Package ‘MASSEextra’

May 2, 2021

Type Package

Title Some 'MASS' Enhancements

Version 1.1.2

Author Bill Venables

Maintainer Bill Venables <bill.venables@gmail.com>

Description Some enhancements, extensions and additions
    to the facilities of the recommended 'MASS' package
    that are useful mainly for teaching purposes, with
    more convenient default settings and user interfaces.
    Key functions from 'MASS' are imported and re-exported
    to avoid masking conflicts. In addition we provide
    some additional functions mainly used to illustrate
    coding paradigms and techniques, such as Gramm-Schmidt
    orthogonalisation and generalised eigenvalue problems.

License GPL-2 | GPL-3

Depends R (>= 4.0.0)

Imports methods, graphics, stats, MASS, utils, grDevices, demoKde

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, patchwork, visreg, tidyverse

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2021-05-02 07:10:02 UTC

R topics documented:

 .normalise ................................................................. 2
 bc ................................................................. 3
Description

Similar to base::scale() but returning a vector with class attribute. Used for safe prediction

Usage

.normalise(x, location, scale)

Arguments

x A numeric vector
location A numeric vector of length 1
scale A numeric vector of length 1, usually positive

Value

A normalised vector inheriting from class "normalise"
**bc**

**Box-Cox transform**

**Description**
Compute the box-cox transform of a vector of values, handling the region near lambda = 0 with some care

**Usage**
bc(y, lambda, eps = 1e-04)

**Arguments**
- **y** numeric, the original observations
- **lambda** numeric, the box-cox power
- **eps** numeric, a guard around lambda = 0

**Value**
A vector of transformed quantities

**Examples**
```
plot(12:50, bc(12:50, -1), type = "l", xlab = "MPG", ylab = "bc(MPG, -1)",
    las = 1, col = "sky blue", panel.first = grid())
points(bc(MPG.city, -1) ~ MPG.city, data = Cars93, pch = 16, cex = 0.7)
```

**bc_inv**

**Box-Cox transform inverse**

**Description**
Find the original value corresponding to a box-cox transform

**Usage**
bc_inv(z, lambda, eps = 1e-05)

**Arguments**
- **z** numeric, the transformed value
- **lambda** numeric, the power of the box-cox transform
- **eps** numeric, a guard around lambda = 0
**Value**

A vector of original quantities

**Examples**

```r
invy <- with(Cars93, bc(MPG.city, lambda = -1))
mpgc <- bc_inv(invy, lambda = -1)
range(mpgc - Cars93$MPG.city)
```

**Description**

Taken from the MASS data sets. See MASS::<data set> for more information

**Usage**

Boston

**Format**

A data frame with 506 rows and 14 columns:

- **crim** numeric: As for MASS dataset of the same name.
- **zn** numeric: As for MASS dataset of the same name.
- **indus** numeric: As for MASS dataset of the same name.
- **chas** integer: As for MASS dataset of the same name.
- **nox** numeric: As for MASS dataset of the same name.
- **rm** numeric: As for MASS dataset of the same name.
- **age** numeric: As for MASS dataset of the same name.
- **dis** numeric: As for MASS dataset of the same name.
- **rad** integer: As for MASS dataset of the same name.
- **tax** numeric: As for MASS dataset of the same name.
- **ptratio** numeric: As for MASS dataset of the same name.
- **black** numeric: As for MASS dataset of the same name.
- **lstat** numeric: As for MASS dataset of the same name.
- **medv** numeric: As for MASS dataset of the same name.
A front-end to boxcox with slicker display and better defaults

Usage

box_cox(object, ...)  

