

Package ‘ravetools’

June 2, 2024

Type Package

Title Signal and Image Processing Toolbox for Analyzing Intracranial Electroencephalography Data

Version 0.1.6

Language en-US

Description Implemented fast and memory-efficient Notch-filter, Welch-periodogram, discrete wavelet spectrogram for minutes of high-resolution signals, fast 3D convolution, image registration, 3D mesh manipulation; providing fundamental toolbox for intracranial Electroencephalography (iEEG) pipelines. Documentation and examples about 'RAVE' project are provided at <https://openwetware.org/wiki/RAVE>, and the paper by John F. Magnotti, Zhengjia Wang, Michael S. Beauchamp (2020) [doi:10.1016/j.neuroimage.2020.117341](https://doi.org/10.1016/j.neuroimage.2020.117341); see 'citation("ravetools")' for details.

BugReports <https://github.com/dipterix/ravetools/issues>

URL <https://dipterix.org/ravetools/>

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.1

Depends R (>= 4.0.0)

SystemRequirements fftw3 (libfftw3-dev (deb), or fftw-devel (rpm)), pkg-config

Copyright Karim Rahim (author of R package 'fftwtools', licensed under 'GPL-2' or later) is the original author of 'src/ffts.h' and 'src/ffts.cpp'. Prerau's Lab wrote the original 'R/multitaper.R', licensed under 'MIT'. Marcus Geelnard wrote the source code of 'TinyThread' library ('MIT' license) located at 'inst/include/tthread'. Stefan Schlager wrote the original code that converts R objects to 'vcg' (see 'src/vcgCommon.h', licensed under 'GPL-2' or later). Visual Computing Lab is the copyright holder of 'vcglib' source code (see 'src/vcglib', licensed under GPL-2 or later).

Imports graphics, stats, filearray ($\geq 0.1.3$), Rcpp, waveslim ($\geq 1.8.2$), signal ($\geq 0.7.7$), pracma, digest ($\geq 0.6.29$), splines, RNiftyReg ($\geq 2.7.1$), R6 ($\geq 2.5.1$)

LinkingTo Rcpp, RcppEigen

Suggests fftwtools, bit64, grDevices, microbenchmark, freesurferformats, testthat

LazyData true

NeedsCompilation yes

Author Zhengjia Wang [aut, cre],
 John Magnotti [aut],
 Michael Beauchamp [aut],
 Trustees of the University of Pennsylvania [cph] (All files in this package unless explicitly stated in the file or listed in the 'Copyright' section below.),
 Karim Rahim [cph, ctb] (Contributed to src/ffts.h and stc/ffts.cpp),
 Thomas Possidente [cph, ctb] (Contributed to R/multitaper.R),
 Michael Prerau [cph, ctb] (Contributed to R/multitaper.R),
 Marcus Geelnard [ctb, cph] (TinyThread library, tinythreadpp.bitsnbites.eu, located at inst/include/tthread/),
 Stefan Schlager [ctb, cph] (R-vcg interface, located at src/vcgCommon.h),
 Visual Computing Lab, ISTI [ctb, cph] (Copyright holder of vcglib, located at src/vcglib/)

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2024-06-02 09:40:07 UTC

R topics documented:

band_pass	3
baseline_array	5
collapse	8
convolve	10
decimate	11
detrend	12
diagnose_channel	13
fast_cov	15
fast_quantile	16
fill_surface	18
filter_signal	19
filtfilt	20
grow_volume	21
internal_rave_function	22
interpolate_stimulation	22
left_hippocampus_mask	24

matlab_palette	24
mesh_from_volume	24
multitaper	26
new_matrix4	28
new_quaternion	28
new_vector3	29
notch_filter	30
parallel-options	31
plot_signals	32
pwelch	33
raw-to-sexp	36
register_volume	38
rgl-call	40
shift_array	41
vcg_isosurface	42
vcg_mesh_volume	43
vcg_smooth	44
vcg_sphere	46
vcg_uniform_remesh	47
vcg_update_normals	48
wavelet	49

Index	52
--------------	-----------

band_pass	<i>Band-pass signals</i>
-----------	--------------------------

Description

Band-pass signals

Usage

```
band_pass1(x, sample_rate, lb, ub, domain = 1, ...)
```

```
band_pass2(
  x,
  sample_rate,
  lb,
  ub,
  order,
  method = c("fir", "butter"),
  direction = c("both", "forward", "backward"),
  window = "hamming",
  ...
)
```

Arguments

x	input signals, numeric vector or matrix. x must be row-major if input is a matrix: each row is a channel, and each column is a time-point.
sample_rate	sampling frequency
lb	lower frequency bound of the band-passing filter, must be positive
ub	upper frequency bound of the band-passing filter, must be greater than the lower bound and smaller than the half of sampling frequency
domain	1 if x is in time-domain, or 0 if x is in frequency domain
...	ignored
order	the order of the filter, must be positive integer and be less than one-third of the sample rate
method	filter type, choices are 'fir' and 'butter'
direction	filter direction, choices are 'forward', 'backward', and 'both' directions
window	window type, can be a character, a function, or a vector. For character, window is a function name in the signal package, for example, 'hanning'; for a function, window takes one integer argument and returns a numeric vector with length of that input; for vectors, window is a numeric vector of length order+1.

