Package ‘cooltools’

July 24, 2025

Type Package

Title Practical Tools for Scientific Computations and Visualizations

Version 2.18

Description

Collection of routines for efficient scientific computations in physics and astrophysics. These routines include utility functions, numerical computation tools, as well as visualisation tools. They can be used, for example, for generating random numbers from spherical and custom distributions, information and entropy analysis, special Fourier transforms, two-point correlation estimation (e.g. as in Landy & Szalay (1993) <doi:10.1086/172900>), binning & gridding of point sets, 2D interpolation, Monte Carlo integration, vector arithmetic and coordinate transformations. Also included is a non-exhaustive list of important constants and cosmological conversion functions. The graphics routines can be used to produce and export publication-ready scientific plots and movies, e.g. as used in Obreschkow et al. (2020, MNRAS Vol 493, Issue 3, Pages 4551–4569). These routines include special color scales, projection functions, and bitmap handling routines.

Imports plotrix, celestial, data.table, pracma, utils, png, jpeg, MASS, raster, sp, cubature, bit64, float, randtoolbox, Rcpp, FNN, hdf5r

Suggests EBImage

LinkingTo Rcpp

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation yes

Author Danail Obreschkow [aut, cre] (ORCID: <https://orcid.org/0000-0002-1527-0762>)

Maintainer Danail Obreschkow <danail.obreschkow@gmail.com>

Repository CRAN

Date/Publication 2025-07-23 22:40:08 UTC

Contents

cooltools-package *Practical Tools for Scientific Computations and Visualizations*

Description

Collection of routines for efficient scientific computations in physics and astrophysics. These routines include utility functions, advanced computation tools, as well as visualisation tools. They can be used, for example, for generating random numbers from spherical and custom distributions, information and entropy analysis, special Fourier transforms, two-point correlation estimation (e.g. as in Landy & Szalay (1993) <doi:10.1086/172900>), binning & gridding of point sets, 2D interpolation, Monte Carlo integration, vector arithmetic and coordinate transformations. Also included are a non-exhaustive list of important constants and cosmological conversion functions. The graphics routines can be used to produce and export publication-ready scientific plots and movies, e.g. as used in Obreschkow et al. (2020) <doi:10.1093/mnras/staa445>. These routines include special color scales, projection functions, and bitmap handling routines.

Author(s)

Danail Obreschkow

.cooltools.env *Package environment*

Description

Environment used to store global variables in the package, e.g. used for subplot routine.

Usage

```
.cooltools.env
```

Format

An object of class `environment` of length 0.

Value

None

Author(s)

Danail Obreschkow

alp — *Associated Legendre Polynomials*

Description

Compute associated Legendre polynomials P_l^m(x), defined as the canonical solutions of the general Legendre equation. These polynomials are used, for instance, to compute the spherical harmonics.

Usage

```
alp(x, l = 0, m = 0)
```

Arguments

`x`	real argument between -1 and +1, can be a vector
`l`	degree of the polynomial (0,1,2,...); accurate to about 25
`m`	order of the polynomial (-l,-l+1,...,l); for negative values the standard convention is used: if m>0, then P(x,l,-m) = P(x,l,m) (-1)^m*factorial(l-m)/factorial(l+m).

Value

Returns a vector with the same number of elements as `x`

Author(s)

Danail Obreschkow

See Also

`sphericalharmonics`

approxfun2 — *Bilinear interpolation function of data on a regular grid*

Description

Generates a fast function f(x,y) that interpolates gridded data, based on the analogous subroutine `approxfun` in 1D.

Usage

```
approxfun2(x, y, z, outside = NA)
```

Arguments

x	n-vector of x-coordinates; must be strictly monotonically increasing, but not necessarily equally spaced
y	m-vector of y-coordinates; must be strictly monotonically increasing, but not necessarily equally spaced
z	n-by-m matrix containing the known function values at the (x,y)-coordinates
outside	value of the approximation function outside the grid (default is NA)

Value

Returns a fast and vectorized interpolation function f(x,y)

Author(s)

Danail Obreschkow

See Also

approxfun

Examples

```
x = seq(3)
y = seq(4)
z = array(c(x+1,x+2,x+3,x+4),c(3,4))
f = approxfun2(x,y,z)
print(f(1.7,2.4))
```

---

bindata *Bin two-dimensional data in one dimension*

Description

Divides a vector of values x into finite intervals; returns the counts and other statistics in each interval.

Usage

```
bindata(x, y = NULL, bins = 20, method = "regular", xlim = NULL)
```

Arguments

x	N-element vector of x-coordinates
y	optional N-element vector of values associated with the different points in x
bins	If method is 'regular' or 'equal', this is a scalar specifying the number of bins. If method is 'custom' this is a vector of (n+1) x-values delimiting the n bins.
method	Character string. Choose 'regular' for regularly space bins, 'equal' for bins containing an equal number of points (+-1), or 'custom' for bins with custom edges.
xlim	optional 2-element vector specifying the data range (data cropped if necessary). If not given, xlim is set to the full range of x.

Value

Returns a list of items

n	number of bins
xlim	considered range of x-coordinates, same as input argument xlim, if given
xleft	n-element vector containing the x-coordinates of the left bin edges
xmid	n-element vector containing the x-coordinates of the bin centres
xright	n-element vector containing the x-coordinates of the right bin edges
dx	n-element vector containing the widths of the bins
count	n-element vector containing the number of points in each bin
x	n-element vector containg the mean x-values in each bin
y	n-element vector containg the mean y-values in each bin
xmedian	n-element vector containg the median of the x-values in each bin
ymedian	n-element vector containg the median of the y-values in each bin
yerr	n-element vector giving the uncertainty on the mean
ysd	n-element vector giving the standard deviations of the y-values
y16	n-element vector giving the 15.86-percentile of the y-values
y84	n-element vector giving the 84.13-percentile of the y-values

Author(s)

Danail Obreschkow

See Also

griddata

Examples

```
# make and plot 100 random (x,y)-points
set.seed(1)
x = runif(200)
y = x+rnorm(200)
plot(x,y,pch=16,cex=0.5)

# bin the data into 10 bins of 20 points each
bin = bindata(x,y,10,'equal')
segments(bin$xleft,bin$y,bin$xright,bin$y,col='red')
segments(bin$x,bin$y16,bin$x,bin$y84,col='red')
segments(bin$x,bin$y-bin$yerr,bin$x,bin$y+bin$yerr,col='red',lwd=3)
points(bin$x,bin$y,pch=16,col='red')
```

---

car2pol *Cartesian to polar/cylindrical coordinate conversion*

Description

Convert 2D/3D Cartesian to polar/cylindrical coordinates

Usage

```
car2pol(x)
```

Arguments

x	2/3-vector or n-by-2/3 matrix representing the Cartesian components x,y,(z) of n two/three-dimensional vectors

Value

Returns a 2/3-vector or a n-by-2/3 matrix representing the polar/cylindrical coordinates r,phi,(z), where phi=0...2*pi is the azimuth measured positively from the x-axis.

Author(s)

Danail Obreschkow

See Also

pol2car

car2sph — *Cartesian to spherical coordinate conversion*

Description

Convert 3D Cartesian to spherical coordinates

Usage

```
car2sph(x)
```

Arguments

`x`	3-vector or n-by-3 matrix representing the Cartesian components (x,y,z) of n three-dimensional vectors

Value

Returns a 3-vector or a n-by-3 element matrix representing the spherical coordinates (r,theta,phi), where theta=0...pi is the polar angle measured from the north pole and phi=0...2*pi is the azimuth measured positively from the x-axis (ISO 80000-2:2019 physics convention).

Author(s)

Danail Obreschkow

See Also

`sph2car`

cmplx2col — *Convert complex numbers to color*

Description

Converts a complex number (or a vector/array thereof) into a color-string, such that brightness represents the complex amplitude and hue represents the complex phase.

Usage

```
cmplx2col(z, max = NULL, hue = 0, saturation = 1, gamma = 1)
```

Arguments

z	complex number or vector/array of complex numbers
max	value of the complex module that corresponds to full brightness; if not given, this is set equal to the maximum complex amplitude in z.
hue	hue value of positive reals
saturation	saturation value of colors
gamma	positive number to adjust the non-linearity of the color scale (1=linear)

Value

Returns a single color or vector/array of colors

Author(s)

Danail Obreschkow

---

colorbar *Vertical color bar*

Description

Adds a vertical color bar to a plot with a custom axis.

Usage

```
colorbar(
  xleft,
  ybottom,
  xright,
  ytop,
  col = gray.colors(256, 0, 1),
  clim = c(0, 1),
  show.border = TRUE,
  text = "",
  line = 2,
  show.axis = TRUE,
  side = "right",
  lwd = 1,
  nticks = 5,
  at = NULL,
  srt = 0,
  ticklength = 0.1,
  shift = 0,
  ...
)
```

Arguments

`xleft`	left x-coordinate of the bar
`ybottom`	bottom y-coordinate of the bar
`xright`	right x-coordinate of the bar
`ytop`	top y-coordinate of the bar
`col`	vector of colors
`clim`	2-vector specifying the range of values, linearly represented by the full color scale
`show.border`	logical flag specifying whether to draw a rectangular border around the bar
`text`	axis label
`line`	distance between label and color bar
`show.axis`	logical flag specifying whether to draw an axis
`side`	character string, which specifies the location of the axis on the bar. This has to be 'left' or 'right' (default).
`lwd`	linewidth of border and ticks
`nticks`	number of ticks on the axis
`at`	vector of values specifying the tick positions on the axis, this overrides nticks.
`srt`	rotation angle of tick text
`ticklength`	length of ticks
`shift`	extra distance between axis numbers and color bar
`...`	optional arguments to be passed text function.

Value

None

Author(s)

Danail Obreschkow

Examples

```
## Plot a spherical function with color bar
nplot(xlim=c(0,1.2), asp=1)
f = function(theta,phi) cos(10*theta+phi)
sp = sphereplot(f, 200, col=planckcolors(200), phi0=0.1, theta0=pi/3,
add=TRUE, center=c(0.5,0.5), radius=0.4, clim=c(-1,1))
colorbar(1,0.1,1.1,0.9,col=sp$col,clim=sp$clim)
```

---

contourlevel *Find contour levels of a d-dimensional density field*

Description

Given a vector d-dimensional vector/array f or function f(x) of a d-element vector x, this routine evaluates the value l, such that the sum/integral of f over the domain f>l makes up a fraction p of the total sum/integral of f. The main purpose of this routine is to determine the iso-contour levels of a likelihood or density function, containing a cumulative probability-mass p.

Usage

```
contourlevel(
  f,
  p = c(0.6826895, 0.9544997),
  xmin = NULL,
  xmax = NULL,
  neval = 10000,
  napprox = 10,
  ...
)
```

Arguments

f	either a d-dimensional vector/array or a function f(x) of a d-element vector x. There is no need for f to be normalized.
p	vector of probabilities
xmin, xmax	(only used if f is a function) vectors with of lower and upper limits of x, defining the domain on which the function f(x) is evaluated. Outside this domain, f is assumed to vanish. These limits should be chosen as narrow as possible for the algorithm to converge quickly.
neval	(only used if f is a function) maximum number of function evaluations in numerical integrals
napprox	(only used if f is a function) number of points used interpolate the cumulative probability density. If set to 0, no interpolation is used.
...	(only used if f is a function) additional parameters to be passed to the function f.

Value

Returns a vector of levels l, which has the same length as p.

Author(s)

Danail Obreschkow

See Also

dpqr

Examples

```
## f(x) is a 1D PDF
# compute 1-sigma and 2-sigma contour levels of a normal distribution, i.e.
# the values l, such that int_{dnorm(x)>l} dnorm(x) dx = p (=68.3%, 95.4%).
l = contourlevel(dnorm,xmin=-10,xmax=10,napprox=0)
print(l)

# compare these values to the solutions dnorm(1), dnorm(2)
print(dnorm(c(1,2)))


## f(x) is a 2D likelihood function
# Produce 20%, 40%, 60% and 80% iso contours on the 2D likelihood function f(x)
f = function(x) cos(2*x[1]-x[2]-1)^2*exp(-x[1]^2-x[2]^2-x[1]*x[2])
p = c(0.2,0.4,0.6,0.8) # cumulative probability
l = contourlevel(f,p,c(-5,-5),c(5,5)) # values l, such that int_{f>l}(f(x)dx)=p int(f(x)*dx)

# Plot function and contours at the levels l
x = seq(-3,3,length=200)
m = pracma::meshgrid(x)
z = array(Vectorize(function(x,y) f(c(x,y)))(m$Y,m$X),dim(m$X))
image(x,x,z,col=terrain.colors(100))
contour(x,x,z,levels=l,add=TRUE,labels=sprintf('%.0f%%',p*100),labcex=0.7)


## f is a 20-by20 array representing a gridded pointset
# produce a set of 1000 points in 2D, drawn from a 2D normal distribution
set.seed(1)
x = MASS::mvrnorm(n=1000, mu=c(0,0), matrix(c(3,1,1,2),2,2))

# grid these points onto a regular 20-by-20 grid
g = griddata(x, min=-6, max=6)

# find 1-sigma and 2-sigma contour levels and draw contours at these levels
l = contourlevel(g$field)
plot(x, xlim=g$grid[[1]]$lim, ylim=g$grid[[2]]$lim, pch=20, cex=0.5)
contour(g$grid[[1]]$mid,g$grid[[2]]$mid,g$field,
        levels=l,add=TRUE,col='red',lwd=c(2,1),labels=NA)
```

---

cosmofct *Fast cosmology conversion functions*

Description

Generates all 20 conversion functions between redshift (z), luminosity distance (dl), comoving distance (dc) and angular diameter distance (da), and lookback time (t = light travel time from specified redshift); based on the *celestial* package.