## S4 method for signature 'formula'
box_cox(object, data = sys.parent(), ...)

## S4 method for signature 'lm'
box_cox(object, ..., plotit, flap = 0.4)

## S3 method for class 'box_cox'
plot(
  x,
  ...,
  las = 1,
  xlab = expression(lambda),
  ylab,
  col.lines = "steel blue"
)

## S3 method for class 'box_cox'
print(
  x,
  ...,
  las = 1,
  xlab = expression(lambda),
  ylab,
  col.lines = "steel blue"
)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>either a &quot;box_cox&quot; object, a formula, data pair, a linear model object or an xy-plot</td>
</tr>
<tr>
<td>...</td>
<td>additional arguments passed on to methods</td>
</tr>
<tr>
<td>data</td>
<td>a data frame or environment</td>
</tr>
<tr>
<td>plotit</td>
<td>currently ignored. Plotting is done by plot or print methods</td>
</tr>
<tr>
<td>flap</td>
<td>fraction of the central 95% notional confidence to expand the range of lambda for the display</td>
</tr>
</tbody>
</table>
x  a "box_cox" object to be displayed
xlab, ylab, las as for plot
col.lines colour to use for indicator lines in the display

Value
an object of class "box_cox"

Examples
box_cox(MPG.city ~ Weight, Cars93)

Description
Taken from the MASS data sets. See MASS::<data set> for more information

Usage
Cars93

Format
A data frame with 93 rows and 27 columns:

Manufacturer  factor: As for MASS dataset of the same name.
Model  factor: As for MASS dataset of the same name.
Type  factor: As for MASS dataset of the same name.
Min.Price  numeric: As for MASS dataset of the same name.
Price  numeric: As for MASS dataset of the same name.
Max.Price  numeric: As for MASS dataset of the same name.
MPG.city  integer: As for MASS dataset of the same name.
MPG.highway  integer: As for MASS dataset of the same name.
AirBags  factor: As for MASS dataset of the same name.
DriveTrain  factor: As for MASS dataset of the same name.
Cylinders  factor: As for MASS dataset of the same name.
EngineSize  numeric: As for MASS dataset of the same name.
Horsepower  integer: As for MASS dataset of the same name.
RPM  integer: As for MASS dataset of the same name.
Rev.per.mile  integer: As for MASS dataset of the same name.
Man.trans.avail  factor: As for MASS dataset of the same name.
| Description | G{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{}\text{
Arguments

object a fitted model object accommodated by \texttt{dropterm}

Value

A character string, one of "F", "Chisq", or "none"

Examples

\begin{verbatim}
fm <- glm.nb(Days ~ .^3, quine)
default_test(fm)
\end{verbatim}

\begin{verbatim}
eigen2
\end{verbatim}

\textit{Generalized eigenvalue problem}

\textbf{Description}

Solves the generalized eigenvalue problem \((B - \lambda W)\alpha = 0\), where \(B\) and \(W\) are symmetric matrices of the same size, \(W\) is positive definite, \(\lambda\) is a scalar and \(\alpha\) and \(\theta\) are vectors.

\textbf{Usage}

\texttt{eigen2(B, W)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{B, W} Similarly sized symmetric matrices with \(W\) positive definite.
\end{itemize}

\textbf{Details}

If \(W\) is not specified, \(W = I\) is assumed.

\textbf{Value}

A list with components \texttt{values} and \texttt{vectors} as for \texttt{eigen}

\textbf{Examples}

\begin{verbatim}
X <- as.matrix(subset(iris, select = -Species))
W <- crossprod(resid(aov(X ~ Species, iris)))
B <- crossprod(resid(aov(X ~ 1,      iris))) - W
n <- nrow(iris)
p <- length(levels(iris$Species))
(ev <- eigen2(B/(p - 1), W/(n - p))) ## hand-made discriminant analysis
DF <- X %*% ev$vectors[,1:2]
with(iris, {
  plot(DF, col = Species, pch = 20,
       xlab = expression(DF[1]), ylab = expression(DF[2]))
})
\end{verbatim}
GIC

Intermediate Information Criterion

Description

An AIC-variant criterion that weights complexity with a penalty mid-way between 2 (as for AIC) and log(n) (as for BIC). I.e. "not too soft" and "not too hard", just "Glodilocks".

