# Package 'HDInterval'

June 9, 2018

**Type** Package

**Title** Highest (Posterior) Density Intervals

**Version** 0.2.0

**Date** 2018-06-09

**Suggests** coda

**Author** Mike Meredith and John Kruschke

**Maintainer** Mike Meredith <mmeredith@wcs.org>

**Description** A generic function and a set of methods to calculate highest density intervals for a variety of classes of objects which can specify a probability density distribution, including MCMC output, fitted density objects, and functions.

**License** GPL-3

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2018-06-09 15:46:43 UTC

## R topics documented:



---

`hdi` *Highest (Posterior) Density Interval*

---

### Description

Calculate the highest density interval (HDI) for a probability distribution for a given probability mass. This is often applied to a Bayesian posterior distribution and is then termed "highest posterior density interval", but can be applied to any distribution, including priors.

The function is an S3 generic, with methods for a range of input objects.

## Usage

```
hdi(object, credMass = 0.95, ...)

## Default S3 method:
hdi(object, credMass = 0.95, ...)

## S3 method for class 'function'
hdi(object, credMass = 0.95, tol, ...)

## S3 method for class 'matrix'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'data.frame'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'list'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'density'
hdi(object, credMass = 0.95, allowSplit=FALSE, ...)

## S3 method for class 'mcmc'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'mcmc.list'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'mcarray'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'bugs'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'jagsUI'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'rjags'
hdi(object, credMass = 0.95, ...)

## S3 method for class 'runjags'
hdi(object, credMass = 0.95, ...)
```

## Arguments

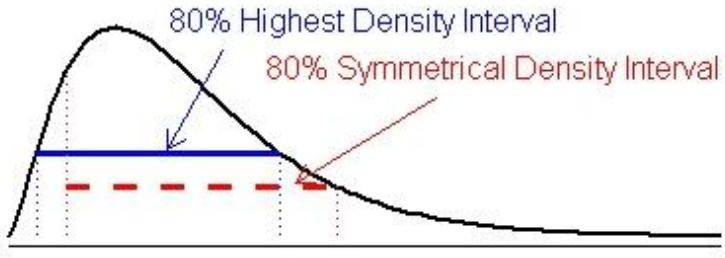
| | |
|---|---|
| `object` | an object specifying the target distribution; see Details. |
| `credMass` | a scalar [0, 1] specifying the mass within the credible interval. |
| `tol` | the desired accuracy; see `optimize`; default is 1e-8. |

`allowSplit` only available for objects of class `density`; if FALSE and the proper HDI is discontinuous, a single credible interval is returned, but this is not HDI; see Value.

`...` named parameters to be passed to other methods; see Examples.

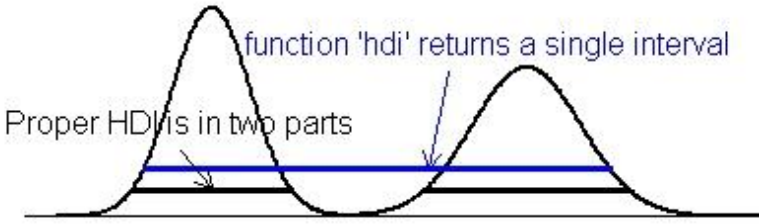
## Details

The HDI is the interval which contains the required mass such that all points within the interval have a higher probability density than points outside the interval.




In contrast, a symmetric density interval defined by (eg.) the 10% and 90% quantiles may include values with lower probability than those excluded.

For a distribution that is not severely multimodal, the HDI is the narrowest interval containing the specified mass, and the `hdi` function actually returns the narrowest interval. This does not always work properly for multimodel densities, where the HDI may be discontinuous (the horizontal black line in the Figure below). The single interval returned by `hdi` (the blue line) may incorrectly include values between the modes with low probability density. The `density` method with `allowSplit = TRUE` gives separate limits for discontinuous HDIs.




The default method expects a vector representing draws from the target distribution, such as is produced by an MCMC process. Missing values are silently ignored; if the vector has no non-missing values, NAs are returned.

The `matrix` and `data.frame` methods expect an object with vectors of the above type for each parameter in columns. The result is a matrix with parameters in columns and rows with the upper and lower limits of the HDI.

The `list` method is a wrapper for `lapply(object, hdi, ...)`. It is intended for output from `rjags::jags.samples` which returns a list of `mcarray` objects.

The `mcmc.list`, `mcmc` and `mcarray` methods expect objects of the respective types as defined in package **coda**.

The packages **R2winbugs** and **R2openbugs** produce bugs objects; **R2jags** produces rjags; **jagsUI** produces jagsUI; **runjags** produces runjags. For the **rjags** package, rjags::coda.samples produces a mcmc.list object and rjags::jags.samples produces a list of mcarray objects.

