

# Package ‘ymse’

November 21, 2019

**Title** Ymse (Various)

**Version** 0.6.7

**Description** Supplies a number of more or less useful functions and methods suitable for, eg. estimating dice roll probabilities, calculate latin squares, perform binary search, adjust colours in HSV space, produce prime numbers, find maximum acf/pacf/ccf, convert floats to simple ratio, produce averaged shifted histogram drop variables from formulae using regex, flatten a nested list, compute the similarity between two character vectors, plot a simple loess smooth, and other assorted tasks.

**Depends** R (>= 3.5.0)

**Imports** stats, utils, graphics, grDevices

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.0.0

## R topics documented:

acf_max . . . . .	3
addrows . . . . .	4
adjustcolorHSV . . . . .	5
ahist . . . . .	6
align_char . . . . .	7
align_num . . . . .	8
arfilter . . . . .	9
arfit . . . . .	9
arimpulse . . . . .	10
armodel . . . . .	11
as.array.list . . . . .	12
aug_median . . . . .	13
bartlett . . . . .	14
binsearch . . . . .	15
bix . . . . .	16
caledoscope . . . . .	17
cbapply . . . . .	17
central.tendency . . . . .	18
combodice . . . . .	20

comparison_with_ties . . . . .	21
default_par . . . . .	22
dice . . . . .	23
dput2 . . . . .	24
drop_pattern . . . . .	25
drop_randfx . . . . .	26
dtf_clean . . . . .	26
dusd . . . . .	28
Elo_rating . . . . .	30
entropy . . . . .	31
every_nth . . . . .	32
expand . . . . .	33
explode . . . . .	34
factorise . . . . .	35
factors . . . . .	36
file_ext . . . . .	37
fingerknit . . . . .	38
fitrange . . . . .	38
flatten . . . . .	39
gcd . . . . .	40
incdiff . . . . .	40
indexvalue . . . . .	41
intsect . . . . .	42
in_range . . . . .	43
is_coprime . . . . .	43
is_prime . . . . .	44
keep_finite . . . . .	44
lag_vector . . . . .	45
latin_sq . . . . .	46
markov_seq . . . . .	47
math_constants . . . . .	47
math_constants_char . . . . .	48
means . . . . .	48
merge_multiple . . . . .	49
multidensity . . . . .	50
narm . . . . .	52
norma . . . . .	53
pairwise . . . . .	54
pcamean . . . . .	55
plot.histogram . . . . .	56
plot.stl . . . . .	56
primes . . . . .	56
quartz.png . . . . .	57
quick_table . . . . .	58
rainbowHCL . . . . .	58
randcolours . . . . .	59
resolve_dup . . . . .	60
revert_par . . . . .	61
rle2 . . . . .	61
seq_range . . . . .	62
set_mar . . . . .	63
similarity . . . . .	64

<i>acf_max</i>	3
simple_loess . . . . .	65
simple_table . . . . .	66
smat . . . . .	68
speedskate . . . . .	69
spread_seq . . . . .	70
summary.stl . . . . .	71
tied_triple_test . . . . .	71
var_th . . . . .	73
weekday . . . . .	74
ymse . . . . .	75
<b>Index</b>	<b>76</b>

---

<i>acf_max</i>	<i>Maximum ACF, PACF and CCF</i>
----------------	----------------------------------

---

## Description

Find lag that maximizes correlation

## Usage

```
acf_max(x, ..., plot = FALSE, show = plot, ci = 0.95, ma.ci = TRUE,
        max.type = c("pos", "neg", "abs"), most.signif = FALSE)
```

```
pacf_max(x, ..., plot = FALSE, show = plot, ci = 0.95,
        max.type = c("pos", "neg", "abs"))
```

```
ccf_max(x, y, ..., plot = FALSE, show = plot, ci = 0.95,
        max.type = c("pos", "neg", "abs"))
```

## Arguments

<i>x</i> , <i>y</i>	univariate numeric vector or time series
<i>...</i>	further arguments passed to <i>acf</i> , <i>pacf</i> , <i>ccf</i>
<i>plot</i>	logical; return a plot
<i>show</i>	indicate on the plot the maximum correlation
<i>ci</i>	confidence interval used, by default 95%
<i>ma.ci</i>	should the confidence limits assume an MA input (TRUE, the default), or white noise as is default for <i>plot.acf</i> ?
<i>max.type</i>	what maximum should be returned, the positive (default), negative, or absolute maximum?
<i>most.signif</i>	should the most significant correlation be returned. Only applicable if <i>ma.ci</i> =TRUE

## Examples

```
x <- c(5, 5, 3, 6, 3, 6, 9, 6, 3, 1, 3, 2, 8, 9, 4, 3, 6, 6,
      6, 7, 5, 2, 5, 1, 5, 5, 0, 3, 7, 3, 6, 6, 2, 2, 6, 5)
y <- c(8, 9, 7, 5, 3, 5, 6, 9, 6, 3, 4, 5, 9, 7, 8, 5, 5, 7,
      4, 7, 7, 2, 5, 6, 5, 7, 5, 3, 5, 6, 7, 0, 5, 3, 8, 4)

acf_max(x, plot=TRUE, max.type="abs")
acf_max(x, max.type="neg")
acf_max(x, max.type="neg", most.signif=TRUE)

pacf_max(x, plot=TRUE)
pacf_max(x, max.type="abs")

ccf_max(x, y, plot=TRUE)
ccf_max(x, y, max.type="neg")

# Same plot
plot(acf(x, plot=FALSE), ci.type="ma")
acf_max(x, plot=TRUE)

acf_max(x, ci=0.99, plot=TRUE)
ccf_max(x, y, ci=0, max.type="pos", plot=TRUE)
```

---

addrows

*Add rows to a data.frame*


---

## Description

An "rbind for data.frames", sort of.

## Usage

```
addrows(dtf, nrw, top = FALSE)
```

## Arguments

dtf	data.frame; original data.frame
nrw	data.frame; the new row(s) to be added
top	logical; should the new rows be added to the top or the bottom (default)?

## Details

Can only bind two objects at a time, but will bind data.frames with non-matching column names and -classes. In such cases the original data.frame will serve as template.

**Examples**

```

dtf <- data.frame(A=letters[1:5],
                  B=1:5,
                  C=as.factor(5:1),
                  D=as.Date(0:4, origin="2000-01-01"),
                  stringsAsFactors=FALSE)

nrw <- data.frame(A=letters[1:5],
                  B=4:8,
                  C=5:1,
                  D=as.Date(5:1, origin="1990-01-01"),
                  stringsAsFactors=FALSE)

str(dtf)

dtf.a <- addrows(dtf, nrw, top=FALSE)
str(dtf.a)

# adding a single row with little concern for data types and column names
b <- type.convert(beaver1[80:90,])
b$activ <- as.logical(b$activ)

addrows(b, data.frame(350, 1200, 37.02, 1))

```

adjustcolorHSV

*Adjust Colors in One or More Directions Conveniently.***Description**

Adjust or modify a vector of colors by "turning knobs" on one or more coordinates in (h,s,v, $\alpha$ ) space, typically by up or down scaling them.

**Usage**

```

adjustcolorHSV(col, alpha.f = 1, h.f = 1, s.f = 1, v.f = 1,
               offset = c(0, 0, 0, 0), transform = diag(c(h.f, s.f, v.f, alpha.f)),
               h = NULL, s = NULL, v = NULL, alpha = NULL)

```

**Arguments**

<code>col</code>	vector of colors, in any format that <code>col2rgb()</code> accepts
<code>alpha.f</code> , <code>h.f</code> , <code>s.f</code> , <code>v.f</code>	factors scaling the opacity, hue, saturation and value of <code>col</code>
<code>offset</code>	a length 4 numeric vector specifying the linear offset applied to the <i>hue</i> , <i>saturation</i> , <i>value</i> and <i>alpha</i> values
<code>transform</code>	a 4x4 diagonal matrix specifying the scaling applied to the <i>hue</i> , <i>saturation</i> , <i>value</i> and <i>alpha</i> values
<code>h</code> , <code>s</code> , <code>v</code> , <code>alpha</code>	fixed vlues for hue, saturation, value and alpha. Overrides any corresponding scaling factor or offset

**Details**

Essentially an HSV version of the RGB-based [adjustcolor](#). One important distinction is that the `h.f` value wraps around to fit the [0, 1] range, rather than simply "clamping" it between 0 and 1.

**Value**

A character vector the same length as `col` containing color data in standard hexadecimal RGBA format.

**Examples**

```
# Halve the saturation and value of the default palette colours
plot(2:8, cex=5, lwd=4, pch=21, bg=2:8,
     col=adjustcolorHSV(2:8, s.f=0.5, v.f=0.6))

# Offset the hue of the default palette colours by 0.5, inverting the colours
plot(2:8, cex=5, lwd=4, pch=21, bg=2:8,
     col=adjustcolorHSV(2:8, offset=c(0.5, 0, 0, 0)))
```

---

ahist

---

*Average shifted histogram*


---

**Description**

Create a smoothed histogram by averaging several histograms shifted by fractions of a bin-width

**Usage**

```
ahist(x, n.breaks = nclass.FD(x), n.shifts = 3, type = c("histogram",
  "polygon", "line", "table"), freq = FALSE, plot = TRUE,
  add = FALSE, ...)
```

**Arguments**

<code>x</code>	a vector of values for which the histogram is desired
<code>n.breaks</code>	an integer giving the number of bins to be used
<code>n.shifts</code>	an integer giving the number of shifts to be performed
<code>type</code>	if <code>plot=TRUE</code> , the type of plot to be used
<code>freq</code>	should frequency counts be used, or density (default)
<code>plot</code>	logical; if <code>TRUE</code> (default), a graphical output will be returned
<code>add</code>	logical; if <code>TRUE</code> the plot will be added to the current plot
<code>...</code>	further graphical parameters to <code>ymse::plot.histogram</code> , <code>polygon</code> , or <code>lines</code>

**Value**

an object of class "histogram"

**Examples**

```

set.seed(1)
n <- 6

x <- sample(sample(0:20, 8), 6*n, replace=TRUE) + rnorm(6*n, -8, 0.5)
x <- c(x, rgamma(5*n, 3, 0.5), rnorm(4*n, 15, 2))
x <- round(x*5)/5

hist(x, freq=FALSE, breaks="FD", col="lightblue")
ahist(x, type="hist", border=2, col=7, freq=FALSE, lwd=2)
ahist(x, type="poly", border=2, col=7, freq=FALSE, lwd=2)
ahist(x, type="line", col=2, freq=FALSE, lwd=2)
ahist(x, type="table", col=2, freq=FALSE, lwd=2)
ahist(x, plot=FALSE)

```

align\_char

*Align character strings***Description**

Align character strings

**Usage**

```
align_char(x, pattern = ".", ..., lpad = " ", rpad = " ")
```

**Arguments**

x	numeric or character vector
pattern	pattern, passed to <code>regexpr</code> , whose match the character strings will be aligned by
...	further arguments passed to <code>regexpr</code>
lpad, rpad	character strings used for padding. Repeated to length

**See Also**

[align\\_num](#), for alignment more aimed at numeric strings

**Examples**

```

x <- c("Tom und Jerry", "Abbott og Costello", "Milo och Stich", "et alii")
cat(align_char(x, pat="[:alpha:]", sep="\n"))
cat(align_char(x, pat=" "), sep="\n")
cat(align_char(x, pat=" [A-Z]", sep="\n"))
cat(align_char(x, pat=" [a-z]", sep="\n"))
cat(align_char(x, pat="t", ignore.case=TRUE), sep="\n")
cat(align_char(x, pat="x"), sep="\n")

```

align\_num

*Align numbers***Description**

Align numbers for neat vertical printing

**Usage**

```
align_num(x, lpad = " ", rpad = "0", dec = ".", min.dec = 0,
          rm.dec = TRUE)
```

**Arguments**

x	numeric or character vector
lpad, rpad	character strings used for padding. Repeated to length
dec	decimal separator, or any other character to align by
min.dec	pad with zeros to reach a minimum number of decimal points
rm.dec	remove zeros at the end of whole numbers

**See Also**

[align\\_char](#), for alignment more aimed at character strings

**Examples**

```
x <- c(22100, 100, 1015, 13.018, 0.1, 0.01234)
cat(align_num(x), sep="\n") # Default
cat(format(x, scientific=FALSE), sep="\n")

cat(align_num(x, rpad=" ", sep="\n")
cat(align_num(x, rpad=" ", rm.dec=FALSE), sep="\n")
cat(align_num(x, rpad=" ", min.dec=1), sep="\n")
cat(align_num(x, rpad=" ", min.dec=2), sep="\n")

cat(align_num(x, rpad="\U00b7", min.dec=0), sep="\n")
cat(align_num(x, lpad="'", rpad="'", min.dec=0), sep="\n")

cat(align_num(c("1.000.000", "10.000.000", "1.000,85"), dec=",", sep="\n")

# corner cases
x <- c("100.", "1.2", ".1111")
cat(align_num(x, rpad=" ", rm.dec=TRUE), sep="\n")
cat(align_num(round(as.numeric(x)), rpad=" ", rm.dec=TRUE), sep="\n")

# matching on more than one character
# so far not much more advanced than this
# working on align_num2 more suited for character strings
s <- c("cataract", "hematology", "pancreatic")
cat(align_num(s, dec="a", rpad=" ", sep="\n")
cat(align_num(s, dec="at", rpad=" ", sep="\n")
```



```
a <- c("Tom and Jerry", "Milo and Stich", "Abbott and Costello")
cat(align_num(a, dec="and", rpad=" "), sep="\n")
```

---

arfilter	<i>AR filter</i>
----------	------------------

---

## Description

Filter a time series using AR coefficients

## Usage

```
arfilter(x, mod, x.mean = mod$x.mean, init = "focb")
```

## Arguments

x	a time series
mod	an AR model
x.mean	the mean used. By default the mean of the original model. Set to zero for no demeaning
init	how the initial values should be chosen. First observation carried backwards (default), mean of the first values, or the first values in reverse.