Usage

GIC(object)

Arguments

object a fitted model object for which the criterion is desired

Value

The GIC criterion value

Examples

gm <- glm.nb(Days ~ Sex/(Age + Eth*Lrn), quine)
c(AIC = AIC(gm), GIC = GIC(gm), BIC = BIC(gm))

givens_orth

Givens orthogonalisation

Description

Orthogonalization using Givens’ method.

Usage

givens_orth(X, nullspace = FALSE)

Arguments

X a numeric matrix with ncol(X) <= nrow(X)
nullspace logical: do you want an orthogonal basis for the null space?
gs_orth_modified

Value

A list with components Q, R, as normally defined, and if nullspace is TRUE a further component N giving the basis for the requested null space of X

Examples

set.seed(1234)
X <- matrix(rnorm(7*6), 7)
givens_orth(X, nullspace = TRUE)

---

gs_orth_modified     Gram-Schmidt orthogonalization

Description

Either classical or modified algorithms. The modified algorithm is the more accurate.

Usage

gs_orth_modified(X)

gs_orth(X)

Arguments

X       a numerical matrix with ncol(X) <= nrow(X)

Value

A list with two components, Q, R, as usually defined.

Examples

set.seed(1234)
X <- matrix(rnorm(10*7), 10)
gs_orth_modified(X)
all.equal(gs_orth(X), gs_orth_modified(X))
all.equal(gs_orth_modified(X), givens_orth(X))
Description

kernelBiweight <- function(x, mean = 0, sd = 1) h <- sqrt(7)*sd ifelse(z <- abs(x-mean)) < h, 15/16*(1 - (z/h)^2)^2/h, 0)

Usage

hr_levels(x, ...)

## Default S3 method:
hr_levels(x, p = (1:9)/10, ...)

## S3 method for class 'kde_2d'
hr_levels(x, ...)

Arguments

x an object whose z component represents the KDE

... extra arguments (currently not used)

p a vector of probability levels

Details

kernelCosine <- function(x, mean = 0, sd = 1) h <- sqrt(1/(1-8/pi^2))*sd ifelse(z <- abs(x-mean)) < h, pi/4*cos((pi*z)/(2*h))/h, 0)

kernelEpanechnikov <- function(x, mean = 0, sd = 1) h <- sqrt(5)*sd ifelse(z <- abs(x-mean)) < h, 3/4*(1 - (z/h)^2)/h, 0)

kernelGaussian <- function(x, mean = 0, sd = 1) dnorm(x, mean = mean, sd = sd)

kernelLogistic <- function(x, mean = 0, sd = 1) stats::dlogis(x, mean, sqrt(3)/pi*sd)

kernelOptCosine <- function(x, mean = 0, sd = 1) h <- sqrt(1/(1-8/pi^2))*sd ifelse(z <- abs(x-mean)) < h, pi/4*cos((pi*z)/(2*h))/h, 0)

kernelRectangular <- function(x, mean = 0, sd = 1) h <- sqrt(3)*sd ifelse(abs(x-mean) < h, 1/(2*h), 0)

kernelSquaredCosine <- function(x, mean = 0, sd = 1) h <- sqrt(3/(1-6/pi^2))*sd ifelse((z <- abs(x-mean)) < h, cos(pi*z/(2*h))^2/h, 0)

kernelTriangular <- function(x, mean = 0, sd = 1) h <- sqrt(24)*sd/2 ifelse(z <- abs(x-mean)) < h, (1 - z/h)/h, 0)

kernelTricube <- function(x, mean = 0, sd = 1) h <- sqrt(243/35)*sd ifelse((z <- abs(x - mean)) < h, 70/81*(1 - (z/h)^3)^3/h, 0)
Home Range levels

For an object representing a 2-dimensional kernel density estimate find the level(s) defining a central "home range" region, that is, a region of probability content p for which all density points within the region are higher than any density point outside the region. This makes it a region of probability p with smallest area.

