

# Package ‘ymse’

May 27, 2019

**Title** Ymse (Various)

**Version** 0.6.1

**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 sample loess smooth, and other assorted tasks.

**Depends** R (>= 3.5.0)

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

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1

## R topics documented:

acf_max . . . . .	2
addrows . . . . .	4
adjustcolorHSV . . . . .	5
ahist . . . . .	6
arfilter . . . . .	7
arfit . . . . .	7
arimpulse . . . . .	8
armodel . . . . .	9
as.array.list . . . . .	10
bartlett . . . . .	11
binsearch . . . . .	12
caleidoscope . . . . .	13
central.tendency . . . . .	14
combodice . . . . .	14
compare_forecasts . . . . .	16
comparison_with_ties . . . . .	17
default_par . . . . .	18
dput2 . . . . .	18
drop_pattern . . . . .	19

drop_randfx . . . . .	20
dtf_clean . . . . .	21
dusd . . . . .	23
entropy . . . . .	25
every_nth . . . . .	26
explode_obj . . . . .	27
factorise . . . . .	28
factors . . . . .	29
fitrange . . . . .	29
flatten . . . . .	30
forecasts . . . . .	31
gcd . . . . .	31
incdiff . . . . .	32
indexvalue . . . . .	33
intersect . . . . .	34
is_coprime . . . . .	34
is_prime . . . . .	35
latin_sq . . . . .	35
math_constants . . . . .	36
math_constants_char . . . . .	37
means . . . . .	37
narm . . . . .	38
norma . . . . .	39
pairwise . . . . .	40
pcamean . . . . .	41
plot.histogram . . . . .	42
plot.stl . . . . .	42
primes . . . . .	42
randcolours . . . . .	43
revert_par . . . . .	44
set_mar . . . . .	44
similarity . . . . .	45
simple_loess . . . . .	46
simple_table . . . . .	48
smat . . . . .	49
speedskate . . . . .	50
tied_triple_test . . . . .	51
ymse . . . . .	52

**Index****53**

acf\_max

*Maximum ACF, PACF and CCF***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**

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

**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

<code>dtf</code>	data.frame; original data.frame
<code>nrw</code>	data.frame; the new row(s) to be added
<code>top</code>	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**

col	vector of colors, in any format that col2rgb() accepts
alpha.f, h.f, s.f, v.f	factors scaling the opacity, hue, saturation and value of col
offset	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
transform	a 4x4 diagonal matrix specifying the scaling applied to the <i>hue</i> , <i>saturation</i> , <i>value</i> and <i>alpha</i> values
h, s, v, alpha	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 containng 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**

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

**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)
```

---

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	<i>Create an AR model object</i>
---------	----------------------------------

---

## 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)
```

---

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)
```

---

caleidoscope

*Caleidoscopic effect on a matrix*


---

**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))
```

---

central.tendency	<i>Central tendency measures</i>
------------------	----------------------------------

---

**Description**

Central tendency measures

**Usage**

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

cmode(x, ...)
```

**Arguments**

x	numeric vector
na.rm	remove NAs before starting calculations
...	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)
```

---

combodice	<i>Combine dice</i>
-----------	---------------------

---

**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, "*"))
```

```
# 3d6, and d6+d10+d20. Discard lowest
discard_lowest <- function(x) sum(sort(x)[-1])
combodice(list(1:6, 1:6, 1:6), discard_lowest, method="ex")
combodice(list(1:6, 1:10, 1:20), discard_lowest, method="ex")

# 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

# Equivalent using binomial PDF
dbinom(0:3, 3, 0.4)*100

# 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)
```

---

compare_forecasts	<i>Compare forecast accuracies</i>
-------------------	------------------------------------

---

## Description

Test the efficacy of time series models by comparing forecasts with actual data

## Usage

```
compare_forecasts(m, y = NULL, holdout = NULL)
```

## Arguments

<code>m</code>	a list of models to compare
<code>y</code>	a monovariate time series; the data to train and test the models on
<code>holdout</code>	single integer; the last n points will be forecasted

## Examples