Usage

```
cosmofct(zmin = 0, zmax = 1, dz = 0.02, H0 = 70, OmegaM = 0.3, ...)
```

Arguments

`zmin`	minimum redshift for which the conversion functions are used
`zmax`	maximum redshift for which the conversion functions are used
`dz`	redshift interval on which the conversion functions are interpolated (default of 0.02 is normally largely sufficient)
`H0`	local Hubble constant in units of km/s/Mpc (default 70).
`OmegaM`	local normalised matter density (default 0.3).
`...`	other cosmological parameters accepted by `cosdist` of the *celestial* package. Defaults are OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0.

Value

Returns a list of 20 vectorized functions; e.g. dc2z to convert from comoving distance to redshift. Also contains the age of the universe at z=0. All distances are in units of Mpc and times are in units of Gyr.

Author(s)

Danail Obreschkow (based on *celestial* package by Aaron Robotham)

Examples

```
## uses a flat LCDM cosmology with h=0.68, OmegaM=0.32 and OmegaL=0.68
cosmo = cosmofct(0,1,H0=68,OmegaM=0.32)
curve(cosmo$z2dl(x),0,1,xlab='z',ylab='distance',col='red')
curve(cosmo$z2dc(x),0,1,col='black',add=TRUE)
curve(cosmo$z2da(x),0,1,col='blue',add=TRUE)
d = seq(500,5000,500)
points(cosmo$dl2z(d),d,pch=16,col='red')
points(cosmo$dc2z(d),d,pch=16,col='black')
points(cosmo$da2z(d),d,pch=16,col='blue')
```

cshift *Circularly shift each dimension of an array*

Description

Circulates each dimension of an array. This routine is identical to `circshift`, but works with arrays up to 5 dimensions.

Usage

```
cshift(x, s)
```

Arguments

x	vector or array (up to rank 5)
s	scalar, if x is a vector, or a vector of length matching the rank of x, if x is an array

Value

Returns a vector or array of the same shape as x.

Author(s)

Danail Obreschkow

cst *Scientific constants*

Description

The list `cst` contains useful scientific constants in SI units, mainly for astrophysics.

Usage

```
cst
```

Format

An object of class `list` of length 22.

Value

None

Author(s)

Danail Obreschkow

Examples

```
# print all the constants to console
for (i in seq(length(cst))) cat(sprintf('%6s = %.12e\n',names(cst)[i],cst[i]))
```

---

cubehelix — *Cube Helix colour palette*

Description

Generate the Cube Helix colour palette, designed to appropriately display of intensity images, as the brightness increases monotonically when displayed in greyscale.

Usage

```
cubehelix(n, r = 1.5, hue = 1, gamma = 1, rev = FALSE)
```

Arguments

`n`	integer number of colours in the scale
`r`	real number specifying the number of rotations of the helix over the scale
`hue`	non-negative number specifying the colour intensity from grey (0) to normal (1) and over-saturated (>1)
`gamma`	positive number specifying the relative importance of low vs high values
`rev`	logical flag indicating whether the ordering of the colors should be reversed

Details

This scheme was published by Green, D. A., 2011, "A colour scheme for the display of astronomical intensity images." Bulletin of the Astronomical Society of India, 39, 289.

Value

Returns an n-vector of RGB colour strings.

Author(s)

Danail Obreschkow

dft — *Discrete Fourier Transform*

Description

Discrete Fourier Transform (DFT) with longest modes at the center in Fourier space and normalized such that dft(dft(f),inverse)=f. This is the discretization scheme described in Appendix D of Obreschkow et al. 2013, ApJ 762. Relies on `fft`.

Usage

```
dft(f, inverse = FALSE, shift = -floor(dim(as.array(f))/2), simplify = TRUE)
```

Arguments

`f`	real or complex D-dimensional array containing the values to be transformed.
`inverse`	logical flag; if TRUE the inverse Fourier transform is performed.
`shift`	D-vector specifying the integer shift of the coordinates in Fourier space. Set to `shift=rep(0,D)` to produced a DFT with the longest mode at the corner in Fourier space.
`simplify`	logical flag; if TRUE the complex output array will be simplified to a real array, if it is real within the floating point accuracy

Value

Returns an array of the same shape as `f`, containing the (inverse) Fourier Transform.

Author(s)

Danail Obreschkow

See Also

`fft`

Examples

```
## DFT of a 2D normal Gaussian function
n = 30
f = array(0,c(n,n))
for (i in seq(n)) {
  for (j in seq(n)) f[i,j] = exp(-(i-6)^2/4-(j-8)^2/2-(i-6)*(j-8)/2)
}
plot(NA,xlim=c(0,2.1),ylim=c(0,1.1),asp=1,bty='n',xaxt='n',yaxt='n',xlab='',ylab='')
rasterImage(f,0,0,1,1,interpolate=FALSE)
g = dft(f)
img = array(hsv((pracma::angle(g)/2/pi)%%1,1,abs(g)/max(abs(g))),c(n,n))
rasterImage(img,1.1,0,2.1,1,interpolate=FALSE)
```

```
text(0.5,1,'Input function f',pos=3)
text(1.6,1,'DFT(f)',pos=3)
```

`dftgrid` *Produce coordinates for Discrete Fourier Transform*

Description

Produces the direct space coordinates (x) and Fourier space frequencies (f) and angular frequencies (k), corresponding to the Discrete Fourier Transform `dft` of this package.

Usage

```
dftgrid(N, L, x0 = 0, k0 = -floor(N/2) * 2 * pi/L)
```

Arguments

`N`	number of divisions along one dimension used in direct and Fourier space
`L`	side-length (=period) of the data long one dimension
`x0`	zero-point of x-coordinates (=0 in most applications)
`k0`	zero-point of k-coordinates (=0 or -floor(N/2)*dk in most applications)

Value

Returns a list with:

`x`	vector of direct space coordinates
`f`	vector of Fourier space coordinates (frequencies)
`k`	vector of Fourier space coordinates (angular frequencies)
`dx`	spacing of x-values
`df`	spacing of f-values
`dk`	spacing of k-values
`L`	input value of L
`N`	input value of N

Author(s)

Danail Obreschkow

`dpqr` *d/p/q/r-family for a custom distribution*

Description

Produces the family of d/p/q/r functions associated with a custom one-dimensional distribution function; similarly to the standard families dnorm/pnorm/qnorm/rnorm, dunif/punif/...

Usage

```
dpqr(fun, min, max)
```

Arguments

`fun`	distribution function of a single variable; does not have to be normalized
`min`, `max`	domain of distribution function; outside this domain `fun` will be considered equal to 0. In practice, this should be the most restrictive domain containing (almost) all the mass of `fun`.

Value

Returns a list of items:

`d(x)`	Probability distribution function (PDF), i.e. a normalised version of `fun`, limited to the domain `[xmin,xmax]`.
`p(x)`	Cumulative distributiont function, defined as the integrated PDF up to x.
`q(p)`	Quantile function, returning the position x, at which the cumulative probability equals `p`.
`r(n)`	A vector of `n` random numbers drawn from the PDF.

Author(s)

Danail Obreschkow

See Also

`rng`, `contourlevel`

Examples

```
f = function(x) sin(x)
rsin = dpqr(f,0,pi)$r
x = rsin(1e3)
hist(x,freq=FALSE)
curve(sin(x)/2,0,pi,add=TRUE)
```

entropy — *Information entropy*

Description

Computes the information entropy H=sum(p*log_b(p)), also known as Shannon entropy, of a probability vector p.

Usage

```
entropy(p, b = exp(1), normalize = TRUE)
```

Arguments

`p`	vector of probabilities; typically normalized, such that sum(p)=1.
`b`	base of the logarithm (default is e)
`normalize`	logical flag. If TRUE (default), the vector p is automatically normalized.

Value

Returns the information entropy in units that depend on b. If b=2, the units are bits; if b=exp(1), the units are nats; if b=10, the units are dits.

Author(s)

Danail Obreschkow

errlines — *Draw a line with uncertainty regions*

Description

Draw a line with uncertainty regions

Usage

```
errlines(
  x,
  y,
  errp,
  errn = errp,
  col = "black",
  alpha = 0.5,
  smooth = FALSE,
  df = NULL,
  ...
)
```

Arguments

`x`	vector of x-coordinates
`y`	vector of y-coordinates
`errp`	vector of y-errors in the positive direction (upwards)
`errn`	vector of y-errors in the negative direction (downwards)
`col`	color of the line
`alpha`	transparency of the uncertainty region
`smooth`	logical flag indicating whether the line should be smoothed
`df`	df (=degrees of freedom) parameter of `smooth.spline` function
`...`	additional parameters used by `lines`

Value

None

Author(s)

Danail Obreschkow

`error` *Stop code and produce an error message*

Description

Stops the code and produces an error message, like the stop() function of the base library, but terminating a timer started with `tick`, if running.

Usage

```
error(...)
```

Arguments

`...`	zero or more objects which can be coerced to character (and which are pasted together with no separator) or a single condition object.

Value

None. geterrmessage() returns the last error message, as a character string.

Author(s)

Danail Obreschkow

See Also

`tick` `tock` `progress`

`fibonaccisphere` *Evenly distributed n points on a sphere*

Description

Distributes n points on a sphere in a relatively even fashion following the generalised Fibonacci algorithm, described at http://extremelearning.com.au/evenly-distributing-points-on-a-sphere/

Usage

```
fibonaccisphere(n = 1000, r = 1, out.xyz = TRUE, out.sph = FALSE)
```

Arguments

`n`	number of points to be placed on the sphere
`r`	radius
`out.xyz`	logical flag specifying whether to return Cartesian coordinates (default is TRUE)
`out.sph`	logical flag specifying whether to return spherical coordinates (default is FALSE); `theta`=colatitute (=polar angle=angle from z-axis), `phi`=longitude (=azimuth angle),

Value

matrix of n points (in n rows), in Cartesian and/or spherical coordinates.

Author(s)

Danail Obreschkow

See Also

`runif3`

Examples

```
## plot standard projections of a 1000-point Fibonacci sphere
xyz = fibonaccisphere()
plot(xyz, asp=1, pch=16, cex=0.5)
```

gradient — *Compute gradient*

Description

Evaluates the gradient of a vector of array of rank 2 or 3

Usage

```
gradient(f, circular = FALSE)
```

Arguments

`f`	vector or array of rank 2 or 3
`circular`	logical scalar or vector specifying whether periodic boundaries are used along each dimension

Value

returns a list with the different components of the gradient; each component has the same dimension as f.

Author(s)

Danail Obreschkow

grf — *Gaussian Random Field generator*

Description

Generates a scalar Gaussian Random Field (GRF) in 1, 2 or 3 dimensions, with a power-law power spectrum of custom slope alpha. The field is normalized such that its mean is zero (up to floating point errors) and its expected standard deviation is one, for alpha=0. The Fourier phases are sampled in order of increasing frequency, such that the random structure is preserved when changing the output size.

Usage

```
grf(nside = 100, dim = 2, alpha = 0, seed = NULL)
```

Arguments

nside	integer number of elements per dimension in the output matrix
dim	integer number of dimensions
alpha	spectral index, such that Fourier amplitudes scale as k^alpha. alpha=0 corresponds to uncorrelated white noise, alpha<0 is red noise, while alpha>1 is blue noise.
seed	optional seed for random number generator

Value

Returns a vector, matrix or 3D-array with `nside` elements per dimension

Author(s)

Danail Obreschkow

Examples

```
# Generate the same 2D GRF in two different resolutions
nplot(c(0,2.1), asp=1)
lowres = grf(nside=30, alpha=-2, seed=1)
graphics::rasterImage(stretch(lowres),0,0,1,1)
highres = grf(nside=120, alpha=-2, seed=1)
graphics::rasterImage(stretch(highres),1.1,0,2.1,1)

# Check the power spectrum of a general GRF
nside = 50 # change this to any integer >1
alpha = -1.7 # change this to any power
dim = 3 # change this to any positive integer (but keep nside^dim reasonable)
x = grf(nside=nside, dim=dim, alpha=alpha)
fourier.grid = dftgrid(N=nside,L=1)
knorm = vectornorm(expand.grid(rep(list(fourier.grid$k),dim)))
power = abs(dft(x))^2
b = bindata(knorm,power,method='equal')
plot(b$xmedian,b$ymedian,log='xy',pch=16,
     xlab='Fourier mode |k|',ylab='Power p(k)',main='Power spectrum')
graphics::curve(x^alpha/nside^dim,col='blue',add=TRUE) # expected power-law
```

---

griddata *Distribute a point set onto a regular grid*

Description

Distributes a set of points in D dimensions onto a regular, D-dimensional grid, using a fast nearest neighbor algorithm. Weights can be used optionally.