Value

Filtered signals, vector if x is a vector, or matrix of the same dimension as x

Examples

```
t <- seq(0, 1, by = 0.0005)
x <- sin(t * 0.4 * pi) + sin(t * 4 * pi) + 2 * sin(t * 120 * pi)

oldpar <- par(mfrow = c(2, 2), mar = c(3.1, 2.1, 3.1, 0.1))
# ---- Using band_pass1 -----

y1 <- band_pass1(x, 2000, 0.1, 1)
y2 <- band_pass1(x, 2000, 1, 5)
y3 <- band_pass1(x, 2000, 10, 80)

plot(t, x, type = 'l', xlab = "Time", ylab = "",
      main = "Mixture of 0.2, 2, and 60Hz")
lines(t, y1, col = 'red')
lines(t, y2, col = 'blue')
lines(t, y3, col = 'green')
legend(
  "topleft", c("Input", "Pass: 0.1-1Hz", "Pass 1-5Hz", "Pass 10-80Hz"),
  col = c(par("fg"), "red", "blue", "green"), lty = 1,
  cex = 0.6
)

# plot pwelch
```

```

pwelch(x, fs = 2000, window = 4000, noverlap = 2000, plot = 1)
pwelch(y1, fs = 2000, window = 4000, noverlap = 2000,
       plot = 2, col = "red")
pwelch(y2, fs = 2000, window = 4000, noverlap = 2000,
       plot = 2, col = "blue")
pwelch(y3, fs = 2000, window = 4000, noverlap = 2000,
       plot = 2, col = "green")

# ---- Using band_pass2 with FIR filters -----

order <- floor(2000 / 3)
z1 <- band_pass2(x, 2000, 0.1, 1, method = "fir", order = order)
z2 <- band_pass2(x, 2000, 1, 5, method = "fir", order = order)
z3 <- band_pass2(x, 2000, 10, 80, method = "fir", order = order)

plot(t, x, type = 'l', xlab = "Time", ylab = "",
     main = "Mixture of 0.2, 2, and 60Hz")
lines(t, z1, col = 'red')
lines(t, z2, col = 'blue')
lines(t, z3, col = 'green')
legend(
  "topleft", c("Input", "Pass: 0.1-1Hz", "Pass 1-5Hz", "Pass 10-80Hz"),
  col = c(par("fg"), "red", "blue", "green"), lty = 1,
  cex = 0.6
)

# plot pwelch
pwelch(x, fs = 2000, window = 4000, noverlap = 2000, plot = 1)
pwelch(z1, fs = 2000, window = 4000, noverlap = 2000,
       plot = 2, col = "red")
pwelch(z2, fs = 2000, window = 4000, noverlap = 2000,
       plot = 2, col = "blue")
pwelch(z3, fs = 2000, window = 4000, noverlap = 2000,
       plot = 2, col = "green")

# ---- Clean this demo -----
par(oldpar)

```

Description

Provides five methods to baseline an array and calculate contrast.

Usage

```
baseline_array(x, along_dim, unit_dims = seq_along(dim(x))[-along_dim], ...)
```

```

## S3 method for class 'array'
baseline_array(
  x,
  along_dim,
  unit_dims = seq_along(dim(x))[-along_dim],
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore",
    "subtract_mean"),
  baseline_indexpoints = NULL,
  baseline_subarray = NULL,
  ...
)

```

Arguments

<code>x</code>	array (tensor) to calculate contrast
<code>along_dim</code>	integer range from 1 to the maximum dimension of <code>x</code> . baseline along this dimension, this is usually the time dimension.
<code>unit_dims</code>	integer vector, baseline unit: see Details.
<code>...</code>	passed to other methods
<code>method</code>	character, baseline method options are: "percentage", "sqrt_percentage", "decibel", "zscore", and "sqrt_zscore"
<code>baseline_indexpoints</code>	integer vector, which index points are counted into baseline window? Each index ranges from 1 to <code>dim(x)[[along_dim]]</code> . See Details.
<code>baseline_subarray</code>	sub-arrays that should be used to calculate baseline; default is NULL (automatically determined by <code>baseline_indexpoints</code>).

Details

Consider a scenario where we want to baseline a bunch of signals recorded from different locations. For each location, we record n sessions. For each session, the signal is further decomposed into frequency-time domain. In this case, we have the input x in the following form:

$$session \times frequency \times time \times location$$

Now we want to calibrate signals for each session, frequency and location using the first 100 time points as baseline points, then the code will be

```
baseline_array(x, along_dim = 3, baseline_window = 1 : 100, unit_dims = c(1, 2, 4))
```

`along_dim=3` is dimension of time, in this case, it's the third dimension of x . `baseline_indexpoints=1:100`, meaning the first 100 time points are used to calculate baseline. `unit_dims` defines the unit signal. Its value `c(1, 2, 4)` means the unit signal is per session (first dimension), per frequency (second) and per location (fourth).

In some other cases, we might want to calculate baseline across frequencies then the unit signal is *frequency × time*, i.e. signals that share the same session and location also share the same baseline. In this case, we assign `unit_dims=c(1, 4)`.

There are five baseline methods. They fit for different types of data. Denote z is an unit signal, z_0 is its baseline slice. Then these baseline methods are:

"percentage"

$$\frac{z - \bar{z}_0}{\bar{z}_0} \times 100\%$$

"sqrt_percentage"

$$\frac{\sqrt{z} - \sqrt{\bar{z}_0}}{\sqrt{\bar{z}_0}} \times 100\%$$

"decibel"

$$10 \times (\log_{10}(z) - \log_{10}(\bar{z}_0))$$

"zscore"

$$\frac{z - \bar{z}_0}{sd(z_0)}$$

"sqrt_zscore"

$$\frac{\sqrt{z} - \sqrt{\bar{z}_0}}{sd(\sqrt{z_0})}$$

Value

Contrast array with the same dimension as x .

Examples

```
# Set ncores = 2 to comply to CRAN policy. Please don't run this line
ravetools_threads(n_threads = 2L)
```

```
library(ravetools)
set.seed(1)
```

```
# Generate sample data
dims = c(10,20,30,2)
x = array(rnorm(prod(dims))^2, dims)
```

```
# Set baseline window to be arbitrary 10 timepoints
baseline_window = sample(30, 10)
```

```
# ----- baseline percentage change -----
```

```
# Using base functions
re1 <- aperm(apply(x, c(1,2,4), function(y){
  m <- mean(y[baseline_window])
  (y/m - 1) * 100
}), c(2,3,1,4))
```

```
# Using ravetools
re2 <- baseline_array(x, 3, c(1,2,4),
  baseline_indexpoints = baseline_window,
```

```

        method = 'percentage')

# Check different, should be very tiny (double precisions)
range(re2 - re1)

# Check speed for large dataset, might take a while to profile
ravetools_threads(n_threads = -1)

dims <- c(200,20,300,2)
x <- array(rnorm(prod(dims))^2, dims)
# Set baseline window to be arbitrary 10 timepoints
baseline_window <- seq_len(100)
f1 <- function(){
  aperm(apply(x, c(1,2,4), function(y){
    m <- mean(y[baseline_window])
    (y/m - 1) * 100
  }), c(2,3,1,4))
}
f2 <- function(){
  # equivalent as bl = x[,baseline_window, ]
  #
  baseline_array(x, along_dim = 3,
                 baseline_indexpoints = baseline_window,
                 unit_dims = c(1,2,4), method = 'percentage')
}
range(f1() - f2())
microbenchmark::microbenchmark(f1(), f2(), times = 10L)

```

collapse

Collapse array

Description

Collapse array

Usage

```
collapse(x, keep, ...)
```

```
## S3 method for class 'array'
collapse(
  x,
  keep,
```



```

    average = TRUE,
    transform = c("asis", "10log10", "square", "sqrt"),
    ...
  )

```

Arguments

x	A numeric multi-mode tensor (array), without NA
keep	Which dimension to keep
...	passed to other methods
average	collapse to sum or mean
transform	transform on the data before applying collapsing; choices are 'asis' (no change), '10log10' (used to calculate decibel), 'square' (sum-squared), 'sqrt' (square-root and collapse)

