other Fields.Birkhauser. doi:10.1007/9783764373993.

See Also

`gpd_sum_stats` to calculate summary statistics for use in `gpd_loglik`.

`rgpd` for simulation from a generalized Pareto

`find_lambda` to produce (somewhat) automatically a list for the argument `lambda` of `ru`.

Examples

```
# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = 0, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
ss <- gpd_sum_stats(gpd_data)
# Calculate initial estimates
do.call(gpd_init, ss)
```

`gpd_logpost` *Generalized Pareto posterior log-density*

Description

Calculates the generalized Pareto posterior log-density based on a particular prior for the generalized Pareto parameters, a Maximal Data Information (MDI) prior truncated to $\xi \geq -1$ in order to produce a posterior density that is proper.

Usage

```
gpd_logpost(pars, ss)
```

Arguments

`pars`	A numeric vector containing the values of the generalized Pareto parameters σ and ξ.
`ss`	A numeric list. Summary statistics to be passed to the generalized Pareto log-likelihood. Calculated using `gpd_sum_stats`

Value

A numeric scalar. The value of the log-likelihood.

References

Northrop, P. J. and Attalides, N. (2016) Posterior propriety in Bayesian extreme value analyses using reference priors. Statistica Sinica, 26(2), 721-743, doi:10.5705/ss.2014.034.

See Also

`gpd_sum_stats` to calculate summary statistics for use in `gpd_loglik`.

`rgpd` for simulation from a generalized Pareto

Examples

```
# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = 0, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
ss <- gpd_sum_stats(gpd_data)
# Calculate the generalized Pareto log-posterior
gpd_logpost(pars = c(1, 0), ss = ss)
```

`gpd_sum_stats` *Generalized Pareto summary statistics*

Description

Calculates summary statistics involved in the Generalized Pareto log-likelihood.

Usage

```
gpd_sum_stats(gpd_data)
```

Arguments

`gpd_data`	A numeric vector containing positive values.

Value

A list with components

`gpd_data`	A numeric vector. The input vector with any missings removed.
`m`	A numeric scalar. The sample size, i.e., the number of non-missing values.
`xm`	A numeric scalar. The sample maximum
`sum_gp`	A numeric scalar. The sum of the non-missing sample values.

See Also

`rgpd` for simulation from a generalized Pareto distribution.

Examples

```
# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = 0, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
ss <- gpd_sum_stats(gpd_data)
```

plot.ru *Plot diagnostics for an ru object*

Description

plot method for class "ru". For d = 1 a histogram of the simulated values is plotted with a the density function superimposed. The density is normalized crudely using the trapezium rule. For d = 2 a scatter plot of the simulated values is produced with density contours superimposed. For d > 2 pairwise plots of the simulated values are produced.

Usage

```
## S3 method for class 'ru'
plot(
  x,
  y,
  ...,
  n = ifelse(x$d == 1, 1001, 101),
  prob = c(0.1, 0.25, 0.5, 0.75, 0.95, 0.99),
  ru_scale = FALSE,
  rows = NULL,
  xlabs = NULL,
  ylabs = NULL,
  var_names = NULL,
  points_par = list(col = 8)
)
```

Arguments

x	an object of class "ru", a result of a call to ru.
y	Not used.
...	Additional arguments passed on to hist, lines, contour or points.
n	A numeric scalar. Only relevant if x$d = 1 or x$d = 2. The meaning depends on the value of x$d. • For d = 1 : n + 1 is the number of abscissae in the trapezium method used to normalize the density. • For d = 2 : an n by n regular grid is used to contour the density.
prob	Numeric vector. Only relevant for d = 2. The contour lines are drawn such that the respective probabilities that the variable lies within the contour are approximately equal to the values in prob.
ru_scale	A logical scalar. Should we plot data and density on the scale used in the ratio-of-uniforms algorithm (TRUE) or on the original scale (FALSE)?
rows	A numeric scalar. When d > 2 this sets the number of rows of plots. If the user doesn't provide this then it is set internally.

`xlabs, ylabs` Numeric vectors. When `d > 2` these set the labels on the x and y axes respectively. If the user doesn't provide these then the column names of the simulated data matrix to be plotted are used.

`var_names` A character (or numeric) vector of length `x$d`. This argument can be used to replace variable names set using `var_names` in the call to `ru` or `ru_rcpp`.

`points_par` A list of arguments to pass to `points` to control the appearance of points depicting the simulated values. Only relevant when `d = 2`.

Value

No return value, only the plot is produced.

See Also

`summary.ru` for summaries of the simulated values and properties of the ratio-of-uniforms algorithm.

Examples

```
# Log-normal density ----------------
x <- ru(logf = dlnorm, log = TRUE, d = 1, n = 1000, lower = 0, init = 1)

plot(x)

# Improve appearance using arguments to plot() and hist()

plot(x, breaks = seq(0, ceiling(max(x$sim_vals)), by = 0.25),
  xlim = c(0, 10))

# Two-dimensional normal with positive association ----------------
rho <- 0.9
covmat <- matrix(c(1, rho, rho, 1), 2, 2)
log_dmvnorm <- function(x, mean = rep(0, d), sigma = diag(d)) {
  x <- matrix(x, ncol = length(x))
  d <- ncol(x)
  - 0.5 * (x - mean) %*% solve(sigma) %*% t(x - mean)
}
x <- ru(logf = log_dmvnorm, sigma = covmat, d = 2, n = 1000, init = c(0, 0))

plot(x)
```

`print.ru` *Print method for an "ru" object*

Description

`print` method for class `"ru"`.