```
library(forecast)
set.seed(1)
extr <- aggregate(sunspot.month, nfrequency=2, mean)[100:349]
extr <- ts(extr, f=21)

mod1 <- StructTS(extr)
```



```

mod2 <- ar(extr)
mod3 <- nnetar(extr)
mod4 <- arfima(extr)
mod5 <- Arima(extr, order=c(3, 0, 1))
mod6 <- Arima(extr, order=c(2, 0, 2), seasonal=c(2, 1, 0))

mod.l <- list(mod1, mod2, mod3, mod4, mod5, mod6)

l <- compare_forecasts(mod.l, extr, 21)

diffs <- sapply(l, function(y) y[["fcast"]] - y[["test"]])
matplot(diffs, type="l",
        col=c("red", "lightgreen", "blue", "orange", "pink", "cyan"), lty=1)

par(mfrow=c(3, 2), mar=c(3, 3, 2, 1), mgp=c(2, 0.6, 0), oma=c(0, 0, 0, 0))
invisible(lapply(l, function(x) {
  plot(x$fcast.obj, shaded=FALSE, PI=FALSE, include=66, type="l",
       cex.main=0.9, xpd=NA)
  lines(x$test, col="#00FF4488")
}))
summary(l)
head(forecasts(l))
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](#)

---

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](#)

**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 (hhhhhhh\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

*Drop predictors*


---

**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 containing 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 containg 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**

```
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)
```

---

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)))

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

## End(Not run)
# 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)
```

---

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	<i>Select every n'th element</i>
-----------	----------------------------------

---

**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)
```

---

explode_obj	<i>Explode object</i>
-------------	-----------------------

---

### Description

Presents an R object in an exploded form

### Usage

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

### Arguments

x                an R object, or a character string describing an R object  
indent          how many spaces for indentation (and exdentation) 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 exdentation after each open parenthesis, newline and indentation 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_obj(xc)

x1 <- list(0=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_obj(x1)

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

factorise

*Factorise***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**

```
x <- 2 * 2 * 2 * 3 * 3 * 5
factorise(x)

prod(factorise(5641324))

factorise(nextn(60000000, c(2, 3)))
factorise(72*999983)
```

---

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)
```

---

fitrange	<i>Fit to a range</i>
----------	-----------------------

---

**Description**

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

**Usage**

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

**Arguments**

x	a numeric vector
lower	the lower bound of the new vector
upper	the upper bound of the new vector

See Also

[norma](#)

Examples

```
range(fitrangle(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(
  0=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)
```

---

forecasts	<i>Return forecasts</i>
-----------	-------------------------

---

### Description

Return forecasts and actual data from compare\_forecasts object

### Usage

```
forecasts(x)
```

### Arguments

x                      a compare\_forecasts object

### Value

A multivariate time series (mts) with the actual data, the holdout, on the first column, and the forecasts on the rest.

---

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 permutate 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	<i>Intersect</i>
---------	------------------

---

**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)])
fcl <- split(fc, sample(1:3, 20, rep=TRUE))

intsect(fcl)
```

---

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

---

**Description**

Test whether to 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)
```

---

**is\_prime***Primality check*

---

**Description**

Test integers for whether they are prime or not

**Usage**

```
is_prime(x)
```

**Arguments**

x                      vector of integers

**See Also**

[primes](#)

---

**latin\_sq***Latin square*

---

**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 increses 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
```

---

math\_constants

*Mathematical constants*


---

**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      *High precision mathematical constants*


---

**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      *Generalized means*


---

**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**

<code>x</code>	numeric vector of values whose *mean is to be computed
<code>na.rm</code>	logical; should NA values be removed? (default TRUE)
<code>zero.rule</code>	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.
<code>p</code>	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.

**Examples**

```
fun1 <- 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(fun1), function(f) f(x)))
rownames(m) <- as.character(fun1)[-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")
```

---

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**

<code>x</code>	a vector or matrix
<code>...</code>	further arguments passed to methods
<code>margin</code>	if <code>x</code> is matrix, which margin to remove NAs by
<code>keep</code>	if <code>x</code> 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

*PCA mean***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))

```

---

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

---

### Description

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](#)

---

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

---

### Description

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](#)

---

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

---

### Description

Prime generator based on the sieve of Eratosthenes

### Usage

`primes(n)`

## Arguments

n 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](#)

---

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 l, 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)
```

---

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](#)

---

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)
```

### 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, no title
right	margin width for the right edge, default 1

### 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

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	<i>Similarity measure</i>
------------	---------------------------

---

**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)