Value

a collapsed array with values to be mean or summation along collapsing dimensions

Examples

```

# Set ncores = 2 to comply to CRAN policy. Please don't run this line
ravetools_threads(n_threads = 2L)

# Example 1
x = matrix(1:16, 4)

# Keep the first dimension and calculate sums along the rest
collapse(x, keep = 1)
rowMeans(x) # Should yield the same result

# Example 2
x = array(1:120, dim = c(2,3,4,5))
result = collapse(x, keep = c(3,2))
compare = apply(x, c(3,2), mean)
sum(abs(result - compare)) # The same, yield 0 or very small number (1e-10)

ravetools_threads(n_threads = -1)

# Example 3 (performance)

# Small data, no big difference
x = array(rnorm(240), dim = c(4,5,6,2))
microbenchmark::microbenchmark(
  result = collapse(x, keep = c(3,2)),
  compare = apply(x, c(3,2), mean),

```

```

times = 1L, check = function(v){
  max(abs(range(do.call('-', v)))) < 1e-10
}
)

# large data big difference
x = array(rnorm(prod(300,200,105)), c(300,200,105,1))
microbenchmark::microbenchmark(
  result = collapse(x, keep = c(3,2)),
  compare = apply(x, c(3,2), mean),
  times = 1L , check = function(v){
    max(abs(range(do.call('-', v)))) < 1e-10
  })

```

convolve

Convolution of 1D, 2D, 3D data via FFT

Description

Use the 'Fast-Fourier' transform to compute the convolutions of two data with zero padding.

Usage

```
convolve_signal(x, filter)
```

```
convolve_image(x, filter)
```

```
convolve_volume(x, filter)
```

Arguments

<code>x</code>	one-dimensional signal vector, two-dimensional image, or three-dimensional volume; numeric or complex
<code>filter</code>	kernel with the same number of dimensions as <code>x</code>

Details

This implementation uses 'Fast-Fourier' transform to perform 1D, 2D, or 3D convolution. Compared to implementations using original mathematical definition of convolution, this approach is much faster, especially for image and volume convolutions.

The input `x` is zero-padded beyond edges. This is most common in image or volume convolution, but less optimal for periodic one-dimensional signals. Please use other implementations if non-zero padding is needed.

The convolution results might be different to the ground truth by a precision error, usually at $1e-13$ level, depending on the 'FFTW3' library precision and implementation.

Value

Convolution results with the same length and dimensions as `x`. If `x` is complex, results will be complex, otherwise results will be real numbers.

Examples

```
# ---- 1D convolution -----
x <- cumsum(rnorm(100))
filter <- dnorm(-2:2)
# normalize
filter <- filter / sum(filter)
smoothed <- convolve_signal(x, filter)

plot(x, pch = 20)
lines(smoothed, col = 'red')

# ---- 2D convolution -----
x <- array(0, c(100, 100))
x[
  floor(runif(10, min = 1, max = 100)),
  floor(runif(10, min = 1, max = 100))
] <- 1

# smooth
kernel <- outer(dnorm(-2:2), dnorm(-2:2), FUN = "*")
kernel <- kernel / sum(kernel)

y <- convolve_image(x, kernel)

oldpar <- par(mfrow = c(1,2))
image(x, asp = 1, axes = FALSE, main = "Origin")
image(y, asp = 1, axes = FALSE, main = "Smoothed")
par(oldpar)
```

decimate

Decimate with 'FIR' or 'IIR' filter

Description

Decimate with 'FIR' or 'IIR' filter

Usage

```
decimate(x, q, n = if (ftype == "iir") 8 else 30, ftype = "fir")
```

Arguments

x	signal to be decimated
q	integer factor to down-sample by
n	filter order used in the down-sampling; default is 30 if ftype='fir', or 8 if ftype='iir'
ftype	filter type, choices are 'fir' (default) and 'iir'

Details

This function is migrated from signal package, but with bugs fixed on 'FIR' filters. The result agrees with 'Matlab' decimate function with 'FIR' filters. Under 'IIR' filters, the function is identical with signal::decimate, and is slightly different with 'Matlab' version.

Value

Decimated signal

Examples

```
x <- 1:100
y <- decimate(x, 2, ftype = "fir")
y

# compare with signal package
z <- signal::decimate(x, 2, ftype = "fir")

# Compare decimated results
plot(x, type = 'l')
points(seq(1,100, 2), y, col = "green")
points(seq(1,100, 2), z, col = "red")
```

detrend

Remove the trend for one or more signals

Description

'Detrending' is often used before the signal power calculation.

Usage

```
detrend(x, trend = c("constant", "linear"), break_points = NULL)
```

Arguments

x	numerical or complex, a vector or a matrix
trend	the trend of the signal; choices are 'constant' and 'linear'
break_points	integer vector, or NULL; only used when trend is 'linear' to remove piecewise linear trend; will throw warnings if trend is 'constant'

Value

The signals with trend removed in matrix form; the number of columns is the number of signals, and number of rows is length of the signals

Examples

```
x <- rnorm(100, mean = 1) + c(
  seq(0, 5, length.out = 50),
  seq(5, 3, length.out = 50))
plot(x)

plot(detrend(x, 'constant'))
plot(detrend(x, 'linear'))
plot(detrend(x, 'linear', 50))
```

diagnose_channel *Show channel signals with diagnostic plots*

Description

The diagnostic plots include 'Welch Periodogram' ([pwelch](#)) and histogram ([hist](#))

Usage

```
diagnose_channel(
  s1,
  s2 = NULL,
  s3 = NULL,
  srate,
  name = "",
  try_compress = TRUE,
  max_freq = 300,
  window = ceiling(srate * 2),
  noverlap = window/2,
  std = 3,
  which = NULL,
  main = "Channel Inspection",
  col = c("black", "red"),
```

```

    cex = 1.2,
    cex.lab = 1,
    lwd = 0.5,
    plim = NULL,
    nclass = 100,
    start_time = 0,
    boundary = NULL,
    mar = c(3.1, 4.1, 2.1, 0.8) * (0.25 + cex * 0.75) + 0.1,
    mgp = cex * c(2, 0.5, 0),
    xaxs = "i",
    yaxs = "i",
    xline = 1.66 * cex,
    yline = 2.66 * cex,
    tck = -0.005 * (3 + cex),
    ...
)