## See Also

[armodel](#)

## Examples

```
set.seed(1)
arap <- ar(AirPassengers)
spec.ar(arap)
spec.pgram(arfilter(rnorm(10000), arap), span=21, na.action=na.omit)

arm <- armodel(c(1.3, -0.4))
spec.ar(arm)
plot(x <- rnorm(200), type="l")
lines(scale(arfilter(x, arm), center=FALSE), col="red", lwd=2)
```

---

arfit	<i>AR model fit</i>
-------	---------------------

---

## Description

Fit a specified AR model to a univariate time series

## Usage

```
arfit(x, mod, x.mean = mod$x.mean)
```

**Arguments**

x	a time series
mod	an AR model
x.mean	the mean used. By default the mean of the original model. Set to zero for no demeaning

**See Also**

[armodel](#) for examples

**Examples**

```
set.seed(1)
x <- runif(50) + sin(1:50/10)
plot(x); lines(arfilter(x, armodel(c(1.5, -0.5, 0.5)), x.mean=mean(x)))
```

---

arimpulse

*Impulse response of an AR model*

---

**Description**

Get and plot the impulse response of an AR model

**Usage**

```
arimpulse(mod, pulse = 1, n.ahead = 20, plot = TRUE, ...)
```

**Arguments**

mod	an AR model
pulse	numeric vector; the initial pulse. Magnitude is added to the model mean
n.ahead	the length of the computed response
plot	logical; should the result be plotted?
...	further arguments to plot

**See Also**

[armodel](#) for examples

---

armodel	Create an AR model object
---------	---------------------------

---

## Description

Specify the characteristics of an AR model

## Usage

```
armodel(coefs, mean = 0, intercept = 0, var.pred = 1,  
        frequency = 1, x.name = "Synthetic AR model")
```

## Arguments

coefs	a vector of model coefficients
mean	the mean of the process
intercept	the intercept in the model
var.pred	the portion of the variance not explained by this model
frequency	the sampling frequency of the process
x.name	name of the series

## See Also

[arimpulse](#)

## Examples

```
# short decay  
ar.mod <- armodel(c(0.5))  
arimpulse(ar.mod, pulse=1)  
  
# long decay  
ar.mod <- armodel(c(0.8))  
arimpulse(ar.mod, pulse=1)  
  
# negative second coefficient reduce damping, signal returns to normal  
# more quickly  
ar.mod <- armodel(c(0.8, -0.1))  
arimpulse(ar.mod, pulse=1)  
  
# second coefficient reduce damping too much, overdamping, oscillations  
ar.mod <- armodel(c(0.8, -0.5))  
arimp <- arimpulse(ar.mod, pulse=1, n.ahead=40)$pred  
polyroot(c(1, -ar.mod$ar)) # complex conjugate roots  
acf(arimp) # period ~= 6?  
phi1 <- ar.mod$ar[1]  
phi2 <- ar.mod$ar[2]  
f <- (1/(2*pi)) * acos((phi1*(phi2-1))/(4*phi2))  
1/f # period = 6.78  
sp <- spec.ar(ar.mod, plot=FALSE)  
1/sp$freq[which.max(sp$spec)] # period = 6.79
```

```
# decaying oscillations
ar.mod1 <- armodel(c(0.8, -0.6, -0.5, 0.2, -0.2))
arimpulse(ar.mod1, n.ahead=100)
Mod(1/polyroot(c(1, -ar.mod1$ar))) # barely inside the unit circle

# growing oscillations
ar.mod2 <- armodel(c(0.8, -0.7, -0.5, 0.2, -0.2))
arimpulse(ar.mod2, n.ahead=100)
Mod(1/polyroot(c(1, -ar.mod2$ar))) # barely outside the unit circle

ar.mod3 <- armodel(c(1.8, -1.1, 0.2, -0.2, 0.2))
arimpulse(ar.mod3, n.ahead=100)
spec.ar(ar.mod3)

resid(arfit(rnorm(10), armodel(c(0.5, -0.1), frequency=2)))
```

---

as.array.list

*Coerce a list to an array*


---

### Description

Coerce a list consisting of data.frames or matrices of equal size to a 3d array

### Usage

```
## S3 method for class 'list'
as.array(x, ...)
```

### Arguments

x	a list of equal sized data.frames or matrices
...	(not used)

### Value

A list of length  $l$  with elements of  $m$  rows and  $n$  columns will result in an  $m \times n \times l$  array.

### Examples

```
df1 <- data.frame(x=c(1, 2, 3), y=c(2, 3, 4), z=c(3, 4, 5))
df2 <- data.frame(x=c(4, 2, 3), y=c(2, 5, 4), z=c(3, 4, 6))
df3 <- data.frame(x=c(1, 4, 2), y=c(3, 3, 8), z=c(4, 3, 5))

l <- list(df1, df2, df3)

as.array(l)

llm <- list(matrix(LETTERS[1:6], 2),
            matrix(LETTERS[7:12], 2))

as.array(llm)

as.array(speedskate)
```

---

aug_median	<i>Centre weighted mean</i>
------------	-----------------------------

---

## Description

Offering a continuous link between median and arithmetic mean

## Usage

```
aug_median(x, p = 1, rank = FALSE, na.rm = FALSE)
```

## Arguments

x	numeric vector
p	positive numeric narrowness of the weight. 1 gives triangular weighting. Higher values gives narrower weights, approaching median, lower values gives broader weights, approaching arithmetic mean
rank	logical. Should ranks or numeric values determine relative weights?
na.rm	logical. Should missing values be removed?

## Details

A weighted arithmetic mean is calculated over the input vector, where most weight is given to the median value(s), and monotonically less towards either extreme. Falloff depends on p, with small values resulting in a gentler falloff and less difference between minimum and maximum weights.

## Examples

```
x <- c(0, 8, 8, 8, 9)

aug_median(x)

# 0 and 9 are considered equidistant from 8
aug_median(x, rank=TRUE)

# Nearly a point weight placed at the median
aug_median(x, 100)
median(x)

# Nearly uniform weights
aug_median(x, 0.001)
mean(x)
```

---

bartlett

---

*Maurice Stevenson Bartlett's car data*


---

### Description

This is an example data set Bartlett used for a lecture course on stochastic processes, Statistics Department, University College, London. The data represents the times, in seconds, when cars passed an observation point by a road.

Bartlett attributes the data to a Dr A. J. Miller who supplied them as a class example. According to Adery C. A. Hope the data was recorded on a rural Swedish road.

### Usage

```
bartlett
```

### Format

A numeric vector representing time points in seconds

### M. S. Bartlett's notes

Analyse the above data with a view to examining:

- i whether the times of passing constitute a Poisson process;
- ii if not, whether some form of "bunching" or "clustering" seems to be present.

Possible analyses include:

- a testing the homogeneity of the consecutive random time-intervals, by means of a partitioning of the degrees of freedom for the total (approximate)  $\chi^2$ ;
- b testing the homogeneity of counts in consecutive fixed time-intervals, choosing an appropriate interval, and partitioning the degrees of freedom corresponding to the total dispersion by means of an analysis of variance;
- c testing the correlation between the consecutive random time-intervals;
- d examining the overall distribution of counts in fixed time-intervals;
- e examining the overall distribution of the consecutive random time-intervals

You should undertake at least sufficient of these to answer the questions asked.

### Source

The Spectral Analysis of Point Processes (p. 280), M. S. Bartlett, 1963

Also mentioned in:

Statistical Estimation of Density Functions (p. 252), M. S. Bartlett, 1963

A Simplified Monte Carlo Significance Test Procedure (p. 583), Adery C. A. Hope, 1968

**Examples**

```

cpgram(diff(bartlett))

bartlett2 <- bartlett - bartlett[1]

x <- rep(0, tail(bartlett2, 1)*10)
x[bartlett2*10] <- 1

par(mfrow=c(2, 1), mar=c(2, 3, 1, 1))
plot(x, type="l", ann=FALSE)
lines(cumsum(x)/sum(x), col="red", lwd=2)

sp <- spectrum(x, main="", xlim=c(0, 0.1), ylim=c(1e-3, 0.04))
spec <- predict(loess(sp$spec[1:3000] ~ sp$freq[1:3000], span=0.15), se=TRUE)
lines(sp$freq[1:3000], spec$fit, col="red", lwd=2)
lines(sp$freq[1:3000], spec$fit - qt((0.99 + 1)/2, spec$df)*spec$se,
      lty=1, col="lightblue")
lines(sp$freq[1:3000], spec$fit + qt((0.99 + 1)/2, spec$df)*spec$se,
      lty=1, col="lightblue")

```

binsearch

*Binary search***Description**

Find the position of a given value in a sorted array

**Usage**

```

binsearch(val, arr, L = 1L, H = length(arr))

binclosest(val, arr, L = 1L, H = length(arr))

```

**Arguments**

val	the value to search for
arr	a sorted array to make the search in
L	a lower bound
H	an upper bound

**Details**

While both `val` and `arr` can be either integer or double, the algorithm is limited by integer storage in how long the array can be. `L` and `H` can be used to limit the range of indices to be search within. `binsearch` will return either the index of the exact match, or the index just below if no exact match is found. This means that if `val` is less than the lowest value in `arr` (and `L=1`), a `0` will be returned, which can lead to issues as such an index does not exist in R. An array indexed by `0` will return a zero length object. `binclosest` will return the index of the closest match, and therefore a `1` in the situation where `binsearch` returns a `0`. If there is a tie the lower index will be returned. In either case, if there are duplicate matches, the lower index will be returned.

**Value**

A single integer representing an index on the input array.

**Examples**

```
binsearch(15, (1:9)*3.333)
binsearch(2, (1:9)*3.333)
binclosest(2, (1:9)*3.333)

binsearch(18, seq_len(2e9))
## Not run:
binsearch(18, seq_len(3e9))
## End(Not run)
binsearch(18, seq_len(3e9), H=2e9)
binsearch(2000, seq_len(3e7)*100 + 0.1)

set.seed(1)
x <- sort(sample(1:300, 30))
r <- sort(sample(1:300, 30))

plot(sapply(r, binsearch, x), type="l")
lines(sapply(r, binclosest, x), col="red")

x <- c(1, 2, 3, 5, 8, 9)
binclosest(6, x)
binclosest(7, x)
binclosest(5, x)
```

---

bix

*Bix attributes*


---

**Description**

bix provides access to the bix attribute of a variable. The first form returns the value of the levels of its argument and the second sets the attribute.

**Usage**

```
bix(d)
```

```
bix(d) <- value
```

**Arguments**

d	a "dice" object
value	value to begin index at

**Examples**

```
d <- dice(6)
d
bix(d)
bix(d) <- 3
```



```
d
expand(d)
```

---

caleidoscope	<i>Caleidoscopic effect on a matrix</i>
--------------	---

---

### Description

Flip a matrix vertically and horizontally before recombining into a new large matrix

### Usage

```
caleidoscope(m, odd = TRUE)
```

### Arguments

m	a matrix
odd	logical; should the resulting matrix have odd dimensions?

### Details

Three copies of `m` will be made. One flipped horizontally, one flipped vertically, and one flipped both horizontally and vertically. Then they are recombined with the original matrix in the upper right corner, and the flipped copies in the upper left, lower right and lower left corners, respectively.

### Value

A matrix of either  $2 \times$  or  $2 \times -1$  the number of rows and columns of the input matrix.

### Examples

```
caleidoscope(matrix(1:4, 2), odd=FALSE)

image(caleidoscope(1:9 %o% 1:9))

image(caleidoscope(matrix(runif(180*200)^2, 180)), col=rainbow(256, start=0.58))
```

---

cbapply	<i>Apply function to contents of clipboard</i>
---------	--

---

### Description

Read in clipboard contents as lines, apply a function on them, and write results back to the clipboard

### Usage

```
cbapply(FUN, ..., collapse = FALSE, write = TRUE, eval = FALSE)
```

**Arguments**

<code>FUN</code>	function to be applied
<code>...</code>	optional arguments to <code>FUN</code>
<code>collapse</code>	collapse the lines into a single string separated by newlines
<code>write</code>	write the results back to the clipboard
<code>eval</code>	parse and evaluate the results

**Examples**

```
## Not run:
# Copy to clipboard
a <- 10
b <- 20
s <- a + b
s
# end

# Run
ev <- cbapply(FUN=function(x) paste(x, "+ 2"), eval=TRUE)
ev; a; b; s

# Clipboard contents changed to
a <- 10 + 2
b <- 20 + 2
s <- a + b + 2
s + 2
# end

# Copy to clipboard
One Two
Three
# end

# Run
cbapply(FUN=toupper, write=FALSE)

# Clipboard contents unchanged
One Two
Three
# end

## End(Not run)
```

---

central.tendency*Central tendency measures*

---

**Description**

Central tendency measures

**Usage**

```
pseudomedian(x, na.rm = TRUE)

cmode(x, single = TRUE, ...)

dmode(x, single = TRUE, na.rm = FALSE)

midrange(x, na.rm = FALSE)

srmean(x, na.rm = FALSE)
```

**Arguments**

x	numeric vector
na.rm	remove NAs before starting calculations
single	return a single value (for cmode and dmode)
...	send further arguments to underlying function, e.g. density for cmode

**See Also**

[means](#)

**Examples**

```
xx <- c(1, 3, 4, 5, 7, 8, 9, 9, 7, 5, 4, 5, 3, 8)
median(xx)
pseudomedian(xx)

# Discrete mode
dmode(c(2, 3, 3, 4, 5))
dmode(c(2, 3, 3, 2, 5))
dmode(c(2, 3, 3, 2, 5), single=FALSE)
dmode(c(2, 1, 3, NA, 1))
dmode(c(2, 1, 3, NA, NA))

# Continuous mode
cmode(c(2, 3, 3, 4, 5))
cmode(c(2, 3, 3, 4, 5))
cmode(c(2, 3, 3, 4, 4, 5), n=512)
cmode(c(2, 2, 3, 3, 6, 6, 6, 7), single=FALSE, adjust=0.5)

# Slightly robust mean
set.seed(1)
r <- round(rexp(12)*c(-100, 100))
mean(r)
srmean(r)
weighted.mean(sort(r), c(0.5, rep(1, length(r)-2), 0.5))
```

combodice

*Combine dice***Description**

Generate probability density functions for combinations of dice.