Usage

```
griddata(x, w = NULL, n = 10, min = NULL, max = NULL, type = "counts")
```

Arguments

`x`	N-element vector (if D=1) or N-by-D matrix (if D>1), giving the Cartesian coordinates of N points in D dimensions.
`w`	optional N-element vector with weights.
`n`	scalar or D-element vector specifying the number of equally space grid cells along each dimension.
`min`	optional scalar or D-element vector specifying the lower bound of the grid. If not given, min is adjusted to the range of x.
`max`	optional scalar or D-element vector specifying the upper bound of the grid. If not given, max is adjusted to the range of x.
`type`	character string ("counts", "density", "probability") specifying the normalization of the output: "counts" (default) returns the number of points (multiplied by their weights, if given) in each cell; thus the total number of points (or total mass, if weights are given) is `sum(field)`. "density" returns the density, such that the total number of points (or total mass, if weights are given) is `sum(field) dV`. "probability" returns a probability density, such that `sum(field) dV`=1.

Value

Returns a list of items

`field`	D-dimensional array representing the value in each grid cell. See parameter `type` for more details.
`grid`	List of D elements with the grid properties along each dimension. n: number of grid cells; mid: n-vector of mid-cell coordinates; breaks: (n+1)-vector of cell edges; lim: 2-vector of considered range; delta: cell width.
`dV`	Single number representing the volume of the D-dimensional grid cells.

Author(s)

Danail Obreschkow

Examples

```
# Distribute 1-dimensional data onto a regular grid
npoints = 1e4
x = rnorm(npoints)
g = griddata(x,min=-3,max=3,n=100,type='probability')
curve(dnorm(x),-3,3)
points(g$grid$mid,g$field,pch=16)

# Distribute 2-dimensional data onto a regular grid
x = runif(100,max=2)
y = runif(100)
```

```
g = griddata(cbind(x,y),min=c(0,0),max=c(2,1),n=c(20,10))
image(g$grid[[1]]$breaks,g$grid[[2]]$breaks,g$field,
      asp=1,col=grey.colors(100,0,1),xlab='x',ylab='y')
points(x,y,col='red',pch=16)

# ... same with weights
w = runif(100)
g = griddata(cbind(x,y),w,min=c(0,0),max=c(2,1),n=c(20,10))
image(g$grid[[1]]$breaks,g$grid[[2]]$breaks,g$field,
      asp=1,col=grey.colors(100,0,1),xlab='x',ylab='y')
points(x,y,col='red',pch=16,cex=w)
```

---

histcoord *Generate histogram coordinates from mid points*

Description

Converts the mid-point x-values and mean densities of binned data into (x,y)-coordinates of a histogram.

Usage

```
histcoord(x, y, yleft = 0, yright = 0)
```

Arguments

`x`	n-vector giving the bin mid-points
`y`	n-vector giving bin values
`yleft`	optional value specifying the value at the left edge
`yright`	optional value specifying the value at the right edge

Value

(2n+2)-by-2 matrix of (x,y)-coordinates to draw histogram as a connected line

Author(s)

Danail Obreschkow

Examples

```
x = seq(5)
y = sin(x)
plot(x,y,xlim=c(0,6))
lines(histcoord(x,y))
```

`inertia` *Inertia tensor*

Description

Computes the symmetric tensor of moments of inertia

Usage

```
inertia(x, m = 1)
```

Arguments

`x`	n-by-3 matrix (x[1:n,1:3]) specifying the 3D Cartesian coordinates of n points
`m`	n-vector with point masses, or single scalar giving a uniform mass for all points (default is unity)

Value

Returns a 3-by-3 symmetric matrix

Author(s)

Danail Obreschkow

See Also

`quadrupole`, `moments`

`invert` *Invert and shift colors of an image*

Description

Invert the brightness of each color channel in an image and/or circularly shifts the hue value. Optionally, a Gaussian blur can be applied.

Usage

```
invert(
  img = NULL,
  invert = TRUE,
  colshift = 0,
  blur = 0,
  file.in = "",
  file.out = "",
```

```
  format = "png",
  show.image = TRUE
)
```

Arguments

`img`	n-by-m-by-3 array or n-by-m-by-4 array representing an rgb(+alpha) image
`invert`	logical flag indicating whether the channel-brightness should be inverted
`colshift`	numerical value between 0 and 1 setting the circular shift of the hue value. If `invert=TRUE`, choosing `colshift=0.5` preserves the colors, while inverting black and white.
`blur`	numerical value >=0 defining the standard deviation of an optional Gaussian blur.
`file.in`	optional input filename, which can be used to load an image instead of providing it via `img`. This filename is ignored if `img` is specified.
`file.out`	optional output filename.
`format`	one of "png" or "jpg" specifying the file format of the input and output image.
`show.image`	logical flag specifying whether the image is displayed in the R console.

Value

Returns an n-by-m-by-3 array or n-by-m-by-4 array of the processed image.

Author(s)

Danail Obreschkow

Examples

```
img = yinyangyong # this is an example image included in the package

# invert brightness of all channels
invert(img)

# invert brightness, but preserve hue
invert(img, colshift=0.5)
```

`is.equal` *Numerical equality check*

Description

Checks if two or more numerical values are identical within a relative numerical tolerance

Usage

```
is.equal(x, ..., stoptext = NULL, eps = sqrt(.Machine$double.eps))
```

Arguments

`x`	vector or array of values to compare
`...`	optional additional values to compare
`stoptext`	optional character string; if given, the routine stops if the values are not all equal and adds the character string to the error message
`eps`	relative numerical tolerance

Value

Returns a logical value. TRUE means that all values of `x` are equal within the specified relative tolerance; also returns TRUE if all values are Inf or NA or NaN.

Author(s)

Danail Obreschkow

Examples

```
# almost identical values
x = c(acos(1/sqrt(2)),pi/4)
print(x)
print(x[1]==x[2])
print(is.equal(x))

# various other examples
print(is.equal(1,2,3))
print(is.equal(1,NA,3))
print(is.equal(Inf,NA))
print(is.equal(NaN,NA))
print(is.equal(NaN,Inf))
print(is.equal(Inf,Inf,Inf))
print(is.equal(NA,NA))
print(is.equal(NaN,NaN))
print(is.equal(1.4,1.4))
print(is.equal(1.4,1.400000001))
print(is.equal(1.4,1.400000001,1.41))
print(is.equal(0,0,0,0))
```

jackknife *Jackknife Estimation*

Description

Computes the "leave-one-out" Jackknife bias and standard error of an estimator `f(x)` of a data-vector `x`, or an estimator `f(x,y)` of vectors `x` and `y`. See Efron and Tibshirani (1993) for details

Usage

```
jackknife(x, f, y = NULL, ...)
```

Arguments

`x`	data vector
`f`	estimator function `f(x,...)` or `f(x,y,...)`
`y`	optional data vector if `f` is a function of two vectors, such as the correlation coefficient `cor(x,y)`.
`...`	optional arguments to be passed to `f`.

Value

Returns a list with the following components:

`value`	Default value of the estimator `f`.
`bias`	Jackknife bias estimate of `f`.
`unbiased`	Bias-corrected value of `f`.
`sd`	Jackknife standard error of `f`.

Author(s)

Danail Obreschkow

kde2 *Multi-dimensional adaptive kernel density estimation*

Description

Produces a 2D kernel density estimation on a 2D grid from a D-dimensional (D>=2) point set

Usage

```
kde2(
  x,
  w = NULL,
  nx = 300,
  xlim = NULL,
  ylim = NULL,
  smoothing = 1,
  sigma = NULL,
  sigma.min = 0,
  sigma.max = Inf,
  reflect = "",
  algorithm = "kdenn",
  probability = FALSE
)
```

Arguments

`x`	N-by-D vector of x-coordinates or N-by-2 matrix of (x,y)-coordinates
`w`	optional N-element vector with weights
`nx`	integer specifying the number of equally spaced grid cells along the x-axis; the number ny of pixels along the y-axis is determined automatically from xlim and ylim.
`xlim`	2-element vector specifying the x-range
`ylim`	2-element vector specifying the y-range; if needed, this range is slightly adjusted to allow for an integer number of pixels.
`smoothing`	positive linear smoothing factor; the larger, the smoother the density field
`sigma`	optional N-vector, specifying the blurring of each pixel individually in length units of x; only used if algorithm=4.
`sigma.min`	optional value, specifying the minimum blurring of any pixel, expressed in standard deviations in length units of x
`sigma.max`	optional value, specifying the maximum blurring of any pixel, expressed in standard deviations in length units of x
`reflect`	vector of strings c('left','right','bottom','top') specifying the edges, where the data should be reflected to avoid probability density leaking outside the window
`algorithm`	integer value or character string specifying the smoothing algorithm: `basic (0)`: basic nearest-neighbor gridding algorithm without smoothing `blur (1)`: simple Gaussian blur of gridded density field `kdefast (2)`: 2D smoothing method that ignores higher dimensional information and applies a smoothing size to each pixel that depends on the number of objects in each pixel `kdenn (3)`: sophisticated Kernel density estimator that uses D-dimensional nearest neighbor separations to smooth each data point individually `manual (4)`: smooths each data point individually using a Gaussian Kernel with point-dependent standard deviations given in the optional vector `sigma`

probability logical flag. If TRUE, the output field is normalized such that sum(field)dpixel^2=1. If FALSE (default), the field is such that sum(field)dpixel^2 equals the effective number of particles (or effective mass, if weights are given) in the range specified by xlim and ylim, including particle fractions that have been smoothed into the field and excluding particle fractions that have been smoothed out of it.

Value

Returns a list of items

field	2D array of smoothed density field.
x	nx-element vector of cell-center x-coordinates.
y	ny-element vector of cell-center y-coordinates.
xbreak	(nx+1)-element vector of cell-edge x-coordinates.
ybreak	(ny+1)-element vector of cell-edge y-coordinates.
dpixel	grid spacing along x-coordinate and y-coordinate.
algorithm	name of algorithm in use.

Author(s)

Danail Obreschkow

See Also

griddata

Examples

```
# make a mock sample of n d-dimensional points from
# three different components (1D line, 2D square, d-D normal distr)
d = 3 # number of dimensions of mock point set; try to choose different values 2, 3, 4, ...
n = 1e4 # number of particles per component
set.seed(1)
x = rbind(cbind(array(rep(runif(n,-1,1),2),c(n,2)),array(0,c(n,d-2))),
          cbind(array(runif(2*n),c(n,2)),array(0,c(n,d-2))),
          array(rnorm(d*n),c(n,d)))

# grid total projected probability density
npixels = 500 # number of pixels along a grid side
q = midseq(-3,3,npixels)
f1 = outer(dnorm(q),dnorm(q),'*')/3+outer(dunif(q),dunif(q),'*')/3
q = seq(round(npixels/3),round(npixels*2/3))
f1[q+npixels*(q-1)] = f1[q+npixels*(q-1)]+(npixels/6)^2/length(q)/3

# recover 2D projected pdf from 3D point sample using different methods
f2 = kde2(x, n=npixels, xlim=c(-3,3), ylim=c(-3,3), algorithm='basic', probability=TRUE)$field
f3 = kde2(x, n=npixels, xlim=c(-3,3), ylim=c(-3,3), algorithm='kdefast', probability=TRUE)$field
```

```
f4 = kde2(x, n=npixels, xlim=c(-3,3), ylim=c(-3,3), algorithm='kdenn', probability=TRUE)$field

# plot the 2D fields
img = function(f,x,y,title) {
  graphics::rasterImage(rasterflip(lim(f)^0.3),x,y,x+0.99,y+0.99)
  graphics::text(x+0.05,y+0.9,title,col='orange',pos=4)
}
oldpar = graphics::par(mar=rep(0.1,4))
nplot(c(0,2),c(0,2),asp=1)
img(f1,0,1,'Input pdf')
img(f2,1,1,'Random sample ("basic")')
img(f3,0,0,'Recovered pdf ("kdefast")')
img(f4,1,0,'Recovered pdf ("kdenn")')
graphics::par(oldpar)
```

`landyszalay` *Two-point correlation estimation*

Description

Evaluates the Landy-Szalay (1993) estimator of the two-point correlation function of a point set D given a random comparison set R. The two point sets D and R can be made of different numbers of points, as the pair-counts are automatically normalized according to the number of points. In fact, it is often preferable to make the R set larger to reduce the R-related shot noise in the two-point estimator.