Value

A vector of density levels defining the home range contours

Examples

krc <- with(Boston, {
  criminality <- log(crim)
  spaciousness <- sqrt(rm)
  kde_2d(criminality, spaciousness)
})
plot(krc, xlab = expression(italic(Criminality)), ylab = expression(italic(Spaciousness)))
home <- hr_levels(krc, p = 0.5)
contour(krc, add = TRUE, levels = home, labels = "50%")

Description

A pure R implementation of an approximate one-dimensional KDE, similar to density but using a different algorithm not involving fft. Two extra facilities are provided, namely (a) the kernel may be given either as a character string to select one of a number of kernel functions provided, or a user defined R function, and (b) the kde may be fitted beyond the prescribed limits for the result, and folded back to emulate the effect of having known bounds for the distribution.

Usage

kde_1d(
  x,
  bw = bw.nrd0,
  kernel = c("gaussian", "biweight", "cosine", "epanechnikov", "logistic", "optCosine", "rectangular", "squaredCosine", "triangular", "tricube", "triweight", "uniform"),
  n = 512,
  cut = 3,
Arguments

- **x**: A numeric vector for which the kde is required or (in methods) an object of class "kde_1d"
- **bw**: The bandwidth or the bandwidth function.
- **kernel**: The kernel function, specified either as a character string or as an R function. Partial matching of the character string is allowed.
- **n**: Integer, the number of equally-spaced values in the abscissa of the kde.
- **limits**: numeric vector of length 2. Prescribed x-range limits for the x-range of the result. May be infinite, but infinite values will be pruned back to an appropriate value as determined by the data.
- **cut**: The number of bandwidths beyond the range of the input x-values to use.
- **na.rm**: Logical value: should any missing values in x be silently removed?
- **adjust**: numeric value: a multiplier to be applied to the computed bandwidth.
- **fold**: Logical value: should the kde be estimated beyond the prescribed limits for the result and 'folded back' to emulate the effect of having known range boundaries for the underlying distribution?
- **...**: currently ignored, except in method functions
- **las, col, xlab, ylab**: base graphics parameters

Value

A list of results specifying the result of the kde computation, of class "kde_1d"
Examples

```r
set.seed(1234)
u <- runif(5000)
kdeu0 <- kde_1d(u, limits = c(-Inf, Inf))
kdeu1 <- kde_1d(u, limits = 0:1, kernel = "epan", fold = TRUE)
plot(kdeu0, col = 4)
lines(kdeu1, col = "dark green")
fun <- function(x) (0 < x & x < 1) + 0
curve(fun, add=TRUE, col = "grey", n = 1000)
```

kde_2d

**A Two-dimensional Kernel Density Estimate**

Description

A pure R implementation of an approximate two-dimensional kde computation, where the approximation depends on the x- and y-resolution being fine, i.e. the number of both x- and y-points should be reasonably large, at least 256. The coding follows the same idea as used in `kde2d`, but scales much better for large data sets.

Usage

```r
kde_2d(
  x,
  y = NULL,
  bw = list(x = bw.nrd0, y = bw.nrd0),
  kernel = c("gaussian", "biweight", "cosine", "epanechnikov", "logistic", "optCosine",
            "rectangular", "squaredCosine", "triangular", "tricube", "triweight", "uniform"),
  n = 128,
  x_limits = c(rx[1] - cut * bw["x"], rx[2] + cut * bw["x"]),
  cut = 1,
  na.rm = FALSE,
  adjust = 53/45,
  ...
)
```

## S3 method for class 'kde_2d'
print(x, ...)

## S3 method for class 'kde_2d'
plot(
  x,
  ...
  las = 1,
  xlab = bquote(italic(.(x$data_name[["x"]]))),
  ylab = bquote(italic(.(x$data_name[["y"]]))),
)
col = hcl.colors(50, "YlOrRd", rev = TRUE)
)

Arguments

x, y  
Numeric vectors of the same length specified in any way acceptable to xy.coords.  
In methods, x will be an object of class "kde_2d"

bw  
bandwidths. May be a numeric vector of length 1 or 2, or a function, or list of 
two bandwidth computation functions. Short entities will be repeated to length 
1. The first relates to the x-coordinate and the second to the y.

