Package ‘classInt’

April 7, 2020

Version 0.4-3
Date 2020-04-05
Title Choose Univariate Class Intervals
Depends R (>= 2.2)
Imports grDevices, stats, graphics, e1071, class, KernSmooth
Suggests spData (>= 0.2.6.2), units, knitr, rmarkdown
NeedsCompilation yes
Description Selected commonly used methods for choosing univariate class intervals for mapping or other graphics purposes.
License GPL (>= 2)
URL https://r-spatial.github.io/classInt/, https://github.com/r-spatial/classInt/
BugReports https://github.com/r-spatial/classInt/issues/
RoxygenNote 6.1.1
Encoding UTF-8
VignetteBuilder knitr
Author Roger Bivand [aut, cre] (<https://orcid.org/0000-0003-2392-6140>),
    Hisaji Ono [ctb],
    Richard Dunlap [ctb],
    Matthieu Stigler [ctb],
    Bill Denney [ctb] (<https://orcid.org/0000-0002-5759-428X>),
    Diego Hernangómez [ctb] (<https://orcid.org/0000-0001-8457-4658>)
Maintainer Roger Bivand <Roger.Bivand@nhh.no>
Repository CRAN
Date/Publication 2020-04-07 11:10:02 UTC
classIntervals

Choose univariate class intervals

Description

The function provides a uniform interface to finding class intervals for continuous numerical variables, for example for choosing colours or symbols for plotting. Class intervals are non-overlapping, and the classes are left-closed — see findInterval. Argument values to the style chosen are passed through the dot arguments. classIntervals2shingle converts a classIntervals object into a shingle. Labels generated in methods are like those found in cut unless cutlabels=FALSE.

Usage

classIntervals(var, n, style = "quantile", rtimes = 3, ..., intervalClosure = c("left", "right"), dataPrecision = NULL, warnSmallN = TRUE, warnLargeN = TRUE, largeN = 3000L, samp_prop = 0.1, gr = c("[", "]"))
## S3 method for class 'classIntervals'
plot(x, pal, ...)
## S3 method for class 'classIntervals'
print(x, digits = getOption("digits"), ..., under="under", over="over", between="-", cutlabels=TRUE, unique=FALSE)
nPartitions(x)
classIntervals2shingle(x)

Arguments

var a continuous numerical variable
n number of classes required, if missing, nclass.Sturges is used; see also the "dpih" and "headtails" styles for automatic choice of the number of classes
style chosen style: one of "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", "jenks", "dpih" or "headtails"
rtimes number of replications of var to catenate and jitter; may be used with styles "kmeans" or "bclust" in case they have difficulties reaching a classification
classIntervals

intervalClosure
default "left", allows specification of whether partition intervals are closed on
the left or the right (added by Richard Dunlap). Note that the sense of inter-
val closure is hard-coded as "right"-closed when style="jenks" (see Details
below).
dataPrecision
default NULL, permits rounding of the interval endpoints (added by Richard
Dunlap)
warnSmallN
default TRUE, if FALSE, quietens warning for n >= nobs
warnLargeN
default TRUE, if FALSE large data handling not used
largeN
default 3000L, the QGIS sampling threshold; over 3000, the observations pre-
sented to "fisher" and "jenks" are either a samp_prop= sample or a sample of
3000, whichever is larger
samp_prop
default 0.1, QGIS 10% sampling proportion
gr
default c("[","]"), if the units package is available, units::units_options("group")
may be used directly to give the enclosing bracket style
arguments to be passed to the functions called in each style
x
"classIntervals" object for printing, conversion to shingle, or plotting
under
character string value for "under" in printed table labels if cutlabels=FALSE
over
character string value for "over" in printed table labels if cutlabels=FALSE
between
character string value for "between" in printed table labels if cutlabels=FALSE
digits
minimal number of significant digits in printed table labels
cutlabels
default TRUE, use cut-style labels in printed table labels
unique
default FALSE; if TRUE, collapse labels of single-value classes
pal
a character vector of at least two colour names for colour coding the class inter-
vals in an ECDF plot; colorRampPalette is used internally to create the correct
number of colours