```

Arguments

s1	the main signal to draw
s2	the comparing signal to draw; usually s1 after some filters; must be in the same sampling rate with s1; can be NULL
sc	decimated s1 to show if srate is too high; will be automatically generated if NULL
srate	sampling rate
name	name of s1, or a vector of two names of s1 and s2 if s2 is provided
try_compress	whether try to compress (decimate) s1 if srate is too high for performance concerns
max_freq	the maximum frequency to display in 'Welch Periodograms'
window, noverlap	see pwelch
std	the standard deviation of the channel signals used to determine boundary; default is plus-minus 3 standard deviation
which	NULL or integer from 1 to 4; if NULL, all plots will be displayed; otherwise only the subplot will be displayed
main	the title of the signal plot
col	colors of s1 and s2
cex, lwd, mar, cex.lab, mgp, xaxs, yaxs, tck, ...	graphical parameters; see par
plim	the y-axis limit to draw in 'Welch Periodograms'
nclass	number of classes to show in histogram (hist)
start_time	the starting time of channel (will only be used to draw signals)
boundary	a red boundary to show in channel plot; default is to be automatically determined by std
xline, yline	distance of axis labels towards ticks

Value

A list of boundary and y-axis limit used to draw the channel

Examples

```
library(ravetools)

# Generate 20 second data at 2000 Hz
time <- seq(0, 20, by = 1 / 2000)
signal <- sin( 120 * pi * time) +
  sin(time * 20*pi) +
  exp(-time^2) *
  cos(time * 10*pi) +
  rnorm(length(time))

signal2 <- notch_filter(signal, 2000)

diagnose_channel(signal, signal2, srate = 2000,
  name = c("Raw", "Filtered"), cex = 1)
```

fast_cov

Calculate massive covariance matrix in parallel

Description

Speed up covariance calculation for large matrices. The default behavior is the same as `cov` ('pearson', no NA handling).

Usage

```
fast_cov(x, y = NULL, col_x = NULL, col_y = NULL, df = NA)
```

Arguments

x	a numeric vector, matrix or data frame; a matrix is highly recommended to maximize the performance
y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x; the default is equivalent to y = x
col_x	integers indicating the subset indices (columns) of x to calculate the covariance, or NULL to include all the columns; default is NULL
col_y	integers indicating the subset indices (columns) of y to calculate the covariance, or NULL to include all the columns; default is NULL
df	a scalar indicating the degrees of freedom; default is <code>nrow(x)-1</code>

Value

A covariance matrix of x and y. Note that there is no NA handling. Any missing values will lead to NA in the resulting covariance matrices.

Examples

```
# Set ncores = 2 to comply to CRAN policy. Please don't run this line
ravetools_threads(n_threads = 2L)

x <- matrix(rnorm(400), nrow = 100)

# Call `cov(x)` to compare
fast_cov(x)

# Calculate covariance of subsets
fast_cov(x, col_x = 1, col_y = 1:2)

# Speed comparison, better to use multiple cores (4, 8, or more)
# to show the differences.

ravetools_threads(n_threads = -1)
x <- matrix(rnorm(100000), nrow = 1000)
microbenchmark::microbenchmark(
  fast_cov = {
    fast_cov(x, col_x = 1:50, col_y = 51:100)
  },
  cov = {
    cov(x[,1:50], x[,51:100])
  },
  unit = 'ms', times = 10
)
```

fast_quantile

Compute quantiles

Description

Compute quantiles

Usage

```
fast_quantile(x, prob = 0.5, na.rm = FALSE, ...)
```

```
fast_median(x, na.rm = FALSE, ...)
```

```
fast_mvquantile(x, prob = 0.5, na.rm = FALSE, ...)
```

```
fast_mvmedian(x, na.rm = FALSE, ...)
```

Arguments

x	numerical-value vector for fast_quantile and fast_median, and column-major matrix for fast_mvquantile and fast_mvmedian
prob	a probability with value from 0 to 1
na.rm	logical; if true, any NA are removed from x before the quantiles are computed
...	reserved for future use

Value

fast_quantile and fast_median calculate univariate quantiles (single-value return); fast_mvquantile and fast_mvmedian calculate multivariate quantiles (for each column, result lengths equal to the number of columns).

Examples

```
fast_quantile(runif(1000), 0.1)
fast_median(1:100)

x <- matrix(rnorm(100), ncol = 2)
fast_mvquantile(x, 0.2)
fast_mvmedian(x)

# Compare speed for vectors (usually 30% faster)
x <- rnorm(10000)
microbenchmark::microbenchmark(
  fast_median = fast_median(x),
  base_median = median(x),
  # bioc_median = Biobase::rowMedians(matrix(x, nrow = 1)),
  times = 100, unit = "milliseconds"
)

# Multivariate cases
# (5~7x faster than base R)
# (3~5x faster than Biobase rowMedians)
x <- matrix(rnorm(100000), ncol = 20)
microbenchmark::microbenchmark(
  fast_median = fast_mvmedian(x),
  base_median = apply(x, 2, median),
  # bioc_median = Biobase::rowMedians(t(x)),
```

```

    times = 10, unit = "milliseconds"
)

```

fill_surface	<i>Fill a volume cube based on water-tight surface</i>
--------------	--

Description

Create a cube volume (256 'voxels' on each margin), fill in the 'voxels' that are inside of the surface.

Usage

```

fill_surface(
  surface,
  inflate = 0,
  IJK2RAS = NULL,
  preview = FALSE,
  preview_frame = 128
)

```

Arguments

surface	a surface mesh, can be mesh objects from rgl or freesurferformats packages
inflate	amount of 'voxels' to inflate on the final result; must be a non-negative integer. A zero inflate value means the resulting volume is tightly close to the surface
IJK2RAS	volume 'IJK' (zero-indexed coordinate index) to 'tkrRAS' transform, default is automatically determined leave it 'NULL' if you don't know how to set it
preview	whether to preview the results; default is false
preview_frame	integer from 1 to 256 the depth frame used to generate preview.

Details

This function creates a volume (256 on each margin) and fill in the volume from a surface mesh. The surface vertex points will be embedded into the volume first. These points may not be connected together, hence for each 'voxel', a cube patch will be applied to grow the volume. Then, the volume will be bucket-filled from a corner, forming a negated mask of "outside-of-surface" area. The inverted bucket-filled volume is then shrunk so the mask boundary tightly fits the surface

Value

A list containing the filled volume and parameters used to generate the volume

Author(s)

Zhengjia Wang

Examples

```
# takes > 5s to run example

# Generate a sphere
surface <- vcg_sphere()
surface$vb[1:3, ] <- surface$vb[1:3, ] * 50

fill_surface(surface, preview = TRUE)
```

filter_signal	<i>Filter one-dimensional signal</i>
---------------	--------------------------------------

Description

The function is written from the scratch. The result has been compared against the 'Matlab' filter function with one-dimensional real inputs. Other situations such as matrix b or multi-dimensional x are not implemented.