kernel  
As for kde_1d though 1 or 2 values may be specified relating to x- and y-
coordinates respectively. Short entities will be repeated to length 2

n  
positive integer vector of length 1 or 2 specifying the resolution required in the 
x- and y-coordinates respectively. Short values will be repeated to length 2.

x_limits, y_limits  
Numeric vectors specifying the limits required for the result

cut  
The number of bandwidths beyond the x- and y-range limits for the result.

na.rm  
Should missing values be silently removed?

adjust  
A factor to adjust both bandwidths to regulate smoothness

...  
currently ignored, except in method functions

las, col, xlab, ylab  
base graphics parameters

Value

A list of results of class "kde_2d". The result may be used directly in image or contour.

Examples

krc <- with(Boston, {  
criminality <- log(crim)  
spaciousness <- sqrt(rm)  
kde_2d(criminality, spaciousness, n = 128, kernel = "biweight")  
})
plot(krc, xlab = expression(italic(Criminality)), ylab = expression(italic(Spaciousness)))
levs <- hr_levels(krc)
contour(krc, add = TRUE, levels = levs, labels = names(levs))

with(krc, persp(x, 10*y, 3*z, border="transparent", col = "powder blue", 
theta = 30, phi = 15, r = 20, scale = FALSE, shade = TRUE, 
xlab = "Criminality", ylab = "Spaciousness", zlab = "density"))
Find the box-cox transform exponent estimate

Description

Estimates the box-cox power transformation appropriate for a linear model

Usage

lambda(bc, ...)

## S3 method for class 'formula'
lambda(bc, data = sys.parent(), ..., span = 5)

## S3 method for class 'lm'
lambda(bc, ..., span = 5)

## S3 method for class 'box_cox'
lambda(bc, ..., span = 5)

## Default S3 method:
lambda(bc, ...)

Arguments

- `bc`: either a "box_cox" object, a formula, data pair, a linear model object or an x-y list
- `...`: additional parameters passed on to `box_cox`
- `data`: a data frame or environment
- `span`: integer: how many steps on either side of the maximum to use for the quadratic interpolation to find the maximum

Value

numeric: the maximum likelihood estimate of the exponent

Examples

lambda(medv ~ ., Boston, span = 10)
makepredictcall.normalise

Method function for safe prediction

Description

This is an internal function not intended to be called directly by the user.

Usage

## S3 method for class 'normalise'
makepredictcall(var, call)

Arguments

var A numeric variable
call A single term from a linear model formula

Value

A call object used in safe prediction

mean_c Mean and variance for a circular sample

Description

Mean and variance for a circular sample

Usage

mean_c(theta)
var_c(theta)

Arguments

theta A vector of angles (in radians)

Value

The mean (rsp. variance) of the angle sample

Examples

th <- 2*base::pi*(rbeta(2000, 1.5, 1.5) - 0.5)
c(mn = mean_c(th), va = var_c(th))
rm(th)
plot.drop_term  

**drop_term plot method**

### Description

drop_term plot method

### Usage

```r
## S3 method for class 'drop_term'
plot(
x,
..., 
horiz = TRUE,
las = ifelse(horiz, 1, 2),
col = c("#DF536B", "#2297E6"),
border = c("#DF536B", "#2297E6"),
show.model = TRUE
)
```

### Arguments

- `x`: An object of class "drop_term" generated by either `drop_term` or `add_term`
- `...`: arguments past on to `graphics::barplot`
- `horiz`: graphics parameter
- `las`: barplot fill and border colour(s) for positive and negative changes to the criterion, respectively
- `col`, `border`: logical: should the model itself be displayed?