```

## 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 space 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	<i>Read a simple table</i>
--------------	----------------------------

---

## 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 teble 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)

```

**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, speeds skate))
pairs(tt)
cor(tt)
outer(
  colnames(tt),
  colnames(tt),
  Vectorize(function(i,j) cor.test(tt[,i],tt[,j])$p.value)
)
```

---

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**

x, y	numeric values to be compared
symbols	should symbols be used instead of numeric values?
...	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()

```

## 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, 11
- math\_constants, 36
- math\_constants\_char, 37
- speedskate, 50
- %tgt%(comparison\_with\_ties), 17
- %tlt%(comparison\_with\_ties), 17
- %ttt%(tied\_triple\_test), 51
- acf\_max, 2
- addrows, 4, 52
- adjustcolor, 5
- adjustcolorHSV, 5
- ahist, 6, 42, 52
- arfilter, 7
- arfit, 7
- arimpulse, 8, 9
- armodel, 7, 8, 9
- as.array.list, 10
- bartlett, 11
- binclosest (binsearch), 12
- binsearch, 12
- caleidoscope, 13
- ccf\_max (acf\_max), 2
- central.tendency, 14
- cmode (central.tendency), 14
- combodice, 14, 23
- compare\_forecasts, 16
- Comparison, 17, 51
- comparison\_with\_ties, 17, 51
- cubi (means), 37
- default\_par, 18, 44
- deparse, 19
- dput, 19, 27
- dput2, 18, 27
- drop\_pattern, 19, 21
- drop\_randfx, 20, 20
- dtf\_clean, 21
- dusd, 15, 23
- dusd1 (dusd), 23
- dusd2 (dusd), 23

- e (math\_constants), 36
- e.char (math\_constants\_char), 37
- entropy, 25
- eu.ma (math\_constants), 36
- eu.ma.char (math\_constants\_char), 37
- every\_nth, 26
- explode\_obj, 27
- factanal, 41
- factorise, 28, 29
- factors, 28, 29
- feig1 (math\_constants), 36
- feig1.char (math\_constants\_char), 37
- feig2 (math\_constants), 36
- feig2.char (math\_constants\_char), 37
- fitrange, 29, 40
- flatten, 30
- forecasts, 31
- gcd, 31
- geom (means), 37
- glai.kin (math\_constants), 36
- glai.kin.char (math\_constants\_char), 37
- grepl, 20
- harm (means), 37
- incdiff, 32
- indexvalue, 33, 36
- intsect, 34
- is.ttt (tied\_triple\_test), 51
- is\_coprime, 34
- is\_prime, 35, 43
- khin (math\_constants), 36
- khin.char (math\_constants\_char), 37
- latin\_sq, 33, 35
- lehm (means), 37
- loess, 47
- math\_constants, 36
- math\_constants\_char, 37
- means, 14, 37

narm, [38](#)  
norma, [30](#), [39](#)  
  
pacf\_max (acf\_max), [2](#)  
pairwise, [40](#)  
pcamean, [41](#)  
phi (math\_constants), [36](#)  
phi.char (math\_constants\_char), [37](#)  
pi (math\_constants), [36](#)  
pi.char (math\_constants\_char), [37](#)  
plot.histogram, [42](#), [42](#)  
plot.stl, [42](#), [42](#)  
powr (means), [37](#)  
prcomp, [41](#)  
primes, [35](#), [42](#)  
princomp, [41](#)  
print.ttt (tied\_triple\_test), [51](#)  
pseudomedian (central.tendency), [14](#)  
  
quad (means), [37](#)  
  
randcolours, [43](#)  
revert\_par, [18](#), [44](#), [44](#)  
  
set\_mar, [18](#), [44](#), [44](#)  
similarity, [40](#), [45](#), [50](#)  
simple\_loess, [46](#)  
simple\_table, [48](#)  
smat, [45](#), [49](#)  
speedskate, [50](#)  
  
table.ttt (tied\_triple\_test), [51](#)  
tgt (comparison\_with\_ties), [17](#)  
tied\_triple\_test, [17](#), [51](#)  
tlt (comparison\_with\_ties), [17](#)  
ttt (tied\_triple\_test), [51](#)  
  
ymse, [52](#)  
ymse-package (ymse), [52](#)