Usage

```
filter_signal(b, a, x, z)
```

Arguments

b	one-dimensional real numerical vector, the moving-average coefficients of an ARMA filter
a	the auto-regressive (recursive) coefficients of an ARMA filter
x	numerical vector input (real value)
z	initial condition, must have length of n-1, where n is the maximum of lengths of a and b; default is all zeros

Value

A list of two vectors: the first vector is the filtered signal; the second vector is the final state of z

Examples

```
t <- seq(0, 1, by = 0.01)
x <- sin(2 * pi * t * 2.3)
bf <- signal::butter(2, c(0.15, 0.3))
```

```

res <- filter_signal(bf$b, bf$a, x)
y <- res[[1]]
z <- res[[2]]

## Matlab (2022a) equivalent:
# t = [0:0.01:1];
# x = sin(2 * pi * t * 2.3);
# [b,a] = butter(2,[.15,.3]);
# [y,z] = filter(b, a, x)

```

 filtfilt

Forward and reverse filter a one-dimensional signal

Description

The result has been tested against 'Matlab' filtfilt function. Currently this function only supports one filter at a time.

Usage

```
filtfilt(b, a, x)
```

Arguments

b	one-dimensional real numerical vector, the moving-average coefficients of an ARMA filter
a	the auto-regressive (recursive) coefficients of an ARMA filter
x	numerical vector input (real value)

Value

The filtered signal, normally the same length as the input signal x.

Examples

```

t <- seq(0, 1, by = 0.01)
x <- sin(2 * pi * t * 2.3)
bf <- signal::butter(2, c(0.15, 0.3))

res <- filtfilt(bf$b, bf$a, x)

## Matlab (2022a) equivalent:
# t = [0:0.01:1];
# x = sin(2 * pi * t * 2.3);
# [b,a] = butter(2,[.15,.3]);

```

```
# res = filtfilt(b, a, x)
```

grow_volume	<i>Grow volume mask</i>
-------------	-------------------------

Description

Grow volume mask

Usage

```
grow_volume(volume, x, y = x, z = x, threshold = 0.5)
```

Arguments

volume	volume mask array, must be 3-dimensional array
x, y, z	size of grow along each direction
threshold	threshold after convolution

Examples

```
oldpar <- par(mfrow = c(2,3), mar = c(0.1,0.1,3.1,0.1))

mask <- array(0, c(21,21,21))
mask[11,11,11] <- 1
image(mask[11,,], asp = 1,
      main = "Original mask", axes = FALSE)
image(grow_volume(mask, 2)[11,,], asp = 1,
      main = "Dilated (size=2) mask", axes = FALSE)
image(grow_volume(mask, 5)[11,,], asp = 1,
      main = "Dilated (size=5) mask", axes = FALSE)

mask[11, sample(11,2), sample(11,2)] <- 1
image(mask[11,,], asp = 1,
      main = "Original mask", axes = FALSE)
image(grow_volume(mask, 2)[11,,], asp = 1,
      main = "Dilated (size=2) mask", axes = FALSE)
image(grow_volume(mask, 5)[11,,], asp = 1,
      main = "Dilated (size=5) mask", axes = FALSE)

par(oldpar)
```

`internal_rave_function`*Get external function from 'RAVE'*

Description

Internal function used for examples relative to 'RAVE' project and should not be used directly.

Usage

```
internal_rave_function(name, pkg, inherit = TRUE, on_missing = NULL)
```

Arguments

<code>name</code>	function or variable name
<code>pkg</code>	'RAVE' package name
<code>inherit</code>	passed to <code>get0</code>
<code>on_missing</code>	default value to return of no function is found

Value

Function object if found, otherwise `on_missing`.

`interpolate_stimulation`*Find and interpolate stimulation signals*

Description

Find and interpolate stimulation signals

Usage

```
interpolate_stimulation(  
  x,  
  sample_rate,  
  duration = 40/sample_rate,  
  ord = 4L,  
  nknots = 100,  
  nsd = 1,  
  nstim = NULL,  
  regularization = 0.5  
)
```

Arguments

x	numerical vector representing a analog signal
sample_rate	sampling frequency
duration	time in second: duration of interpolation
ord	spline order, default is 4
nknots	a rough number of knots to use, default is 100
nsd	number of standard deviation to detect stimulation signals, default is 1
nstim	number of stimulation pulses, default is to auto-detect
regularization	regularization parameter in case of inverting singular matrices, default is 0.5

Value

Interpolated signal with an attribute of which sample points are interpolated

Examples

```
x0 <- rnorm(1000) / 5 + sin(1:1000 / 300)

# Simulates pulase signals
x <- x0
x[400:410] <- -100
x[420:430] <- 100

fitted <- interpolate_stimulation(x, 100, duration = 0.3, nknots = 10, nsd = 2)

oldpar <- par(mfrow = c(2, 1))

plot(fitted, type = 'l', col = 'blue', lwd = 2)
lines(x, col = 'red')
lines(x0, col = 'black')
legend("topleft", c("Interpolated", "Observed", "Underlying"),
      lty = 1, col = c("blue", "red", "black"))

pwelch(x0, 100, 200, 100, plot = 1, col = 'black', ylim = c(-50, 50))
pwelch(x, 100, 200, 100, plot = 2, col = 'red')
pwelch(fitted, 100, 200, 100, plot = 2, col = 'blue')

par(oldpar)
```

left_hippocampus_mask *Left 'Hippocampus' of 'N27-Collin' brain*

Description

Left 'Hippocampus' of 'N27-Collin' brain

Usage

left_hippocampus_mask

Format

A three-mode integer mask array with values of 1 ('Hippocampus') and 0 (other brain tissues)

matlab_palette *'Matlab' heat-map plot palette*

Description

'Matlab' heat-map plot palette

Usage

matlab_palette()

Value

vector of 64 colors

mesh_from_volume *Generate 3D mesh surface from volume data*

Description

This function is soft-deprecated. Please use [vcg_mesh_volume](#), [vcg_uniform_remesh](#), and [vcg_smooth_explicit](#) or [vcg_smooth_implicit](#).

Usage

```

mesh_from_volume(
  volume,
  output_format = c("rgl", "freesurfer"),
  IJK2RAS = NULL,
  threshold = 0,
  verbose = TRUE,
  remesh = TRUE,
  remesh_voxel_size = 1,
  remesh_multisample = TRUE,
  remesh_automerge = TRUE,
  smooth = FALSE,
  smooth_lambda = 10,
  smooth_delta = 20,
  smooth_method = "surfPreserveLaplace"
)

```

Arguments

volume	3-dimensional volume array
output_format	resulting data format, choices are 'rgl' and 'freesurfer'
IJK2RAS	volume 'IJK' (zero-indexed coordinate index) to 'tkrRAS' transform, default is automatically determined
threshold	threshold used to create volume mask; the surface will be created to fit the mask boundaries
verbose	whether to verbose the progress
remesh	whether to re-sample the mesh using vcg_uniform_remesh
remesh_voxel_size, remesh_multisample, remesh_automerge	see arguments in vcg_uniform_remesh
smooth	whether to smooth the mesh via vcg_smooth_explicit
smooth_lambda, smooth_delta, smooth_method	see vcg_smooth_explicit

Value

A 'mesh3d' surface if output_format is 'rgl', or 'fs.surface' surface otherwise.

