

# Package ‘ymse’

December 6, 2018

**Title** Various more or less useful functions and methods

**Version** 0.1.2

**Description** What the package does (one paragraph).

**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:

addrows . . . . .	2
ahist . . . . .	3
arfilter . . . . .	4
arfit . . . . .	4
arimpulse . . . . .	5
armodel . . . . .	6
as.array.list . . . . .	7
bartlett . . . . .	8
binsearch . . . . .	9
caleidoscope . . . . .	10
central.tendency . . . . .	11
compare_forecasts . . . . .	11
comparison_with_ties . . . . .	12
default_par . . . . .	13
dput2 . . . . .	13
dtf.clean . . . . .	14
dusd . . . . .	16
entropy . . . . .	18
fitrange . . . . .	19
forecasts . . . . .	19
incdiff . . . . .	20
indexvalue . . . . .	21
isPrime . . . . .	22
latin_sq . . . . .	22
mathemathical_constants . . . . .	23

mathematical_constants_char . . . . .	23
means . . . . .	24
norma . . . . .	25
pcamean . . . . .	26
plot.histogram . . . . .	27
plot.stl . . . . .	27
primes . . . . .	28
randcolours . . . . .	28
reset_par . . . . .	29
set_mar . . . . .	29
speedskate . . . . .	30
tied_triple_test . . . . .	30
ymse . . . . .	32

<b>Index</b>	<b>33</b>
--------------	-----------

---

addrows	<i>Add rows to a data.frame</i>
---------	---------------------------------

---

## 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))
```

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)
```

---

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	<i>Coerce a list to an array</i>
---------------	----------------------------------

---

## 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)

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)

cmode(x, ...)

```

**Arguments**

x	numeric vector
...	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)

```

---

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**

```

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))
l

```

---

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)
```

---

default_par	<i>Default par</i>
-------------	--------------------

---

**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: [reset\\_par](#), [set\\_mar](#)

---

dput2	<i>Write an Object to console</i>
-------	-----------------------------------

---

**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)
```

**Arguments**

x	an object
width	integer; column width
assign	character; should assignment be included?
breakAtParen	logical; should lines break at parenthesis begins (default FALSE)

**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, 65)
dput2(xmpl, 65, assign="end")
dput2(xmpl, 80, assign="none")
dput2(xmpl[1:10,], 10, "none")

# 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, breakAtParen=TRUE)
dput2(xmpl)
```

---

dtf.clean

*Data cleanup*


---

**Description**

Create a data.frame from a messy table

**Usage**

```
dtf.clean(x, header = TRUE, ...)
```

**Arguments**

**x**                      A messy table the form of a character string  
**header**                Does the table include headers? (default TRUE)  
**...**                   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
"

lapply(c(x1, x2, x3, x4), dtf.clean)

```

---

dusd	<i>Discrete (Uniform) Sum Distributions</i>
------	---

---

**Description**

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

**Usage**

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

```
dusd2(xi = rep(1, 6), n = 2, round, zero.index = FALSE, limit = 1e-13)
```

**Arguments**

<code>xr</code>	numeric vector; a vector of equiprobable values
<code>n</code>	integer; the number of distributions to be summed
<code>xi</code>	numeric vector; a vector of probabilities, with indices representing values
<code>round</code>	integer; number of digits to round to after each convolution
<code>zero.index</code>	logical; should the index of <code>xi</code> start at zero?
<code>limit</code>	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.

**Examples**

```
# five coin flips
plot(table(dusd1(0:1, 5)))
plot(dusd2(c(1, 1), 5, zero.index=TRUE))
plot(dbinom(0:5, 5, 0.5), type="h", lwd=2)

# 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*3, 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	<i>Information entropy</i>
---------	----------------------------

---

## 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)
```

---

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(fitrange(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)
```

---

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.

---

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)
```

---

isPrime	<i>Primality check</i>
---------	------------------------

---

**Description**

Test an integer for whether it is prime or not

**Usage**

```
isPrime(x)
```

**Arguments**

x	integer; one or more prime candidates
---	---------------------------------------

**See Also**

[primes](#)

---

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 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
```

---

mathematical\_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

---

mathematical\_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	<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 lop1p), 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.



**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")

```

---

norma	<i>Normalize</i>
-------	------------------

---

**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))
```

---

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))
```

---

plot.histogram	<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.

### Usage

```
## S3 method for class 'histogram'
plot(x, freq = equidist, density = NULL, angle = 45,
     col = NULL, border = par("fg"), lty = NULL, lwd = 1,
     main = paste("Histogram of", paste(x$xname, collapse = "\n")),
     sub = NULL, xlab = x$xname, ylab, xlim = range(x$breaks), ylim = NULL,
     axes = TRUE, labels = FALSE, add = FALSE, ann = TRUE, ...)
```

### See Also

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

---

plot.stl	<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.