Details
The "fixed" style permits a "classIntervals" object to be specified with given breaks, set in the
fixedBreaks argument; the length of fixedBreaks should be n+1; this style can be used to insert
rounded break values.
The "sd" style chooses breaks based on pretty of the centred and scaled variables, and may have a
number of classes different from n; the returned par= includes the centre and scale values.
The "equal" style divides the range of the variable into n parts.
The "pretty" style chooses a number of breaks not necessarily equal to n using pretty, but likely
to be legible; arguments to pretty may be passed through ....
The "quantile" style provides quantile breaks; arguments to quantile may be passed through ....
The "kmeans" style uses kmeans to generate the breaks; it may be anchored using set.seed; the
par's attribute returns the kmeans object generated; if kmeans fails, a jittered input vector containing
rtimes replications of var is tried — with few unique values in var, this can prove necessary; arguments
to kmeans may be passed through ....
The "hclust" style uses hclust to generate the breaks using hierarchical clustering; the pars attribute returns the hclust object generated, and can be used to find other breaks using getHclustClassIntervals; arguments to hclust may be passed through . . . .

The "bclust" style uses bclust to generate the breaks using bagged clustering; it may be anchored using set.seed; the pars attribute returns the bclust object generated, and can be used to find other breaks using getBclustClassIntervals; if bclust fails, a jittered input vector containing rtimes replications of var is tried — with few unique values in var, this can prove necessary; arguments to bclust may be passed through . . . .

The "fisher" style uses the algorithm proposed by W. D. Fisher (1958) and discussed by Slocum et al. (2005) as the Fisher-Jenks algorithm; added here thanks to Hisaji Ono. This style will subsample by default for more than 3000 observations. This style should always be preferred to "jenks" as it uses the original Fortran code and runs nested for-loops much faster.

The "jenks" style has been ported from Jenks’ code, and has been checked for consistency with ArcView, ArcGIS, and MapInfo (with some remaining differences); added here thanks to Hisaji Ono (originally reported as Basic, now seen as Fortran (as described in a talk last seen at http://www.irlogi.ie/wp-content/uploads/2016/11/NUIM_ChoroHarmful.pdf, slides 26-27)). Note that the sense of interval closure is reversed from the other styles, and in this implementation has to be right-closed - use cut-labels=TRUE in findColours on the object returned to show the closure clearly, and use findCols to extract the classes for each value. This style will subsample by default for more than 3000 observations.

The "dpih" style uses the dpih() function from KernSmooth (Wand, 1995) implementing direct plug-in methodology to select the bin width of a histogram.

The "headtails" style uses the algorithm proposed by Bin Jiang (2013), in order to find groupings or hierarchy for data with a heavy-tailed distribution. This classification scheme partitions all of the data values around the mean into two parts and continues the process iteratively for the values (above the mean) in the head until the head part values are no longer heavy-tailed distributed. Thus, the number of classes and the class intervals are both naturally determined. By default the algorithm uses thr = 0.4, meaning that when the head represents more than 40% of the observations the distribution is not considered heavy-tailed. The threshold argument thr may be modified through . . . (see Examples).

Value

an object of class "classIntervals":

var the input variable
brks a vector of breaks

and attributes:

style the style used
parameters parameter values used in finding breaks
nobs number of different finite values in the input variable
call this function’s call
intervalClosure string, whether closure is “left” or “right”
classIntervals

dataPrecision  the data precision used for printing interval values in the legend returned by findColours, and in the print method for classIntervals objects. If intervalClosure is "left", the value returned is ceiling of the data value multiplied by 10 to the dataPrecision power, divided by 10 to the dataPrecision power.