**Usage**

```
combodice(x, FUN, ..., method = c("outer", "expand.grid", "convolve"),
          name)
```

**Arguments**

x	a list of dice objects, or objects that can be interpreted as such
FUN	function passed on to outer or apply, depending on method
...	further arguments passed to FUN
method	method for computation. One of outer, expand.grid or convolve
name	name used for the resulting PDF. Will use x object if none is given

**Details**

Each of the methods have their advantages and disadvantages. Outer and expand.grid work with roughly the same speed and memory, and can take the same kind of input, but FUN is interpreted differently, reflecting their use of outer and apply respectively. Convolve is much quicker than the other two, but is restricted to only summing distributions. While the first two can handle non-integer values, but only integer probabilities, the third can handle non-integer probabilities, but only integer values.

**Value**

A table giving the relative probability of each value

**See Also**

[dusd](#)

**Examples**

```
# Fudge dice
dF.2 <- as.table(c("-1"=2, "0"=2, "1"=2))
dF.1 <- as.table(c("-1"=1, "0"=4, "1"=1))
fudgedice2221 <- list(dF.2, dF.2, dF.2, dF.1)

combodice(fudgedice2221)

# Heterogeneous-class list and non-integer values
die1 <- as.table(c("2.6"=2, "3"=1, "5"=1))
die2 <- c(0, 1.4)
die3 <- as.dice(as.table(c("1"=2, "2"=2, "3"=2)))
diel <- list(die1, die2, die3)
```

```

combodice(diel)

# Regular d6 pair
re <- combodice(list(1:6, 1:6))

# Sichermann pair
si <- combodice(list(c(1, 2, 2, 3, 3, 4), c(1, 3, 4, 5, 6, 8)))
re; si # Identical

# One regular and one "average" d6
combodice(list(1:6, c(2, 3, 3, 4, 4, 5)))

# One 1/2 coin, one D4 and one d6, multiplied together
combodice(list(1:2, 1:4, 1:6), "*")

# Probability of getting n 1s throwing 1d4, 1d6 and 2d8
f <- function(x) sum(x == 1)
combodice(list(1:4, 1:6, 1:8, 1:8), FUN=f, method="exp")

# 3d6, discarding the lowest
discard_lowest <- function(x) sum(x[-which.min(x)])
combodice(list(1:6, 1:6, 1:6), discard_lowest, method="exp")

# 1d4, 2d6 and 1d20, discarding lowest and highest
olympic <- function(x) sum(x[-c(which.min(x), which.max(x))])
combodice(list(1:4, 1:6, 1:6, 1:20), olympic, method="exp")

# Dice pool. 3 d10 with target value 7
f <- function(x) sum(x >= 7)
combodice(lapply(rep(1, 3), seq, 10), f, method="ex")/10^3

# Equivalent using binomial PDF
dbinom(0:3, 3, 0.4)

# I have a d20 with a slight bump at the 4 and 10 facets,
# which makes 16 and 11 less likely, but the nearby 3, 18, 19 and 20
# correspondingly more likely. How does this affect the PDF?
d20l <- dice(20)
d20l[c(16, 11)] <- 0.6
d20l[c(3, 20, 18, 19)] <- 1.2
mean(d20l)

c0 <- combodice(list(dice(6), dice(10), dice(20)), method="conv", name="fair")
c1 <- combodice(list(dice(6), dice(10), d20l), method="conv", name="uneven")

set_mar()
plot(c0, type="o", pch=16, col="grey")
points(c1, col=2, type="o", lwd=1, pch=16, cex=0.6)
legend("topright", c("fair", "bumpy"), bty="n", col=c("grey", "red"), lwd=2:1)

```

---

comparison\_with\_ties    *Comparison with ties*

---

## Description

Compare numeric values, returning an inbetween value for ties

**Usage**

```
x %tgt% y

tgt(x, y, bias = 0.5)

x %tlt% y

tlt(x, y, bias = 0.5)
```

**Arguments**

x, y	numeric values to be compared
bias	what bias should be given to ties? 0.5, the default, is considered neutral as it's halfway between 1 and 0 (true and false).

**See Also**

[Comparison](#), [tied\\_triple\\_test](#)

**Examples**

```
1:5 %tlt% 3
1:5 %tgt% 3

c(1, 4, 3, 1) %tlt% c(1, 3, 3, 2)
c(1, 4, 3, 1) %tgt% c(1, 3, 3, 2)

# Calculate MannWhitney U statistic
set.seed(1)
x <- sort(round(runif(20)*13, 1))
y <- sort(round(runif(15)*10, 1))
o <- outer(x, y, "%tgt%")

sum(o)
wilcox.test(x, y, exact=FALSE)$statistic
```

---

default\_par

*Default par*

---

**Description**

Sets par settings to their default values

**Usage**

```
default_par()
```

**Details**

Default par settings can be retrieved by `data(.def.par)`. A new default can be specified by editing `def.par` or making a `def.par <- par(no.readonly=TRUE)` type call.

**See Also**

Other `par_and_plot_margins_functions`: [revert\\_par](#), [set\\_mar](#)

---

`dice`*Create, modify or convert from/to dice objects*

---

**Description**

Create, modify or convert from/to dice objects

**Usage**

```
dice(dval)

is.dice(x, ...)

as.dice(x, ...)

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

## S3 method for class 'dice'
as.table(x, ...)
```

**Arguments**

<code>dval</code>	an integer vector
<code>x</code>	an arbitrary R object
<code>...</code>	further arguments passed to methods

**See Also**

[expand](#), [table](#)

**Examples**

```
# Regular d6 dice
dice(6)

# d4 dice with sides 0, 1, 2, 4
dice(c(0:3))

# d4 dice with two 2s and two 5s
dice(c(2, 2, 5, 5))
```

dput2

*Write an Object to console***Description**

Writes an ASCII text representation of an R object to the console for easy copy/paste sharing

**Usage**

```
dput2(x, width = 65, assign = c("front", "end", "none"),
      breakAtParen = FALSE, compact = TRUE, exdent = NULL)
```

**Arguments**

x	an object
width	integer; column width
assign	character; should assignment be included?
breakAtParen	logical; should lines break at parenthesis begins
compact	remove spaces around ' = ' assignments
exdent	a non-negative integer specifying the exdentation of lines after the first. default 2 if assign="front", else 0.

**Details**

This is similar to the way dput is used to print ASCII representations of objects to the console. The differences are that dput2 lets you specify the width of the resulting column, and assignment of the object to the name used in the call will by default be included. Line breaks are by default only done on whitespace, but can be set to happen at parenthesis begins as well. This should not break code and can make for a more compact representation, but it can also make the code harder to read.

**See Also**

[dput](#), [deparse](#), [explode](#)

**Examples**

```
xmpl <- faithful[sort(sample(1:nrow(faithful), 50)), ]
dput(xmpl)
cat(deparse(xmpl, width.cutoff=65), sep='\n')
dput2(xmpl, compact=FALSE)
dput2(xmpl)
dput2(xmpl, assign="end")
dput2(xmpl, assign="none")
dput2(xmpl, 80)

# no line breaks on whitespaces or parens within character strings
xmpl <- mtcars[1:5, ]
rownames(xmpl) <- c("bbbb (hhhhhh\u00A0hhhhhhh)",
                    " rrrrrrrr ( bbbbbb )",
                    "v v v v v v v v v v",
                    "( g-god, d-god, _-___)",
```



```

                                "100*(part)/(total)")
dput2(xmpl, 15)
dput2(xmpl, 15, breakAtParen=TRUE)

```

---

drop_pattern	<i>Drop predictors</i>
--------------	------------------------

---

## Description

Drop predictor variables according to a (regex) pattern

## Usage

```
drop_pattern(form, pattern, ...)
```

## Arguments

form	a formula object
pattern	predictors matching this pattern will be dropped
...	further arguments passed on to <a href="#">grepl</a>

## Details

form is divided into its individual terms, any term matching pattern is removed, before form is updated and returned. In case no match is made, form is returned unmodified. In case all predictors match, only the intercept is retained. In any case the response variable(s) are kept as is.

## Value

A formula object

## See Also

[drop\\_randfx](#)

## Examples

```

f6 <- y ~ aa*bb + aa + ac + cc + acab

drop_pattern(f6, "a") # Drop all containing a
drop_pattern(f6, "a{2}") # Drop all containing exactly 2 consecutive as
drop_pattern(f6, "^[^a]*a[^a]*$") # All containing exactly 1 a
drop_pattern(f6, ":") # Drop interaction
drop_pattern(f6, "^[^:]*a[^:]*$") # Drop all containg a, but not interaction
drop_pattern(f6, "^(?!a).*$", perl=TRUE) # Drop all not containing a

# Degenerate cases
drop_pattern(f6, "[abc]") # Drop all
drop_pattern(f6, "q") # Drop none

```

---

drop_randfx	<i>Drop random effects</i>
-------------	----------------------------

---

**Description**

Drop random effects from a mixed effects model formula

**Usage**

```
drop_randfx(form)
```

**Arguments**

form                      a formula object

**Details**

form is divided into its individual terms, any term containing a vertical bar (|) is removed, before form is updated and returned. In case form has no random effect terms, form is returned unmodified. In case all effects are random, only the intercept is retained. In any case the response variable(s) are kept as is.

**Value**

A formula object

**See Also**

[drop\\_pattern](#)

**Examples**

```
f1 <- Reaction ~ (1 + Days | Subject)
f2 <- Reaction ~ (1 | mygrp/mysubgrp) + (1 | Subject)
f3 <- Reaction ~ x1 + x2 + (1 + Days | Subject)
f4 <- Reaction ~ x1 * x2 + (1 | mygrp/mysubgrp) + (1 | Subject)
f5 <- Reaction ~ x1 + x2

sapply(list(f1, f2, f3, f4, f5), drop_randfx)
```

---

dtf_clean	<i>Data cleanup</i>
-----------	---------------------

---

**Description**

Create a data.frame from a messy table

**Usage**

```
dtf_clean(x, header = TRUE, na.strings = c("NA", "N/A"),
  stringsAsFactors = FALSE, ...)
```

**Arguments**

x	a messy table the form of a character string
header	does the table include headers? (default TRUE)
na.strings	a vector of character strings which will be interpreted as missing values
stringsAsFactors	should strings be read as factors? (default FALSE)
...	further arguments passed to read.table

**Examples**

```
## Not run:
```

```
x1 <- "
```

```
+-----+-----+-----+-----+-----+
|   Date   | Emp1 | Case | Priority | PriorityCountinLast7days |
+-----+-----+-----+-----+-----+
| 2018-06-01 | A    | A1   | 0       | 0 |
| 2018-06-03 | A    | A2   | 0       | 1 |
| 2018-06-02 | B    | B2   | 0       | 2 |
| 2018-06-03 | B    | B3   | 0       | 3 |
+-----+-----+-----+-----+-----+
```

```
"
```

```
x2 <- '
```

```
+-----+-----+-----+-----+-----+
|   Date   | Emp1 | Case | Priority | PriorityCountinLast7days |
+-----+-----+-----+-----+-----+
| 2018-06-01 | A    | "A 1" | 0       | 0 |
| 2018-06-03 | A    | "A 2" | 0       | 1 |
| 2018-06-02 | B    | "B 2" | 0       | 2 |
| 2018-06-03 | B    | "B 3" | 0       | 3 |
+-----+-----+-----+-----+-----+
```

```
'
```

```
x3 <- "
```

```
      Date   | Emp1 | Case | Priority | PriorityCountinLast7days
2018-06-01 | A    | A|1  | 0       | 0
2018-06-03 | A    | A|2  | 0       | 1
2018-06-02 | B    | B|2  | 0       | 2
2018-06-03 | B    | B|3  | 0       | 3
```

```
"
```

```
x4 <- "
```

```
Maths | English | Science | History | Class
0.1   | 0.2     | 0.3     | 0.2     | Y2
0.9   | 0.5     | 0.7     | 0.4     | Y1
0.2   | 0.4     | 0.6     | 0.2     | Y2
0.9   | 0.5     | 0.2     | 0.7     | Y1
"
```

```

x5 <- "
      Season | Team | W | AHWO
-----|-----
1 | 2017/2018 | TeamA | 2 | 1.75
2 | 2017/2018 | TeamB | 1 | 1.85
3 | 2017/2018 | TeamC | 1 | 1.70
4 | 2017/2018 | TeamD | 0 | 3.10
5 | 2016/2017 | TeamA | 1 | 1.49
6 | 2016/2017 | TeamB | 3 | 1.51
7 | 2016/2017 | TeamC | 2 | 1.90
8 | 2016/2017 | TeamD | 0 | N/A
"

lapply(c(x1, x2, x3, x4), dtf_clean)

## End(Not run)

```

---

dusd

*Discrete (Uniform) Sum Distributions*


---

## Description

Generate distributions of the sum of discrete (uniform) random variables. Two different approaches.

## Usage

```
dusd1(xr = 1:6, n = 2, FUN = "+")
```

```
dusd2(xi = rep(1, 6), n = 2, bix = 1, round, limit = 1e-13)
```

## Arguments

xr	numeric vector; a vector of equiprobable values
n	integer; the number of distributions to be summed
FUN	function passed on to outer
xi	numeric vector; a vector of probabilities, with indices representing values
bix	logical; where does the index of xi start?
round	integer; number of digits to round to after each convolution
limit	numeric; values (frequencies or counts) less than this will be omitted.