### Usage

```
## S3 method for class 'stl'
plot(x, labels = colnames(X), set.pars = list(mar = c(0, 6, 0,
  6), oma = c(6, 0, 4, 0), tck = -0.01, mfrow = c(nplot, 1)), main = NULL,
     range.bars = TRUE, ..., col.range = "light gray")
```

### 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**

[isPrime](#)

---

randcolours	<i>Random colours</i>
-------------	-----------------------

---

**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)
```

---

reset_par	<i>Reset par</i>
-----------	------------------

---

## Description

Reverts `par` settings back to `.old.par`

## Usage

```
reset_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 = 2, y = 2, main = 1, right = 1)
```

## Arguments

<code>x</code>	margin width for the x axis, default 2
<code>y</code>	margin width for the x axis, default 2
<code>main</code>	margin width for the main title, default 1, no title
<code>right</code>	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](#), [reset\\_par](#)

---

speedskate	<i>2018 MarbleLympics speed skating times</i>
------------	---

---

**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)
)
```

---

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, symbol = TRUE, ...)

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

**Arguments**

x, y                    numeric values to be compared

**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()
```

---

ymse

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

---

### **Description**

There is go grand "theme" to ymse, other than that none of the functions, and in some cases function groups, seemed to fit too well in any other package

### **ymse functions**

[addrows](#)



# Index

## \*Topic **datasets**

bartlett, 8  
mathematical\_constants, 23  
mathematical\_constants\_char, 23  
speedskate, 30  
%tgt%(comparison\_with\_ties), 12  
%tl%(comparison\_with\_ties), 12  
%ttt%(tied\_triple\_test), 30  
  
addrows, 2, 32  
ahist, 3  
arfilter, 4  
arfit, 4  
arimpulse, 5, 6  
armodel, 4, 5, 6  
as.array.list, 7  
  
bartlett, 8  
binclosest(binsearch), 9  
binsearch, 9  
  
caleidoscope, 10  
central.tendency, 11  
cmode(central.tendency), 11  
compare\_forecasts, 11  
Comparison, 13, 31  
comparison\_with\_ties, 12, 31  
cubi(means), 24  
  
default\_par, 13, 29, 30  
deparse, 14  
dput, 14  
dput2, 13  
dtf.clean, 14  
dusd, 16  
dusd1(dusd), 16  
dusd2(dusd), 16  
  
e(mathematical\_constants), 23  
e.char(mathematical\_constants\_char), 23  
entropy, 18  
eu.ma(mathematical\_constants), 23

eu.ma.char  
(mathematical\_constants\_char), 23  
  
factanal, 26  
feig1(mathematical\_constants), 23  
feig1.char  
(mathematical\_constants\_char), 23  
feig2(mathematical\_constants), 23  
feig2.char  
(mathematical\_constants\_char), 23  
fitrange, 19, 25  
forecasts, 19  
  
geom(means), 24  
glai.kin(mathematical\_constants), 23  
glai.kin.char  
(mathematical\_constants\_char), 23  
  
harm(means), 24  
  
incdiff, 20  
indexvalue, 21, 22  
is.ttt(tied\_triple\_test), 30  
isPrime, 22, 28  
  
khin(mathematical\_constants), 23  
khin.char  
(mathematical\_constants\_char), 23  
  
latin\_sq, 21, 22  
lehm(means), 24  
  
mathematical\_constants, 23  
mathematical\_constants\_char, 23  
means, 11, 24  
  
norma, 19, 25  
  
pcamean, 26  
phi(mathematical\_constants), 23

`phi.char`  
    `(mathematical_constants_char)`,  
    23  
`pi (mathematical_constants)`, 23  
`pi.char (mathematical_constants_char)`,  
    23  
`plot.histogram`, 27, 27  
`plot.stl`, 27, 27  
`powr (means)`, 24  
`prcomp`, 26  
`primes`, 22, 28  
`princomp`, 26  
`print.ttt (tied_triple_test)`, 30  
`pseudomedian (central.tendency)`, 11  
  
`quad (means)`, 24  
  
`randcolours`, 28  
`reset_par`, 13, 29, 30  
  
`set_mar`, 13, 29, 29  
`speedskate`, 30  
  
`table.ttt (tied_triple_test)`, 30  
`tgt (comparison_with_ties)`, 12  
`tied_triple_test`, 13, 30  
`tlt (comparison_with_ties)`, 12  
`ttt (tied_triple_test)`, 30  
  
`ymse`, 32  
`ymse-package (ymse)`, 32