Examples

```

volume <- array(0, dim = c(8,8,8))
volume[4:5, 4:5, 4:5] <- 1

graphics::image(x = volume[4,,])

# you can use rgl::wire3d(mesh) to visualize the mesh

```

```
mesh <- mesh_from_volume(volume, verbose = FALSE)
```

multitaper

Compute 'multitaper' spectral densities of time-series data

Description

Compute 'multitaper' spectral densities of time-series data

Usage

```
multitaper_config(
  data_length,
  fs,
  frequency_range = NULL,
  time_bandwidth = 5,
  num_tapers = NULL,
  window_params = c(5, 1),
  nfft = NA,
  detrend_opt = "linear"
)

multitaper(
  data,
  fs,
  frequency_range = NULL,
  time_bandwidth = 5,
  num_tapers = NULL,
  window_params = c(5, 1),
  nfft = NA,
  detrend_opt = "linear"
)
```

Arguments

data_length	length of data
fs	sampling frequency in 'Hz'
frequency_range	frequency range to look at; length of two
time_bandwidth	a number indicating time-half bandwidth product; i.e. the window duration times the half bandwidth of main lobe; default is 5
num_tapers	number of 'DPSS' tapers to use; default is NULL and will be automatically computed from $\text{floor}(2 \times \text{time_bandwidth} - 1)$

window_params	vector of two numbers; the first number is the window size in seconds; the second number is the step size; default is c(5, 1)
nfft	'NFFT' size, positive; see 'Details'
detrend_opt	how you want to remove the trend from data window; options are 'linear' (default), 'constant', and 'off'
data	numerical vector, signal traces

Details

The original source code comes from 'Prerau' Lab (see 'Github' repository 'multitaper_toolbox' under user 'preraulab'). The results tend to agree with their 'Python' implementation with precision on the order of at $1E-7$ with standard deviation at most $1E-5$. The original copy was licensed under a Creative Commons Attribution 'NC'-'SA' 4.0 International License (<https://creativecommons.org/licenses/by-nc-sa/4.0/>).

This package ('ravetools') redistributes the multitaper function under minor modifications on nfft. In the original copy there is no parameter to control the exact numbers of nfft, and the nfft is always the power of 2. While choosing nfft to be the power of 2 is always recommended, the modified code allows other choices.

Value

multitaper_config returns a list of configuration parameters for the filters; multitaper also returns the time, frequency and corresponding spectral power.

Examples

```
# Takes long to run

time <- seq(0, 3, by = 0.001)
x <- sin(time * 20*pi) + exp(-time^2) * cos(time * 10*pi)

res <- multitaper(
  x, 1000, frequency_range = c(0,15),
  time_bandwidth=1.5,
  window_params=c(2,0.01)
)

image(
  x = res$time,
  y = res$frequency,
  z = 10 * log10(res$spec),
  xlab = "Time (s)",
  ylab = "Frequency (Hz)",
  col = matlab_palette()
)
```

new_matrix4	Create a Matrix4 instance for 'Affine' transform
-------------	--

Description

Create a Matrix4 instance for 'Affine' transform

Usage

```
new_matrix4()
```

```
as_matrix4(m)
```

Arguments

m	a matrix or a vector to be converted to the Matrix4 instance; m must be one of the followings: for matrices, the dimension must be 4x4, 3x4 (the last row will be 0 0 0 1), or 3x3 (linear transform); for vectors, the length must be 16, 12 (will append 0 0 0 1 internally), 3 (translation), or 1 (scale).
---	--

Value

A Matrix4 instance

See Also

[new_vector3](#), [new_quaternion](#)

new_quaternion	Create a Quaternion instance to store '3D' rotation
----------------	---

Description

Create instances that mimic the 'three.js' syntax.

Usage

```
new_quaternion(x = 0, y = 0, z = 0, w = 1)
```

```
as_quaternion(q)
```

Arguments

x, y, z, w numeric of length one
q R object to be converted to Quaternion

Value

A Quaternion instance

See Also

[new_vector3](#), [new_matrix4](#)

new_vector3 *Create a Vector3 instance to store '3D' points*

Description

Create instances that mimic the 'three.js' syntax.

Usage

```
new_vector3(x = 0, y = 0, z = 0)  
  
as_vector3(v)
```

Arguments

x, y, z numeric, must have the same length, 'xyz' positions
v R object to be converted to Vector3 instance

Value

A Vector3 instance

See Also

[new_matrix4](#), [new_quaternion](#)

Examples

```
vec3 <- new_vector3(  
  x = 1:9,  
  y = 9:1,  
  z = rep(c(1,2,3), 3)  
)  
  
vec3[]
```

```
# transform
m <- new_matrix4()

# rotation xy plane by 30 degrees
m$make_rotation_z(pi / 6)

vec3$apply_matrix4(m)

vec3[]

as_vector3(c(1,2,3))
```

notch_filter

Apply 'Notch' filter

Description

Apply 'Notch' filter

Usage

```
notch_filter(
  s,
  sample_rate,
  lb = c(59, 118, 178),
  ub = c(61, 122, 182),
  domain = 1
)
```

Arguments

s	numerical vector if domain=1 (voltage signals), or complex vector if domain=0
sample_rate	sample rate
lb	filter lower bound of the frequencies to remove
ub	filter upper bound of the frequencies to remove; shares the same length as lb
domain	1 if the input signal is in the time domain, 0 if it is in the frequency domain

Details

Mainly used to remove electrical line frequencies at 60, 120, and 180 Hz.

Value

filtered signal in time domain (real numerical vector)

Examples

```
time <- seq(0, 3, 0.005)
s <- sin(120 * pi * time) + rnorm(length(time))

# Welch periodogram shows a peak at 60Hz
pwelch(s, 200, plot = 1, log = "y")

# notch filter to remove 60Hz
s1 <- notch_filter(s, 200, lb = 59, ub = 61)
pwelch(s1, 200, plot = 2, log = "y", col = "red")
```

parallel-options	<i>Set or get thread options</i>
------------------	----------------------------------

Description

Set or get thread options

Usage

```
detect_threads()

ravetools_threads(n_threads = "auto", stack_size = "auto")
```

Arguments

n_threads	number of threads to set
stack_size	Stack size (in bytes) to use for worker threads. The default used for "auto" is 2MB on 32-bit systems and 4MB on 64-bit systems.