## Details

dusd1 works by recursively taking the outer sum of xr, while dusd2 recursively convolves xi. Although convolution is more efficient, it can introduce small errors, and with repeated convolutions those errors can compound. By rounding to a slightly lower precision after each convolution the generation of spurious singletons and general imprecisions can be mitigated.

## Value

dusd1 returns an array of size  $\text{length}(\text{xr})^n$  representing every possible outcome. dusd2 returns a probability mass function in the form of a table.

**See Also**

[combodice](#) for a more flexible implementation of the same ideas

**Examples**

```
# five coin flips
plot(table(dusd1(0:1, 5)))
plot(dusd2(c(1, 1), 5, bix=0))
plot(as.table(dbinom(0:5, 5, 0.5)))

# ten flips with a loaded coin
plot(table(dusd1(c(1, 1, 2), 10)))
plot(dusd2(c(2, 1), 10))
plot(dbinom(0:10, 10, 1/3), type="h", lwd=2)

# sample from a multi-roll d4 distribution
sample(dusd1(1:4, 5), 20, replace=TRUE)
plot(ecdf(dusd1(1:4, 5)))

tt <- dusd2(xi=rep(1, 4), n=3)
plot(tt)
tt <- tt/sum(tt)
rr <- replicate(50000, sample(names(tt), prob=tt))
barplot(apply(rr, 1, table), beside=TRUE)

# distribution of the sum of three d6 rolls
plot(table(dusd1(xr=1:6, 3)))
plot(dusd2(xi=rep(1, 6), n=3))

# d6 die with faces 2, 3, 5, 7, 11, 13 (prime numbers)
plot(table(dusd1(xr=c(2, 3, 5, 7, 11, 13), 3)))

# Probability of getting 7 or 8 with an 8-sided die in n out of 5 throws
l <- 6/8
h <- 1-l
d <- as.dice(c(l, h), bix=0)

dusd2(d, 5)
# need integer "probabilities" for dusd1
table(dusd1(d*4, 5))/(4^5)
# or an equivalent die
table(dusd1(c(0, 0, 0, 1), 5))/(4^5)

# Loaded die
p <- c(0.5, 1, 1, 1, 1, 1.5); sum(p)
plot(dusd2(xi=p, n=2))

# A loaded die with prime number faces
s <- vector(length=13)
s[c(2, 3, 5, 7, 11, 13)] <- c(0.5, 1, 1, 1, 1, 1.5)
plot(dusd2(xi=s, n=3))

# tricky to do with dusd2
plot(table(dusd1(xr=c(0.1105, 2, exp(1)), 10)))

# Demonstrating CLT
```

```

# dusd1 struggles with many iterations
# remember it returns an array of size length(xr)^n
plot(table(dusd1(xr=c(1, 2, 9), 12)))

s <- vector(length=9)
s[c(1, 2, 9)] <- 1
plot(dusd2(xi=s, 12, round=9)) # much quicker
plot(dusd2(xi=s/sum(s), 12)) # for frequencies instead of counts

# Impossible with dusd1
clt <- dusd2(xi=s, 15, round=9)
plot(clt, lwd=0.5, col="#00000088")

# small floating-point errors from convolution.
tail(dusd2(xi=s, 15))

# dusd2 isn't always quicker
## Not run:
plot(table(dusd1(xr=c(1, 220, 3779), 12)), lwd=1)
s2 <- vector(length=3779)
s2[c(1, 220, 3779)] <- 1
plot(dusd2(xi=s2, 12, round=8), lwd=1)

# making sure the length of xi is highly composite (or more precicely 'smooth')
# improves speed
# 3779 is prime, 3780 == 2*2*3*3*3*5*7
s3 <- vector(length=3780)
s3[c(1, 220, 3779)] <- 1
plot(dusd2(xi=s3, 12, round=9), lwd=1)

## End(Not run)

```

---

Elo\_rating

*Elo rating*


---

## Description

Calculate updated Elo ratings based on existing rating and results from matches

## Usage

```
elo_upd(ra, rb, score, k = 16, sum = length(rb) > 1)
```

```
elo_upd_pw(ra, rb, score, k = 16)
```

## Arguments

ra	rating of player A
rb	rating of player B
score	score respective to player A. Numeric or character; 1, 0.5, 0; win, tie, loss
k	a measure of how big the correction should be
sum	calculate the new rating based on sum of scores and opponents ratings

**Examples**

```
# as in example from
# https://en.wikipedia.org/wiki/Elo_rating_system#Mathematical_details
# per 2019-10-04
ra <- 1613
rb <- c(1609, 1477, 1388, 1586, 1720)
score <- c(0, 0.5, 1, 1, 0)
elo_upd(ra, rb, score, k=32, sum=FALSE)
elo_upd(ra, rb, score, k=32, sum=TRUE)

elo_upd_pw(c(1400, 1500), c(1300, 1400), c("w", "t"))

results <- read.table(text="
Player1  Player2  Result
Alice    Bob        Win
Charlie  Dennis     Loss
Elena    Frank      Loss
June     Rashida    Tie", header=TRUE, stringsAsFactors=FALSE)

scores <- read.table(text="
Player  Score
Alice   1150
Charlie 1150
Frank   1150
Bob      800
Dennis  800
Elena   800
June    900
Rashida 1100", header=TRUE, stringsAsFactors=FALSE)

rownames(scores) <- scores$Player

r2 <- results
r2[,1:2] <- scores[as.matrix(r2[,1:2]), 2]

r2u <- elo_upd_pw(r2)

scores.new <- data.frame(Score=c(r2u))
rownames(scores.new) <- as.matrix(results[,1:2])

scores.new <- round(scores.new[rownames(scores),, drop=FALSE])
scores.new$diff <- scores.new$Score - scores$Score
scores.new
```

---

entropy

*Information entropy*


---

**Description**

Computes the information entropy (also called Shannon entropy) of a set of discrete values, or a tabulated such set.

**Usage**

```

entropy(x, ...)

## S3 method for class 'table'
entropy(x, base = 2, ...)

## S3 method for class 'data.frame'
entropy(x, base = 2, ...)

## S3 method for class 'matrix'
entropy(x, base = 2, ...)

## Default S3 method:
entropy(x, base = 2, ...)

```

**Arguments**

x	a vector, table, data.frame or matrix. In the case of table, data.frame and matrix each row is treated as a separate set of counts or proportions, with columns representing species, types, categories etc.
...	further arguments passed to methods
base	the log base to be used.

**Examples**

```

entropy(c(5, 5, 4, 4, 2, 3, 5)) # default is unit bits
entropy(c(5, 5, 4, 4, 2, 3, 5), base=exp(1)) # unit nats

entropy(rep(1:4, 1:4), 4)
entropy(rep(1:4, 1), 4)

entropy(as.factor(c(1, 1, 2, 3, 4, 4)))
entropy(as.character(c(1, 1, 2, 3, 4, 4)))

mtctab <- table(mtcars$cyl, mtcars$carb)
entropy(mtctab, 6)

xx <- data.frame(bee=c(0, 0, 1, 2, 3, 2, 0, 3),
                 wasp=c(1, 3, 2, 0, 1, 1, 2, 1),
                 fly=c(1, 2, 4, 2, 1, 0, 1, 0),
                 beetle=c(1, 0, 0, 1, 2, 2, 0, 2),
                 butterfly=c(0, 0, 0, 0, 3, 1, 0, 1))

entropy(xx)

```

---

every\_nth

*Select every n'th element*


---

**Description**

Select every second, third, fourth etc. element (or slice/hyperplane) of an object



**Usage**

```

every_nth(...)

## Default S3 method:
every_nth(x, n = 2, start = 1, ...)

## S3 method for class 'matrix'
every_nth(x, n = 2, start = 1, margin = 1, ...)

## S3 method for class 'array'
every_nth(x, n = 2, start = 1, margin = 1, ...)

## S3 method for class 'data.frame'
every_nth(x, n = 2, start = 1, margin = 1, ...)

## S3 method for class 'list'
every_nth(x, n = 2, start = 1, ...)

```

**Arguments**

...	further arguments passed to methods
x	an object to be selected from
n	selection "step size"
start	integer in [1:n] specifying the start of selection
margin	what margin to select along

**Examples**

```

m <- matrix(1:64, 8)
every_nth(m, n=3, start=3, margin=2)

d <- data.frame(A=1:8, B=2:9, Q=letters[rep(1:3, length.out=8)])
every_nth(d, start=2)

a <- array(1:6^4, rep(6, 4))
every_nth(a)

l <- list(a=1:3, b=2:6, c=8:5, d=9:7, e=list(ea=1:2, eb=1), f=2:6)
every_nth(l, n=2, start=2)

```

---

expand

*Expand*


---

**Description**

Expand a "table", a "table"-like object, or a list of "table"-like objects

**Usage**

```

expand(x, ...)

```

**Arguments**

`x`                      an object to be expanded  
`...`                    further arguments passed to or from methods

**Value**

A vector with values and their repetitions specified by `x`

**See Also**

[dice](#), [table](#)

**Examples**

```
x <- c(4, 2, 2, 2, 3, 3, 2, 4, 6, 6)
(xt <- table(x))
(xd <- dice(x))

expand(xt)
expand(xd)

expand(list(xt, xd, x))

xn <- as.table(1:4)
names(xn) <- LETTERS[1:length(xn)]
expand(xn)
```

---

explode

*Explode object*

---

**Description**

Presents an R object in an exploded, or expanded, form

**Usage**

```
explode(x, indent = 2)
```

**Arguments**

`x`                      an R object, or a character string describing an R object  
`indent`                how many spaces for indention (and exdention) at each level

**Details**

If `x` is an R object it is first deparsed and converted into a character string describing the object. This string is then unwrapped, or exploded, according to these rules: newline and exdention after each open parenthesis, newline and indention after each close parenthesis, and newline after each comma. Parentheses and commas forming part of character strings are ignored.

Value

An exploded representation of the object is printed to console, and returned invisibly. The output is in most cases a complete and reproducible representation of the object, similarly to `dput`, but less compact and more revealing of its inner structure.

See Also

[dput](#), [dput2](#)

Examples

```
xc <- 'list(v=1, A=c("abv", "bom"), B=c(1:3, 31, 28), list("foo", "bar", 1))'
explode(xc)

x1 <- list(O=NA, R=list(j=1:3, h="(a)", q=structure(list(a=1:2, b=c("A, K",
  "B, L")), class="data.frame", row.names=c(NA, -2L))), N=1, L=FALSE)
explode(x1)

mt <- 'coplot(mpg ~ disp | as.factor(cyl), data = mtcars,
  panel = panel.smooth, rows = 1)'
explode(mt)
```

---

factorise	<i>Factorise</i>
-----------	------------------

---

Description

Find the prime factors of a given integer

Usage

```
factorise(x)
```

Arguments

x                      integer

Value

An integer vector

See Also

[factors](#) for unique prime factors or all integer factors

**Examples**

```
factorise(320)
factorise(2 * 2 * 2 * 3 * 3 * 5)

prod(factorise(5641324))

## Not run:
factorise(nextn(60000000, c(2, 3)))
factorise(72*999983)

## End(Not run)
```

---

factors	<i>Factors</i>
---------	----------------

---

**Description**

Find the integers a given number is divisible by

**Usage**

```
factors(x, prime = FALSE)
```

**Arguments**

x	an integer
prime	should only prime factors be returned?

**Value**

An integer vector

**Note**

The trivial factors 1 and x itself are not included.

**See Also**

[factorise](#) for prime factorisation

**Examples**

```
factors(210)
factors(210, prime=TRUE)
```

---

file_ext	<i>File extension</i>
----------	-----------------------

---

## Description

Separate file name and extension from a file path

## Usage

```
file_ext(x)
```

```
file_name(x)
```

```
file_name_ext(x)
```

## Arguments

x                      a character vector

## Details

If the supplied file name has several extensions, f.ex. like `foobar.tar.bz`, only the last extension will be considered.

## Value

`file_ext` returns the file extension of each file path. `file_name` returns the file name, no extension, of each file path. `file_name_ext` returns both name and extension, but arranged in separate columns of a matrix.

## See Also

[basename](#)

## Examples

```
x <- c("/hg/.gi.tar.gz", "ff/hg/hh.pdf", "git", ".History", ".History.log")

file_ext(x)
file_name(x)
file_name_ext(x)
```

---

`fingerknit`*Render R example*

---

**Description**

Render example from R code stored on the clipboard

**Usage**

```
fingerknit()
```

**Details**

Similar to `reprex::reprex()` or `knitr::spin(text=, envir=new.env(), report=FALSE)`, but stripped down to the very basics. Input is plain valid R code taken from the clipboard. It is run in a fresh environment and both commands and results are captured. Commands are kept as is, but results are commented out. Instead of using three backticks to indicate code for markdown, each line has four whitespaces prepended. `fingerknit` output is also valid R code.

**Value**

The clipboard is used for both input and output, but the output is also returned invisibly as a character string. Warnings and errors are not captured, but printed to console as normal. If an error is encountered nothing is returned and the clipboard data remains unchanged.

---

`fitrange`*Fit to a range*

---

**Description**

Linearly shift and scale a numeric vector so that it fits to a given range.

**Usage**

```
fitrange(x, lower = -1, upper = 1)
```

**Arguments**

<code>x</code>	a numeric vector
<code>lower</code>	the lower bound of the new vector
<code>upper</code>	the upper bound of the new vector

**See Also**

[norma](#)

**Examples**

```
range(fitrang(runif(10, -2, 1.5), 0, 1))

fitrange(c(2, 3, 5, 7, 4), 1, 0)
# same, but without warning
1 - fitrange(c(2, 3, 5, 7, 4), 0, 1)
```

---

flatten	<i>Flatten list</i>
---------	---------------------

---

**Description**

Flatten a (nested) list to a list of its leaves

**Usage**

```
flatten(x, flatten.df = FALSE, keep.order = TRUE)
```

**Arguments**

x	a list object
flatten.df	should data.frames also be flattened?
keep.order	keep the order of the original list, same as seen when using str

**Details**

The nodes of the supplied list is traversed from root to leaf and successively unlisted until no lists are left (except possibly for data.frames).