Note

From version 0.1-11, the default representation has been changed to use cutlabels=TRUE, and representation within intervals has been corrected, thanks to Richard Dunlap. From version 0.1-15, the print method drops the calculation of the possible number of combinations of observations into classes, which generated warnings for n > 170.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


Slocum TA, McMaster RB, Kessler FC, Howard HH 2005 Thematic Cartography and Geographic Visualization, Prentice Hall, Upper Saddle River NJ.;


See Also

findColours, findCols, pretty, quantile, kmeans, hclust, bclust, findInterval, colorRamp, nclass, shingle

Examples

if (!require("spData", quietly=TRUE)) {
  message("spData package needed for examples")
  run <- FALSE
} else {
  run <- TRUE
}
if (run) {
  data(jenks71, package="spData")
  p1 <- c("wheat1", "red3")
opar <- par(mfrow=c(2,3))
plot(classIntervals(jenks71$jenks71, n=5, style="fixed",
fixedBreaks=c(15.57, 25, 50, 75, 100, 155.30)), pal=pal1, main="Fixed")
plot(classIntervals(jenks71$jenks71, n=5, style="sd"), pal=pal1, main="Pretty standard deviations")
plot(classIntervals(jenks71$jenks71, n=5, style="equal"), pal=pal1, main="Equal intervals")
plot(classIntervals(jenks71$jenks71, n=5, style="quantile"), pal=pal1, main="Quantile")
set.seed(1)
plot(classIntervals(jenks71$jenks71, n=5, style="kmeans"), pal=pal1, main="K-means")
plot(classIntervals(jenks71$jenks71, n=5, style="hclust", method="complete"),
pal=pal1, main="Complete cluster")
}
if (run) {
plot(classIntervals(jenks71$jenks71, n=5, style="hclust", method="single"),
pal=pal1, main="Single cluster")
set.seed(1)
plot(classIntervals(jenks71$jenks71, n=5, style="bclust", verbose=FALSE),
pal=pal1, main="Bagged cluster")
plot(classIntervals(jenks71$jenks71, n=5, style="fisher"), pal=pal1,
main="Fisher's method")
plot(classIntervals(jenks71$jenks71, n=5, style="jenks"), pal=pal1,
main="Jenks' method")
plot(classIntervals(jenks71$jenks71, style="dpih"), pal=pal1,
main="dpih method")
plot(classIntervals(jenks71$jenks71, style="headtails", thr = 1), pal=pal1,
main="Head Tails method")
par(opar)
}
if (run) {
print(classIntervals(jenks71$jenks71, n=5, style="fixed",
fixedBreaks=c(15.57, 25, 50, 75, 100, 155.30)))
}
if (run) {
print(classIntervals(jenks71$jenks71, n=5, style="sd"))
}
if (run) {
print(classIntervals(jenks71$jenks71, n=5, style="equal"))
}
if (run) {
print(classIntervals(jenks71$jenks71, n=5, style="quantile"))
}
if (run) {
print(classIntervals(jenks71$jenks71, n=5, style="kmeans"))
}
if (run) {
set.seed(1)
print(classIntervals(jenks71$jenks71, n=5, style="kmeans", intervalClosure="right"))
}
if (run) {
set.seed(1)
print(classIntervals(jenks71$jenks71, n=5, style="kmeans", dataPrecision=0))
}
if (run) {


```r
set.seed(1)
print(classIntervals(jenks71$jenks71, n=5, style="kmeans"), cutlabels=FALSE)
}
if (run) {
  print(classIntervals(jenks71$jenks71, n=5, style="hclust", method="complete"))
}
if (run) {
  print(classIntervals(jenks71$jenks71, n=5, style="hclust", method="single"))
}
if (run) {
  set.seed(1)
  print(classIntervals(jenks71$jenks71, n=5, style="bclust", verbose=FALSE))
}
if (run) {
  print(classIntervals(jenks71$jenks71, n=5, style="bclust",
                     hclust.method="complete", verbose=FALSE))
}
if (run) {
  print(classIntervals(jenks71$jenks71, n=5, style="fisher"))
}
if (run) {
  print(classIntervals(jenks71$jenks71, n=5, style="jenks"))
}
if (run) {
  print(classIntervals(jenks71$jenks71, style="dpih"))
}
if (run) {
  print(classIntervals(jenks71$jenks71, style="dpih", range.x=c(0, 160)))
}
if (run) {
  print(classIntervals(jenks71$jenks71, style="headtails"))
}
if (run) {
  print(classIntervals(jenks71$jenks71, style="headtails", thr = .45))
}
x <- c(0, 0, 0, 1, 2, 50)
print(classIntervals(x, n=3, style="fisher"))
print(classIntervals(x, n=3, style="jenks"))

# Argument 'unique' will collapse the label of classes containing a
# single value. This is particularly useful for 'censored' variables
# that contain for example many zeros.

data_censored<-c(rep(0,10), rnorm(100, mean=20,sd=1),rep(26,10))
plot(density(data_censored))
c12 <- classIntervals(data_censored, n=5, style="jenks", dataPrecision=2)
print(c12, unique=FALSE)
print(c12, unique=TRUE)

## Not run:
set.seed(1)
n <- 1e+05
x <- runif(n)
```
classIntervals(x, n=5, style="sd")
classIntervals(x, n=5, style="pretty")
classIntervals(x, n=5, style="equal")
classIntervals(x, n=5, style="quantile")
# the class intervals found vary a little because of sampling
classIntervals(x, n=5, style="kmeans")
classIntervals(x, n=5, style="fisher")
classIntervals(x, n=5, style="fisher")
classIntervals(x, n=5, style="fisher")
## End(Not run)
have_units <- FALSE
if (require(units, quietly=TRUE)) have_units <- TRUE
if (have_units) {
  set.seed(1)
x_units <- set_units(sample(seq(1, 100, 0.25), 100), km/h)
classIntervals(x_units, n=5, style="sd")
} if (have_units) {
classIntervals(x_units, n=5, style="pretty")
} if (have_units) {
classIntervals(x_units, n=5, style="equal")
} if (have_units) {
classIntervals(x_units, n=5, style="quantile")
} if (have_units) {
classIntervals(x_units, n=5, style="kmeans")
} if (have_units) {
classIntervals(x_units, n=5, style="fisher")
} if (have_units) {
classIntervals(x_units, style="headtails")
}
st <- Sys.time()
x_POSIXt <- sample(st+((0:500)*3600), 100)
fx <- st+((0:5)*3600)*100
classIntervals(x_POSIXt, style="fixed", fixedBreaks=fx)
classIntervals(x_POSIXt, n=5, style="sd")
classIntervals(x_POSIXt, n=5, style="pretty")
classIntervals(x_POSIXt, n=5, style="equal")
classIntervals(x_POSIXt, n=5, style="quantile")
classIntervals(x_POSIXt, n=5, style="kmeans")
classIntervals(x_POSIXt, n=5, style="fisher")
classIntervals(x_POSIXt, style="headtails")