Usage

```
landyszalay(D, R, dr = 0.1, cpp = TRUE)
```

Arguments

`D`	n-element vector or n-by-d matrix of d-dimensional positions of the data points
`R`	m-element vector or m-by-d matrix of d-dimensional positions of the random comparison points
`dr`	bin size for the evaluation of the two-point correlation function
`cpp`	logical flag; if set to TRUE (default) a fast implementation in C++ is used to count the point-pairs in distance bins, otherwise the counting is performed less efficiently in R.

Value

Returns a data frame with the two-point statistics of the data points:

`r`	vector with the mid-points of the distance bins for which the two-point correlation function has been evaluated.
`xi`	values of the two-point correlation function at the distances r.
`err`	Poisson errors of xi.

Author(s)

Danail Obreschkow

See Also

`paircount`

`last` *Last element of a vector*

Description

Returns the last element of a vector or the n-th element counting from the end of a vector.

Usage

```
last(x, n = 1)
```

Arguments

`x`	vector
`n`	optional integer specifying the n-th element from the end to be returned

Value

scalar of the same type as `x`

Author(s)

Danail Obreschkow

`lightness` *Change lightness of a color*

Description

Change lightness of a color

Usage

```
lightness(col, light = 0.5)
```

Arguments

`col`	is a color or vector/array of colors, specified as text (e.g. 'purple') or 7/9-character (e.g. '#A020F0')
`light`	lightness value, according to a HSL scheme, between 0 and 1 or a vector/array thereof

Value

Returns a 9-character color or vector/array of 9-character colors.

Author(s)

Danail Obreschkow

See Also

`transparent`

Examples

```
# Generate different lightnesses of the same color
plot(runif(50),runif(50),pch=20,cex=10,col=lightness('purple',runif(50)))
```

`lim` *Crop values to a custom range*

Description

Limits the values of a vector or array to a desired interval, while keeping the shape of the input argument

Usage

```
lim(x, min = 0, max = 1, clip = NULL, na = NULL)
```

Arguments

`x`	vector or array
`min`	minimum value
`max`	maximum value
`clip`	optional value specifying the value assigned to clipped data, e.g. `clip=NA`
`na`	optional value specifying the value assigned to non-numbers (NA and NaN)

Value

vector/array of the same shape as `x`

Author(s)

Danail Obreschkow

See Also

stretch

---

linuxspaces *Handle spaces in Linux filenames*

Description

Convert spaces in filenames (" ") to linux-type spaces "\ ", needed when calling system() on macOS.

Usage

```
linuxspaces(txt)
```

Arguments

txt	filename, which may contain ordinary spaces, e.g. "my file 1.txt"

Value

filename with modified spaces, e.g. "my\ file\ 1.txt"

Author(s)

Danail Obreschkow

Examples

```
filename = '~/Desktop/my file 1.txt'
command = sprintf('ls -l %s',linuxspaces(filename))
## Not run:
system(command)

## End(Not run)
```

`loadbin` *Read binary data into array*

Description

Reads binary data using the base function `readBin` and recasts it into an array of custom dimensions.

Usage

```
loadbin(
  filename,
  dim,
  bytes = 4,
  type = "numeric",
  signed = FALSE,
  endian = "little"
)
```

Arguments

`filename`	path of the file to be loaded
`dim`	vector specifying the dimensions of the array
`bytes`	number of bytes per number in the binary file
`type`	character vector of length describing the data type: "numeric" (default), "double", "integer", "int", "logical", "complex", "character", "raw"
`signed`	logical. Only used for integers of sizes 1 and 2, when it determines if the quantity on file should be regarded as a signed or unsigned integer.
`endian`	endian-type ("big" or "little") of the file

Value

Returns an array of dimension dim.

Author(s)

Danail Obreschkow

makeframe *Display a single movie frame*

Description

Displays a single movie-frame in the R-console, exactly as used in a movie generated with makemovie.

Usage

```
makeframe(
  frame.draw,
  frame.index,
  width = 1080,
  height = 720,
  cex = 1,
  oversampling = 1,
  pngfile = NULL
)
```

Arguments

frame.draw	function that plots an individual frame. This function must have exactly one argument 'x', which can be integer (e.g. a simple frame index) or real (e.g. a time).
frame.index	list of frame indices 'x' to be included in the movie
width	integer number of pixels along the horizontal axis
height	integer number of pixels along the vertical axis
cex	number defining the overall scaling of line widths, font sizes, etc.
oversampling	integer specifying the oversampling factor along both dimensions. If larger than 1, frames are plotted with (width*oversampling)-by-(height*oversampling) pixels and then resized back to width-by-height. This can be used to make line objects and text move more smoothly.
pngfile	optional path+filename of output file to save the image. R must have write access to this file.

Value

Returns the displayed image as n-by-my-by 4 array, representing the 4 RGBA channels of height n and width m.

Author(s)

Danail Obreschkow

See Also

makemovie

Examples

```
## Example: Movie of a manual clock

# Function to draw a single clock face with two hands
frame = function(time) {
  oldpar = graphics::par(mar=c(0,0,0,0))
  nplot(xlim=c(-1.1,1.1),ylim=c(-1.1,1.1),pty='s')
  plotrix::draw.circle(0,0,1,col='#aaaaff')
  radius = c(0.5,0.9)
  speed = 2*pi/c(720,60)
  lwd = c(4,2)
  graphics::arrows(0,0,radius*sin(speed*time),radius*cos(speed*time),lwd=lwd)
  graphics::par(oldpar)
}

# Produce movie
## Not run:
makeframe(frame,15,200,200)

## End(Not run)
```

---

makemovie *Produce a movie from frame-drawing function*

Description

Generates an MP4-movie provided a custom function that plots individual frames. The routine has been developed and tested for MacOS and it requires on a working installation of ffmpeg.

Usage

```
makemovie(
  frame.draw,
  frame.index,
  output.path,
  output.filename,
  width = 1080,
  height = 720,
  fps = 60,
  keep.frames = FALSE,
  quiet = FALSE,
  separator = "/",
  ffmpeg.cmd = "ffmpeg",
```

```
  ffmpeg.opt = "-vcodec libx264 -crf 18 -pix_fmt yuv420p",
  manual = FALSE,
  cex = 1,
  oversampling = 1,
  first.index = 1,
  last.index = length(frame.index)
)
```

Arguments

`frame.draw`	function that plots an individual frame. This function must have exactly one argument 'x', which can be integer (e.g. a simple frame index) or real (e.g. a time).
`frame.index`	list of frame indices 'x' to be included in the movie
`output.path`	character specifying the directory, where the movie and temporary frames are saved
`output.filename`	movie filename without path. This filename should end on the extension '.mp4'.
`width`	integer number of pixels along the horizontal axis
`height`	integer number of pixels along the vertical axis
`fps`	number of frames per second
`keep.frames`	logical flag specifying whether the temporary directory with the individual frame files should be kept. If `manual` is set to `TRUE`, the frames are always kept.
`quiet`	logical flag; if true, all console outputs produced by 'ffmpeg' are suppressed
`separator`	filename separate of the system ('/' for Mac, Linux, Unix; '\' for Windows)
`ffmpeg.cmd`	command used to call ffmpeg form a terminal. Normally, this is just 'ffmpeg'.
`ffmpeg.opt`	compression and formatting options used with ffmpeg
`manual`	logical flag; if true, ffmpeg is not called from within the code and the frames are never deleted. The suggested linux command line is returned as output.
`cex`	number defining the overall scaling of line widths, font sizes, etc.
`oversampling`	integer specifying the oversampling factor along both dimensions. If larger than 1, frames are plotted with (width*oversampling)-by-(height*oversampling) pixels and then resized back to width-by-height. This can be used to make line objects and text move more smoothly.
`first.index`	integer specifying the first index of the vector frame.index to consider. Choosing a value larger than the default (1) can be used to continue a previously interrupted call of makemovie and/or to call makemovie from different R sessions in parallel.
`last.index`	integer specifying the last index of the vector frame.index to consider. Choosing a value smaller than the default (length(frame.index)) can be used to continue a previously interrupted call of makemovie and/or to call makemovie from different R sessions in parallel.

Value

Linux command line to convert frames into movie using ffmpeg.

Author(s)

Danail Obreschkow

See Also

makeframe

Examples

```
## Example: Movie of a manual clock

# Function to draw a single clock face with two hands
frame = function(time) {
  oldpar = graphics::par(mar=c(0,0,0,0))
  nplot(xlim=c(-1.1,1.1),ylim=c(-1.1,1.1),pty='s')
  plotrix::draw.circle(0,0,1,col='#aaaaff')
  radius = c(0.5,0.9)
  speed = 2*pi/c(720,60)
  lwd = c(4,2)
  graphics::arrows(0,0,radius*sin(speed*time),radius*cos(speed*time),lwd=lwd)
  graphics::par(oldpar)
}

# Produce movie
## Not run:
makemovie(frame,seq(0,60,0.5),'~/testmovie','movie.mp4',200,200)

## End(Not run)
```

`mcintegral` *Monte Carlo and Quasi-Monte Carlo integration in any dimension*

Description

Numerical integration using a Monte Carlo (MC) or Quasi-Monte Carlo (QMC) algorithm, based on a Halton sequence. These algorithms are of low order (1/sqrt(n) for MC, log(n)/n for QMC in one dimension) compared to the typical orders of 1D deterministic integrators, such as those available in the `integrate` function. The MC and QMC integrators are suitable to compute D-dimensional integrals with D»1, since the order of most deterministic methods deteriorates exponentially with D, whereas the order of MC remains 1/sqrt(n), irrespective of D, and the order of QMC only deteriorates slowly with D as log(n)^D/n.

Usage

```
mcintegral(f, a, b, n = 1e+05, qmc = FALSE, seed = NULL, warn = TRUE)
```

Arguments

`f`	scalar function of a D-vector to be integrated numerically; for fast performance, this function should be vectorized, such that it returns an N-element vector if it is given an N-by-D matrix as argument. An automatic warning is produced if the function is not vectorized in this manner.
`a`	D-vector with lower limit(s) of the integration
`b`	D-vector with upper limit(s) of the integration
`n`	approximate number of random evaluations. (The exact number is max(1,round(sqrt(n)))^2.)
`qmc`	logical flag. If false (default), pseudo-random numbers are used; if true, quasi-random numbers from a D-dimensional Halton sequence are used.
`seed`	optional seed for random number generator. Only used if `qmc` is false.
`warn`	logical flag. If true (default), a warning is produced if the function f is not vectorized.

Value

Returns a list of items:

`value`	the best estimate of the integral.
`error`	an estimate of the statistical 1-sigma uncertainty.
`iterations`	exact number of evaluations (close to n).

Author(s)

Danail Obreschkow

Examples

```
## Numerically integrate sin(x)
f = function(x) sin(x)
m = mcintegral(f,0,pi)
cat(sprintf('Integral = %.3f\u00B1%.3f (true value = 2)\n',m$value,m$sigma))

## Numerically compute the volume of a unit sphere
sphere = function(x) as.numeric(rowSums(x^2)<=1) # this is vectorized
vmc = mcintegral(sphere,rep(-1,3),rep(1,3),seed=1)
vqmc = mcintegral(sphere,rep(-1,3),rep(1,3),qmc=TRUE)
cat(sprintf('Volume of unit sphere = %.3f\u00B1%.3f (MC)\n',vmc$value,vmc$error))
cat(sprintf('Volume of unit sphere = %.3f\u00B1%.3f (QMC)\n',vqmc$value,vqmc$error))
cat(sprintf('Volume of unit sphere = %.3f (exact)\n',4*pi/3))
```

midseq *Mid-points of regular grid*

Description

Compute the mid-point positions of a one-dimensional regular grid of n equal intervals.

Usage

```
midseq(min, max, n = 1)
```

Arguments

`min`	left boundary of first bin
`max`	right boundary of last bin
`n`	number of bins

Value

vector of mid points

Author(s)

Danail Obreschkow

mollweide *Mollweide projection*

Description

Performs a Mollweide projection (also known as Babinet projection, homalographic projection, homolographic projection, and elliptical projection) of longitude and latitude coordinates. The most important feature of the Mollweide projection is that it preserves surface areas, which makes it a commonly used projection in geography, astronomy and cosmology. The total surface area of the standard projection is equal to the surface area of the unit sphere (4pi); and the shape of the fully projected sphere is an ellipse (with axes lengths 2*sqrt(2) and sqrt(2)).

Usage

```
mollweide(lon, lat, lon0 = 0, radius = 1, deg = FALSE)
```

Arguments

`lon`	n-vector of longitudes in radian (unless deg=TRUE)
`lat`	n-vector of latitudes in radian (unless deg=TRUE), must lie between -pi/2 and +pi/2
`lon0`	latitude of null meridian, which will be projected on to x=0
`radius`	radius of spherical projection, such that the surface area of the projection equals 4*pi*radius^2
`deg`	logical flag; if set to TRUE, the input arguments `lon`, `lat`, `lon0` are assumed to be in degrees (otherwise in radians)

Value

Returns an n-by-2 matrix of 2D Cartesian coordinates `x` and `y`.

