

# Package ‘MASSExtra’

January 22, 2023

**Type** Package

**Title** Some 'MASS' Enhancements

**Version** 1.2.1

**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** 7.2.3

**Suggests** knitr, rmarkdown, patchwork, visreg, tidyverse

**VignetteBuilder** knitr

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2023-01-22 06:20:02 UTC

## R topics documented:

.normalise . . . . .	2
as_complex . . . . .	3

avoid . . . . .	4
bc . . . . .	5
bc_inv . . . . .	5
Boston . . . . .	6
box_cox . . . . .	7
Cars93 . . . . .	8
default_test . . . . .	9
eigen2 . . . . .	10
GIC . . . . .	11
givens_orth . . . . .	11
gs_orth_modified . . . . .	12
hr_levels . . . . .	13
kde_1d . . . . .	14
kde_2d . . . . .	16
lambda . . . . .	18
makepredictcall.normalise . . . . .	19
mean_c . . . . .	19
plot.drop_term . . . . .	20
print.lambda . . . . .	21
quine . . . . .	21
step_AIC . . . . .	22
step_down . . . . .	23
unitChange . . . . .	24
usr2in . . . . .	24
vcovx . . . . .	25
which_tri . . . . .	26
whiteside . . . . .	27
xy-class . . . . .	27
zs . . . . .	28

## Index 29

---

.normalise	<i>Normalise a vector</i>
------------	---------------------------

---

### 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"

---

as_complex	<i>Coerce to complex</i>
------------	--------------------------

---

**Description**

Utility function to create complex vectors from arguments specified as in `grDevices::xy.coords()` or otherwise

**Usage**

```
as_complex(x, y)

## S4 method for signature 'xy,missing'
as_complex(x)

## S4 method for signature 'numeric,numeric'
as_complex(x, y)

## S4 method for signature 'numeric,missing'
as_complex(x, y)

## S4 method for signature 'missing,numeric'
as_complex(x, y)
```

**Arguments**

x	A numeric vector or missing, or an object inheriting from class "xy"
y	If x is a numeric an optional numeric vector, or missing. If x or y are missing they are taken as 0, but only one may be missing.

**Value**

A complex vector specifying 2-dimensional coordinates

**Examples**

```
as_complex(cbind(1:3, 3:1))
as_complex(y = 1:3) ## real parts all zero
```

---

 avoid

 Avoid overlaps
 

---

### Description

Generate a vector of positions to use to minimise text overlaps in labelled scatterplots

### Usage

```
avoid(x, ...)

## S4 method for signature 'numeric'
avoid(
  x,
  y,
  ...,
  xlog = par("xlog"),
  ylog = par("ylog"),
  usr = par("usr"),
  pin = par("pin"),
  eps = .Machine$double.eps,
  pi = base::pi
)

## S4 method for signature 'xy'
avoid(x, ...)
```

### Arguments

<code>x, y</code>	any of the forms that the coordinates of a scatterplot may be specified
<code>...</code>	additional arguments for methods
<code>xlog, ylog</code>	logicals: are the x- and/or y-scales logarithmic?
<code>usr, pin</code>	graphics parameters <code>par("usr")</code> , <code>par("pin")</code> (or replacements)
<code>eps</code>	numeric: a zero tolerance
<code>pi</code>	numeric: the value of the arithmetic constant of the same name

### Value

a vector of integers all of which are 1, 2, 3, or 4, indicating placement positions.

### Examples

```
set.seed(123)
z <- complex(real = runif(50), imaginary = runif(50))
mz <- mean(z)
z <- z[order(Arg(z - mz))]
```

```

plot(z, axes = FALSE, ann = FALSE)
segments(Re(mz), Im(mz), Re(z), Im(z))
abline(h = Im(mz), v = Re(mz), lwd = 0.5)
box()
text(Re(z), Im(z), pos = avoid(z), cex = 0.7, offset = 0.25,
     col = "red", font = 2, xpd = NA)

```

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**

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

---

Boston

*Boston*

---

**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.

---

box_cox	<i>Box-cox constructor function</i>
---------	-------------------------------------