# Head Tails method is suitable for right-sided heavy-tailed distributions
set.seed(1234)
# Heavy tails-----
# Pareto distributions a=7 b=14
paretodist <- 7 / (1 - runif(1000)) ^ (1 / 14)
```r
classIntervals

# Lognorm
lognormdist <- rlnorm(1000)
# Weibull
weibulldist <- rweibull(1000, 1, scale = 5)

pal1 <- c("wheat1", "red3")
opar <- par(mfrow = c(2, 3))
plot(classIntervals(paretodist, style = "headtails"),
     pal = pal1,
     main = "HeadTails: Pareto Dist.")
plot(classIntervals(lognormdist, style = "headtails"),
     pal = pal1,
     main = "HeadTails: LogNormal Dist.")
plot(classIntervals(weibulldist, style = "headtails"),
     pal = pal1,
     main = "HeadTails: Weibull Dist.")
plot(classIntervals(paretodist, n = 5, style = "fisher"),
     pal = pal1,
     main = "Fisher: Pareto Dist.")
plot(classIntervals(lognormdist, n = 7, style = "fisher"),
     pal = pal1,
     main = "Fisher: LogNormal Dist.")
plot(classIntervals(weibulldist, n = 4, style = "fisher"),
     pal = pal1,
     main = "Fisher: Weibull Dist.")
par(opar)

#Non heavy tails, thr should be increased-----

#Normal dist
normdist <- rnorm(1000)
#Left-tailed truncated Normal distr
leftnorm <- rep(normdist[normdist < mean(normdist)], 2)
# Uniform distribution
unifdist <- runif(1000)
opar <- par(mfrow = c(2, 3))
plot(classIntervals(normdist, style = "headtails"),
     pal = pal1,
     main = "Normal Dist.")
plot(classIntervals(leftnorm, style = "headtails"),
     pal = pal1,
     main = "Truncated Normal Dist.")
plot(classIntervals(unifdist, style = "headtails"),
     pal = pal1,
     main = "Uniform Dist.")
# thr should be increased for non heavy-tailed distributions
plot(
    classIntervals(normdist, style = "headtails", thr = .6),
    pal = pal1,
    main = "Normal Dist. thr = .6"
)
plot(
```
```r
classIntervals(leftnorm, style = "headtails", thr = .6),
pal = pal1,
main = "Truncated Normal Distribution thr = .6"
)
plot(
  classIntervals(unifdist, style = "headtails", thr = .6),
pal = pal1,
main = "Uniform Distribution thr = .6"
)
par(opar)
```