Author(s)

Danail Obreschkow

Examples

```
lon = runif(1e4,0,2*pi)
lat = asin(runif(1e4,-1,1)) # = uniform sampling of the sphere
plot(mollweide(lon,lat),xlim=c(-3,3),ylim=c(-1.5,1.5),pch=16,cex=0.5)
plotrix::draw.ellipse(0,0,2*sqrt(2),sqrt(2),border='orange',lwd=2)
```

`moments` *Second moment tensor*

Description

Compute the tensor of second moments of a set of point masses

Usage

```
moments(x, m = 1)
```

Arguments

`x`	n-by-3 matrix (x[1:n,1:3]) specifying the 3D Cartesian coordinates of n points
`m`	n-vector with point masses, or single scalar giving a uniform mass for all points (default is unity)

Value

Returns a 3-by-3 symmetric matrix

Author(s)

Danail Obreschkow

See Also

inertia, quadrupole

Examples

```
# Make a randomly oriented ellipsoid of semi-axes a=2.1, b=1.73, c=0.8
x = t(t(fibonaccisphere(1e4))*c(2.1,1.73,0.8))
x = x%*%rotation3(c(0.3,1.64,2.31))

# Recover lengths of semi-axes from eigenvalues of second moment tensor
M = moments(x,m=1/dim(x)[1])
v = sqrt(3*eigen(M)$values)
print(v)
```

---

mutual — *Mutual information of two random variables*

Description

Computes the mutual information of two random variables X and Y, given their 2D density represented in a matrix.

Usage

```
mutual(x, y = NULL, b = exp(1), n = NULL, xlim = NULL, ylim = NULL)
```

Arguments

`x`	either of the following: (1) an m-by-n matrix representing the 2D probability mass function of two random variables X and Y; all elements must be non-negative; the normalization is irrelevant. (2) an n-vector of sampled x-values; in this case y must be specified.
`y`	optional vector of sampled y-values (only used if x is a vector of x-values).
`b`	base of the logarithm in mutual information I(X,Y). Default is e.
`n`	scalar or 2-element vector specifying the number of equally space grid cells. Only used if x and y are vectors. If not provided, the default is n=0.2*sqrt(length(x)), bound between 2 and 1000. Note that n~sqrt(length(x)) keeps the mutual information constant for random data sets of different size.
`xlim`	2-element vector specifying the x-range (data cropped if necessary). Only used if x and y are vectors. If not given, xlim is set to the range of x.
`ylim`	2-element vector specifying the y-range (data cropped if necessary). Only used if x and y are vectors. If not given, ylim is set to the range of y.

Value

Returns a list of items:

- `I` standard mutual information I(X,Y).
- `N` normalized mutual information I(X,Y)/sqrt(H(X)*H(Y)), where H is the Shannon information entropy.

Author(s)

Danail Obreschkow

`ndft` *Non-uniform Discrete Fourier Transform*

Description

Compute the one-dimensional Non-uniform Discrete Fourier Transform (NDFT). This is needed if the data are sampled at irregularly spaced times.

Usage

```
ndft(
  f,
  x = seq(0, length(f) - 1)/length(f),
  nu = seq(0, length(f) - 1),
  inverse = FALSE,
  weighing = TRUE,
  simplify = TRUE
)
```

Arguments

- `f` vector of real or complex function values
- `x` vector of points in direct space, typically time or position coordinates. If `inverse=FALSE`, `x` must have the same length as `f`.
- `nu` vector of frequencies in units of [1/units of `x`]. If `inverse=TRUE`, `nu` must have the same length as `f`.
- `inverse` logical flag; if TRUE, the inverse Fourier transform is performed.
- `weighing` logical flag; if TRUE, irregularly spaced evaluations of `f` will be weighted proportionally to their bin width in `x` (if `inverse=FALSE`) or `nu` (if `inverse=FALSE`).
- `simplify` logical flag; if TRUE, the complex output array will be simplified to a real array, if it is real within the floating point accuracy.

Details

The one-dimensional NDFT of a vector $f = (f_1, ..., f_N)$ is defined as

$$F_j = \sum_i w_i f_i exp(-2\pi i * x_i * \nu_j)$$

where w_i are optional weights, proportional to the interval around x_i, only used if `weighing=TRUE`. Likewise, the inverse NDFT is defined as

$$f_i = \sum_j w_j F_j exp(+2\pi i * x_i * nu_j)$$

where w_j are optional weights, proportional to the interval around ν_j. In this implementation NDFTs are computed using a brute force algorithm, scaling as $O(N * N)$, which is considerably worse than the $O(N) * log(N)$ scaling of FFT algorithms. It is therefore important to pick the required frequencies wisely to minimise computing times.

Value

Returns a vector of the same length as `x` (if `inverse=FALSE`) or `nu` (if `inverse=TRUE`).

Author(s)

Danail Obreschkow

See Also

`fft`

Examples

```
# Define an example signal
nu1 = 1 # [Hz] first frequency
nu2 = 8 # [Hz] second frequency in the signal
s = function(t) sin(2*pi*nu1*t)+0.7*cos(2*pi*nu2*t+5)

# Discretize signal
N = 50 # number of samples
t.uniform = seq(0,N-1)/N
t.nonuniform = t.uniform^1.3
s.uniform = s(t.uniform)
s.nonuniform = s(t.nonuniform)

# Plot signal
oldpar = par(mfrow = c(1, 2))
curve(s,0,1,500,xaxs='i',main='Time signal',xlab='Time t',ylab='s(t)',col='grey')
points(t.uniform,s.uniform,pch=16,cex=0.8)
points(t.nonuniform,s.nonuniform,pch=4,col='blue')
legend('topright',c('Continuous signal','Uniform sample','Non-uniform sample'),
       lwd=c(1,NA,NA),pch=c(NA,16,4),col=c('grey','black','blue'),pt.cex=c(1,0.8,1))

# Uniform and non-uniform DFT
```

```
nu = seq(0,N-1) # discrete frequencies
spectrum.uniform = stats::fft(s.uniform)
spectrum.nonuniform = ndft(s.nonuniform,t.nonuniform,nu)
spectrum.wrong = stats::fft(s.nonuniform)

# Evaluate power
power.uniform = Mod(spectrum.uniform)^2
power.nonuniform = Mod(spectrum.nonuniform)^2
power.wrong = Mod(spectrum.wrong)^2

# Plot DFT and NDFT up to Nyquist frequency
plot(nu,power.uniform,pch=16,cex=0.8,xlim=c(0,N/2),xaxs='i',
              main='Power spectrum',xlab=expression('Frequency'~nu~'[Hz]'),ylab='Power')
points(nu,power.nonuniform,pch=4,col='blue')
points(nu,power.wrong,pch=1,col='red')
abline(v=c(nu1,nu2),col='grey',lty=2)
legend('topright',c('DFT of uniform sample','NDFT of non-uniform sample',
'DFT of non-uniform sample (wrong)','Input frequencies'),
       lwd=c(NA,NA,NA,1),lty=c(NA,NA,NA,2),pch=c(16,4,1,NA),
       col=c('black','blue','red','grey'),pt.cex=c(0.8,1,1,NA))
par(oldpar)
```

---

ngon   *Draw a regular n-gon*

Description

Draws a regular polygon with n sides, such as a triangle (n=3) or hexagon (n=6).

Usage

```
ngon(x = 0, y = 0, s = 1, n = 6, angle = 0, fix.aspect = TRUE, ...)
```

Arguments

x	vector of x-coordinates specifying the centres of the n-gons.
y	vector of y-coordinates specifying the centres of the n-gons.
s	side lengths; either a single number or a vector of the same length as x and y. In log-log plots, the value of s is in units of dex.
n	number of sides of the regular n-gons; either a single number or a vector of the same length as x and y.
angle	rotation angle in radians; either a single number or a vector of the same length as x and y.
fix.aspect	logical flag. If TRUE (default), the aspect ratio of the n-gon on the screen is forced to be unity, even if the this makes it irregular in the coordinates of the plot. If FALSE, the n-gon is regular in plot coordinates, which makes it distorted in screen coordinates if the aspect ratio is not one and/or if logarithmic coordinates are used.

... additional arguments used by polygon.

Value

None.

Author(s)

Danail Obreschkow

Examples

```
## Plot random points on the unit sphere in Mollweide projection
# hexagon at the center of a plot
nplot(bty='o', asp=0.5)
ngon(x=0.5, y=0.5, s=0.1, n=6, fix.aspect=FALSE)

ngon(x=0.5, y=0.5, s=0.1, n=6, border='red')

plot(NA,xlim=c(1,1e3),ylim=c(1,1e5),log='xy')
ngon(x=10^runif(10,0,3), y=10^runif(10,0,5), s=1, n=6,  border='red',lwd=3)
```

`nplot` *Make empty plot area*

Description

Open an empty plot

Usage

```
nplot(
  xlim = c(0, 1),
  ylim = c(0, 1),
  xlab = "",
  ylab = "",
  xaxs = "i",
  yaxs = "i",
  xaxt = "n",
  yaxt = "n",
  bty = "n",
  ...
)
```

Arguments

`xlim, ylim`	vectors with plotting limits.
`xlab, ylab`	horizontal and vertical labels.
`xaxs, yaxs`	style of the axis interval (see `par`).
`xaxt, yaxt`	character which specifies the x axis type (see `par`).
`bty`	character specifying the border type (see `par`).
`...`	additional arguments used by `plot`

Value

None

Author(s)

Danail Obreschkow

`paircount` *Count the number of point-pairs in distance bins*

Description

Count the number of point-pairs in equally spaced distances bins. Code works in any dimension. If only one point set is provided, the distances of this point set with itself are used (counting each pairs only once, i.e. only ij, not ji).

Usage

```
paircount(x, y = NULL, dr, rmax, cpp = TRUE)
```

Arguments

`x`	n-element vector or n-by-d matrix of d-dimensional positions of data points
`y`	optional m-element vector or m-by-d matrix of d-dimensional positions of a second point set
`dr`	distance bin size
`rmax`	maximum distance to be considered
`cpp`	logical flag; if set to TRUE (default) a fast implementation in C++ is used, otherwise the counting is performed less efficiently in R.

Value

Returns a data frame of two column vectors:

`r`	mid-points of the distance bins.
`n`	number of pairs in the distance bin.

Author(s)

Danail Obreschkow

See Also

landyszalay

---

pdf2jpg *Convert pdf to jpg*

Description

Calls the console "convert" function to convert a pdf-file into a jpeg-image. Requires "convert" to be installed already.

Usage

```
pdf2jpg(
  pdf.filename,
  jpg.filename,
  quality = 100,
  background = "white",
  dim = c(1600, 1200),
  remove.pdf = FALSE,
  verbose = TRUE
)
```

Arguments

`pdf.filename`	filename of pdf input file
`jpg.filename`	filename of jpeg output file
`quality`	quality of jpeg compression from 1 (worst) to 100 (best)
`background`	color of background
`dim`	size in pixels of the jpeg-image
`remove.pdf`	logical flag. If TRUE, the pdf file is deleted after the conversion.
`verbose`	logical flag to turn on/off console statements.

Value

None

Author(s)

Danail Obreschkow

planckcolors — *Planck CMB colour palette*

Description

Generates color scale matching the one normally used to display the Planck CMB temperature map from -300uK to +300uK.

Usage

```
planckcolors(n)
```

Arguments

`n`	integer number of colors in the scale

Value

Returns an n-vector of RGB colour strings.

Author(s)

Danail Obreschkow

#' @examples nplot() rasterImage(rbind(planck.colors(1e3)),0,0,1,1)

pol2car — *Polar/cylindrical to Cartesian coordinate conversion*

Description

Convert polar/cylindrical coordinates to 2D/3D Cartesian coordinates

Usage

```
pol2car(x)
```

Arguments

`x`	2/3-element or n-by-2/3 matrix representing the polar/cylindrical coordinates (r,phi)/(r,phi,z), where phi=0...2*pi is the azimuth measured positively from the x-axis.

Value

Returns a 2/3-element vector or a n-by-2/3 element matrix representing the Cartesian coordinates (x,y)/(x,y,z).

Author(s)

Danail Obreschkow

See Also

car2pol

`progress` *Show progress while timer in running*

Description

Show progress while a timer is running following a call of tick().

Usage

```
progress(txt = NULL)
```

Arguments

`txt`	custom text to be displayed. If not provided, the time since calling tick() is displayed.

Value

None

Author(s)

Danail Obreschkow

See Also

tick tock error

Examples

```
tick('Test')
Sys.sleep(.1)
for (i in seq(3)) {
  progress(i)
  Sys.sleep(.1)
}
tock()
```

quadrupole *Quadrupole tensor*

Description

Compute the trace-free quadrupole tensor of a set of point masses

Usage

```
quadrupole(x, m = 1)
```

Arguments

x	n-by-3 matrix (x[1:n,1:3]) specifying the 3D Cartesian coordinates of n points
m	n-vector with point masses, or single scalar giving a uniform mass for all points (default is unity)

Value

Returns a 3-by-3 symmetric matrix

Author(s)

Danail Obreschkow

See Also

`inertia`, `moments`

quiet *Suppress console output*

Description

Executes routines and command lines and load packages while suppressing the console output.