Value

detect_threads returns an integer of default threads that is determined by the number of CPU cores; ravetools_threads returns nothing.

Examples

```
detect_threads()

ravetools_threads(n_threads = 2)
```

plot_signals

Plot one or more signal traces in the same figure

Description

Plot one or more signal traces in the same figure

Usage

```
plot_signals(
  signals,
  sample_rate = 1,
  col = graphics::par("fg"),
  space = 0.995,
  space_mode = c("quantile", "absolute"),
  start_time = 0,
  duration = NULL,
  compress = TRUE,
  channel_names = NULL,
  time_shift = 0,
  xlab = "Time (s)",
  ylab = "Electrode",
  lwd = 0.5,
  new_plot = TRUE,
  xlim = NULL,
  cex = 1,
  cex.lab = 1,
  mar = c(3.1, 2.1, 2.1, 0.8) * (0.25 + cex * 0.75) + 0.1,
  mgp = cex * c(2, 0.5, 0),
  xaxs = "r",
  yaxs = "i",
  xline = 1.5 * cex,
  yline = 1 * cex,
  tck = -0.005 * (3 + cex),
  ...
)
```

Arguments

signals	numerical matrix with each row to be a signal trace and each column contains the signal values at a time point
sample_rate	sampling frequency
col	signal color, can be vector of one or more
space	vertical spacing among the traces; for values greater than 1, the spacing is absolute; default is 0.995; for values less equal to 1, this is the percentile of the whole data. However, the quantile mode can be manually turned off is "absolute" is required; see space_mode

space_mode	mode of spacing, only used when space is less equal to one; default is quantile
start_time	the time to start drawing relative to the first column
duration	duration of the signal to draw
compress	whether to compress signals if the data is too large
channel_names	NULL or a character vector of channel names
time_shift	the actual start time of the signal. Unlike start_time, this should be the actual physical time represented by the first column
xlab, ylab, lwd, xlim, cex, cex.lab, mar, mgp, xaxs, yaxs, tck, ...	plot parameters; see plot and par
new_plot	whether to draw a new plot; default is true
xline, yline	the gap between axis and label

Examples

```
n <- 1000
base_signal <- c(rep(0, n/2), sin(seq(0,10,length.out = n/2))) * 10
signals <- rbind(rnorm(n) + base_signal,
                rbinom(n, 10, 0.3) + base_signal,
                rt(n, 5) + base_signal)
plot_signals(signals, sample_rate = 100)
plot_signals(signals, sample_rate = 100, start_time = 5)
plot_signals(signals, sample_rate = 100,
             start_time = 5, time_shift = 100)
```

pwelch

Calculate 'Welch Periodogram'

Description

pwelch is for single signal trace only; mv_pwelch is for multiple traces. Currently mv_pwelch is experimental and should not be called directly.

Usage

```
pwelch(
  x,
  fs,
  window = 64,
  noverlap = 8,
  nfft = 256,
  col = "black",
  xlim = NULL,
```

```

ylim = NULL,
main = "Welch periodogram",
plot = 0,
log = c("xy", "", "x", "y"),
...
)

## S3 method for class 'pwelch'
print(x, ...)

## S3 method for class 'pwelch'
plot(
  x,
  log = c("xy", "x", "y", ""),
  se = FALSE,
  xticks,
  type = "l",
  add = FALSE,
  col = graphics::par("fg"),
  col.se = "orange",
  alpha.se = 0.5,
  lty = 1,
  lwd = 1,
  cex = 1,
  las = 1,
  main = "Welch periodogram",
  xlab,
  ylab,
  xlim = NULL,
  ylim = NULL,
  xaxs = "i",
  yaxs = "i",
  xline = 1.2 * cex,
  yline = 2 * cex,
  mar = c(2.6, 3.8, 2.1, 0.6) * (0.5 + cex/2),
  mgp = cex * c(2, 0.5, 0),
  tck = -0.02 * cex,
  grid = TRUE,
  ...
)

mv_pwelch(x, margin, fs, nfft)

```

Arguments

x numerical vector or a row-major vector, signals. If **x** is a matrix, then each row is a channel. For `plot` function, **x** is the instance returned by `pwelch` function.

fs sample rate, average number of time points per second

window	window length in time points, default size is 64
noverlap	overlap between two adjacent windows, measured in time points; default is 8
nfft	number of basis functions to apply
col, xlim, ylim, main, type, cex, las, xlab, ylab, lty, lwd, xaxs, yaxs, mar, mgp, tck	parameters passed to <code>plot.default</code>
plot	integer, whether to plot the result or not; choices are 0, no plot; 1 plot on a new canvas; 2 add to existing canvas
log	indicates which axis should be log ₁₀ -transformed, used by the plot function. For 'x' axis, it's log ₁₀ -transform; for 'y' axis, it's 10log ₁₀ -transform (decibel unit). Choices are "xy", "x", "y", and "".
...	will be passed to <code>plot.pwelch</code> or ignored
se	logical or a positive number indicating whether to plot standard error of mean; default is false. If provided with a number, then a multiple of standard error will be drawn. This option is only available when power is in log-scale (decibel unit)
xticks	ticks to show on frequency axis
add	logical, whether the plot should be added to existing canvas
col.se, alpha.se	controls the color and opacity of the standard error
xline, yline	controls how close the axis labels to the corresponding axes
grid	whether to draw rectangular grid lines to the plot; only respected when add=FALSE; default is true
margin	the margin in which <code>pwelch</code> should be applied to

Value

A list with class 'ravetools-pwelch' that contains the following items:

freq frequencies used to calculate the 'periodogram'
spec resulting spectral power for each frequency
window window function (in numerical vector) used
noverlap number of overlapping time-points between two adjacent windows
nfft number of basis functions
fs sample rate
x_len input signal length
method a character string 'Welch'

Examples

```
x <- rnorm(1000)
pwel <- pwelch(x, 100)
pwel

plot(pwel, log = "xy")
```

raw-to-sexp	<i>Convert raw vectors to R vectors</i>
-------------	---

Description

Convert raw vectors to R vectors

Usage

raw_to_uint8(x)

raw_to_uint16(x)

raw_to_uint32(x)

raw_to_int8(x)

raw_to_int16(x)

raw_to_int32(x)

raw_to_int64(x)

raw_to_float(x)

raw_to_string(x)

Arguments

x raw vector of bytes

Details

For numeric conversions, the function names are straightforward. For example, `raw_to_uintN` converts raw vectors to unsigned integers, and `raw_to_intN` converts raw vectors to signed integers. The number 'N' stands for the number of bits used to store the integer. For example `raw_to_uint8` uses 8 bits (1 byte) to store an integer, hence the value range is 0-255.