### findColours

*assign colours to classes from classInterval object*

#### Description

This helper function is a wrapper for `findCols` to extract classes from a "classInterval" object and assign colours from a palette created by `colorRampPalette` from the two or more colours given in the `pal` argument. It also returns two attributes for use in constructing a legend.

#### Usage

```r
findColours(clI, pal, under="under", over="over", between="-",
digits = getOption("digits"), cutlabels=TRUE)
```

#### Arguments

- **clI**: a "classIntervals" object
- **pal**: a character vector of at least two colour names; `colorRampPalette` is used internally to create the required number of colours
- **under**: character string value for "under" in legend if `cutlabels=FALSE`
- **over**: character string value for "over" in legend if `cutlabels=FALSE`
- **between**: character string value for "between" in legend if `cutlabels=FALSE`
- **digits**: minimal number of significant digits in legend
- **cutlabels**: use cut-style labels in legend

#### Value

A character vector of colours with attributes: "table", a named frequency table; "palette", a character vector of colours corresponding to the specified breaks.

#### Author(s)

Roger Bivand <Roger.Bivand@nhh.no>
findCols

extract classes from classInterval object

Description

This helper function is a wrapper for `findInterval` to extract classes from a "classInterval" object

Usage

`findCols(cI)`

Arguments

cI a "classIntervals" object
getBclustClassIntervals

Value

an integer vector of class indices

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

classIntervals, findInterval

Examples

```r
if (!require("spData", quietly=TRUE)) {
  message("spData package needed for examples")
  run <- FALSE
} else {
  run <- TRUE
}
if (run) {
  data(jenks71, package="spData")
  fix5 <- classIntervals(jenks71$jenks71, n=5, style="fixed",
                        fixedBreaks=c(15.57, 25, 50, 75, 100, 155.30))
  print(fix5)
} else {
  print(findCols(fix5))
}
```

Change breaks in a "classIntervals" object

Description

Because "classIntervals" objects of style "hclust" or "bclust" contain hierarchical classification trees in their "par" attribute, different numbers of classes can be chosen without repeating the initial classification. This function accesses the "par" attribute and modifies the "brks" member of the returned "classIntervals" object.