Usage

```
quiet(x)
```

Arguments

x	routine to be called

Value

Returns whatever the called routine returns in invisible form.

Author(s)

Danail Obreschkow

Examples

```
# Test function
test = function(x) {
  cat('This routine is likes to talk a lot!\n')
  return(x^2)
}

# Standard call call:
y = test(5)
print(y)

# Quiet call:
y = quiet(test(6))
print(y)
```

`rasterflip` *Flip array to be displayed with rasterImage()*

Description

Flips the array A to be displayed with rasterImage, such that the first index runs from left to right and second index runs from bottom to top, like in standard Cartesian coordinates. In this way `rasterImage(rasterflip(A))` has the same orientation as `image(A)`.

Usage

```
rasterflip(A)
```

Arguments

`A`	n-by-m array of a monochromatic image or n-by-m-by-k array of a color image (where k is 3 or 4)

Value

A m-by-n array of a monochromatic image or m-by-n-by-k array of a color image.

Author(s)

Danail Obreschkow

readhdf5 *Read data from an HDF5 file*

Description

Reads data from an HDF5 file, including attributes, into a nested list that preserves the hierarchical structure of the HDF5 data. The routine can read 64-bit integers and represent them as 64-bit integers in R via the `bit64` package. It can also read 32-bit floating points values, not part of native R. These are automatically converted into 64-bit doubles.

Usage

```
readhdf5(file, subtree = "*", group.attr.as.data = FALSE, empty = FALSE)
```

Arguments

`file` Character string specifying the file name of the input HDF5 file.

`subtree` A structure specifying the HDF5 groups and datasets to be read. Use an asterisk `"*"` (default) to read the entire file. To read only part of the file, provide a named list reflecting the hierarchy of groups, subgroups, and datasets. For (sub)groups, use nested lists containing the items to read, or use `'*'` to load everything in the group. An empty list `list()` reads only the attributes of a group. For datasets, use `NULL` to read only attributes, or any other content to read the full data.

`group.attr.as.data` Logical flag. If `TRUE`, group attributes are converted to datasets, which is useful for formats where parameters are stored as attributes (e.g., Gadget simulation outputs).

`empty` Logical flag. If `TRUE`, only names of groups and datasets are returned, with all data equal to NA. This is a fast way of reading the hierarchical structure.

Details

This function, based on the `hdf5r` package, recursively parses and reads HDF5 files into nested lists that preserve the original hierarchy. Attributes in groups and datasets are included in the output.

Value

Nested list representing the contents of the HDF5 file. Groups are nested sublists, datasets are represented by their data. Attributes of groups and datasets are attached as attributes to the corresponding sublists and data elements.

See Also

`writehdf5` for examples.

`rebindensity` *Re-bin density histograms*

Description

Transform density histogram data into histogram data with different bins

Usage

```
rebindensity(x, y, xout)
```

Arguments

`x`	n-vector giving the mid-points of the input histogram bins, must be equally spaced
`y`	n-vector giving the values of the input histogram values
`xout`	m-vector giving the mid-points of the output histogram bins, must be equally spaced

Value

m-vector of y-values associated with the bins specified by xout.

Author(s)

Danail Obreschkow

Examples

```
# original binning
x = seq(0.5,4.5)
y = seq(5)
plot(x,y,xlim=c(-1,6),ylim=c(0,6),pch=16)
lines(histcoord(x,y),lwd=3)

# rebinning
xout = seq(-0.9,6,0.3)
yout = rebindensity(x,y,xout)
points(xout,yout,col='red',pch=16)
lines(histcoord(xout,yout),col='red')
```

`rng` *Random number generator for a custom d-dimensional distribution*

Description

Brute-force algorithm for drawing random numbers from a d-dimensional distribution.

Usage

```
rng(f, n, min, max, fmax = NULL, quasi = FALSE, start = 1, warn = TRUE)
```

Arguments

`f`	function of a d-vector representing a d-dimensional distribution function. This function must be non-negative on the whole domain. It does not need to be normalized. For fast performance, this function should be vectorized, such that it returns an N-element vector if it is given an N-by-D matrix as argument. An automatic warning is produced if the function is not vectorized in this manner.
`n`	number of random numbers to be generated
`min, max`	are d-vectors specifying the domain of distribution function; the domain must be finite and should be as restrictive as possible to keep the number of random trials as low as possible.
`fmax`	maximum value of `f` on its domain. If set to `NULL` (default), this value will be determined automatically, using the `optimize` (if d=1) and `optim` (if d>1) function with its default options. A value for `fmax` should be set, if the automatically determined value (see output list) is incorrect.
`quasi`	logical flag. If true, quasi-random numbers with low-discrepancy are drawn, based on a Halton sequence. Otherwise, the standard internal pseudo-random generator of `runif()` is used.
`start`	starting index of Halton sequence. Only used if `quasi=TRUE`.
`warn`	logical flag. If true, a warning is produced if the function f is not vectorized.

Value

Returns list of items:

`x`	n-by-d matrix of n random d-vectors.
`fmax`	maximum value of the distribution function `f` on the domain.
`n`	number of random vectors (same as argument `n`).
`ntrials`	number of trials.

Author(s)

Danail Obreschkow

See Also

dpqr

Examples

```
## 1D random number generation from a sine-function
f = function(x) sin(x)
out.pseudo = rng(f,1e3,0,pi)
out.quasi = rng(f,1e3,0,pi,quasi=TRUE)
hist(out.pseudo$x,100,freq=FALSE,border=NA,xlab='x',main='sine-distribution')
hist(out.quasi$x,100,freq=FALSE,border=NA,col='#ff000066',add=TRUE)
curve(sin(x)/2,0,pi,add=TRUE)

## 2D quasi-random sampling of a disk with exponentially declining surface density
f = function(x) exp(-sqrt(x[,1]^2+x[,2]^2))
out = rng(f,1e4,min=c(-5,-5),max=c(5,5),quasi=TRUE)
plot(out$x,cex=0.3,pch=16,asp=1,main='Quasi-random exponential disk')

## 5D random number generation (5-dimensional sphere)
f = function(x) as.numeric(sum(x^2)<=1)
out = rng(f,1e4,rep(-1,5),rep(1,5))
cat(sprintf('Number of successes over number of trials : %.4f\n',out$n/out$ntrials))
cat(sprintf('Expected ratio for n=\u221E : %.4f\n',pi^(5/2)/gamma(1+5/2)/2^5))
```

---

rotation2 — *2D rotation matrix*

Description

Compute a 2D rotation matrix given a rotation angle

Usage

```
rotation2(angle)
```

Arguments

angle	rotation angle in radians (counter-clockwise)

Value

Returns a 2-by-2 anti-symmetric rotation matrix

Author(s)

Danail Obreschkow

See Also

`rotation3`

`rotation3` *3D rotation matrix*

Description

Compute a 3D rotation matrix given an axis and an angle

Usage

```
rotation3(u, angle = NULL)
```

Arguments

`u`	3-vector specifying the rotation axis
`angle`	rotation angle in radians; if not given, the norm of the vector `u` is interpreted as the angle

Value

Returns a 3-by-3 rotation matrix

Author(s)

Danail Obreschkow

See Also

`rotation2`

`runif2` *Generate randomly oriented vectors in 2D*

Description

Generate randomly oriented vectors in 2D, following an isotropic distribution (optionally truncated to a region).

Usage

```
runif2(n = 1, r = c(0, 1), azimuth = c(0, 2 * pi), quasi = FALSE, start = 1)
```

Arguments

`n`	number of random vectors to be generated
`r`	2-vector specifying the range of radii
`azimuth`	2-vector specifying the range of azimuth angles
`quasi`	logical flag. If true, quasi-random numbers with low-discrepancy are drawn, based on a Halton sequence. Otherwise, the standard internal pseudo-random generator of `runif()` is used.
`start`	starting index of Halton sequence. Only used if `quasi=TRUE`.

Value

Returns an n-by-2 array of n vectors.

Author(s)

Danail Obreschkow

See Also

`runif3`

Examples

```
## generate 500 unit vectors with radii between 0.5 and 1
x = runif2(500,r=c(0.5,1))
oldpar = par(pty='s')
plot(x,pch=20)
par(oldpar)
```

`runif3` *Generate randomly oriented vectors in 3D*

Description

Generate randomly oriented vectors in 3D, following an isotropic distribution (optionally truncated to a region).

Usage

```
runif3(
  n = 1,
  r = c(0, 1),
  azimuth = c(0, 2 * pi),
  polarangle = c(0, pi),
  quasi = FALSE,
  start = 1
)
```

Arguments

`n`	number of random vectors to be generated
`r`	2-vector specifying the range of radii
`azimuth`	2-vector specifying the range of azimuth angles (maximum range 0..2*pi)
`polarangle`	2-vector specifying the range of polar angles (maximum range 0..pi)
`quasi`	logical flag. If true, quasi-random numbers with low-discrepancy are drawn, based on a Halton sequence. Otherwise, the standard internal pseudo-random generator of `runif()` is used.
`start`	starting index of Halton sequence. Only used if `quasi=TRUE`.

Value

Returns an n-by-3 array of n vectors.

Author(s)

Danail Obreschkow

See Also

`runif2`

Examples

```
## draw 20 unit vectors on a sphere
x = runif3(20,r=c(1,1))
print(rowSums(x^2))
```

`scalarproduct` *Scalar product*

Description

Compute scalar product of two vectors

Usage

```
scalarproduct(x, y)
```

Arguments

`x, y`	d-element vectors or n-by-d matrices representing n d-element vectors

Value

Returns a scalar or a n-element vector with the scalar products.

Author(s)

Danail Obreschkow

See Also

`vectorproduct`

`sigmoid` *Sigmoid function*

Description

Returns S-shaped functions, monotonically increasing between two constants with a point symmetry at the half-way point.

Usage

```
sigmoid(x, type = 1, fmin = 0, fmax = 1, xmid = 0, slope = 1)
```

Arguments

`x`	real argument between -Inf and +Inf, can be a vector

`type`	number or character string, specifying the type of function. The following expressions vary between -1 and +1 with a slope of 1 at the symmetry x=0.
	`tanh` (1): hyperbolic tangent function, `tanh(x)`, which becomes the logistic function, `1/(1+exp(-x))`, if all arguments kept at default.
	`atan` (2): arctangent function, `2/pi*atan(pi/2*x)`
	`err` (3): error function, `erf(sqrt(pi)/2*x)`
	`sqrt` (4): algebraic function with root, `x/sqrt(1+x^2)`
	`abs` (5): algebraic function with absolute value, `x/(1+abs(x))`
	`gd` (6): Gudermannian function, `4/pi*atan(tanh(pi/4*x))`

`fmin`	asymptotic function minimum at `x=-Inf`
`fmax`	asymptotic function maximum at `x=+Inf`
`xmid`	point of symmetry, where the function value is `(fmin+fmax)/2`
`slope`	slope of the function at `x=xmid`

Value

Returns a vector with the same number of elements as `x`

Author(s)

Danail Obreschkow

Examples

```
col = c('red','orange','#cccc00','#00bb00','blue','purple')
for (i in seq(6)) {
  graphics::curve(sigmoid(x,i),-3,3,add=i>1,col=col[i],ylab='sigmoid(x)')
}
graphics::lines(c(-1.5,1.5),c(-1,2),lty=2)
```

---

smartround *Round a vector of floating-point values while preserving their sum*

Description

Rounds the values in a vector up and down, preserving the sum of the vector and minimizing the total rounding error under this condition. An example where this is useful is when rounding a vector of percentages, where the total should add up to 100 percent.

Usage

```
smartround(x, digits = 0)
```

Arguments

`x`	vector of values
`digits`	optional non-negative integer specifying the number of digits of the rounded numbers

Value

Returns a vector of rounded values of the same length as `x`.

Author(s)

Danail Obreschkow

Examples

```
x = runif(5)
x = x/sum(x)*100
print(x)
print(sum(x))
y = smartround(x)
print(y)
print(sum(y))
y2 = smartround(x,2)
print(y2)
print(sum(y2))
```

`smoothcontour` *Draw smoothed contours*

Description

Draw smoothed iso-countours for a density field. The contours are computed using the `contourLines` routine and smoothed using the `smooth.spline` function. Both open and closed contour lines are handled correctly.

Usage

```
smoothcontour(
  x = seq(0, 1, length.out = nrow(z)),
  y = seq(0, 1, length.out = ncol(z)),
  z,
  levels,
  smoothing = 0.5,
  min.radius = 1,
  lwd = 1,
  lty = 1,
  col = "black",
  ...
)
```

Arguments

`x, y`	vectors containing the locations of grid lines at which the values of z are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used.
`z`	matrix representing the density field on which the contours are plotted.
`levels`	vector of the iso-contour levels.
`smoothing`	value between 0 and 1 specifying the degree of smoothing.
`min.radius`	numerical value. If larger than 0, all contours with a mean radius (in pixels) below `min.radius` are removed.
`lwd`	vector of line widths (see `par`)
`lty`	vector of line types (see `par`)
`col`	vector of colors (see `par`)
`...`	additional parameters to be passed to the function `lines`.

