Package ‘ymse’

November 30, 2018

Title Various more or less useful functions and methods

Version 0.0.0.9000

Description What the package does (one paragraph).

Depends R (>= 3.5.0), forecast

Imports forecast

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

R topics documented:

`addrows` *Add rows to a data.frame*

Description

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

Usage

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

Arguments

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

Details

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

Examples

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

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

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

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

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

arfilter — *AR filter*

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

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 — *AR model fit*

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; sgould the result be plotted?
...	further arguments to plot

See Also

armodel for examples

---

armodel *Create an AR model object*

Description

Specify the characteristics of an AR model

Usage

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

Arguments

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

See Also

arimpulse

Examples

```
# short decay
ar.mod <- armodel(c(0.5))
arimpulse(ar.mod, pulse=1)

# long decay
ar.mod <- armodel(c(0.8))
arimpulse(ar.mod, pulse=1)

# negative second coefficient reduce damping, signal returns to normal
more quickly
ar.mod <- armodel(c(0.8, -0.1))
arimpulse(ar.mod, pulse=1)

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

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

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

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

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

`as.array.list` *Coerce a list to an array*

Description

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

Usage

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

Arguments

`ll`	a list of equal sized data.frames or matrices

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) χ^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))
binsearch(18, seq_len(3e9))
binsearch(18, seq_len(3e9), H=2e9)
binsearch(2000, seq_len(3e7)*100 + 10.71)

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 righ and lower left corners, respectively.

Value

A matrix of either $2\times$ or 2×-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 *Central tendency measures*

Description

Central tendency measures

Usage

```
pseudomedian(x)

cmode(x, ...)
```

See Also

means

Examples

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

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)
```

---

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)
```

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 (hhhhhhh\u00A0hhhhhhhh)",
                    " rrrrrrrr ( bbbbbbb )",
                    "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 *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)

dusd2(xi = rep(1, 6), n = 2, round, zero.index = FALSE,
  limit = 0.0000000000001)
```

Arguments

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

Value

dusd1 returns an array of size length(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)

s2 <- vector(length=3779)
s2[c(1, 220, 3779)] <- 1
plot(dusd2(xi=s2, 12, round=8), lwd=1)

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

---

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(...)

## 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.
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 *Fit to a range*

Description

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

Usage

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

Arguments

W	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(9, -2, 0.1), 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)
```

forecast.test *Compare forecast accuracies*

Description

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

Usage

```
## S3 method for class 'test'
forecast(m, y = NULL, holdout = NULL)
```

Arguments

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

Examples

```
library(forecast)
extr <- aggregate(sunspot.month, nfrequency=2, mean)
extr <- ts(extr, f=21)

extr <- head(extr, 342)

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 <- forecast.test(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
```

hist.avg *Average shifted histogram*

Description

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

Usage

```
## S3 method for class 'avg'
hist(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")
hist.avg(x, type="hist", border=2, col=7, freq=FALSE, lwd=2)
hist.avg(x, type="poly", border=2, col=7, freq=FALSE, lwd=2)
hist.avg(x, type="line", col=2, freq=FALSE, lwd=2)
hist.avg(x, type="table", col=2, freq=FALSE, lwd=2)
hist.avg(x, plot=FALSE)
```

`incdiff` *Increase difference*

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)
```

index.value *Index–value representation of arrays*

Description

Represent an array as columns of dimensional indices and value

Usage

```
## S3 method for class 'value'
index(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 <- index.value(arr)

# can be used to permutate an array
index.value(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 <- index.value(arr2)
index.value(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 <- index.value(lsq)
iv

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

`isPrime` *Primality check*

Description

Test an integer for whether it is prime or not

Usage

```
isPrime(x)
```

Arguments

`n`	integer; one or more prime candidates

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?
`redude`	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

`index.symbol`

Examples

```
set.seed(1)
ls <- latin.sq(10, reduce=TRUE)
image(ls, col=randcolours(ncol(ls)))

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

---

mean.pca *PCA mean*

Description

Takes the average of several PCA objects

Usage

```
## S3 method for class 'pca'
mean(...)
```

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(mean.pca(xxl))
biplot(princomp(xxm))
```

`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

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 ≤ 0, the wise call would be to reconsider whether using a geometric mean really makes sense in this case.

Examples

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

xl <- 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(xl, 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(xl[[1]]); powr(xl[[1]], -1); lehm(xl[[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 *Normalize*

Description

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

Usage

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

Arguments

W	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))
```

---

plot.histogram *Plot* `histogram` *object*

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

stats::plot.histogram, ymse::plot.stl

---

plot.stl *Plot* stl *object*

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

stats::plot.stl, ymse::plot.histogram

---

primes *Prime number generator*

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 *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 exeed this range, at an icreased risk of clipping.

Examples

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

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

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` *Tied triple test*

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), family="PT Mono")
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