Usage

```
## S3 method for class 'ru'
print(x, ...)
```

Arguments

x	an object of class "ru", a result of a call to ru or ru_rcpp.
...	Additional arguments. None are used in this function.

Details

Simply prints the call to ru or ru_rcpp.

Value

The argument x, invisibly.

See Also

summary.ru for summaries of the simulated values and properties of the ratio-of-uniforms algorithm.

plot.ru for a diagnostic plot.

---

rgpd — *Generalized Pareto simulation*

Description

Simulates a sample of size m from a generalized Pareto distribution.

Usage

```
rgpd(m = 1, sigma = 1, xi = 0)
```

Arguments

m	A numeric scalar. The size of sample required.
sigma	A numeric scalar. The generalized Pareto scale parameter σ.
xi	A numeric scalar. The generalized Pareto shape parameter ξ.

Value

A numeric vector. A generalized Pareto sample of size m.

Examples

```
# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = 0, sigma = 1)
```

ru *Generalized ratio-of-uniforms sampling*

Description

Uses the generalized ratio-of-uniforms method to simulate from a distribution with log-density $\log f$ (up to an additive constant). The density f must be bounded, perhaps after a transformation of variable.

Usage

```
ru(
  logf,
  ...,
  n = 1,
  d = 1,
  init = NULL,
  mode = NULL,
  trans = c("none", "BC", "user"),
  phi_to_theta = NULL,
  log_j = NULL,
  user_args = list(),
  lambda = rep(1L, d),
  lambda_tol = 1e-06,
  gm = NULL,
  rotate = ifelse(d == 1, FALSE, TRUE),
  lower = rep(-Inf, d),
  upper = rep(Inf, d),
  r = 1/2,
  ep = 0L,
  a_algor = if (d == 1) "nlminb" else "optim",
  b_algor = c("nlminb", "optim"),
  a_method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  b_method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  a_control = list(),
  b_control = list(),
  var_names = NULL,
  shoof = 0.2
)
```

Arguments

`logf` A function returning the log of the target density f evaluated at its first argument. This function should return `-Inf` when the density is zero. It is better to use `logf =` explicitly, for example, `ru(logf = dnorm, log = TRUE, init = 0.1)`, to avoid argument matching problems. In contrast, `ru(dnorm, log =`

`TRUE, init = 0.1)` will throw an error because partial matching results in `logf` being matched to `log = TRUE`.

`...` Further arguments to be passed to `logf` and related functions.

`n` A non-negative integer scalar. The number of simulated values required. If `n = 0` then no simulation is performed but the component box in the returned object gives the ratio-of-uniforms bounding box that would have been used.

`d` A positive integer scalar. The dimension of f.

`init` A numeric vector of length `d`. Initial estimate of the mode of `logf`. If `trans = "BC"` or `trans = "user"` this is *after* Box-Cox transformation or user-defined transformation, but *before* any rotation of axes. If `init` is not supplied then `rep(1, d)` is used. If `length(init) = 1` and `d > 1` then `init <- rep(init, length.out = d)` is used.

`mode` A numeric vector of length `d`. The mode of `logf`. If `trans = "BC"` or `trans = "user"` this is *after* Box-Cox transformation or user-defined transformation, but *before* any rotation of axes. Only supply `mode` if the mode is known: it will not be checked. If `mode` is supplied then `init` is ignored.

`trans` A character scalar. `trans = "none"` for no transformation, `trans = "BC"` for Box-Cox transformation, `trans = "user"` for a user-defined transformation. If `trans = "user"` then the transformation should be specified using `phi_to_theta` and `log_j` and `user_args` may be used to pass arguments to `phi_to_theta` and `log_j`. See **Details** and the **Examples**.

`phi_to_theta` A function returning (the inverse) of the transformation from `theta` (θ) to `phi` (ϕ) that may be used to ensure positivity of ϕ prior to Box-Cox transformation. The argument is `phi` and the returned value is `theta`. If `phi_to_theta` is undefined at the input value then the function should return `NA`. See **Details**. If `lambda$phi_to_theta` (see argument `lambda` below) is supplied then this is used instead of any function supplied via `phi_to_theta`.

`log_j` A function returning the log of the Jacobian of the transformation from `theta` (θ) to `phi` (ϕ), i.e., based on derivatives of ϕ with respect to θ. Takes `theta` as its argument. If `lambda$log_j` (see argument `lambda` below) is supplied then this is used instead of any function supplied via `log_j`.

`user_args` A list of numeric components. If `trans = "user"` then `user_args` is a list providing arguments to the user-supplied functions `phi_to_theta` and `log_j`.

`lambda` Either

- A numeric vector. Box-Cox transformation parameters, or
- A list with components

  lambda A numeric vector. Box-Cox parameters (required).

  gm A numeric vector. Box-Cox scaling parameters (optional). If supplied this overrides any `gm` supplied by the individual `gm` argument described below.

  init_psi A numeric vector. Initial estimate of mode *after* Box-Cox transformation (optional).

  sd_psi A numeric vector. Estimates of the marginal standard deviations of the Box-Cox transformed variables (optional).

phi_to_theta As above (optional).

log_j As above (optional).