**Value**

A single level list of x's leaves.

**Examples**

```
x1 <- list(
  O=NA,
  R=list(
    j=1:3,
    h="(a)",
    q=data.frame(
      a=1:2,
      b=c("A, K", "B, L"),
      stringsAsFactors=FALSE
    )
  ),
  N=1,
  L=FALSE
)

flatten(x1, flatten.df=TRUE, keep.order=FALSE)
flatten(x1, flatten.df=TRUE, keep.order=TRUE)
str(x1)
```

---

gcd	<i>Greatest common divisor</i>
-----	--------------------------------

---

**Description**

Find the largest integer, that when two numbers are divided by it, returns an integer in both cases

**Usage**

```
gcd(x, y)
```

**Arguments**

x, y	integers whose greatest common divisor is to be found
------	---

**Examples**

```
gcd(sequence(10:16), rep(10:16, 10:16))
```

---

incdiff	<i>Increase difference</i>
---------	----------------------------

---

**Description**

Rearrange a sorted numeric sequence so that the difference between subsequent elements is increased

**Usage**

```
incdiff(x, step = 2)
```

**Arguments**

x	a numeric sequence
step	how long a step the difference is considered for.

**Details**

With step=2 (default) only the difference between immediate neighbours are considered; the difference between every second element will remain small, or rather reduced, compared to the original sequence. With step=3 say, differences of both lag 1 and 2 is increased, but the difference of lag 1 will be less than if a step of 2 was used.



**Examples**

```

x <- 1:100
diff(x)

diff(incdiff(x, 2))
diff(incdiff(x, 3))

diff(incdiff(x, 2), 2)
diff(incdiff(x, 3), 2)

# incdiff will introduce a periodicity equal to the step length
acf(incdiff(x, 10))

# useful for making a sequence of colours more distinct
y <- seq(0.4, 1, l=18)
cols1 <- hsv(y, 1, y)
cols2 <- hsv(y, 1, incdiff(y, 3))

plot(y, col=cols1, pch=16, cex=5, ylim=c(0.4, 1.5))
points(y+0.5, col=cols2, pch=16, cex=5)

```

indexvalue

*Index-value representation of arrays***Description**

Represent an array as columns of dimensional indices and value

**Usage**

```
indexvalue(x, reverse = FALSE)
```

**Arguments**

x	an array or something that can be coerced into an array
reverse	logical; convert from Index-value representation to regular array representation?

**Details**

An n-dimensional array will be unfolded to a n+1-column data.frame where the first n columns represent the indices of the n dimensions, and the last column gives the value found at each index tuple. The reverse process can also be performed.

**See Also**

[latin\\_sq](#)

**Examples**

```

arr <- array(1:(2*3*4), dim=c(2, 3, 4))
arr.is <- indexvalue(arr)

# can be used to permute an array
indexvalue(arr.is[,c(2, 1, 3, 4)], rev=TRUE)
aperm(arr, c(2, 1, 3))

# can interpret values (symbols) as dimensional indices and permute them as well
arr2 <- array(rep(1:6, 4), dim=c(2, 3, 4))
arr2.is <- indexvalue(arr2)
indexvalue(arr2.is[,c(1, 2, 4, 3)], rev=TRUE)

# a latin square will produce an "orthogonal array"
set.seed(1)
lsq <- latin_sq(5)
iv <- indexvalue(lsq)
iv

# any permutation of a latin square is also a latin square
indexvalue(iv[, c(1, 3, 2)], reverse=TRUE)

```

intsect

*Intersect***Description**

Performs set intersection on a list of vectors

**Usage**

```
intsect(x)
```

**Arguments**

x                      list of sets (vectors of same mode or factors)

**Details**

The intersection between the sets in the list is found. This means no duplicate values are returned, whether or not there were any in the input.

**Value**

A vector of same mode as input, or a single factor object if input was factor.

**Examples**

```

intsect(list(0:6, c(2, 4, 6, 8), 3:8))

fc <- factor(LETTERS[sample(1:5, 20, rep=TRUE)])
fc1 <- split(fc, sample(1:3, 20, rep=TRUE))

intsect(fc1)

```

---

in_range	<i>Test if values is in a given range</i>
----------	---

---

**Description**

Checks either whether both extrema are in the given range, or if each individual value is.

**Usage**

```
in_range(x, lower, upper, inc = c(TRUE, TRUE), na = NA)
```

**Arguments**

x	any atomic or vector-like object
lower	lower range
upper	upper range
inc	logical vector of length one or two; should the lower and upper ranges, respectively, be considered inclusive?
na	should NAs in x return TRUE, FALSE, NA, or something else?

**Examples**

```
in_range(c(1:3, NA), 1, 4, na=NA)

in_range(1:4, 1, 4)
in_range(matrix(1:4, 2), 1, 4)

in_range(1:4, 1, 4, inc=1:0)
in_range(1:4, 1, 4, inc=0:1)
in_range(1:4, 1, 4, inc=0)

x <- as.Date(0:3, origin="2000-01-01")
in_range(x, "2000-01-01", "2000-01-04")

in_range(letters[1:4], "a", "d", inc=1:0)
in_range(letters[1:4], "a", "da", inc=1:0)

# no upper range
in_range(c(10^rnorm(9), NA), 0, NA)
in_range(c(10^rnorm(9), NA), 0, NA, na=TRUE)
```

---

is_coprime	<i>Coprimality check</i>
------------	--------------------------

---

**Description**

Test whether two integers are coprime, that is, have no factors in common

**Usage**

```
is_coprime(x, y)
```

**Arguments**

`x`, `y` integers to be tested for coprimality

**Value**

A logical vector

**Examples**

```
is_coprime(sequence(10:16), rep(10:16, 10:16))
is_coprime(2*3*5*7, 11*13)
```

---

<code>is_prime</code>	<i>Primality check</i>
-----------------------	------------------------

---

**Description**

Test integers for whether they are prime or not

**Usage**

```
is_prime(x)
```

**Arguments**

`x` vector of integers

**See Also**

[primes](#)

---

<code>keep_finite</code>	<i>Keep finite values</i>
--------------------------	---------------------------

---

**Description**

Remove NAs codeNaNs and codeInfs from data

**Usage**

```
keep_finite(x, ...)

## Default S3 method:
keep_finite(x, ...)

## S3 method for class 'matrix'
keep_finite(x, margin = 1, keep = c("any",
  "complete"), ...)

## S3 method for class 'data.frame'
keep_finite(x, margin = 1, keep = c("any",
  "complete"), ...)
```

**Arguments**

x	a vector or matrix
...	further arguments passed to methods
margin	if x is matrix, which margin to keep finites by
keep	if x is matrix, keep rows/columns with any finite values, or keep only complete rows/columns.

**Value**

If x is a matrix and margin is 1 or 2, a matrix is returned. Else a vector.

**Examples**

```
m1 <- matrix(c(10, 20, 30, 43,
               10, NA, 32, 50,
               NA, NA, NA, NA,
               13, 22, 70, 81,
               NA, 29, NA, 41), 5, byrow=TRUE,
             dimnames=list(letters[1:5], LETTERS[1:4]))

keep_finite(m1)
matplot(keep_finite(apply(m1, 2, sort, na.last=TRUE)), type="l")

m1[complete.cases(m1),]
keep_finite(m1, 1, "c") #same
keep_finite(m1, 2, "complete") #no complete columns

m1.df <- as.data.frame(t(m1))
keep_finite(m1.df, 2, "complete")
```

---

lag\_vector

*Lag an arbitrary vector*


---

**Description**

Lag an arbitrary vector

**Usage**

```
lag_vector(x, k, type = c("cycle", "na.fill", "trim"), ...)
```

**Arguments**

x	vector to be lagged
k	integer vector specifying the number of lags
type	how to deal with non-overlapping sections
...	further arguments passed to methods

**Examples**

```
x <- 1:9

lag_vector(x, c(0, 1, -2, 3))
lag_vector(x, c(0, 1, -2, 3), "na")
lag_vector(x, c(0, 1, -2, 3), "trim")
```

---

latin_sq	<i>Latin square</i>
----------	---------------------

---

**Description**

Generate latin squares, either randomly or ordered

**Usage**

```
latin_sq(n, random = TRUE, reduce = TRUE)
```

**Arguments**

n	integer; number of unique values (aka. symbols)
random	logical; should the square be generated randomly?
reduce	logical; should the square be in reduced form?

**Details**

Computation time increases rapidly with n. On my computer generating a random square with n=12 takes about ten minutes, marking the upper limit of practicability, or even stretching it a little. A latin square in reduced form will have elements in the first row and the first column in a sorted order. By setting reduced=TRUE the first row and the first column will always be 1:n.

**Value**

A square integer matrix of size  $n^2$

**See Also**

[indexvalue](#)

**Examples**

```
set.seed(1)
ls <- latin_sq(9, reduce=TRUE)
image(ls, col=randcolours(ncol(ls)))

# The more "classic" representation with latin capital letters
ls[] <- LETTERS[ls]
ls
```

---

markov_seq	<i>Discrete markov sequence</i>
------------	---------------------------------

---

**Description**

Generate a random discrete markov sequence

**Usage**

```
markov_seq(n = 100, tmat = rbind(1:3, 3:1, 2:0), init = 1)
```

**Arguments**

n	length of the sequence
tmat	a transition matrix
init	the initial state

**Examples**

```
m <- matrix(c(0.5, 0.3, 0.2,
              0.2, 0.6, 0.2,
              0.2, 0.3, 0.5), 3, byrow=TRUE)

set.seed(1)
ms <- markov_seq(n=1000, tmat=m)

colMeans(m)
prop.table(table(ms))
round(prop.table(table(head(ms, -1), tail(ms, -1), dnn=c("n", "n+1")), 1), 2)
```

---

math_constants	<i>Mathematical constants</i>
----------------	-------------------------------

---

**Description**

Various mathematical constants available as global variables

**Format**

An object of class `numeric` of length 1.

**Details**

e Euler's number  
 pi Archimedes' number, the circle constant  
 phi Golden ratio  
 feig1 Feigenbaum's first constant,  $\delta$ ; bifurcation velocity  
 feig2 Feigenbaum's second constant,  $\alpha$ ; reduction parameter  
 eu.ma Euler–Mascheroni constant  
 khin Khintchine's constant  
 glai.kin Glaisher–Kinkelin constant

---

math_constants_char	<i>High precision mathematical constants</i>
---------------------	--

---

### Description

Character strings representing various mathematical constants to ~100 decimal points

### Format

An object of class character of length 1.

### Details

e.char Euler's number  
 pi.char Archimedes' number, the circle constant  
 phi.char Golden ratio  
 feig1.char Feigenbaum's first constant,  $\delta$ ; bifurcation velocity  
 feig2.char Feigenbaum's second constant,  $\alpha$ ; reduction parameter  
 eu.ma.char Euler–Mascheroni constant  
 khin.char Khintchine's constant  
 glai.kin.char Glaisher-Kinkelin constant

---

means	<i>Generalized means</i>
-------	--------------------------

---

### Description

Harmonic, geometric, quadratic, cubic, power and Lehmer means.

### Usage

```
harm(x, na.rm = TRUE)

geom(x, zero.rule = c("1p", "rm", "1"), na.rm = TRUE)

quad(x, na.rm = TRUE)

cubi(x, na.rm = TRUE)

powr(x, p = 1.5, na.rm = TRUE)

lehm(x, p = 2, na.rm = TRUE)
```



**Arguments**

x	numeric vector of values whose *mean is to be computed
na.rm	logical; should NA values be removed? (default TRUE)
zero.rule	for the geometric mean, how should zeros be dealt with? Add one before, and subtract one after the calculation (see <code>lop1p</code> ), remove all zeros, or replace all zeros with 1.
p	exponential power. For the power mean $p=-1$ , $p=2$ and $p=3$ gives the harmonic, quadratic and cubic means, respectively. For the Lehmer mean $p=0$ , $p=1$ and $p=2$ gives the harmonic, arithmetic and contraharmonic means, respectively.

**Notice**

For some of these means zeros and/or negative values are undefined, or make otherwise little sense in context. Workarounds are given for the geometric mean, but if you end up using it on data  $\leq 0$ , the wise call would be to reconsider whether using a geometric mean really makes sense in that case.

**See Also**

[central.tendency](#)

**Examples**

```
funl <- substitute(c(harm, geom, mean, quad, cubi))

x1 <- list(c( 1, 2, 3, 5),
          c(-1, 1, 2, 3, 5),
          c( 0, 1, 2, 3, 5),
          c(-1, 0, 1, 2, 3, 5))

m <- sapply(x1, function(x) sapply(eval(funl), function(f) f(x)))
rownames(m) <- as.character(funl)[-1]
colnames(m) <- c("posi", "1neg", "zero", "1ngz")
round(m, 3)

harm(x1[[1]]); powr(x1[[1]], -1); lehm(x1[[1]], 0)

y <- c(0, 1, 5, 0, 6, 5, 9)

geom(y, zero.rule="1p")
geom(y, zero.rule="rm")
geom(y, zero.rule="1")
```

---

merge\_multiple

---

Merge multiple data.frames

---

**Description**

Merge multiple data.frames

**Usage**

```
merge_multiple(x, by, all = FALSE, sort = TRUE, incomparables = NULL,
  include = NULL, exclude = NULL)
```

**Arguments**

<code>x</code>	a list of <code>data.frames</code> with at least one column in common
<code>by</code>	name of the column to be merged by, by default the full intersect of column names between <code>data.frames</code>
<code>all</code>	include all rows, including those with no match
<code>sort</code>	sort the output on the <code>by</code> column(s)
<code>incomparables</code>	values which cannot be matched.
<code>include, exclude</code>	numeric, logical or character vector specifying which columns to include in or exclude from the merge

**Details**

If there are duplicate columns that aren't being used to merge by, one of two things will happen. If the parent `data.frames` of the duplicate columns are named, then that name will be appended to the relevant column names. If the `data.frames` aren't named, then the `data.frames` indices in the parent list are appended to the relevant column names.