Value

None

Author(s)

Danail Obreschkow

See Also

contourLines, smooth.spline

Examples

```
set.seed(1)
f = function(x) cos(2*x[1]-x[2]-1)^2*exp(-x[1]^2-x[2]^2-x[1]*x[2])
x = seq(-3,3,length=100)
m = pracma::meshgrid(x)
z = array(Vectorize(function(x,y) f(c(x,y)))(m$Y,m$X)+rnorm(4e4,sd=0.1),dim(m$X))
image(x,x,z,col=terrain.colors(100))
contour(x,x,z,levels=c(0.2,0.5),add=TRUE)
smoothcontour(x,x,z,levels=c(0.2,0.5),lwd=3,smoothing=0.8,min.radius=2)
```

---

smoothfun *Smoothed Function*

Description

Generates a cubic smoothed spline function y=f(x) approximating supplied (x,y)-data with a custom number of degrees of freedom. The routines builds on smooth.spline, but directly returns the smoothed function rather than the fitted model.

Usage

```
smoothfun(x, y = NULL, w = NULL, df = NULL, ...)
```

Arguments

x	a vector giving the values of the predictor variable, or a list or a two-column matrix specifying x and y.
y	responses. If y is missing or NULL, the responses are assumed to be specified by x, with x the index vector.
w	optional vector of weights of the same length as x; defaults to all 1.
df	the desired equivalent number of degrees of freedom. Must be in [2,nx], where nx is the number of unique x values. If not given, nx is set to the square root of the number of unique x-values.
...	additional optional arguments used by smooth.spline.

Value

Returns a fast and vectorized smoothed function f(x).

Author(s)

Danail Obreschkow

See Also

smooth.spline

Examples

```
# make random data set
set.seed(1)
x = runif(100)
y = sin(2*pi*x)+rnorm(100, sd=0.5)
plot(x,y,pch=16)

# smoothed spline
f = smoothfun(x, y)
curve(f, add=TRUE, col='red')

# smoothed spline with custom degree of freedom
g = smoothfun(x, y, df=5)
curve(g, add=TRUE, col='blue')
```

---

sortlist *Sort List Entries Alphabetically*

Description

This function sorts the entries of a structured list in alphabetical order, including all nested sublists recursively, making it easier to navigate nested list structures and compare different lists. Unnammed items are listed last. Attributes of the original list are preserved.

Usage

```
sortlist(lst, recursive = TRUE, convert.data.frames = FALSE)
```

Arguments

`lst`	list to be sorted.
`recursive`	logical flag indicating whether to apply sorting recursively to all sublists. Defaults to `TRUE`. If `FALSE`, only the top-level entries in the list will be sorted.
`convert.data.frames`	logical flag. If `TRUE`, data frames are converted to lists.

Value

Structured list with alphabetically sorted entries. If `recursive = TRUE`, all sublists are also sorted alphabetically. If the list contains unnamed elements, they are left unsorted.

Examples

```
# Create a nested list
my_list = list(
b = list(d = 4, a = 2),
empty = list(),
'unnamed',
a = 1,
f = list(b = 3, a = list(d = seq(5), 'unnamed', c = 2))
)

# Sort the list recursively
sorted_list_recursive = sortlist(my_list)
print(sorted_list_recursive)
```

---

spectrumcolors *Spectrum colour palette*

Description

Generates smooth rainbow color scale from red to purple, similar to rainbow, but with improved smoothness

Usage

```
spectrumcolors(n, alpha = 1, rev = FALSE)
```

Arguments

n	integer number of colors in the scale
alpha	alpha transparency value (0=fully transparent, 1=fully opaque)
rev	logical flag indicating whether the ordering of the colors should be reversed

Value

Returns an n-vector of RGB colour strings.

Author(s)

Danail Obreschkow

#' @examples nplot() rasterImage(rbind(spectrumcolors(1e3)),0,0,1,0.5) rasterImage(rbind(rainbow(1e3,end=5/6)),0,0.5,1,1) text(0.5,0.25,'spectrum') text(0.5,0.75,'rainbow') abline(h=0.5)

sph2car — *Spherical to Cartesian coordinate conversion*

Description

Convert 3D spherical to Cartesian coordinates

Usage

```
sph2car(x)
```

Arguments

x	3-element or n-by-3 matrix representing the spherical components (r,theta,phi) of n three-dimensional vectors. Here, theta=0...pi is the polar angle measured from the north pole and phi=0...2*pi is the azimuth measured positively from the x-axis (ISO 80000-2:2019 physics convention).

Value

Returns a 3-element vector or a n-by-3 element matrix representing the Cartesian coordinates (x,y,z)

Author(s)

Danail Obreschkow

See Also

car2sph

sphereplot — *Plot a spherical function or point set*

Description

Plots a spherical function or a point set in a 2D projection using only standard R graphics. This avoids compatibility issues of rgl, e.g. knitting markdown documents.

Usage

```
sphereplot(
  f = NULL,
  n = 100,
  theta0 = pi/2,
  phi0 = 0,
  angle = 0,
  projection = "globe",
  col = gray.colors(256, 0, 1),
  clim = NULL,
  add = FALSE,
  center = c(0, 0),
  radius = 1,
  nv = 500,
  show.border = TRUE,
  show.grid = TRUE,
  grid.phi = seq(0, 330, 30)/180 * pi,
  grid.theta = seq(30, 150, 30)/180 * pi,
  pch = 16,
  pt.col = "black",
  pt.cex = 0.5,
  lwd = 0.5,
  lty = 1,
  line.col = "black",
  background = "white",
  ...
)
```

Arguments

`f`	must be either of: (1) NULL to plot just grid without spherical function (2) a vectorized real function f(theta,phi) of the polar angle theta [0,pi] and azimuth angle [0,2pi] (3) an n-by-2 array of values theta and phi
`n`	number of grid cells in each dimension used in the plot
`theta0`	polar angle in radians at the center of the projection
`phi0`	azimuth angle in radians at the center of the projection
`angle`	angle in radians between vertical axis and central longitudinal great circle
`projection`	type of projection: "globe" (default), "cylindrical", "mollweide"
`col`	color map
`clim`	2-element vector specifying the values of f corresponding to the first and last color in col
`add`	logical flag specifying whether the sphere is to be added to an existing plot
`center`	center of the sphere on the plot

radius	radius of the sphere on the plot
nv	number or vertices used for grid lines and border
show.border	logical flag specifying whether to show a circular border around the sphere
show.grid	logical flag specifying whether to show grid lines
grid.phi	vector of phi-values of the longitudinal grid lines
grid.theta	vector of theta-values of the latitudinal grid lines
pch	point type
pt.col	point color
pt.cex	point size
lwd	line width of grid lines and border
lty	line type of grid lines and border
line.col	color of grid lines and border
background	background color
...	additional arguments to be passed to the function f

Value

Returns a list containing the vector `col` of colors and 2-vector `clim` of values corresponding to the first and last color.

Author(s)

Danail Obreschkow

Examples

```
## Plot random points on the unit sphere in Mollweide projection
set.seed(1)
f = cbind(acos(runif(5000,-1,1)),runif(5000,0,2*pi))
sphereplot(f,theta0=pi/3,projection='mollweide',pt.col='red')

## Plot real spherical harmonics up to third degree
oldpar = par(mar=c(0,0,0,0))
nplot(xlim=c(-4,3.5),ylim=c(0,4),asp=1)
for (l in seq(0,3)) { # degree of spherical harmonic
  for (m in seq(-l,l)) { # order of spherical harmonic

    # make spherical harmonic function in real-valued convention
    f = function(theta,phi) sphericalharmonics(l,m,cbind(theta,phi))

    # plot spherical harmonic
   sphereplot(f, 50, col=planckcolors(100), phi0=0.1, theta0=pi/3, add=TRUE, clim=c(-0.7,0.7),
               center=c(m,3.5-l), radius=0.47)

    if (l==3) text(m,-0.15,sprintf('m=%+d',m))
  }
  text(-l-0.55,3.5-l,sprintf('l=%d',l),pos=2)
```

```
}
par(oldpar)
```

---

sphericalharmonics *Spherical Harmonics*

Description

Evaluates spherical harmonics Y, either in the real-valued or complex-valued basis.

Usage

```
sphericalharmonics(l, m, x, basis = "real")
```

Arguments

`l`	degree of the spherical harmonic, accurate to about l=500; (0=monopole, 1=dipole, 2=quadrupole, 3=octupole, 4=hexadecapole,...)
`m`	order of the spherical harmonic (-l,-l+1,...,+l)
`x`	either an n-by-2 matrix specifying the polar angle `theta` (0...pi) and azimuthal angle `phi` (0...2*pi); or an n-by-3 matrix specifying the 3D coordinates of n vectors (whose normalization is irrelevant).
`basis`	a string specifying the type of spherical harmonics; this has to be either "complex" for the standard complex-valued harmonics with Condon-Shortley phase convention, or "real" (default) for the standard real-valued harmonics.

Value

Returns an n-vector of the spherical harmonics; for points x=c(0,0,0), a value of 0 is returned

Author(s)

Danail Obreschkow

Examples

```
## Check orthonormalization of all spherical harmonics up to 3rd degree

# make indices l and m up to 3rd degree
l = c(0,rep(1,3),rep(2,5),rep(3,7))
m = c(0,seq(-1,1),seq(-2,2),seq(-3,3))

# check orthonormalization for all pairs
for (i in seq(16)) {
  for (j in seq(16)) {

    # compute scalar product
```

```
      f = function(theta,phi) {
        Yi = sphericalharmonics(l[i],m[i],cbind(theta,phi))
        Yj = sphericalharmonics(l[j],m[j],cbind(theta,phi))
        return(Re(Yi*Conj(Yj))*sin(theta))
      }
      g = Vectorize(function(phi) integrate(f,0,pi,phi)$value)
      scalar.product = integrate(g,0,2*pi)$value

      # compare scalar product to expected value
      ok = abs(scalar.product-(i==j))<1e-6
      cat(sprintf('(l=%1d,m=%+1d|l=%1d,m=%+1d)=%5.3f  %s\n',l[i],m[i],l[j],m[j],
                  scalar.product+1e-10,ifelse(ok,'ok','wrong')))
    }
  }
```

`stretch` *Stretch values to a custom range*

Description

Shifts and stretches the values of a vector or array to a desired interval, while maintaining the shape of the input argument

Usage

```
stretch(x, min = 0, max = 1, invert = FALSE, gamma = NULL, na = NULL)
```

Arguments

`x`	vector or array
`min`	minimum value
`max`	maximum value
`invert`	logical flag specifying whether the data should be inverted, such that the smallest input value maps to max and the largest input value maps to min.
`gamma`	optional argument specifying a non-linear transformation x->x^gamma, if gamma>0, or x->1-(1-x)^(-gamma), if gamma<0.
`na`	optional value specifying the value assigned to non-numbers (NA and NaN)

Value

vector/array of the same shape as `x`

Author(s)

Danail Obreschkow

See Also

`lim`

`subplot` *Insert a sub-panel into plot*

Description

Insert a sub-panel into an existing plotting area. To open a subplot, call `par(subplot(...))`, to close it, you must call `par(subplot('off'))`.

Usage

```
subplot(xleft = NA, ybottom, xright, ytop)
```

Arguments

`xleft`	lower x-coordinate of the sub-panel relative to the current plot.
`ybottom`	lower y-coordinate of the sub-panel relative to the current plot.
`xright`	upper x-coordinate of the sub-panel relative to the current plot.
`ytop`	upper y-coordinate of the sub-panel relative to the current plot.

Value

graphical parameters to user with `par`.

Author(s)

Danail Obreschkow

Examples

```
# main plot
f = function(x) x*sin(1/(x+.Machine$double.eps))
curve(f,0,1,n=1000,ylim=c(-1,1),xlab='',ylab='',xaxs='i',yaxs='i')
rect(0.02,-0.1,0.1,0.1,border='blue',col=transparent('blue',0.2))

# subplot
par(subplot(0.55,-0.8,0.93,0))
curve(f,0.02,0.1,n=500,ylim=c(-0.1,0.1),xlab='',ylab='',xaxs='i',yaxs='i')
rect(0.02,-0.1,0.1,0.1,border=NA,col= transparent('blue',0.2))
par(subplot('off'))
```

tick — *Start timer*

Description

Start timer and write a custom text into the console.

Usage

```
tick(txt = "Start")
```

Arguments

txt	custom text to be displayed.

Value

None

Author(s)

Danail Obreschkow

See Also

tock progress error

Examples

```
tick('Sum 10 million random numbers')
x = sum(runif(1e7))
tock()
```

tock — *Stop timer*

Description

Stop timer and write the computation in seconds since the last call of tick().

Usage

```
tock(txt = "", fmt = " (%.2fs). %s\n")
```

Arguments

`txt`	custom text to be displayed.
`fmt`	character vector of format strings. It must contain exactly one %s as placeholder for `txt` and one numerical format, such as %f or %e, as placeholder for the time.

