Package 'ymse'

September 27, 2018

Title What the Package Does (one line, title case)

Version 0.0.0.9000

Description What the package does (one paragraph).

Depends R (>= 3.5.0)

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; the object that shall be added to
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 first 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))
```

---

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

```
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))
```

---

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\u00A0hhhhhhhhh)",
                    " 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)
```

---

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 = 1e-13)
```

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

```
## 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)
```

`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`

`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 sort of calculation has any kind 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")
```

---

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

---

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)
)
```

Index