This list may be created using find_lambda_one_d (for d = 1) or find_lambda (for any d).

lambda_tol
: A numeric scalar. Any values in lambda that are less than lambda_tol in magnitude are set to zero.

gm
: A numeric vector. Box-Cox scaling parameters (optional). If lambda$gm is supplied in input list lambda then lambda$gm is used, not gm.

rotate
: A logical scalar. If TRUE (d > 1 only) use Choleski rotation. If d = 1 and rotate = TRUE then rotate will be set to FALSE with a warning. See **Details**.

lower, upper
: Numeric vectors. Lower/upper bounds on the arguments of the function *after* any transformation from theta to phi implied by the inverse of phi_to_theta. If rotate = FALSE these are used in all of the optimisations used to construct the bounding box. If rotate = TRUE then they are use only in the first optimisation to maximise the target density.‘ If trans = "BC" components of lower that are negative are set to zero without warning and the bounds implied after the Box-Cox transformation are calculated inside ru.

r
: A numeric scalar. Parameter of generalized ratio-of-uniforms.

ep
: A numeric scalar. Controls initial estimates for optimisations to find the b-bounding box parameters. The default (ep = 0) corresponds to starting at the mode of logf small positive values of ep move the constrained variable slightly away from the mode in the correct direction. If ep is negative its absolute value is used, with no warning given.

a_algor, b_algor
: Character scalars. Either "nlminb" or "optim". Respective optimisation algorithms used to find $a(r)$ and $(b_i^-(r), b_i^+(r))$.

a_method, b_method
: Character scalars. Respective methods used by optim to find $a(r)$ and $(b_i^-(r), b_i^+(r))$. Only used if optim is the chosen algorithm. If d = 1 then a_method and b_method are set to "Brent" without warning.

a_control, b_control
: Lists of control arguments to optim or nlminb to find $a(r)$ and $(b_i^-(r), b_i^+(r))$ respectively.

var_names
: A character (or numeric) vector of length d. Names to give to the column(s) of the simulated values.

shoof
: A numeric scalar in [0, 1]. Sometimes a spurious non-zero convergence indicator is returned from optim or nlminb). In this event we try to check that a minimum has indeed been found using different algorithm. shoof controls the starting value provided to this algorithm. If shoof = 0 then we start from the current solution. If shoof = 1 then we start from the initial estimate provided to the previous minimisation. Otherwise, shoof interpolates between these two extremes, with a value close to zero giving a starting value that is close to the current solution. The exception to this is when the initial and current solutions are equal. Then we start from the current solution multiplied by 1 - shoof.

Details

For information about the generalised ratio-of-uniforms method and transformations see the Introducing rust vignette. This can also be accessed using `vignette("rust-a-vignette", package = "rust")`.

If `trans = "none"` and `rotate = FALSE` then `ru` implements the (multivariate) generalized ratio of uniforms method described in Wakefield, Gelfand and Smith (1991) using a target density whose mode is relocated to the origin ('mode relocation') in the hope of increasing efficiency.

If `trans = "BC"` then marginal Box-Cox transformations of each of the `d` variables is performed, with parameters supplied in `lambda`. The function `phi_to_theta` may be used, if necessary, to ensure positivity of the variables prior to Box-Cox transformation.

If `trans = "user"` then the function `phi_to_theta` enables the user to specify their own transformation.

In all cases the mode of the target function is relocated to the origin *after* any user-supplied transformation and/or Box-Cox transformation.

If `d` is greater than one and `rotate = TRUE` then a rotation of the variable axes is performed *after* mode relocation. The rotation is based on the Choleski decomposition (see chol) of the estimated Hessian (computed using `optimHess` of the negated log-density after any user-supplied transformation or Box-Cox transformation. If any of the eigenvalues of the estimated Hessian are non-positive (which may indicate that the estimated mode of `logf` is close to a variable boundary) then `rotate` is set to `FALSE` with a warning. A warning is also given if this happens when `d = 1`.

The default value of the tuning parameter `r` is 1/2, which is likely to be close to optimal in many cases, particularly if `trans = "BC"`.

Value

An object of class `"ru"` is a list containing the following components:

`sim_vals` An `n` by `d` matrix of simulated values.

`box` A (2 * `d` + 1) by `d` + 2 matrix of ratio-of-uniforms bounding box information, with row names indicating the box parameter. The columns contain

- **column 1** values of box parameters.
- **columns 2 to (2+`d`-1)** values of variables at which these box parameters are obtained.
- **column 2+`d`** convergence indicators.

Scaling of f within `ru` and relocation of the mode to the origin means that the first row of `box` will always be `c(1, rep(0, d))`.

`pa` A numeric scalar. An estimate of the probability of acceptance.

`r` The value of `r`.

`d` The value of `d`.

`logf` A function. `logf` supplied by the user, but with f scaled by the maximum of the target density used in the ratio-of-uniforms method (i.e. `logf_rho`), to avoid numerical problems in contouring f in `plot.ru` when `d = 2`.

`logf_rho` A function. The target function actually used in the ratio-of-uniforms algorithm.

sim_vals_rho	An n by d matrix of values simulated from the function used in the ratio-of-uniforms algorithm.
logf_args	A list of further arguments to logf.
f_mode	The estimated mode of the target density f, after any Box-Cox transformation and/or user supplied transformation, but before mode relocation.
trans_fn	An R function that performs the inverse transformation from the transformed variable ρ, on which the generalised ratio-of-uniforms method is performed, back to the original variable θ. **Note**: trans_fn is **not** vectorised with respect to ρ.