### Value

`x` invisibly

### Examples

```r
boston_quad <- lm(medv ~ . + (rm + tax + lstat)^2 + poly(rm, 2) +
poly(tax, 2) + poly(lstat, 2), Boston)
dboston_quad <- drop_term(boston_quad, k = "bic")
plot(dboston_quad)
plot(dboston_quad, horiz = FALSE)
```
Description
Print method for Box-Cox objects

Usage
## S3 method for class 'lambda'
print(x, ...)

Arguments
x an object of class "box_cox"
... ignored

Value
x, invisibly

Description
Taken from the MASS data sets. See MASS::<data set> for more information

Usage
quine

Format
A data frame with 146 rows and 5 columns:

- **Eth** factor: As for MASS dataset of the same name.
- **Sex** factor: As for MASS dataset of the same name.
- **Age** factor: As for MASS dataset of the same name.
- **Lrn** factor: As for MASS dataset of the same name.
- **Days** integer: As for MASS dataset of the same name.
Stepwise model construction and inspection

Description

Front-ends to `stepAIC` and `dropterm` with changed defaults. `step_BIC` implements a stepwise selection with BIC as the criterion and `step_GIC` uses an experimental criterion with a penalty midway between AIC and BIC: the "Goldilocks" criterion.

Usage

```r
step_AIC(object, ..., trace = 0, k = 2)
step_BIC(object, ..., trace = 0, k = max(2, log(nobs(object))))
step_GIC(object, ..., trace = 0, k = (2 + log(nobs(object)))/2)
drop_term(
  object,
  ..., 
  test = default_test(object),
  k,
  sorted = TRUE,
  decreasing = TRUE,
  delta = TRUE
)
add_term(
  object,
  ..., 
  test = default_test(object),
  k,
  sorted = TRUE,
  decreasing = TRUE,
  delta = TRUE
)
```

Arguments

- `object` as for `stepAIC`
- `...` additional arguments passed on to main function in MASS
- `trace, k` as for `stepAIC`
- `sorted, test` as for `dropterm` and `addterm`
- `decreasing` in `drop_term` should the rows be displayed in decreasing order, that is best to worst terms, from that of `dropterm`?
- `delta` Should the criterion be displayed (FALSE) or the change in the criterion relative to the present model (TRUE)?
Value

A fitted model object after stepwise refinement, or a data frame with extra class membership for single term functions.

Examples

```r
fm <- glm.nb(Days ~ .^3, quine)
drop_term(fm_aic <- step_AIC(fm))
drop_term(fm_bic <- step_BIC(fm))
```

Description

A simple facility to refine models by backward elimination. Covers cases where `drop_term` works but `step_AIC` does not.

Usage

```r
step_down(object, ..., trace = FALSE, k)
```

Arguments

- `object` A fitted model object
- `...` additional arguments passed to `drop_term` such as `k`
- `trace` logical: do you want a trace of the process printed?
- `k` penalty (default 2, as for AIC)

Value

A refined fitted model object

Examples

```r
fm <- lm(medv ~ . + (rm + tax + lstat)^2 +
         I((rm - 6)^2) + I((tax - 400)^2) + I((lstat - 12)^2), Boston)
sfm <- step_down(fm, trace = TRUE, k = "bic")
```
**vcovx**

*Extended variance matrix*

**Description**

An extension to the `vcov` function mainly to cover the additional parameter involved in negative binomial models. (Currently the same as `vcov` apart from negative binomial models.)

**Usage**

```r
vcovx(object, ...)
```

## Default S3 method:
```
vcovx(object, ...)
```

## S3 method for class 'negbin'
```
vcovx(object, ...)
```

**Arguments**

- **object**
  - A fitted model object
- **...**
  - currently ignored

**Value**

An extended variance matrix including parameters addition to the regression coefficients

**Examples**

```r
fm <- glm.nb(Days ~ Sex/(Age + Eth*Lrn), quine)
Sigma <- vcovx(fm)
```

---

**which_tri**

*Which in lower/upper triangle*

**Description**

Find where the original positions of components are in a matrix given a logical vector corresponding to the lower or upper triangle stored by columns. Similar to which(., arr.ind = TRUE)

**Usage**

```r
which_tri(cond, diag = FALSE, lower = TRUE)
```
whiteside

Arguments

cond   logical vector of length that of the lower triangle
diag   logical: are the diagonal entries included?
lower  logical: is this the lower triangle? If FALSE it is the upper.