Inclusion and exclusion are performed in sequence, so that if both `include` and `exclude` are specified, `exclude` acts on the result from `include`.

**Examples**

```
dtf1 <- data.frame(ast=1:4, bar=1:4, kat=c("A", "B", "C", "D"))
dtf2 <- data.frame(ast=1:6, bar=1:6, jun=9:4)
dtf3 <- data.frame(ast=2:6, bar=2:6, kat=c("A", "B", "C", "D", "E"))
dtf4 <- data.frame(ast=3:4, bar=3:4)
dtf5 <- data.frame(ast=1:-3, bar=0:4, git=0:4)

ll <- list(d1=dtf1, d2=dtf2, dtf3, A=dtf4, dtf5)

merge_multiple(ll, by="bar")
merge_multiple(ll, by="bar", all=TRUE, include=1:2)
merge_multiple(ll, by="bar", all=TRUE, exclude="kat")
merge_multiple(x=ll, by=c("bar", "ast"), all=TRUE)
```

**Description**

Plot multiple kernel density estimates in the same window, together with a legend

**Usage**

```
multidensity(x, main, xlab = "", ylab = "Density", xlim, ylim,
  col = 1:9, lty = 1:2, lwd = 1, add = FALSE, frame.plot = TRUE,
  legend = TRUE, x.legend = "topleft", y.legend = NULL, bty = "o",
  box.col = "#FFFFFF00", bg.legend = "#FFFFFFAA", cex.legend = 0.7,
  x.intersp = 1, y.intersp = 1.5, inset = 0, xpd.legend = NA,
  horiz = FALSE, ...)
```

**Arguments**

<code>x</code>	a list or data.frame of numeric values
<code>main</code>	a main title for the plot. Defaults to the call made to density
<code>xlab</code> , <code>ylab</code>	labels for the x and y axes
<code>xlim</code> , <code>ylim</code>	the x and y limits of the plot
<code>col</code> , <code>lty</code> , <code>lwd</code>	the line colours, types and widths for lines appearing in plot and legend
<code>add</code>	if TRUE, add to the current plot
<code>frame.plot</code>	an integer indicating whether a box should be drawn around the plot before the legend (1), after the legend (2), or not at all (0). Logical values are coerced to integer, so TRUE implies 1, and FALSE implies 0
<code>legend</code>	logical; if TRUE (the default) a legend is included with the plot
<code>x.legend</code> , <code>y.legend</code>	the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by <code>xy.coords</code>
<code>bty</code>	legend box type
<code>box.col</code>	line colour for the legend box
<code>bg.legend</code>	background colour for the legend box
<code>cex.legend</code>	character expansion faftor for legend
<code>x.intersp</code> , <code>y.intersp</code>	horizontal and vertical character interspacing for legend
<code>inset</code>	the legends inset distance from the margins as a fraction of the plot region
<code>xpd.legend</code>	the value of <code>xpd</code> to be used while drawing the legend
<code>horiz</code>	logical; if TRUE, set the legend horizontally rather than vertically
<code>...</code>	further arguments passed to density

**Value**

An invisible list of the "density" objects the plot is based on.

**See Also**

[density](#), [ahist](#)

## Examples

```
set.seed(1)
dl <- list("Unif-1"=runif(80, -2.1, 2.1),
           "Unif-2"=runif(70, -1.5, 1.5),
           "Normal-1"=rnorm(50, 0, 0.866),
           "Normal-2"=rnorm(90, 0, 1))

# sqrt((sd^2)*12) # sd to unif range

md <- multidensity(dl)
head(md, 2)

multidensity(dl, adj=1.2, x.legend="topright", frame=FALSE, inset=-0.02, lty=1)
multidensity(dl, x.legend="top", horiz=TRUE, cex.legend=0.5,
             inset=-0.05, bg.legend="white")
```

---

narm

Remove NAs

---

## Description

Remove NAs from vector or matrix

## Usage

```
narm(x, ...)

## Default S3 method:
narm(x, ...)

## S3 method for class 'matrix'
narm(x, margin = 1, keep = c("any", "complete"), ...)

## S3 method for class 'data.frame'
narm(x, margin = 1, keep = c("any", "complete"),
     ...)
```

## Arguments

x	a vector or matrix
...	further arguments passed to methods
margin	if x is matrix, which margin to remove NAs by
keep	if x is matrix, keep rows/columns with any non-NA values, or keep only complete rows/columns.

## Value

If x is a matrix and margin is 1 or 2, a matrix is returned. Else a vector.

**Examples**

```

m1 <- matrix(c(10, 20, 30, 43,
               10, NA, 32, 50,
               NA, NA, NA, NA,
               13, 22, 70, 81,
               NA, 29, NA, 41), 5, byrow=TRUE,
             dimnames=list(letters[1:5], LETTERS[1:4]))

narm(m1)
matplot(narm(apply(m1, 2, sort, na.last=TRUE))), type="l")

m1[complete.cases(m1),]
narm(m1, 1, "c") #same
narm(m1, 2, "complete") #no complete columns

m1.df <- as.data.frame(t(m1))
narm(m1.df, 2, "complete")

```

norma

*Normalize***Description**

Linearly shift and scale a numeric vector so that it has a given range, about a given centre.

**Usage**

```
norma(x, c = 0, r = 2)
```

**Arguments**

x	a numeric vector
c	the centre (as in the midrange) for the new vector
r	the range of the new vector

**See Also**

[fitrange](#)

**Examples**

```
range(norma(runif(9, -2, 0.1), 0, 2))
```

pairwise

*Apply function to columns/elements pairwise***Description**

Pairwise application of a function to the columns of a matrix/data.frame or elements of a list

**Usage**

```
pairwise(x, FUN, ..., comm = FALSE)
```

**Arguments**

x	a matrix or data.frame
FUN	any function that takes two vectors as input and returns a single value
...	further arguments passed to FUN
comm	logical; is FUN commutative? If true, only the lower triangle, including the diagonal, is computed

**Value**

An  $n \times n$  square matrix with  $n$  the number of columns of x.

**See Also**

[similarity](#) for a few more examples

**Examples**

```
dtf <- data.frame(aa=c(1, 1, 2, 2, 3, 2, 4),
                  bb=c(1, 1, 2, 3, 3, 3, 4),
                  cc=c(3, 3, 2, 1, 1, 1, 1),
                  dd=c(1, 2, 2, 2, 1, 1, 2))

# Root Mean Square Deviation
pairwise(dtf, function(x, y) sqrt(mean((x-y)^2)))

# using with cor.test() to accompany cor()
pv <- pairwise(dtf, function(x, y) cor.test(x, y)$p.val)
pvn <- 6^(1.1-pv)-5
pvn[pvn<1] <- 1

set_mar(1, 1, 1, 1)
plot(0, xlim=c(0.5, 4.5), ylim=c(0.5, 4.5), cex=0, ann=FALSE, xaxt="n", yaxt="n")
text(rep(1:4, 4), rep(4:1, each=4), t(round(cor(dtf), 2)), cex=pvn,
     col=c("black", "darkgrey")[(pv>0.1)+1])
```

---

pcamean	<i>PCA mean</i>
---------	-----------------

---

## Description

Takes the average of several PCA objects

## Usage

```
pcamean(...)
```

## Arguments

...                      prcomp, princomp or factanal objects, or a single list of such objects

## Details

I don't know if this kind of calculation has any sort of merit. It was written more as an impromptu challenge than as a solution to any problem

## See Also

[prcomp](#), [princomp](#), [factanal](#)

## Examples

```
xx <- data.frame(bee=c(0, 0, 1, 2, 3, 2, 0, 3),
                 wasp=c(1, 3, 2, 0, 1, 1, 2, 1),
                 fly=c(1, 2, 4, 2, 1, 0, 1, 0),
                 beetle=c(1, 0, 0, 1, 2, 2, 0, 2))

set.seed(1)
r <- 1000
xxs <- replicate(r, {
  xx$random <- sample(c(0:1, 0:4), 8, r=TRUE)
  xx
}, simplify=FALSE)

xxm <- Reduce("+", xxs) / r
xxl <- lapply(xxs, princomp)

biplot(pcamean(xxl))
biplot(princomp(xxm))
```

---

<code>plot.histogram</code>	<i>Plot histogram object</i>
-----------------------------	------------------------------

---

**Description**

A a very minor modification of `graphics::plot.histogram`.  
Only difference is that `lwd` now specifies the width of the histogram bars' outline.

**See Also**

[plot.histogram](#), [plot.stl](#), [ahist](#)

---

<code>plot.stl</code>	<i>Plot stl object</i>
-----------------------	------------------------

---

**Description**

A a very minor modification of `stats::stl`.  
Only difference is that the distance between the plotting window and the x and y labels is set by `par("mgp")[1]`, as it is for regular plots.

**See Also**

[plot.stl](#), [plot.histogram](#)

---

<code>primes</code>	<i>Prime number generator</i>
---------------------	-------------------------------

---

**Description**

Prime generator based on the sieve of Eratosthenes

**Usage**

```
primes(n)
```

**Arguments**

<code>n</code>	integer; all prime numbers up to this will be returned
----------------	--

**Details**

Effective for primes up to ~100,000,000.  
On my lightweight laptop: 1e7 -> 0.32s, 5e7 -> 1.7s, 1e8 -> 3.7s, 2e8 -> 7.6s, 3e8 -> 15s

**Source**

<https://stackoverflow.com/questions/3789968/generate-a-list-of-primes-up-to-a-certain-number/3791284#3791284>



**See Also**[is\\_prime](#)


---

quartz.png	<i>Save as PNG</i>
------------	--------------------

---

**Description**

Save the contents of the current Quartz window as PNG file

**Usage**

```
quartz.png(file = "%Y%m%d_%H", width = 550, dir, force = FALSE)
```

**Arguments**

file	file name. If it contains any "%" it is passed on as a format string to <code>format(Sys.time(), file)</code> . A .png file extension is added automatically.
width	pixel width of the PNG file
dir	directory to save to. Defaults to current working directory
force	force overwriting of existing file with same name. By default duplicate path names are resolved by appending _N, using successive integers, to the end of the file name.

**Value**

A PNG file is written to disk and a message is written to the console, giving the new file's path and pixel dimensions. The file path is also returned invisibly.

**Examples**

```
## Not run:
set.seed(1)
i_h100 <- round(runif(100, 2, 30), 2)
i_cd <- rexp(100, 1/i_h100)
mydata <- data.frame(i_cd, i_h100)

mydata$i_h100_2m <- cut(mydata$i_h100, seq(2, 30, by=2))

i_cd_2m <- aggregate(i_cd ~ i_h100_2m, mydata, mean)

set_mar(x=2.5)
plot.default(i_cd_2m, xaxt="n", main="Groupwise means", xlab="", cex.main=1)
axis(1, i_cd_2m[,1], as.character(i_cd_2m[,1]), cex.axis=0.6, las=2)

quartz.png()
p <- quartz.png("test", 550)
file.info(p)

## End(Not run)
```

---

quick_table	<i>Tabulate data</i>
-------------	----------------------

---

**Description**

Quick and simple function for creating contingency tables

**Usage**

```
quick_table(x, na.rm = FALSE, order = c("frequency", "value", "none"))
```

**Arguments**

x	a vector or factor object
na.rm	should NAs be included
order	how should the results be ordered, if any?

**Value**

A data.frame with columns val (the original values and class of x) and freq (the count, or frequency, of each value in x, integer). The rows are sorted by frequency in descending order.

**Examples**

```
set.seed(1)
m <- sample(c(rep(NA, 5), rpois(45, 3)))
quick_table(m)

x <- LETTERS[c(2, 2, 2, 2, 3, 1, 1)]
quick_table(x, order="freq")
quick_table(x, order="value")
quick_table(x, order="none")
```

---

rainbowHCL	<i>HCL rainbow palette</i>
------------	----------------------------

---

**Description**

HCL version of rainbow. Create a vector of n contiguous colours by specifying a range of Hues, and fixed Chroma and Luminance

**Usage**

```
rainbowHCL(n, c = 100, l = 75, start = 0, end = max(1, n - 1)/n,
  alpha = 1, s = NULL, v = NULL)
```

**Arguments**

n	number of colours
c, l	the ‘chroma’ and ‘luminance’ to be used to complete the HSV color descriptions
start, end	the hue in [0,1] at which the rainbow begins/ends
alpha	the alpha transparency, a number in [0,1], see argument alpha in <a href="#">hsv</a>
s, v	‘saturation’ and ‘value’ passed to <a href="#">adjustcolorHSV</a> . Overrides ‘chroma’ and ‘luminance’ if specified.

**Examples**

```
mat2grid <- function(x) {
  eg <- expand.grid(1:NCOL(x), NROW(x):1)
  gd <- data.frame(eg, c(t(x)), stringsAsFactors=FALSE)
  colnames(gd) <- c("x", "y", "z")
  gd
}

n <- 25
hcl0 <- rainbowHCL(n)
hcl1 <- rainbowHCL(n, c=150, l=85)
hcl2 <- rainbowHCL(n, s=1, v=1)
hsv0 <- rainbow(n)

cols <- rbind(hcl0, hcl1, hcl2, hsv0)

pos <- mat2grid(cols)
plot(pos[,1:2], pch=17, cex=3.5, col=pos[,3], ylim=c(0.5, 4.5))
```

randcolours

*Random colours***Description**

Generate a randomly selected colour palette

**Usage**

```
randcolours(n, l = c(0.2, 0.9), c1 = c(0, 1), c2 = c(0, 1),
  alpha = 1, space = c("Luv", "Lab"))
```

**Arguments**

n	number of colours
l	lightness range
c1	colour channel one range
c2	colour channel two range
alpha	alpha channel range
space	should the parameters be interpreted as Luv or Lab components?