References

Wakefield, J. C., Gelfand, A. E. and Smith, A. F. M. (1991) Efficient generation of random variates via the ratio-of-uniforms method. *Statistics and Computing* (1991), **1**, 129-133. doi:10.1007/BF01889987.

See Also

ru_rcpp for a version of ru that uses the Rcpp package to improve efficiency.

summary.ru for summaries of the simulated values and properties of the ratio-of-uniforms algorithm.

plot.ru for a diagnostic plot.

find_lambda_one_d to produce (somewhat) automatically a list for the argument lambda of ru for the d = 1 case.

find_lambda to produce (somewhat) automatically a list for the argument lambda of ru for any value of d.

optim for choices of the arguments a_method, b_method, a_control and b_control.

nlminb for choices of the arguments a_control and b_control.

optimHess for Hessian estimation.

chol for the Choleski decomposition.

Examples

```
# Normal density ===================

# One-dimensional standard normal ----------------
x <- ru(logf = function(x) -x ^ 2 / 2, d = 1, n = 1000, init = 0.1)

# Two-dimensional standard normal ----------------
x <- ru(logf = function(x) -(x[1]^2 + x[2]^2) / 2, d = 2, n = 1000,
        init = c(0, 0))

# Two-dimensional normal with positive association ----------------
rho <- 0.9
covmat <- matrix(c(1, rho, rho, 1), 2, 2)
log_dmvnorm <- function(x, mean = rep(0, d), sigma = diag(d)) {
  x <- matrix(x, ncol = length(x))
```

```
  d <- ncol(x)
  - 0.5 * (x - mean) %*% solve(sigma) %*% t(x - mean)
}

# No rotation.
x <- ru(logf = log_dmvnorm, sigma = covmat, d = 2, n = 1000, init = c(0, 0),
        rotate = FALSE)

# With rotation.
x <- ru(logf = log_dmvnorm, sigma = covmat, d = 2, n = 1000, init = c(0, 0))

# three-dimensional normal with positive association ----------------
covmat <- matrix(rho, 3, 3) + diag(1 - rho, 3)

# No rotation.  Slow !
x <- ru(logf = log_dmvnorm, sigma = covmat, d = 3, n = 1000,
        init = c(0, 0, 0), rotate = FALSE)

# With rotation.
x <- ru(logf = log_dmvnorm, sigma = covmat, d = 3, n = 1000,
        init = c(0, 0, 0))

# Log-normal density ===================

# Sampling on original scale ----------------
x <- ru(logf = dlnorm, log = TRUE, d = 1, n = 1000, lower = 0, init = 1)

# Box-Cox transform with lambda = 0 ----------------
lambda <- 0
x <- ru(logf = dlnorm, log = TRUE, d = 1, n = 1000, lower = 0, init = 0.1,
        trans = "BC", lambda = lambda)

# Equivalently, we could use trans = "user" and supply the (inverse) Box-Cox
# transformation and the log-Jacobian by hand
x <- ru(logf = dlnorm, log = TRUE, d = 1, n = 1000, init = 0.1,
        trans = "user", phi_to_theta = function(x) exp(x),
        log_j = function(x) -log(x))

# Gamma(alpha, 1) density ===================

# Note: the gamma density in unbounded when its shape parameter is < 1.
# Therefore, we can only use trans="none" if the shape parameter is >= 1.

# Sampling on original scale ----------------

alpha <- 10
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        lower = 0, init = alpha)

alpha <- 1
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        lower = 0, init = alpha)
```

```
# Box-Cox transform with lambda = 1/3 works well for shape >= 1. -----------

alpha <- 1
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        trans = "BC", lambda = 1/3, init = alpha)
summary(x)

# Equivalently, we could use trans = "user" and supply the (inverse) Box-Cox
# transformation and the log-Jacobian by hand

# Note: when phi_to_theta is undefined at x this function returns NA
phi_to_theta  <- function(x, lambda) {
  ifelse(x * lambda + 1 > 0, (x * lambda + 1) ^ (1 / lambda), NA)
}
log_j <- function(x, lambda) (lambda - 1) * log(x)
lambda <- 1/3
x <- ru(logf = dgamma, shape = alpha, log = TRUE, d = 1, n = 1000,
        trans = "user", phi_to_theta = phi_to_theta, log_j = log_j,
        user_args = list(lambda = lambda), init = alpha)
summary(x)


# Generalized Pareto posterior distribution ===================

# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = -0.5, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
ss <- gpd_sum_stats(gpd_data)
# Calculate an initial estimate
init <- c(mean(gpd_data), 0)

# Mode relocation only ----------------
n <- 1000
x1 <- ru(logf = gpd_logpost, ss = ss, d = 2, n = n, init = init,
         lower = c(0, -Inf), rotate = FALSE)
plot(x1, xlab = "sigma", ylab = "xi")
# Parameter constraint line xi > -sigma/max(data)
# [This may not appear if the sample is far from the constraint.]
abline(a = 0, b = -1 / ss$xm)
summary(x1)

# Rotation of axes plus mode relocation ----------------
x2 <- ru(logf = gpd_logpost, ss = ss, d = 2, n = n, init = init,
         lower = c(0, -Inf))
plot(x2, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x2)

# Cauchy ========================

# The bounding box cannot be constructed if r < 1.  For r = 1 the
# bounding box parameters b1-(r) and b1+(r) are attained in the limits
# as x decreases/increases to infinity respectively.  This is fine in
```

```
# theory but using r > 1 avoids this problem and the largest probability
# of acceptance is obtained for r approximately equal to 1.26.

res <- ru(logf = dcauchy, log = TRUE, init = 0, r = 1.26, n = 1000)

# Half-Cauchy ===================

log_halfcauchy <- function(x) {
  return(ifelse(x < 0, -Inf, dcauchy(x, log = TRUE)))
}

# Like the Cauchy case the bounding box cannot be constructed if r < 1.
# We could use r > 1 but the mode is on the edge of the support of the
# density so as an alternative we use a log transformation.

x <- ru(logf = log_halfcauchy, init = 0, trans = "BC", lambda = 0, n = 1000)
x$pa
plot(x, ru_scale = TRUE)

# Example 4 from Wakefield et al. (1991) ===================

# Bivariate normal x bivariate student-t
log_norm_t <- function(x, mean = rep(0, d), sigma1 = diag(d), sigma2 = diag(d)) {
  x <- matrix(x, ncol = length(x))
  log_h1 <- -0.5 * (x - mean) %*% solve(sigma1) %*% t(x - mean)
  log_h2 <- -2 * log(1 + 0.5 * x %*% solve(sigma2) %*% t(x))
  return(log_h1 + log_h2)
}

rho <- 0.9
covmat <- matrix(c(1, rho, rho, 1), 2, 2)
y <- c(0, 0)

# Case in the top right corner of Table 3
x <- ru(logf = log_norm_t, mean = y, sigma1 = covmat, sigma2 = covmat,
  d = 2, n = 10000, init = y, rotate = FALSE)
x$pa

# Rotation increases the probability of acceptance
x <- ru(logf = log_norm_t, mean = y, sigma1 = covmat, sigma2 = covmat,
  d = 2, n = 10000, init = y, rotate = TRUE)
x$pa

# Normal x log-normal: different Box-Cox parameters ==================
norm_lognorm <- function(x, ...) {
  dnorm(x[1], ...) + dlnorm(x[2], ...)
}
x <- ru(logf = norm_lognorm, log = TRUE, n = 1000, d = 2, init = c(-1, 0),
        trans = "BC", lambda = c(1, 0))
plot(x)
plot(x, ru_scale = TRUE)
```