Value

a two column matrix with the row and column indices as the rows

Examples

set.seed(123)
X <- matrix(rnorm(20*2), 20, 2)
plot(X, asp = 1, pch = 16, las = 1, xlab = "x", ylab = "y")
dX <- dist(X)
ij <- which_tri(dX == max(dX))
points(X[as.vector(ij), ], col = "red", cex = 2, pch = 1)
segments(X[ij[1], 1], X[ij[1], 2],
         X[ij[2], 1], X[ij[2], 2], col = "red")
ij <- which_tri(dX == sort(dX, decreasing = TRUE)[2])
points(X[as.vector(ij), ], col = "blue", cex = 2, pch = 1)
segments(X[ij[1], 1], X[ij[1], 2],
         X[ij[2], 1], X[ij[2], 2], col = "blue")
polygon(X[chull(X), ], border = "sky blue")
rm(X, dX, ij)

whiteside

whiteside

Description

Taken from the MASS data sets. See MASS::<data set> for more information

Usage

whiteside

Format

A data frame with 56 rows and 3 columns:

Insul  factor: As for MASS dataset of the same name.
Temp   numeric: As for MASS dataset of the same name.
Gas    numeric: As for MASS dataset of the same name.
Description

These functions are for use in fitting linear models (or allies) with scaled predictors, in such a way that when the fitted model objects are used for prediction (or visualisation) the same scaling parameters will be used with the new data.

Usage

zs(x)
zu(x)
zr(x)
zq(x)

Arguments

x

A numeric vector

Value

a standardised vector containing the parameters needed for use in prediction with new data

Examples

fm <- lm(Gas ~ Insul/zs(Temp), whiteside)
gm <- lm(Gas ~ Insul/zu(Temp), whiteside)
hm <- lm(Gas ~ Insul/Temp, whiteside)
c(fm = unname(predict(fm, data.frame(Insul = "Before", Temp = 0.0))),
gm = unname(predict(gm, data.frame(Insul = "Before", Temp = 0.0))),
hm = unname(predict(hm, data.frame(Insul = "Before", Temp = 0.0))))
rm(fm, gm, hm)
Index

* datasets
  Boston, 4
  Cars93, 6
  quine, 19
  whiteside, 23
  .normalise, 2

add_term (step_AIC), 20

addterm, 20

bc, 3
bc_inv, 3
Boston, 4
box_cox, 5
box_cox, formula-method (box_cox), 5
box_cox, lm-method (box_cox), 5
boxcox, 5

Cars93, 6
contour, 15

default_test, 7
density, 12
drop_term, 21
drop_term (step_AIC), 20
dropterm, 7, 8, 20

eigen, 8
eigen2, 8

fft, 12

GIC, 9
givens_orth, 9
gs_orth (gs_orth_modified), 10
gs_orth_modified, 10

hr_levels, 11

image, 15

kde2d, 14

cars_1d, 12, 15
cars_2d, 14

lambda, 16

makepredictcall.normalise, 17
mean_c, 17

plot.box_cox (box_cox), 5
plot.drop_term, 18
plot.kde_1d (kde_1d), 12
plot.kde_2d (kde_2d), 14
print.box_cox (box_cox), 5
print.kde_1d (kde_1d), 12
print.kde_2d (kde_2d), 14
print.lambda, 19

quine, 19

step_AIC, 20, 21
step_BIC (step_AIC), 20
step_down, 21
step_GIC (step_AIC), 20
stepAIC, 20

var_c (mean_c), 17
vcov, 22
vcovx, 22

which_tri, 22
whiteside, 23

xy.coords, 15

zq (zs), 24
zr (zs), 24
zs, 24
zu (zs), 24