Usage

```r
getBclustClassIntervals(cI, k)
gETCHclustClassIntervals(cI, k)
```

Arguments

cI a "classIntervals" object

k number of classes required
Value

A "classIntervals" object with a "modified" attribute set.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

classIntervals

Examples

```r
if (!require("spData", quietly=TRUE)) {
  message("spData package needed for examples")
  run <- FALSE
} else {
  run <- TRUE
}
if (run) {
  data(jenks71, package="spData")
  pal1 <- c("wheat1", "red3")
  opar <- par(mfrow=c(2,2))
  hCI5 <- classIntervals(jenks71$jenks71, n=5, style="hclust", method="complete")
  plot(attr(hCI5, "par"))
  plot(hCI5, pal=pal1, main="hclust k=5")
  plot(getHclustClassIntervals(hCI5, k=7), pal=pal1, main="hclust k=7")
  plot(getHclustClassIntervals(hCI5, k=9), pal=pal1, main="hclust k=9")
  par(opar)
}
if (run) {
  set.seed(1)
  bCI5 <- classIntervals(jenks71$jenks71, n=5, style="bclust")
  plot(attr(bCI5, "par"))
}
if (run) {
  opar <- par(mfrow=c(2,2))
  plot(getBclustClassIntervals(bCI5, k=3), pal=pal1, main="bclust k=3")
  plot(bCI5, pal=pal1, main="bclust k=5")
  plot(getBclustClassIntervals(bCI5, k=7), pal=pal1, main="bclust k=7")
  plot(getBclustClassIntervals(bCI5, k=9), pal=pal1, main="bclust k=9")
  par(opar)
}
```
Description

The function returns values of two indices for assessing class intervals: the goodness of variance fit measure, and the tabular accuracy index; optionally the overview accuracy index is also returned if the area argument is not missing.

Usage

jenks.tests(clI, area)

Arguments

clI a "classIntervals" object
area an optional vector of object areas if the overview accuracy index is also required

Details

The goodness of variance fit measure is given by Armstrong et al. (2003, p. 600) as:

$$GVF = 1 - \frac{\sum_{j=1}^{k} \sum_{i=1}^{N_j} (z_{ij} - \bar{z}_j)^2}{\sum_{i=1}^{N} (z_i - \bar{z})^2}$$

where the $z_i$, $i = 1, \ldots, N$ are the observed values, $k$ is the number of classes, $\bar{z}_j$ the class mean for class $j$, and $N_j$ the number of counties in class $j$.

The tabular accuracy index is given by Armstrong et al. (2003, p. 600) as:

$$TAI = 1 - \frac{\sum_{j=1}^{k} \sum_{i=1}^{N_j} |z_{ij} - \bar{z}_j|}{\sum_{i=1}^{N} |z_i - \bar{z}|}$$

The overview accuracy index for polygon observations with known areas is given by Armstrong et al. (2003, p. 600) as:

$$OAI = 1 - \frac{\sum_{j=1}^{k} \sum_{i=1}^{N_j} |z_{ij} - \bar{z}_j|a_{ij}}{\sum_{i=1}^{N} |z_i - \bar{z}|a_i}$$

where $a_i$, $i = 1, \ldots, N$ are the polygon areas, and as above the $a_{ij}$ term is indexed over $j = 1, \ldots, k$ classes, and $i = 1, \ldots, N_j$ polygons in class $j$.