---

### Description

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

object	either a "box_cox" object, a formula,data pair, a linear model object or an xy- list
...	additional arguments passed on to methods
data	a data frame or environment
plotit	currently ignored. Plotting is done by plot or print methods
flap	fraction of the central 95% notional confidence to expand the range of lambda for the display

**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)
```

---

Cars93

*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.



**Fuel.tank.capacity** numeric: As for MASS dataset of the same name.  
**Passengers** integer: As for MASS dataset of the same name.  
**Length** integer: As for MASS dataset of the same name.  
**Wheelbase** integer: As for MASS dataset of the same name.  
**Width** integer: As for MASS dataset of the same name.  
**Turn.circle** integer: As for MASS dataset of the same name.  
**Rear.seat.room** numeric: As for MASS dataset of the same name.  
**Luggage.room** integer: As for MASS dataset of the same name.  
**Weight** integer: As for MASS dataset of the same name.  
**Origin** factor: As for MASS dataset of the same name.  
**Make** factor: As for MASS dataset of the same name.

---

 default\_test

*Guess the default test*


---

## Description

Find an appropriate test to use in [dropterm](#) if not specified

## Usage

```
default_test(object)

## Default S3 method:
default_test(object)

## S3 method for class 'negbin'
default_test(object)

## S3 method for class 'lmerMod'
default_test(object)

## S3 method for class 'glmerMod'
default_test(object)

## S3 method for class 'multinom'
default_test(object)

## S3 method for class 'polr'
default_test(object)

## S3 method for class 'glm'
default_test(object)

## S3 method for class 'lm'
default_test(object)
```

**Arguments**

object            a fitted model object accommodated by [dropterm](#)

**Value**

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

**Examples**

```
fm <- glm.nb(Days ~ .^3, quine)
default_test(fm)
```

---

eigen2                            *Generalized eigenvalue problem*

---

**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 0 are vectors.

**Usage**

```
eigen2(B, W)
```

**Arguments**

B, W                            Similarly sized symmetric matrices with W positive definite.

**Details**

If W is not specified, W = I is assumed.

**Value**

A list with components values and vectors as for [eigen](#)

**Examples**

```
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]))
})
```

```

    legend("topleft", levels(Species), pch = 20, col = 1:3)
  })

```

---

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  $\text{ncol}(X) \leq \text{nrow}(X)$   
`nullspace` logical: do you want an orthogonal basis for the null space?

**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  $\text{ncol}(X) \leq \text{nrow}(X)$

**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))
```

---

```
hr_levels      #' @rdname kde_1d #' @export 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)
```

---

## Description

```
#' @rdname kde_1d #' @export 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)
```

## 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

```
#' @rdname kde_1d #' @export 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)

#' @rdname kde_1d #' @export kernelGaussian <- function(x, mean = 0, sd = 1) dnorm(x, mean =
mean, sd = sd)

#' @rdname kde_1d #' @export kernelLogistic <- function(x, mean = 0, sd = 1) stats::dlogis(x,
mean, sqrt(3)/pi*sd)

#' @rdname kde_1d #' @export 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)

#' @rdname kde_1d #' @export kernelRectangular <- function(x, mean = 0, sd = 1) h <- sqrt(3)*sd
ifelse(abs(x-mean) < h, 1/(2*h), 0)

#' @rdname kde_1d #' @export 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)

#' @rdname kde_1d #' @export kernelTriangular <- function(x, mean = 0, sd = 1) h <- sqrt(24)*sd/2
ifelse((z <- abs(x-mean)) < h, (1 - z/h)/h, 0)

#' @rdname kde_1d #' @export 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)
```

```
#' @rdname kde_1d #' @export kernelTriweight <- function(x, mean = 0, sd = 1) h <- sqrt(9)*sd
ifelse((z <- abs(x-mean)) < h, 35/32*(1 - (z/h)^2)^3/h, 0)
```

```
#' @rdname kde_1d #' @export kernelUniform <- function(x, mean = 0, sd = 1) h <- sqrt(3)*sd
ifelse(abs(x-mean) < h, 1/(2*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%")
```

---

kde\_1d

*One-dimensional Kernel Density Estimate*

---

### 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,
  limits = c(rx[1] - cut * bw, rx[2] + cut * bw),
  cut = 3,
```