ru_rcpp *Generalized ratio-of-uniforms sampling using C++ via Rcpp*

Description

Uses the generalized ratio-of-uniforms method to simulate from a distribution with log-density $\log f$ (up to an additive constant). The density f must be bounded, perhaps after a transformation of variable. The file `user_fns.cpp` that is sourced before running the examples below is available at the rust Github page at https://raw.githubusercontent.com/paulnorthrop/rust/master/src/user_fns.cpp.

Usage

```
ru_rcpp(
  logf,
  ...,
  n = 1,
  d = 1,
  init = NULL,
  mode = NULL,
  trans = c("none", "BC", "user"),
  phi_to_theta = NULL,
  log_j = NULL,
  user_args = list(),
  lambda = rep(1L, d),
  lambda_tol = 1e-06,
  gm = NULL,
  rotate = ifelse(d == 1, FALSE, TRUE),
  lower = rep(-Inf, d),
  upper = rep(Inf, d),
  r = 1/2,
  ep = 0L,
  a_algor = if (d == 1) "nlminb" else "optim",
  b_algor = c("nlminb", "optim"),
  a_method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  b_method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  a_control = list(),
  b_control = list(),
  var_names = NULL,
  shoof = 0.2
)
```

Arguments

`logf` An external pointer to a compiled C++ function returning the log of the target density f evaluated at its first argument. This function should return `-Inf` when the density is zero. It is better to use `logf =` explicitly, for example, `ru(logf`

= dnorm, log = TRUE, init = 0.1), to avoid argument matching problems. In contrast, ru(dnorm, log = TRUE, init = 0.1) will throw an error because partial matching results in logf being matched to log = TRUE.

See the Passing user-supplied C++ functions in the Rcpp Gallery and the **Providing a C++ function to** ru_rcpp section in the Rusting faster: Simulation using Rcpp vignette.

... Further arguments to be passed to logf and related functions.

n A non-negative integer scalar. The number of simulated values required. If n = 0 then no simulation is performed but the component box in the returned object gives the ratio-of-uniforms bounding box that would have been used.

d A positive integer scalar. The dimension of f.

init A numeric vector of length d. Initial estimate of the mode of logf. If trans = "BC" or trans = "user" this is *after* Box-Cox transformation or user-defined transformation, but *before* any rotation of axes. If init is not supplied then rep(1, d) is used. If length(init) = 1 and d > 1 then init <- rep(init, length.out = d) is used.

mode A numeric vector of length d. The mode of logf. If trans = "BC" or trans = "user" this is *after* Box-Cox transformation or user-defined transformation, but *before* any rotation of axes. Only supply mode if the mode is known: it will not be checked. If mode is supplied then init is ignored.

trans A character scalar. trans = "none" for no transformation, trans = "BC" for Box-Cox transformation, trans = "user" for a user-defined transformation. If trans = "user" then the transformation should be specified using phi_to_theta and log_j and user_args may be used to pass arguments to phi_to_theta and log_j. See **Details** and the **Examples**.