## Details

The range of 1, c1, c2 and alpha, will be interpreted as the wanted range of each colour component, whether their length is 1, 2, or more. Although they all should nominally lie within [0, 1], only alpha must do so to achieve a valid output. The others can exceed this range, at an increased risk of clipping.

## Examples

```
set.seed(3)
n <- 20
plot(1:n, col=randcolours(n), pch=16, cex=5)
```

---

resolve_dup	<i>Resolve duplicate</i>
-------------	--------------------------

---

## Description

Resolve duplicate names by appending successive integers

## Usage

```
resolve_dup(x, candidates, ignore.extension = FALSE)
```

## Arguments

x	character string; name to be resolved
candidates	character vector; possible duplicate names
ignore.extension	logical; append to the end of x, even if it has something that can be interpreted as an extension

## Examples

```
x <- c("my.var", "aaa.png", "aaa.jpg", "aaa_1.png", "doc-folder")
resolve_dup("aaa.jpg", x)
resolve_dup("aaa.png", x)
resolve_dup("aaa_1.png", x)
resolve_dup("doc-folder", x)
resolve_dup("New Document", x)
resolve_dup("my.var", x, ignore.ext=TRUE)
x <- c(x, resolve_dup("aaa.png", x))
resolve_dup("aaa.png", x)
```

---

revert_par	<i>Revert par</i>
------------	-------------------

---

**Description**

Reverts par settings back to old.par

**Usage**

```
revert_par()
```

**See Also**

Other par\_and\_plot\_margins\_functions: [default\\_par](#), [set\\_mar](#)

---

rle2	<i>Run Length Encoding</i>
------	----------------------------

---

**Description**

Compute the lengths and values of runs of equal values in a vector

**Usage**

```
rle2(x, na.unique = FALSE, output = c("data.frame", "rle",
  "named vector", "lengths", "values"))
```

**Arguments**

x	a numeric or character vector
na.unique	should every NA be considered unique?
output	what form of output

**Value**

Return value depends on output.

`data.frame` A data.frame with lengths and values columns

`rle` An object of class "rle"

`named vector` A vector of lengths with values as names

`lengths` The lengths as a single vector

`values` The values as a single vector

## Examples

```
x <- c(NA, NA, 1, 2, 3, 3, NA, NA, NA, 2, 2, 2, NA, 1, 1, NA, NA)
rle2(x)

m <- matrix(c(
  0.7, 0.2, 0.1,
  0.2, 0.6, 0.2,
  0.1, 0.2, 0.7
), 3, byrow=TRUE)

set.seed(1)
y <- LETTERS[markov_seq(n=100, m)]
rle2(y, out="named")

# Same result as rle
rle2(x, na.unique=TRUE, output="rle")
rle(x)

# inverse.rle works as long as output is "rle"
inverse.rle(rle2(x, output="rle"))
```

---

seq\_range

---

*Generate a sequence spanning a given range*


---

## Description

Generate a sequence spanning a given range

## Usage

```
seq_range(x, ..., spread)
```

## Arguments

x	a single numeric, a range, or a sequence
...	further arguments passed to seq
spread	use spread_seq to spread out the range by a given factor

## Details

If x is a single number, the range is interpreted to be  $[0, x]$ . If x is length two, the numbers are interpreted as the left and right extrema of the sequence interval. If x is longer than two, the sequence is based upon its range.

## Examples

```
seq_range(c(1, 4), by=0.5)
seq_range(c(1, 4), by=0.5, spread=2)
seq_range(4)
```

```
x <- sample(1:10, 3)
seq_range(x)
```

---

set_mar	<i>Set plot margins</i>
---------	-------------------------

---

## Description

Moves axis titles and labels closer to the plotting window and shrinks the margins

## Usage

```
set_mar(x = 1.8, y = 1.8, main = 1, right = 1, cex.main = 1, ...)
```

## Arguments

x	margin width for the x axis, default 2
y	margin width for the x axis, default 2
main	margin width for the main title, default 1
right	margin width for the right edge, default 1
cex.main	The magnification to be used for main titles relative to the current setting of cex, default 1
...	further arguments passed to par

## Details

Old par settings are stored in `.old.par` before a call to `par` of the form `par(mar=c(x, y, main, right), mgp=c(1.9, 0, 0))` is made.

## See Also

[par](#)

Other `par_and_plot_margins_functions`: [default\\_par](#), [revert\\_par](#)

## Examples

```
ymse:::.old.par
get("old.par", envir=ymse::ymseEnv)
ls(envir=ymse::ymseEnv)

par(col.axis=2)
plot(1:4)

set_mar()
plot(1:4)

default_par()
plot(1:4)

revert_par()
plot(1:4)
```

```
ymse:::old.par
head(get("old.par", envir=ymse::ymseEnv))
```

---

similarity

*Similarity measure*


---

## Description

Calculate the similarity between two character vectors based on a similarity matrix

## Usage

```
similarity(x, y, sm = smat(x, y), sfun = sum, ...)
```

## Arguments

x	a character vecor or two-column data.frame/matrix
y	a character vector. Ignored if x is data.frame/matrix
sm	a similarity matrix. By default a unit matrix
sfun	function used to summarise the elementwise similarities
...	further arguments passed to sfun

## See Also

[smat](#)

## Examples

```
# In its most basic form similarity() gives the Hamming distance
similarity(c(1, 0, 1, 0), c(1, 1, 0, 0))
```

```
# Symmetry not required.
bef <- c(1, 2, 3, 1, 2, 3, 1, 2, 3)
aft <- c(0, 2, 2, 1, 2, 2, 1, 1, 2)
```

```
# Here a decrease in value of 1 is considered
# more similar than an increase in value of 1.
sm1 <- t(structure(c(
  3, 0, 0, 0,
  2, 3, 0, 0,
  0, 2, 3, 0,
  0, 0, 2, 3),
  .Dim=c(4L, 4L),
  .Dimnames=list(c("0", "1", "2", "3"), c("0", "1", "2", "3"))))
```

```
# Symmetric version
sm2 <- t(structure(c(
  3, 1, 0, 0,
  1, 3, 1, 0,
  0, 1, 3, 1,
```



```

0, 0, 1, 3),
.Dim=c(4L, 4L),
.Dimnames=list(c("0", "1", "2", "3"), c("0", "1", "2", "3"))))

similarity(bef, aft, sm1)
similarity(bef, aft, sm2)

# Pre-aligned fragments of insulin genes
data(insulin)

# Transition-transversion matrix
data(smt)

# Using pairwise() to run similarity() over all column pairs
pairwise(insulin, similarity, smt, sfun=mean)

# Imagined result from questionnaire
qu <- data.frame(
  Alice=c("happy", "sad", "angry", "unsure", "happy", "sad", "happy", "angry"),
  Bob=c("happy", "sad", "angry", "angry", "happy", "angry", "angry", "sad"),
  Charlie=c("sad", "sad", "unsure", "unsure", "happy", "sad", "angry", "sad"),
  stringsAsFactors=FALSE
)

# Similarity matrix describing the relative similitudes of the moods
emsm <- as.matrix(read.table(text="
      happy  sad  angry unsure
happy    5    0    1     1
sad      0    5    2     1
angry    1    2    4     2
unsure   1    1    2     3",
header=TRUE))

pairwise(qu, similarity, sm=emsm/5, sfun=mean)

```

---

simple\_loess

---

*Simplified Local Polynomial Regression Fitting*


---

## Description

A simplified interface to the loess and predict.loess combo.

## Usage

```

simple_loess(...)

## Default S3 method:
simple_loess(y, x = seq_along(y), xout = sort(x),
  span = 0.75, periodic = FALSE, ...)

## S3 method for class 'data.frame'
simple_loess(df, xout = sort(df[, 1]), ...)

```

**Arguments**

<code>...</code>	further arguments passed to <a href="#">loess</a>
<code>y</code>	the response values to be regressed
<code>x</code>	the regressor, by default an integer sequence along <code>y</code>
<code>xout</code>	values used for prediction, unless it is an integer of length 1. In that case <code>xout</code> specifies the number of equally spaced values on the interval of <code>x</code> to be used. By default the same as <code>x</code>
<code>span</code>	parameter controlling the degree of smoothing
<code>periodic</code>	should the input be treated as periodic?
<code>df</code>	a <code>data.frame</code> with <code>x</code> -values in the first column and <code>y</code> -values in the second

**Value**

A `data.frame` with columns `xout` and `y.predicted`

**Examples**

```
# Simple equally spaced vector
h <- c(-0.63, 0.2, -0.44, 1.6, 0.33, -0.74, -0.82, 0.29, 0.74, 0.58, -0.3)

plot(h)
lines(simple_loess(h))

# More complicated unequally spaced x-values
x <- c(4, 3, 2, 5, 6, 7, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19)
y <- c(3, 2, 4, 5, 6, 5, 5, 3, 4, 7, 10, 10, 8, 9, 7, 8)

plot(x, y)
lines(simple_loess(y, x), col="gray40")
points(simple_loess(y=y, x=x, xout=5L), col=2, cex=2)
points(simple_loess(y=y, x=x, xout=17), col=3, cex=2)
points(simple_loess(y=y, x=x, xout=seq(8, 12, 0.3)), col=3, pch=16)
lines(simple_loess(y=y, x=x, xout=50L), col=4, lty=2)

# data.frame input
dtf <- data.frame(x, y)
simple_loess(dtf)
```

---

simple\_table

*Read a simple table*

---

**Description**

Read tables given in more or less elaborate human-readable formats

**Usage**

```
simple_table(x, header = TRUE, rem.dup.header = header,
  na.strings = c("NA", "N/A"), stringsAsFactors = FALSE, ...)
```

**Arguments**

**x** a table represented as a character string  
**header** are the table columns named? By default TRUE  
**rem.dup.header** remove duplicated headers.  
**na.strings** a character vector of strings which are to be interpreted as NA values  
**stringsAsFactors** should character vectors be converted to factors? By default FALSE  
**...** further arguments passed to `read.table`

**Value**

A `data.frame` containing a representation of the data.

**Examples**

```
x1 <- "
+-----+-----+-----+-----+-----+
|   Date   | Emp1 | Case | Priority | PriorityCountinLast7days |
+-----+-----+-----+-----+-----+
| 2018-06-01 | A   | A1  |      0 |                0 |
| 2018-06-03 | A   | A2  |      0 |                1 |
| 2018-06-02 | B   | B2  |      0 |                2 |
| 2018-06-03 | B   | B3  |      0 |                3 |
+-----+-----+-----+-----+-----+
"
```

```
x2 <- "
      Date   | Emp1 | Case | Priority | PriorityCountinLast7days
2018-06-01 | A   | A|1  |      0 |                0
2018-06-03 | A   | A|2  |      0 |                1
2018-06-02 | B   | B|2  |      0 |                2
2018-06-03 | B   | B|3  |      0 |                3
"
```

```
x3 <- "
Maths | English | Science | History | Class
0.1   | 0.2     | 0.3     | 0.2     | Y2
0.9   | 0.5     | 0.7     | 0.4     | Y1
0.2   | 0.4     | 0.6     | 0.2     | Y2
0.9   | 0.5     | 0.2     | 0.7     | Y1
"
```

```
x4 <- "
      Season | Team | W | AHWO
-----
1 | 2017/2018 | TeamA | 2 | 1.75
2 | 2017/2018 | TeamB | 1 | 1.85
"
```

```
3 | 2017/2018 | TeamC | 1 | 1.70
4 | 2017/2018 | TeamD | 0 | 3.10
5 | 2016/2017 | TeamA | 1 | 1.49
6 | 2016/2017 | TeamB | 3 | 1.51
7 | 2016/2017 | TeamC | 2 | 1.90
8 | 2016/2017 | TeamD | 0 | N/A
"

x5 <- "
  A   T   G   C
-----
A | 6 | 0 | 4 | 0 |
|---:---:---:---
T | 0 | 6 | 0 | 4 |
|---:---:---:---
G | 4 | 0 | 6 | 0 |
|---:---:---:---
C | 0 | 4 | 0 | 6 |
-----
"

x6 <- "
-----
|date           |Material          |Description      |
|-----|-----|-----|
|10/04/2013     |WM.5597394        |PNEUMATIC        |
|11/07/2013     |GB.D040790        |RING              |
|-----|-----|-----|

-----
|date           |Material          |Description      |
|-----|-----|-----|
|08/06/2013     |WM.4M01004A05     |TOUCHEUR         |
|08/06/2013     |WM.4M010108-1     |LEVER             |
|-----|-----|-----|
"
```

```
lapply(c(x1, x2, x3, x4, x5, x6), simple_table)
```

---

smat	<i>Similarity matrix</i>
------	--------------------------

---

**Description**

Create a similarity matrix

**Usage**

```
smat(x, y, s, byrow = FALSE)
```

**Arguments**

x                      an object containing the values the similarity matrix should be computed for

y	same as x. If given the union of values in x and y are used, if not the unique values of x are used
s	a vector for filling the matrix. By default producing an identity matrix
byrow	should s fill the matrix by row?

**Value**

A square matrix with the values of s and row-/colnames of the unique values in {x, y}.

**See Also**

[similarity](#)

**Examples**

```
smat(1:3)

smat(c("f", "e", "d"), s=c(
4, 1, 1,
1, 3, 2,
1, 2, 3
))
```

---

speedskate

*2018 MarbleLympics speed skating times*

---

**Description**

Intermediate and total times for all 16 runs, arranged by lane and heat number.

**Usage**

```
speedskate
```

**Format**

A list containing two data.frames, one for each lane. Columns are heat and rows are time checks in seconds.