Value

Returns the elapsed time in seconds since calling tick().

Author(s)

Danail Obreschkow

See Also

`tick` `progress` `error`

Examples

```
tick('Sum 10 million random numbers')
x = sum(runif(1e7))
tock()
```

`transparent` *Add transparency to a color*

Description

Add transparency to a color

Usage

```
transparent(col, alpha = 0.5)
```

Arguments

`col`	is a color or vector/array of colors, specified as text (e.g. 'purple') or 7/9-character (e.g. '#A020F0')
`alpha`	transparency value between 0 and 1 or a vector/array thereof

Value

Returns a 9-character color or vector/array of 9-character colors.

Author(s)

Danail Obreschkow

See Also

lightness

Examples

```
# Add different transparencies of the same color
plot(runif(50),runif(50),pch=20,cex=10,col=transparent('purple',runif(50)))

# Add the same transparency to different colors
plot(runif(50),runif(50),pch=20,cex=10,col=transparent(rainbow(50)))
```

---

transzoom *Zoom, translate and rotate array image*

Description

Zoom/rotate/translate an image relative to its center for images represented as simple arrays.

Usage

```
transzoom(
  img = NULL,
  zoom = 1,
  shift = c(0, 0),
  angle = 0,
  size = NULL,
  col = "black",
  filter = "bilinear",
  file.in = "",
  file.out = "",
  format = "png",
  show.image = TRUE
)
```

Arguments

img	n-by-m-by-3 array or n-by-m-by-4 array representing an rgb(+alpha) image.
zoom	zoom factor (>0).
shift	2-vector specifying the vertical+horizontal translation in units of output pixels (i.e. after zooming).
angle	rotation angle in degrees.
size	2-vector specifying the vertical+horizontal dimensions of the output image. If not given, this is taken to be identical to the input image.
col	background color

`filter`	affine transformation filter; either 'none' or 'bilinear'
`file.in`	optional input filename, which can be used to load an image instead of providing it via `img`. This filename is ignored if `img` is specified.
`file.out`	optional output filename.
`format`	one of "png" or "jpg" specifying the file format of the input and output image.
`show.image`	logical flag specifying whether the image is displayed in the R console.

Value

Returns an n-by-m-by-3 array or n-by-m-by-4 array of the processed image.

Author(s)

Danail Obreschkow

Examples

```
img = yinyangyong # this is an example image included in the package
transzoom(img, zoom=2) # zoom by a factor 2
```

`uniquedouble` *Map positive 64-bit integers onto unique doubles*

Description

Uses a reversible one-to-one mapping to converts non-negative 64-bit integers, produced via the `bit64` package, to double-precision floating-point values for faster handling of 64-bit integer indices, e.g. comparison of indices, count number of unique indices, etc. The double values are completely different from the corresponding integer values.

Usage

```
uniquedouble(x, inverse = FALSE)
```

Arguments

`x`	If inverse=`FALSE` (default), this is a scalar/vector/array of non-negative integers, typically 64-bit integers. If inverse=`TRUE`, it is a scalar/vector/array of doubles.
`inverse`	Logical flag. If `TRUE`, doubles are converted back to their original 64-bit integers.

Value

Scalar/vector/array of double values, if inverse=`FALSE`, or integer64 values, if inverse=`TRUE`.

Author(s)

Danail Obreschkow

Examples

```
# produce example vector of non-negative 64-bit integers
n = 1e3
x = c(bit64::as.integer64(0:n),
      bit64::as.integer64('9218868437227405311')+c(-n:n),
      bit64::lim.integer64()[2]-c(0:n))

# map this vector onto unique doubles
y = uniquedouble(x)

# check if the double-vector can be inverted back to the integer-vector
cat(sprintf("Check inversion = %s.\n", all(x==uniquedouble(y, inverse=TRUE))))

# measure the time taken to compare all the int64 values
tick("Compare original integer64 values. ")
comparison = TRUE
for(i in seq_along(x)) comparison = comparison & x[i]==x[i]
tock(comparison, fmt="Time taken = %.3fs. Test result = %s.\n")

# measure the time taken to compare all the corresponding double values
tick("Compare corresponding double values. ")
comparison = TRUE
for(i in seq_along(y)) comparison = comparison & y[i]==y[i]
tock(comparison, fmt="Time taken = %.3fs. Test result = %s.\n")
```

`unitvector` *Normalize vectors to unit length*

Description

Compute the unit vectors for for a matrix of non-normalised vectors

Usage

```
unitvector(x)
```

Arguments

x	m-element vector or n-by-m matrix `x[1:n,1:m]` representing n m-element vectors

Value

Returns a data type identical to `x`, but with normalised vectors. Zero-vectors are returned for vectors of length zero.

Author(s)

Danail Obreschkow

See Also

vectornorm

---

userattributes *Retrieve Custom Object Attributes*

Description

This function takes a generic R object and returns the custom attributes added by the user, distinguishing them from the object's inherent attributes.

Usage

```
userattributes(obj)
```

Arguments

obj	R object whose custom attributes to extract.

Value

Named list of custom attributes, mimicking the output of the base function attributes, but only including custom attributes

See Also

attributes

Examples

```
# Create a list containing objects of different classes
example = list(1,
               as.integer(1:3),
               bit64::as.integer64(1),
               diag(5),
               array(0,c(2,3,5)),
               list(a=1,b='c'),
               data.frame(x=1, y=1:10, char=LETTERS[1:10])
)

# Loop over all the objects to check if the custom attribute is correctly extracted.
for (i in seq_along(example)) {
  x = example[[i]]
  attr(x,'MyAttribute') = 'Test'
    cat(sprintf('Class = %s, Custom attribute(s) = %s\n',class(x)[1],
                paste(names(userattributes(x)), collapse = ', ')))
}
```

vectornorm *Vector norm*

Description

Compute the norm of a vector or a matrix of vectors

Usage

```
vectornorm(x)
```

Arguments

`x`	m-element vector or n-by-m matrix (x[1:n,1:m]) representing n m-element vectors

Value

Returns a scalar or an n-element vector with the vector norms

Author(s)

Danail Obreschkow

See Also

`vectornorm`

vectorproduct *Vector product*

Description

Compute cross product of two 3-element vectors

Usage

```
vectorproduct(x, y, normalize = FALSE)
```

Arguments

`x, y`	3-element vectors or n-by-3 matrices representing n 3-element vectors
`normalize`	logical flag; if set to TRUE the cross-product(s) is/are automatically normalized

Value

Returns a 3-element vector or a n-by-3 element matrix with the cross products

Author(s)

Danail Obreschkow

See Also

scalarproduct, vectornorm and unitvector

---

wavelength2col *Convert wavelength to RGB*

Description

Converts a given wavelength of light to an approximate RGB color value, using black in the invisible range.

Usage

```
wavelength2col(wavelength)
```

Arguments

`wavelength`	wavelength value (or vector), in nanometers.

Value

Returns a color string or vector of color strings with the same number of elements as `wavelength`.

Author(s)

Danail Obreschkow

Source

Smoothed implementation of the original Fortran version by Dan Bruton (http://www.physics.sfasu.edu/astro/color/spectra.htm) and the R-function by Michael Friendly (https://gist.github.com/friendly).

Examples

```
lambda = seq(300,800)
col = matrix(wavelength2col(lambda),nrow=1)
plot(NA,xlim=range(lambda),ylim=c(0,1),xaxs='i',xlab='wavelength [nm]',yaxs='i',yaxt='n',ylab='')
rasterImage(col,min(lambda),0,max(lambda),1)
```

writehdf5 *Write structured list to an HDF5 file*

Description

Writes a named list, including its hierarchy of nested sublists, to an HDF5 file with HDF5 groups, subgroups and datasets preserving the hierarchical structure. The routine supports standard HDF5 data types, including double-precision floating point numbers (64 bit), integers (32 bit), characters (8 bit), and booleans, as well as vectors and arrays of these types. It also supports 64-bit integers (`H5T_NATIVE_INT64`), available via the `bit64` package, as well as 32-bit floating points, available via the `float` package. Custom attributes of sublists and data, assigned in R via `attr`, are automatically transcribed to group and dataset attributes in the HDF5 file. Such attributes can also be provided for empty groups (produced by `list()`) and datasets (produced by `numeric(0)`).

Usage

```
writehdf5(obj, file, inherent.attributes = FALSE, level = 6, overwrite = TRUE)
```

Arguments

`obj`	List containing the data to be written. Nested sublists are interpreted as subgroups within the HDF5 file. If a sublist is empty or if an element inside a list is 'NULL', it creates an empty group/dataset that can hold only attributes.
`file`	Character string specifying the file name of the output HDF5 file.
`inherent.attributes`	Logical flag indicating whether to include inherent attributes that some R-objects possess, such as 'dim' for matrices or 'names' for named vectors and arrays.
`level`	Integer specifying the compression level for datasets, typically between 0 (no compression) and 9 (maximum compression). Not all dataset types support compression.
`overwrite`	Logical value indicating whether to overwrite an existing file.

Details

The function relies on the `hdf5r` package and on the `bit64` package.

The nested list `obj` should only contain data types available to HDF5. The only exception are data frames, which, if included, are automatically converted to lists. Normally, the list `obj` and its nested sublists, should all be named lists, i.e. `list(a=1, b=2)` rather than `list(1,2)`. Unnamed elements are automatically assigned a name 'unnamed_#' in the HDF5 file.

Some data types in R have inherent attributes, such as 'names' for data frames and 'dim' for arrays. By default, these inherent attributes are not written to the HDF5 file. They are, however, automatically regenerated when the HDF5 file is loaded back via `readhdf5`. The argument `inherent.attributes` can be used to force writing all attributes, including the inherent ones, to the HDF5 file.

If a structured R list is saved with `writehdf5` and then reloaded using `readhdf5`, the recovered list is identical to the input list up to the ordering of list elements, which is alphabetic by default in HDF5. The only other difference between the input and recovered data occurs when the input data contain data frames, as these are automatically converted do lists. A workaround is to set `inherent.attributes=TRUE` when writing the HDF5 file. In this case, the reloaded HDF5 data becomes a data frame again.

Value

None

See Also

`readhdf5`

Examples

```
# Create example data
input = list(
  group_empty = list(),
  dataset_empty = numeric(0),
  group_dataframe = data.frame(
    ID = as.integer(1:3),
    Name = c("Alice", "Bob", "Charlie"),
    Salary = c(2341.2, 3534.2, 4541.9),
    Employed = c(TRUE, TRUE, FALSE)
  ),
  dataset_parameters = 1.234,
  group_integers = list(int32 = as.integer(123),
                      int64 = bit64::as.integer64(123),
                      vector = bit64::as.integer64((1:3)+12345678912345)),
  dataset_nonnumeric = c(NA, NaN, Inf, -Inf),
  group_mixed = list(
    header = 'test header',
    subgroup1 = list(
      dataset1 = c("A", "%}~&^", "x1y2z3", "", " "),
      dataset2 = matrix(as.integer(1:10), nrow = 2),
      dataset3 = array(runif(30), dim = c(2, 5, 3))
    ),
    subgroup2 = list(date = as.character(as.Date("2025-01-01")),
                   location = 'Perth')
  )
)

# Add attributes to some datasets
attr(input$dataset_empty,'Comment') = 'This is a test file.'
attr(input$group_mixed$subgroup1$dataset3,'Type') = '3D array'
attr(input$group_integers$vector,'Comment') = 'Vector of long integers'
attr(input$group_integers$vector,'Package') = 'bit64'

# Add attributes to some groups
```

```
attr(input$group_dataframe,'Company branch') = 'Sales'
attr(input$group_integers,'Comment') = 'Testing different integers'
attr(input$group_empty,'Timestamp') = date()
attr(input$group_empty,'Working directory') = getwd()

# Write list to HDF5 file
filename = tempfile()
writehdf5(input, filename)

# Read HDF5 file into a new list
output = readhdf5(filename)

# Check if input and output lists are identical
# (up to alphabetic ordering and data frame-to-list conversion)
print(all.equal(sortlist(input, convert.data.frames = TRUE), output))

# Write list to HDF5 file again, this time with inherent attributes, allowing
# to keep track of the data frames
filename = tempfile()
writehdf5(input, filename, inherent.attributes = TRUE)

# Read HDF5 file into a new list
output = readhdf5(filename)

# Check if input and output lists are identical
# (up to alphabetic ordering only)
print(all.equal(sortlist(input), output))
```

`yinyangyong` *Yin-Yang-Yong image*

Description

A data set containing a 500-by-500-by-4 array, representing an image of a Yin-Yang-Yong symbol with transparency. This symbol was 'reinvented' many times. The particular version in this package is the one drawn by the package author Danail Obreschkow in 1999.

Usage

```
yinyangyong
```

Format

An object of class `array` of dimension 500 x 500 x 4.

Value

None

Author(s)

Danail Obreschkow

Examples

```
nplot(asp=1)
rasterImage(yinyangyong,0,0,1,1)
```

Index