phi_to_theta An external pointer to a compiled C++ function returning (the inverse) of the transformation from theta (θ) to phi (ϕ) that may be used to ensure positivity of ϕ prior to Box-Cox transformation. The argument is phi and the returned value is theta. If phi_to_theta is undefined at the input value then the function should return NA. See **Details**. If lambda$phi_to_theta (see argument lambda below) is supplied then this is used instead of any function supplied via phi_to_theta.

log_j An external pointer to a compiled C++ function returning the log of the Jacobian of the transformation from theta (θ) to phi (ϕ), i.e., based on derivatives of ϕ with respect to θ. Takes theta as its argument. If lambda$log_j (see argument lambda below) is supplied then this is used instead of any function supplied via log_j.

user_args A list of numeric components. If trans = ``user'' then user_args is a list providing arguments to the user-supplied functions phi_to_theta and log_j.

lambda Either

- A numeric vector. Box-Cox transformation parameters, or
- A list with components

  lambda A numeric vector. Box-Cox parameters (required).

  gm A numeric vector. Box-Cox scaling parameters (optional). If supplied this overrides any gm supplied by the individual gm argument described below.

init_psi A numeric vector. Initial estimate of mode *after* Box-Cox transformation (optional).

sd_psi A numeric vector. Estimates of the marginal standard deviations of the Box-Cox transformed variables (optional).

phi_to_theta as above (optional).

log_j As above (optional).

user_args As above (optional).

This list may be created using find_lambda_one_d_rcpp (for d = 1) or find_lambda_rcpp (for any d).

lambda_tol
: A numeric scalar. Any values in lambda that are less than lambda_tol in magnitude are set to zero.

gm
: A numeric vector. Box-Cox scaling parameters (optional). If lambda$gm is supplied in input list lambda then lambda$gm is used, not gm.

rotate
: A logical scalar. If TRUE (d > 1 only) use Choleski rotation. If d = 1 and rotate = TRUE then rotate will be set to FALSE with a warning. See **Details**.

lower, upper
: Numeric vectors. Lower/upper bounds on the arguments of the function *after* any transformation from theta to phi implied by the inverse of phi_to_theta. If rotate = FALSE these are used in all of the optimisations used to construct the bounding box. If rotate = TRUE then they are use only in the first optimisation to maximise the target density.‘ If trans = "BC" components of lower that are negative are set to zero without warning and the bounds implied after the Box-Cox transformation are calculated inside ru.

r
: A numeric scalar. Parameter of generalized ratio-of-uniforms.

ep
: A numeric scalar. Controls initial estimates for optimisations to find the b-bounding box parameters. The default (ep = 0) corresponds to starting at the mode of logf small positive values of ep move the constrained variable slightly away from the mode in the correct direction. If ep is negative its absolute value is used, with no warning given.

a_algor, b_algor
: Character scalars. Either "nlminb" or "optim". Respective optimisation algorithms used to find $a(r)$ and $(b_i^-(r), b_i^+(r))$.

a_method, b_method
: Character scalars. Respective methods used by optim to find $a(r)$ and $(b_i^-(r), b_i^+(r))$. Only used if optim is the chosen algorithm. If d = 1 then a_method and b_method are set to "Brent" without warning.

a_control, b_control
: Lists of control arguments to optim or nlminb to find $a(r)$ and $(b_i^-(r), b_i^+(r))$ respectively.

var_names
: A character (or numeric) vector of length d. Names to give to the column(s) of the simulated values.

shoof
: A numeric scalar in [0, 1]. Sometimes a spurious non-zero convergence indicator is returned from optim or nlminb). In this event we try to check that a minimum has indeed been found using different algorithm. shoof controls the starting value provided to this algorithm. If shoof = 0 then we start from the current solution. If shoof = 1 then we start from the initial estimate provided

to the previous minimisation. Otherwise, shoof interpolates between these two extremes, with a value close to zero giving a starting value that is close to the current solution. The exception to this is when the initial and current solutions are equal. Then we start from the current solution multiplied by 1 - shoof.

Details

For information about the generalised ratio-of-uniforms method and transformations see the Introducing rust vignette. See also Rusting faster: Simulation using Rcpp.

These vignettes can also be accessed using vignette("rust-a-vignette", package = "rust") and vignette("rust-c-using-rcpp-vignette", package = "rust").

If trans = "none" and rotate = FALSE then ru implements the (multivariate) generalized ratio of uniforms method described in Wakefield, Gelfand and Smith (1991) using a target density whose mode is relocated to the origin ('mode relocation') in the hope of increasing efficiency.

If trans = "BC" then marginal Box-Cox transformations of each of the d variables is performed, with parameters supplied in lambda. The function phi_to_theta may be used, if necessary, to ensure positivity of the variables prior to Box-Cox transformation.

If trans = "user" then the function phi_to_theta enables the user to specify their own transformation.

In all cases the mode of the target function is relocated to the origin *after* any user-supplied transformation and/or Box-Cox transformation.

If d is greater than one and rotate = TRUE then a rotation of the variable axes is performed *after* mode relocation. The rotation is based on the Choleski decomposition (see chol) of the estimated Hessian (computed using optimHess of the negated log-density after any user-supplied transformation or Box-Cox transformation. If any of the eigenvalues of the estimated Hessian are non-positive (which may indicate that the estimated mode of logf is close to a variable boundary) then rotate is set to FALSE with a warning. A warning is also given if this happens when d = 1.

The default value of the tuning parameter r is 1/2, which is likely to be close to optimal in many cases, particularly if trans = "BC".