```

    na.rm = FALSE,
    adjust = 1,
    fold = FALSE,
    ...
)

## S3 method for class 'kde_1d'
print(x, ...)

## S3 method for class 'kde_1d'
plot(
  x,
  ...,
  col = "steel blue",
  las = 1,
  xlab = bquote(x == italic(. (x$data_name))),
  ylab = expression(kde(italic(x)))
)

```

### 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**

```

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**

```

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"]),
  y_limits = c(ry[1] - cut * bw["y"], ry[2] + cut * bw["y"]),
  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

<code>x, y</code>	Numeric vectors of the same length specified in any way acceptable to <code>xy.coords</code> . In methods, <code>x</code> will be an object of class "kde_2d"
<code>bw</code>	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.
<code>kernel</code>	As for <code>kde_1d</code> though 1 or 2 values may be specified relating to x- and y-coordinates respectively. Short entities will be repeated to length 2
<code>n</code>	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.
<code>x_limits, y_limits</code>	Numeric vectors specifying the limits required for the result
<code>cut</code>	The number of bandwidths beyond the x- and y-range limits for the results.
<code>na.rm</code>	Should missing values be silently removed?
<code>adjust</code>	A factor to adjust both bandwidths to regulate smoothness
<code>...</code>	currently ignored, except in method functions
<code>las, col, xlab, ylab</code>	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"))
```

---

`lambda`*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**

<code>bc</code>	either a "box_cox" object, a formula,data pair, a linear model object or an xy-list
<code>...</code>	additional parameters passed on to box_cox
<code>data</code>	a data frame or environment
<code>span</code>	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

```
## 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 tither drop_term or add_term
..., horiz	arguments past on to graphics::barplot
las	graphics parameter
col, border	barplot fill and border colour(s) for positive and negative changes to the criterion, respectively
show.model	logical: should the model itself be displayed?

## Value

x invisibly

## Examples

```
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)
```

---

print.lambda	<i>Print method for Box-Cox objects</i>
--------------	---

---

**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

---

quine	<i>quine</i>
-------	--------------

---

**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.

---

step\_AIC

*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

```
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 <a href="#">stepAIC</a>
...	additional arguments passed on to main function in MASS
trace, k	as for <a href="#">stepAIC</a>
sorted, test	as for <a href="#">dropterm</a> and <a href="#">addterm</a>
decreasing	in <code>drop_term</code> should the rows be displayed in decreasing order, that is best to worst terms, from that of <a href="#">dropterm</a> ?
delta	Should the criterion be displayed (FALSE) or the change in the 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**

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

---

step\_down

*Naive backward elimination*


---

**Description**

A simple facility to refine models by backward elimination. Covers cases where [drop\\_term](#) works but [step\\_AIC](#) does not

**Usage**

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

**Arguments**

object	A fitted model object
...	additional arguments passed to <a href="#">drop_term</a> 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**

```
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")
```

---

unitChange	<i>Unit change functions</i>
------------	------------------------------

---

**Description**

Convert imperial to metric units, and vice versa.

**Usage**

```
cm2in(cm)
```

```
mm2in(mm)
```

```
in2cm(inch)
```

```
in2mm(inch)
```

**Arguments**

cm, inch, mm      numeric vectors in the appropriate units

**Value**

a numeric vector of values in the new units

---

usr2in	<i>Conversion functions for plotting</i>
--------	--

---

**Description**

Convert user coordinates to inch-based coordinates for the open display, and back again

**Usage**

```
usr2in(x, ...)
```

```
## S4 method for signature 'numeric'
```

```
usr2in(
  x,
  y,
  usr = par("usr"),
  pin = par("pin"),
  xlog = par("xlog"),
  ylog = par("ylog"),
  ...
)
```



```

## S4 method for signature 'xy'
usr2in(x, ...)

in2usr(x, ...)

## S4 method for signature 'numeric'
in2usr(
  x,
  y,
  usr = par("usr"),
  pin = par("pin"),
  xlog = par("xlog"),
  ylog = par("ylog"),
  ...
)

## S4 method for signature 'xy'
in2usr(x, ...)