Value

a named vector of index values

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>
References


See Also

classIntervals

Examples

if (!require("spData", quietly=TRUE)) {
  message("spData package needed for examples")
  run <- FALSE
} else {
  run <- TRUE
}
if (run) {
data(jenks71, package="spData")
fix5 <- classIntervals(jenks71$jenks71, n=5, style="fixed",
  fixedBreaks=c(15.57, 25, 50, 75, 100, 155.30))
print(jenks.tests(fix5, jenks71$area))
}
if (run) {
q5 <- classIntervals(jenks71$jenks71, n=5, style="quantile")
print(jenks.tests(q5, jenks71$area))
}
if (run) {
set.seed(1)
k5 <- classIntervals(jenks71$jenks71, n=5, style="kmeans")
print(jenks.tests(k5, jenks71$area))
}
if (run) {
h5 <- classIntervals(jenks71$jenks71, n=5, style="hclust", method="complete")
print(jenks.tests(h5, jenks71$area))
}
if (run) {
print(jenks.tests(getHclustClassIntervals(h5, k=7), jenks71$area))
}
if (run) {
print(jenks.tests(getHclustClassIntervals(h5, k=9), jenks71$area))
}
if (run) {
set.seed(1)
b5 <- classIntervals(jenks71$jenks71, n=5, style="bclust")
print(jenks.tests(b5, jenks71$area))
}
if (run) {
print(jenks.tests(getBclustClassIntervals(b5, k=7), jenks71$area))
}
if (run) {
logLik.classIntervals Log-likelihood for classIntervals objects

Description

Log-likelihood for classIntervals objects

Usage

## S3 method for class 'classIntervals'
logLik(object, ...)

Arguments

object A classIntervals object
...
Ignored.

Details

Generally, the likelihood is a method for minimizing the standard deviation within an interval, and with the AIC, a per-interval penalty can be used to maximize the information and self-similarity of data in the interval.

Based on Birge 2006 and Davies 2009 (see references), interval binning selections may be compared by likelihood to optimize the number of intervals selected for a set of data. The `logLik()` function (and associated `AIC()` function) can be used to optimize binning by maximizing the likelihood across choices of intervals.

As illustrated by the examples below (the AIC comparison does not specifically select 3 intervals when comparing 2, 3, and 4 intervals for data with 3 intervals), while likelihood-based methods can provide evidence toward optimization of binning, they are not infallible for bin selection.

Value

A ‘logLik’ object (see ‘stats::logLik’).

References


print(jenks.tests(getBclustClassIntervals(b5, k=9), jenks71$area))
}
Examples

```r
x <- classIntervals(rnorm(100), n=5, style="fisher")
logLik(x)
AIC(x) # By having a logLik method, AIC.default is used.

# When the intervals are made of a limited number of discrete values, the
# logLik is zero by definition (the standard deviation is zero giving a dirac
# function at the discrete value indicating a density of 1 and a log-density
# of zero).
x <- classIntervals(rep(1:2, each=10), n=2, style="jenks")
logLik(x)
x <- classIntervals(rep(1:3, each=10), n=2, style="jenks")
logLik(x)

# With slight jitter but notable categorical intervals (at 1, 2, and 3), the
# AIC will make selection of the optimal intervals easier.
data <- rep(1:3, each=100) + runif(n=300, min=-0.01, max=0.01)
x_2 <- classIntervals(data, n=2, style="jenks")
x_3 <- classIntervals(data, n=3, style="jenks")
x_4 <- classIntervals(data, n=4, style="jenks")
AIC(x_2, x_3, x_4)
```
Index

*Topic spatial
  classIntervals, 2
  findColours, 10
  findCols, 11
  getBclustClassIntervals, 12
  jenks.tests, 13

bclust, 5

classIntervals, 2, 11–13, 15
classIntervals2shingle
  (classIntervals), 2

colorRamp, 5, 11

cut, 2

findColours, 5, 10
findCols, 5, 11, 11
findInterval, 5, 11, 12

getBclustClassIntervals, 12
getHclustClassIntervals
  (getBclustClassIntervals), 12

hclust, 5

jenks.tests, 13

kmeans, 5

logLik.classIntervals, 16

nclass, 5

nPartitions (classIntervals), 2

plot.classIntervals (classIntervals), 2

pretty, 5

print.classIntervals (classIntervals), 2

quantile, 5

shingle, 5

18