Value

An object of class "ru" is a list containing the following components:

sim_vals: An n by d matrix of simulated values.

box: A (2 * d + 1) by d + 2 matrix of ratio-of-uniforms bounding box information, with row names indicating the box parameter. The columns contain

- **column 1** values of box parameters.
- **columns 2 to (2+d-1)** values of variables at which these box parameters are obtained.
- **column 2+d** convergence indicators.

Scaling of f within ru and relocation of the mode to the origin means that the first row of box will always be c(1, rep(0, d)).

pa: A numeric scalar. An estimate of the probability of acceptance.

r: The value of r.

d	The value of d.
logf	A function. logf supplied by the user, but with f scaled by the maximum of the target density used in the ratio-of-uniforms method (i.e. logf_rho), to avoid numerical problems in contouring f in plot.ru when d = 2.
logf_rho	A function. The target function actually used in the ratio-of-uniforms algorithm.
sim_vals_rho	An n by d matrix of values simulated from the function used in the ratio-of-uniforms algorithm.
logf_args	A list of further arguments to logf.
logf_rho_args	A list of further arguments to logf_rho. Note: this component is returned by ru_rcpp but not by ru.
f_mode	The estimated mode of the target density f, after any Box-Cox transformation and/or user supplied transformation, but before mode relocation.

References

Wakefield, J. C., Gelfand, A. E. and Smith, A. F. M. (1991) Efficient generation of random variates via the ratio-of-uniforms method. *Statistics and Computing* (1991), **1**, 129-133. doi:10.1007/BF01889987.

Eddelbuettel, D. and Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, **40**(8), 1-18. doi:10.18637/jss.v040.i08

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*, Springer, New York. ISBN 978-1-4614-6867-7.

See Also

ru for a version of ru_rcpp that accepts R functions as arguments.

summary.ru for summaries of the simulated values and properties of the ratio-of-uniforms algorithm.

plot.ru for a diagnostic plot.

find_lambda_one_d_rcpp to produce (somewhat) automatically a list for the argument lambda of ru for the d = 1 case.

find_lambda_rcpp to produce (somewhat) automatically a list for the argument lambda of ru for any value of d.

optim for choices of the arguments a_method, b_method, a_control and b_control.

nlminb for choices of the arguments a_control and b_control.

optimHess for Hessian estimation.

chol for the Choleski decomposition.