```

### Arguments

<code>x, y</code>	any of the forms that the coordinates of a scatterplot may be specified
<code>...</code>	additional arguments for methods
<code>usr, pin</code>	graphics parameters <code>par("usr")</code> , <code>par("pin")</code> (or replacements)
<code>xlog, ylog</code>	logicals: are the x- and/or y-scales logarithmic?

### Value

a complex vector of converted coordinates

---

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

```

vcovx(object, ...)

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

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

```

**Arguments**

object            A fitted mdel objeds  
 ...                currently ignored

**Value**

An extended variance matrix including parameters addition to the regression coefficients

**Examples**

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

---

which_tri	<i>Which in lower/upper triangle</i>
-----------	--------------------------------------

---

**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**

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

**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.

---

xy-class

*An S4 class to represent alternative complex, matrix or list input forms.*

---

### Description

An S4 class to represent alternative complex, matrix or list input forms.

**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, [6](#)
  - Cars93, [8](#)
  - quine, [21](#)
  - whiteside, [27](#)
- .normalise, [2](#)
- add\_term (step\_AIC), [22](#)
- addterm, [22](#)
- as\_complex, [3](#)
- as\_complex,missing,numeric-method (as\_complex), [3](#)
- as\_complex,numeric,missing-method (as\_complex), [3](#)
- as\_complex,numeric,numeric-method (as\_complex), [3](#)
- as\_complex,xy,missing-method (as\_complex), [3](#)
- avoid, [4](#)
- avoid,numeric-method (avoid), [4](#)
- avoid,xy-method (avoid), [4](#)
  
- bc, [5](#)
- bc\_inv, [5](#)
- Boston, [6](#)
- box\_cox, [7](#)
- box\_cox,formula-method (box\_cox), [7](#)
- box\_cox,lm-method (box\_cox), [7](#)
- boxcox, [7](#)
  
- Cars93, [8](#)
- cm2in (unitChange), [24](#)
- contour, [17](#)
  
- default\_test, [9](#)
- density, [14](#)
- drop\_term, [23](#)
- drop\_term (step\_AIC), [22](#)
- dropterm, [9](#), [10](#), [22](#)
  
- eigen, [10](#)
  
- eigen2, [10](#)
  
- fft, [14](#)
  
- GIC, [11](#)
- givens\_orth, [11](#)
- gs\_orth (gs\_orth\_modified), [12](#)
- gs\_orth\_modified, [12](#)
  
- hr\_levels, [13](#)
  
- image, [17](#)
- in2cm (unitChange), [24](#)
- in2mm (unitChange), [24](#)
- in2usr (usr2in), [24](#)
- in2usr,numeric-method (usr2in), [24](#)
- in2usr,xy-method (usr2in), [24](#)
  
- kde2d, [16](#)
- kde\_1d, [14](#), [17](#)
- kde\_2d, [16](#)
  
- lambda, [18](#)
  
- makepredictcall.normalise, [19](#)
- mean\_c, [19](#)
- mm2in (unitChange), [24](#)
  
- plot.box\_cox (box\_cox), [7](#)
- plot.drop\_term, [20](#)
- plot.kde\_1d (kde\_1d), [14](#)
- plot.kde\_2d (kde\_2d), [16](#)
- print.box\_cox (box\_cox), [7](#)
- print.kde\_1d (kde\_1d), [14](#)
- print.kde\_2d (kde\_2d), [16](#)
- print.lambda, [21](#)
  
- quine, [21](#)
  
- step\_AIC, [22](#), [23](#)
- step\_BIC (step\_AIC), [22](#)

step\_down, [23](#)  
step\_GIC (step\_AIC), [22](#)  
stepAIC, [22](#)

unitChange, [24](#)  
usr2in, [24](#)  
usr2in,numeric-method (usr2in), [24](#)  
usr2in,xy-method (usr2in), [24](#)

var\_c (mean\_c), [19](#)  
vcov, [25](#)  
vcovx, [25](#)

which\_tri, [26](#)  
whiteside, [27](#)

xy-class, [27](#)  
xy.coords, [17](#)

zq (zs), [28](#)  
zr (zs), [28](#)  
zs, [28](#)  
zu (zs), [28](#)