None of the above use interpolation: the values returned correspond to specific values in the data object, and will be conservative (ie, too wide rather than too narrow). Results thus depend on the random draws, and will be unstable if few values are provided. For a 95% HDI, 10,000 independent draws are recommended; a smaller number will be adequate for a 80% HDI, many more for a 99% HDI.

The function method requires the name for the inverse cumulative density function (ICDF) of the distribution; standard R functions for this have a q- prefix, eg. qbeta. Arguments required by the ICDF must be specified by their (abbreviated) names; see the examples.

## Value

a vector of length 2 or a 2-row matrix with the lower and upper limits of the HDI, with an attribute "credMass".

The density method with allowSplit=TRUE produces a matrix with a row for each component of a discontinuous HDI and columns for begin and end. It has an additional attribute "height" giving the probability density at the limits of the HDI.

## Author(s)

Mike Meredith and John Kruschke. Code for hdi.function based on hpd by Greg Snow, corrected by John Kruschke.

## References

Kruschke, J. K. 2011. *Doing Bayesian data analysis: a tutorial with R and BUGS.* Elsevier, Amsterdam, section 3.3.5.

## Examples

```
# for a vector:
tst <- rgamma(1e5, 2.5, 2)
hdi(tst)
hdi(tst, credMass=0.8)
# For comparison, the symmetrical 80% CrI:
quantile(tst, c(0.1,0.9))

# for a density:
dens <- density(tst)
hdi(dens, credMass=0.8)

# Now a data frame:
tst <- data.frame(mu = rnorm(1e4, 4, 1), sigma = rlnorm(1e4))
hdi(tst, 0.8)
apply(tst, 2, quantile, c(0.1,0.9))
tst$txt <- LETTERS[1:25]
hdi(tst, 0.8)
```

```
# For a function:
hdi(qgamma, 0.8, shape=2.5, rate=2)
# and the symmetrical 80% CrI:
qgamma(c(0.1, 0.9), 2.5, 2)

# A severely bimodal distribution:
tst2 <- c(rnorm(1e5), rnorm(5e4, 7))
hist(tst2, freq=FALSE)
(hdiMC <- hdi(tst2))
segments(hdiMC[1], 0, hdiMC[2], 0, lwd=3, col='red')
# This is a valid 95% CrI, but not a Highest Density Interval

dens2 <- density(tst2)
lines(dens2, lwd=2, col='blue')
(hdiD1 <- hdi(dens2))  # default allowSplit = FALSE; note the warning
(ht <- attr(hdiD1, "height"))
segments(hdiD1[1], ht, hdiD1[2], ht, lty=3, col='blue')
(hdiD2 <- hdi(dens2, allowSplit=TRUE))
segments(hdiD2[, 1], ht, hdiD2[, 2], ht, lwd=3, col='blue')
# This is the correct 95% HDI.
```

---

`inverseCDF` *Inverse Cumulative Density Function*

---

## Description

Given a cumulative density function, calculates the quantiles coresponding to given probabilities, ie, "converts" a CDF to an ICDF. The function method for `hdi` requires an ICDF, which is not always available for custom distributions.

## Usage

```
inverseCDF(p, CDF, ...)
```

## Arguments

| | |
|---|---|
| `p` | a vector of probabilities |
| `CDF` | a cummulative density function; standard CDFs in R begin with `p-`, eg, `pnorm`. |
| `...` | named parameters to be passed to the CDF function; see Examples; `log.p` and `lower.tail` are not supported and generate an error. |

## Details

The function uses a search algorithm to find the value of `q` which corresponds to `p`. This suffers from imprecision, especially for sections of the CDF which are relatively flat, as is usually the case close to p = 0 or 1.

## Value

a vector of the same length as p with the corresponding quantiles.

## Author(s)

Mike Meredith

## Examples

```
# Try with pgamma/qgamma
inverseCDF(c(0.025, 0.975), pgamma, shape=2.5, rate=2) # 95% interval
qgamma(c(0.025, 0.975), shape=2.5, rate=2) # for comparison

# Plug inverseCDF into hdi, need to specify the CDF
hdi(inverseCDF, CDF=pgamma, shape=2.5, rate=2)
hdi(qgamma, shape=2.5, rate=2) # for comparison

# for a custom density, here a mixture of normals
# the PDF
dmixg <- function(x)
  0.6 * dnorm(x, 0, 1) + 0.4 * dnorm(x, 4, 2^0.5)
curve(dmixg, -5, 10)
# and the CDF
pmixg <- function(q)
  0.6 * pnorm(q, 0, 1) + 0.4 * pnorm(q, 4, 2^0.5)
curve(pmixg, -5, 10)
# Now plug into inverseCDF and hdi
inverseCDF(c(0.025, 0.975), pmixg)
hdi(inverseCDF, CDF=pmixg)
```

# Index