**Source**

[https://www.youtube.com/watch?v=fA-O6f\\_jArk](https://www.youtube.com/watch?v=fA-O6f_jArk)

**Examples**

```
tt <- t(do.call(cbind, speedskate))
pairs(tt)
cor(tt)
outer(
  colnames(tt),
  colnames(tt),
  Vectorize(function(i,j) cor.test(tt[,i],tt[,j])$p.value)
)
```

---

spread_seq	<i>Spread sequence</i>
------------	------------------------

---

## Description

Spread out the values of a numeric vector

## Usage

```
spread_seq(x, f = 1.1, ..., node = c("midrange", "first", "last",
  "mean", "median", "min", "max"))
```

## Arguments

x	a numeric vector
f	numeric or item matching a function. If numeric, a multiplicative factor applied to the distance between points, otherwise a function to be applied to differences
...	further arguments passed to f if matching a function
node	the location of x that will remain unchanged

## Examples

```
x <- c(-1, 0, 2, NA, 4, 5, 6, 8, 9)

spread_seq(c(-1, 1))
spread_seq(x, 1.5, "midrange")
spread_seq(x, 1.5, "first")
spread_seq(x, 1.5, "last")
spread_seq(x, 1.5, "mean")
spread_seq(x, 1.5, "median")

spread_seq(c(3, 4, 1, 9, 6))
spread_seq(c(3, 4, 1, 9, 6), 2, "min")
spread_seq(c(3, 4, 1, 9, 6), 2, "max")

spread_seq(c(3, 4, 1, 9, 6), "/", 2)
spread_seq(c(3, 4, 1, 9, 6), 0.5)

y <- sort(c(3, 4, 1, 9, 6))
plot(y)
lines(spread_seq(y, "log1p"))

f <- function(x) sqrt(abs(x)*2)*sign(x)
y <- c(3, 4, 1, 9, 6)
plot(y)
lines(spread_seq(y, f=f))
```

---

summary.stl	<i>Summarizing seasonal decomposition</i>
-------------	---

---

### Description

Summary method for class "stl".

### Usage

```
## S3 method for class 'stl'
summary(object, digits = getOption("digits"), ...)
```

### Arguments

object	an object of class "stl"
digits	the number of significant digits to use when printing
...	further arguments passed to or from other methods

### Details

This function is a slight modification to `stats::summary.stl`, the main change being the addition of the variance statistic, which can be considered a parametric (normal) compliment to the existing IQR statistic.

### Examples

```
set.seed(1)
x <- ts(rnorm(1e4, sd=1), frequency=12)
a <- stl(x, s.window="periodic")
stats::summary.stl(a)
summary(a)

b <- stl(co2, s.window="periodic")
summary(b)
```

---

tied_triple_test	<i>Tied triple test</i>
------------------	-------------------------

---

### Description

Compare numeric values, returning an inbetween value for ties

**Usage**

```
x %ttt% y

ttt(x, y)

is.ttt(x)

## S3 method for class 'ttt'
print(x, symbols = TRUE, ...)

## S3 method for class 'ttt'
table(...)
```

**Arguments**

<code>x, y</code>	numeric values to be compared
<code>symbols</code>	should symbols be used instead of numeric values?
<code>...</code>	further arguments passed to methods

**See Also**

[Comparison](#), [comparison\\_with\\_ties](#)

**Examples**

```
1:5 %ttt% 3

ttt(1:3, 2)
print(ttt(1:3, 2), FALSE)

c(1, 6, 3, 0) %ttt% c(1, 3, 3, 2)

# Equivalent
as.integer(c(1, 6, 3, 0) %ttt% c(1, 3, 3, 2))
sign(c(1, 6, 3, 0) - c(1, 3, 3, 2))

# Demonstrating table method
dtf <- data.frame(x=1:5, y=3)
dtf$`?'` <- ttt(dtf$x, dtf$y)
dtf

x <- c(8, 4, 6, 8, 9, 6, 5, 7, 0, 3, 2, 1, 5, 6, 4, 7, 6,
      3, 1, 9, 5, 6, 7, 7, 4, 5, 8, 6, 2, 5, 9, 5, 4, 8)
y <- c(1, 3, 2, 4, 6, 0, 5, 3, 7, 5, 7, 4, 5, 6, 0, 1, 4,
      2, 4, 3, 1, 5, 3, 9, 2, 2, 4, 7, 5, 6, 8)

ou <- outer(sort(x), sort(y), "%ttt%")
ta <- table(ou)

pa <- capture.output(ta)

par(mar=c(1, 2, 3, 2))
image(ou, col=topo.colors(length(ta)), axes=FALSE)
title(pa)
```



```
box()
```

---

var_th	<i>Theoretical variance</i>
--------	-----------------------------

---

## Description

Use Calculate the theoretical variance of base probability distributions

## Usage

```
var_th(p, distribution = c("uniform", "exponential", "gamma", "t",
  "students-t", "bates", "binomial", "nbinom", "negative binomial", "beta",
  "f", "geometric", "hypergeometric", "lognormal", "log-normal", "weibull",
  "signed-rank", "rank-sum", "logistic"))
```

## Arguments

p	a named vector of parameter values, or a single unnamed numeric if only one parameter. Use a data.frame with appropriately named columns to calculate several variances of the same distribution.
distribution	the name of the distribution to calculate the variance of

## Details

The parameters and their names are the same as used in their respective density function. In some cases, like gamma, (negative) binomial etc. more than one convention is followed.

## See Also

[Distributions](#)

## Examples

```
var_th(p=data.frame(min=1:2, max=5:6), dist="unif")
var(runif(1e5, 1, 5))

var_th(p=2:3, dist="exp")
var(rexp(1e5, 2))

var_th(p=data.frame(shape=3:2, scale=c(0.8, 1)), dist="gamma")
var(rgamma(1e5, shape=3, scale=0.8))

var_th(p=c(shape=3, rate=1.25), dist="gamma")
var(rgamma(1e5, shape=3, rate=1.25))

var_th(p=18:20, dist="t")
var(rt(1e5, 18))

var_th(p=c(a=1, b=2, n=3), dist="bates")
var(rbates(1e5, a=1, b=2, nr=3))
```

```

var_th(p=c(size=10, prob=0.8), dist="binom")
var(rbinom(1e5, 10, 0.8))

var_th(p=c(size=10, prob=0.8), dist="nbinom")
var(rnbinom(1e5, size=10, prob=0.8))

var_th(p=c(size=10, mu=2), dist="nbinom")
var(rnbinom(1e5, size=10, mu=2))

var_th(p=data.frame(shape1=c(1, 2), shape2=c(1.5, 1)), dist="beta")
var(rbeta(1e5, shape1=1, shape2=1.5))
var(rbeta(1e5, shape1=2, shape2=1))

var_th(p=c(df1=6, df2=11), dist="f")
var(rf(1e5, 6, 11))

var_th(p=c(m=3, n=3, k=2), dist="hypergeom")
var(rhyper(1e5, m=3, n=3, k=2))

var_th(p=c(meanlog=0, sdlog=1), dist="log-normal")
var(rlnorm(1e5, meanlog=0, sdlog=1))

var_th(p=c(shape=2, scale=1), dist="weibull")
var(rweibull(1e5, shape=2, scale=1))

var_th(p=20, dist="signed-rank")
var(rsignrank(1e5, n=20))

var_th(p=c(m=13, n=10), dist="rank-sum")
var(rwilcox(1e5, m=13, n=10))

```

---

weekday	<i>Week-day names</i>
---------	-----------------------

---

## Description

Convert numeric, character, factor and date-time vectors to week-day names

## Usage

```

weekday(x, ...)

## Default S3 method:
weekday(x, short = TRUE, language = c("english",
  "nn norwegian", "bm norwegian"), ...)

## S3 method for class 'Date'
weekday(x, ...)

## S3 method for class 'POSIXt'
weekday(x, ...)

```

### Arguments

x	a vector
...	further arguments passed to methods
short	if TRUE the names will be returned in shortened form
language	what language the names should be returned in

### Details

This function follows the ISO 8601 standard, meaning that Monday is considered the first day of the week.

### Examples

```
weekday(c("c", "b", "a"))
weekday(c("3", "2", "1"))
weekday(3:1)

weekday(Sys.Date())
weekday(Sys.Date(), short=FALSE, lang="nn nor")
```

---

ymse

*ymse: A collection of more or less useful functions*

---

### Description

There is no grand "theme" to ymse, other than that none of the functions, and in some cases function groups and classes, seemed to fit too well in any other package or merit their own package entirely.

### ymse functions

[addrows](#) Add rown to a data.frame [ahist](#) Create an average shifted histogram

# Index

## \*Topic **datasets**

- bartlett, [14](#)
- math\_constants, [47](#)
- math\_constants\_char, [48](#)
- speedskate, [69](#)
- %tgt%(comparison\_with\_ties), [21](#)
- %tlt%(comparison\_with\_ties), [21](#)
- %ttt%(tied\_triple\_test), [71](#)
  
- acf\_max, [3](#)
- addrows, [4](#), [75](#)
- adjustcolor, [5](#)
- adjustcolorHSV, [5](#), [59](#)
- ahist, [6](#), [51](#), [56](#), [75](#)
- align\_char, [7](#), [8](#)
- align\_num, [7](#), [8](#)
- arfilter, [9](#)
- arfit, [9](#)
- arimpulse, [10](#), [11](#)
- armodel, [9](#), [10](#), [11](#)
- as.array.list, [12](#)
- as.dice(dice), [23](#)
- as.table.dice(dice), [23](#)
- aug\_median, [13](#)
  
- bartlett, [14](#)
- basename, [37](#)
- binclosest(binsearch), [15](#)
- binsearch, [15](#)
- bix, [16](#)
- bix<-(bix), [16](#)
  
- caleidoscope, [17](#)
- cbapply, [17](#)
- ccf\_max(acf\_max), [3](#)
- central.tendency, [18](#), [49](#)
- cmode(central.tendency), [18](#)
- combodice, [20](#), [29](#)
- Comparison, [22](#), [72](#)
- comparison\_with\_ties, [21](#), [72](#)
- cubi(means), [48](#)
  
- default\_par, [22](#), [61](#), [63](#)
- density, [51](#)
  
- deparse, [24](#)
- dice, [23](#), [34](#)
- Distributions, [73](#)
- dmode(central.tendency), [18](#)
- dput, [24](#), [35](#)
- dput2, [24](#), [35](#)
- drop\_pattern, [25](#), [26](#)
- drop\_randfx, [25](#), [26](#)
- dtf\_clean, [26](#)
- dusd, [20](#), [28](#)
- dusd1(dusd), [28](#)
- dusd2(dusd), [28](#)
  
- e(math\_constants), [47](#)
- e.char(math\_constants\_char), [48](#)
- Elo\_rating, [30](#)
- elo\_upd(Elo\_rating), [30](#)
- elo\_upd\_pw(Elo\_rating), [30](#)
- entropy, [31](#)
- eu.ma(math\_constants), [47](#)
- eu.ma.char(math\_constants\_char), [48](#)
- every\_nth, [32](#)
- expand, [23](#), [33](#)
- explode, [24](#), [34](#)
  
- factanal, [55](#)
- factorise, [35](#), [36](#)
- factors, [35](#), [36](#)
- feig1(math\_constants), [47](#)
- feig1.char(math\_constants\_char), [48](#)
- feig2(math\_constants), [47](#)
- feig2.char(math\_constants\_char), [48](#)
- file\_ext, [37](#)
- file\_name(file\_ext), [37](#)
- file\_name\_ext(file\_ext), [37](#)
- fingerknit, [38](#)
- fitrange, [38](#), [53](#)
- flatten, [39](#)
  
- gcd, [40](#)
- geom(means), [48](#)
- glai.kin(math\_constants), [47](#)
- glai.kin.char(math\_constants\_char), [48](#)
- grepl, [25](#)

harm (means), 48  
hsv, 59  
  
in\_range, 43  
incdiff, 40  
indexvalue, 41, 46  
intsect, 42  
is.dice (dice), 23  
is.ttt (tied\_triple\_test), 71  
is\_coprime, 43  
is\_prime, 44, 57  
  
keep\_finite, 44  
khin (math\_constants), 47  
khin.char (math\_constants\_char), 48  
  
lag\_vector, 45  
latin\_sq, 41, 46  
lehm (means), 48  
loess, 66  
  
markov\_seq, 47  
math\_constants, 47  
math\_constants\_char, 48  
means, 19, 48  
merge\_multiple, 49  
midrange (central.tendency), 18  
multidensity, 50  
  
narm, 52  
norma, 38, 53  
  
pacf\_max (acf\_max), 3  
pairwise, 54  
par, 63  
pcamean, 55  
phi (math\_constants), 47  
phi.char (math\_constants\_char), 48  
pi (math\_constants), 47  
pi.char (math\_constants\_char), 48  
plot.histogram, 56, 56  
plot.stl, 56, 56  
powr (means), 48  
prcomp, 55  
primes, 44, 56  
princomp, 55  
print.dice (dice), 23  
print.ttt (tied\_triple\_test), 71  
pseudomedian (central.tendency), 18  
  
quad (means), 48  
quartz.png, 57  
quick\_table, 58  
  
rainbowHCL, 58  
randcolours, 59  
resolve\_dup, 60  
revert\_par, 23, 61, 63  
rle2, 61  
  
seq\_range, 62  
set\_mar, 23, 61, 63  
similarity, 54, 64, 69  
simple\_loess, 65  
simple\_table, 66  
smat, 64, 68  
speedskate, 69  
spread\_seq, 70  
srmean (central.tendency), 18  
summary.stl, 71  
  
table, 23, 34  
table.ttt (tied\_triple\_test), 71  
tgt (comparison\_with\_ties), 21  
tied\_triple\_test, 22, 71  
tlt (comparison\_with\_ties), 21  
ttt (tied\_triple\_test), 71  
  
var\_th, 73  
  
weekday, 74  
  
ymse, 75  
ymse-package (ymse), 75