Examples

```
n <- 1000

# Normal density ===================
```

```
# One-dimensional standard normal ----------------
ptr_N01 <- create_xptr("logdN01")
x <- ru_rcpp(logf = ptr_N01, d = 1, n = n, init = 0.1)

# Two-dimensional standard normal ----------------
ptr_bvn <- create_xptr("logdnorm2")
rho <- 0
x <- ru_rcpp(logf = ptr_bvn, rho = rho, d = 2, n = n,
  init = c(0, 0))

# Two-dimensional normal with positive association ===================
rho <- 0.9
# No rotation.
x <- ru_rcpp(logf = ptr_bvn, rho = rho, d = 2, n = n, init = c(0, 0),
             rotate = FALSE)

# With rotation.
x <- ru_rcpp(logf = ptr_bvn, rho = rho, d = 2, n = n, init = c(0, 0))

# Using general multivariate normal function.
ptr_mvn <- create_xptr("logdmvnorm")
covmat <- matrix(rho, 2, 2) + diag(1 - rho, 2)
x <- ru_rcpp(logf = ptr_mvn, sigma = covmat, d = 2, n = n, init = c(0, 0))

# Three-dimensional normal with positive association ----------------
covmat <- matrix(rho, 3, 3) + diag(1 - rho, 3)

# No rotation.
x <- ru_rcpp(logf = ptr_mvn, sigma = covmat, d = 3, n = n,
             init = c(0, 0, 0), rotate = FALSE)

# With rotation.
x <- ru_rcpp(logf = ptr_mvn, sigma = covmat, d = 3, n = n,
             init = c(0, 0, 0))

# Log-normal density ===================

ptr_lnorm <- create_xptr("logdlnorm")
mu <- 0
sigma <- 1
# Sampling on original scale ----------------
x <- ru_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma, d = 1, n = n,
             lower = 0, init = exp(mu))

# Box-Cox transform with lambda = 0 ----------------
lambda <- 0
x <- ru_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma, d = 1, n = n,
             lower = 0, init = exp(mu), trans = "BC", lambda = lambda)

# Equivalently, we could use trans = "user" and supply the (inverse) Box-Cox
# transformation and the log-Jacobian by hand
ptr_phi_to_theta_lnorm <- create_phi_to_theta_xptr("exponential")
ptr_log_j_lnorm <- create_log_j_xptr("neglog")
```

```
x <- ru_rcpp(logf = ptr_lnorm, mu = mu, sigma = sigma, d = 1, n = n,
  init = 0.1, trans = "user", phi_to_theta = ptr_phi_to_theta_lnorm,
  log_j = ptr_log_j_lnorm)

# Gamma (alpha, 1) density ===================

# Note: the gamma density in unbounded when its shape parameter is < 1.
# Therefore, we can only use trans="none" if the shape parameter is >= 1.

# Sampling on original scale ----------------

ptr_gam <- create_xptr("logdgamma")
alpha <- 10
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = n,
  lower = 0, init = alpha)

alpha <- 1
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = n,
  lower = 0, init = alpha)

# Box-Cox transform with lambda = 1/3 works well for shape >= 1. -----------

alpha <- 1
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = n,
  trans = "BC", lambda = 1/3, init = alpha)
summary(x)

# Equivalently, we could use trans = "user" and supply the (inverse) Box-Cox
# transformation and the log-Jacobian by hand

lambda <- 1/3
ptr_phi_to_theta_bc <- create_phi_to_theta_xptr("bc")
ptr_log_j_bc <- create_log_j_xptr("bc")
x <- ru_rcpp(logf = ptr_gam, alpha = alpha, d = 1, n = n,
  trans = "user", phi_to_theta = ptr_phi_to_theta_bc, log_j = ptr_log_j_bc,
  user_args = list(lambda = lambda), init = alpha)
summary(x)


# Generalized Pareto posterior distribution ===================

# Sample data from a GP(sigma, xi) distribution
gpd_data <- rgpd(m = 100, xi = -0.5, sigma = 1)
# Calculate summary statistics for use in the log-likelihood
ss <- gpd_sum_stats(gpd_data)
# Calculate an initial estimate
init <- c(mean(gpd_data), 0)

n <- 1000
# Mode relocation only ----------------
ptr_gp <- create_xptr("loggp")
for_ru_rcpp <- c(list(logf = ptr_gp, init = init, d = 2, n = n,
                 lower = c(0, -Inf)), ss, rotate = FALSE)
```

```
x1 <- do.call(ru_rcpp, for_ru_rcpp)
plot(x1, xlab = "sigma", ylab = "xi")
# Parameter constraint line xi > -sigma/max(data)
# [This may not appear if the sample is far from the constraint.]
abline(a = 0, b = -1 / ss$xm)
summary(x1)

# Rotation of axes plus mode relocation ----------------
for_ru_rcpp <- c(list(logf = ptr_gp, init = init, d = 2, n = n,
                 lower = c(0, -Inf)), ss)
x2 <- do.call(ru_rcpp, for_ru_rcpp)
plot(x2, xlab = "sigma", ylab = "xi")
abline(a = 0, b = -1 / ss$xm)
summary(x2)

# Cauchy ========================

ptr_c <- create_xptr("logcauchy")

# The bounding box cannot be constructed if r < 1.  For r = 1 the
# bounding box parameters b1-(r) and b1+(r) are attained in the limits
# as x decreases/increases to infinity respectively.  This is fine in
# theory but using r > 1 avoids this problem and the largest probability
# of acceptance is obtained for r approximately equal to 1.26.

res <- ru_rcpp(logf = ptr_c, log = TRUE, init = 0, r = 1.26, n = 1000)

# Half-Cauchy ===================

ptr_hc <- create_xptr("loghalfcauchy")

# Like the Cauchy case the bounding box cannot be constructed if r < 1.
# We could use r > 1 but the mode is on the edge of the support of the
# density so as an alternative we use a log transformation.

x <- ru_rcpp(logf = ptr_hc, init = 0, trans = "BC", lambda = 0, n = 1000)
x$pa
plot(x, ru_scale = TRUE)

# Example 4 from Wakefield et al. (1991) ===================
# Bivariate normal x bivariate student-t

ptr_normt <- create_xptr("lognormt")
rho <- 0.9
covmat <- matrix(c(1, rho, rho, 1), 2, 2)
y <- c(0, 0)

# Case in the top right corner of Table 3
x <- ru_rcpp(logf = ptr_normt, mean = y, sigma1 = covmat, sigma2 = covmat,
  d = 2, n = 10000, init = y, rotate = FALSE)
x$pa

# Rotation increases the probability of acceptance
```

```
x <- ru_rcpp(logf = ptr_normt, mean = y, sigma1 = covmat, sigma2 = covmat,
  d = 2, n = 10000, init = y, rotate = TRUE)
x$pa
```

---

summary.ru *Summarizing ratio-of-uniforms samples*

Description

summary method for class "ru".

print method for an object object of class "summary.ru".

Usage

```
## S3 method for class 'ru'
summary(object, ...)

## S3 method for class 'summary.ru'
print(x, ...)
```

Arguments

object	an object of class "ru", a result of a call to ru.
...	For summary.lm: additional arguments passed to summary. For print.lm: additional optional arguments passed to print.
x	an object of class "summary.ru", a result of a call to summary.ru.

Value

For summary.lm: a list of the following components from object:

- information about the ratio-of-uniforms bounding box, i.e., object$box
- an estimate of the probability of acceptance, i.e., object$pa
- a summary of the simulated values, via summary(object$sim_vals)

For print.summary.ru: the argument x, invisibly.

See Also

ru for descriptions of object$sim_vals and object$box.

plot.ru for a diagnostic plot.

Examples

```
# one-dimensional standard normal ----------------
x <- ru(logf = function(x) -x ^ 2 / 2, d = 1, n = 1000, init = 0)
summary(x)

# two-dimensional normal with positive association ----------------
rho <- 0.9
covmat <- matrix(c(1, rho, rho, 1), 2, 2)
log_dmvnorm <- function(x, mean = rep(0, d), sigma = diag(d)) {
  x <- matrix(x, ncol = length(x))
  d <- ncol(x)
  - 0.5 * (x - mean) %*% solve(sigma) %*% t(x - mean)
}
x <- ru(logf = log_dmvnorm, sigma = covmat, d = 2, n = 1000, init = c(0, 0))
summary(x)
```

Index