The input data length must be multiple of the element size represented by the underlying data. For example `uint16` integer uses 16 bits, and one raw number uses 8 bits, hence two raw vectors can form one unsigned integer-16. That is, `raw_to_uint16` requires the length of input to be multiple of two. An easy calculation is: the length of x times 8, must be divided by 'N' (see last paragraph for definition).

The returned data uses the closest available R native data type that can fully represent the data. For example, R does not have single float type, hence `raw_to_float` returns double type, which can represent all possible values in float. For `raw_to_uint32`, the potential value range is $0 - (2^{32}-1)$. This exceeds the limit of R integer type $(-2^{31}) - (2^{31}-1)$. Therefore, the returned values will be real (double float) data type.

There is no native data type that can store integer-64 data in R, package `bit64` provides `integer64` type, which will be used by `raw_to_int64`. Currently there is no solution to convert raw to unsigned integer-64 type.

`raw_to_string` converts raw to character string. This function respects null character, hence is slightly different than the native `rawToChar`, which translates raw byte-by-byte. If each raw byte represents a valid character, then the above two functions returns the same result. However, when the characters represented by raw bytes are invalid, `raw_to_string` will stop parsing and returns only the valid characters, while `rawToChar` will still try to parse, and most likely to result in errors. Please see Examples for comparisons.

Value

Numeric vectors, except for `raw_to_string`, which returns a string.

Examples

```
# 0x00, 0x7f, 0x80, 0xFF
x <- as.raw(c(0, 127, 128, 255))

raw_to_uint8(x)

# The first bit becomes the integer sign
# 128 -> -128, 255 -> -1
raw_to_int8(x)

## Comments based on little endian system

# 0x7f00 (32512), 0xFF80 (65408 unsigned, or -128 signed)
raw_to_uint16(x)
raw_to_int16(x)

# 0xFF807F00 (4286611200 unsigned, -8356096 signed)
raw_to_uint32(x)
raw_to_int32(x)

# ----- String -----

# ASCII case: all valid
x <- charToRaw("This is an ASCII string")

raw_to_string(x)
rawToChar(x)

x <- c(charToRaw("This is the end."),
      as.raw(0),
      charToRaw("*** is invalid"))

# rawToChar will raise error
raw_to_string(x)

# ----- Integer64 -----
```

```

# Runs on little endian system
x <- as.raw(c(0x80, 0x00, 0x7f, 0x80, 0xFF, 0x50, 0x7f, 0x00))

# Calculate bitstring, which concatenates the followings
# 10000000 (0x80), 00000000 (0x00), 01111111 (0x7f), 10000000 (0x80),
# 11111111 (0xFF), 01010000 (0x50), 01111111 (0x7f), 00000000 (0x00)

if(.Platform$endian == "little") {
  bitstring <- paste0(
    "00000000011111110101000011111111",
    "10000000011111110000000010000000"
  )
} else {
  bitstring <- paste0(
    "00000001000000001111111000000001",
    "11111111000010101111111000000000"
  )
}

# This is expected value
bit64::as.integer64(structure(
  bitstring,
  class = "bitstring"
))

# This is actual value
raw_to_int64(x)

```

register_volume

Imaging registration using 'NiftyReg'

Description

Registers 'CT' to 'MRI', or 'MRI' to another 'MRI'

Usage

```

register_volume(
  source,
  target,
  method = c("rigid", "affine", "nonlinear"),
  interpolation = c("cubic", "trilinear", "nearest"),
  threads = detect_threads(),
  symmetric = TRUE,
  verbose = TRUE,
  ...
)

```

Arguments

source source imaging data, or a 'nifti' file path; for example, 'CT'
 target target imaging data to align to; for example, 'MRI'
 method method of transformation, choices are 'rigid', 'affine', or 'nonlinear'
 interpolation how volumes should be interpolated, choices are 'cubic', 'trilinear', or 'nearest'
 threads, symmetric, verbose, ...
 see [niftyreg](#)

Value

See [niftyreg](#)

Examples

```
source <- system.file("extdata", "epi_t2.nii.gz", package="RNiftyReg")
target <- system.file("extdata", "flash_t1.nii.gz", package="RNiftyReg")
aligned <- register_volume(source, target, verbose = FALSE)

source_img <- aligned$source[[1]]
target_img <- aligned$target
aligned_img <- aligned$image

oldpar <- par(mfrow = c(2, 2), mar = c(0.1, 0.1, 3.1, 0.1))

pal <- grDevices::grey.colors(256, alpha = 1)
image(source_img[, , 30], asp = 1, axes = FALSE,
      col = pal, main = "Source image")
image(target_img[, , 64], asp = 1, axes = FALSE,
      col = pal, main = "Target image")
image(aligned_img[, , 64], asp = 1, axes = FALSE,
      col = pal, main = "Aligned image")

# bucket fill and calculate differences
aligned_img[is.nan(aligned_img) | aligned_img <= 1] <- 1
target_img[is.nan(target_img) | aligned_img <= 1] <- 1
diff <- abs(aligned_img / target_img - 1)
image(diff[, , 64], asp = 1, axes = FALSE,
      col = pal, main = "Percentage Difference")

par(oldpar)
```

 rgl-call

Safe ways to call package 'rgl' without requiring 'x11'

Description

Internally used for example show-cases. Please install package 'rgl' manually to use these functions.

Usage

```
rgl_call(FUN, ...)
```

```
rgl_view(expr, quoted = FALSE, env = parent.frame())
```

```
rgl_plot_normals(x, length = 1, lwd = 1, col = 1, ...)
```

Arguments

FUN	'rgl' function name
...	passed to 'rgl' function
expr	expression within which 'rgl' functions are called
quoted	whether expr is quoted
env	environment in which expr is evaluated
x	triangular 'mesh3d' object
length, lwd, col	normal vector length, size, and color

Examples

```
# Make sure the example does not run when compiling
# or check the package
if(FALSE) {

  volume <- array(0, dim = c(8,8,8))
  volume[4:5, 4:5, 4:5] <- 1
  mesh <- mesh_from_volume(volume, verbose = FALSE)

  rgl_view({

    rgl_call("shade3d", mesh, col = 3)
    rgl_plot_normals(mesh)

  })

}
```

shift_array	<i>Shift array by index</i>
-------------	-----------------------------

Description

Re-arrange arrays in parallel

Usage

```
shift_array(x, along_margin, unit_margin, shift_amount)
```

Arguments

x	array, must have at least matrix
along_margin	which index is to be shifted
unit_margin	which dimension decides shift_amount
shift_amount	shift amount along along_margin

Details

A simple use-case for this function is to think of a matrix where each row is a signal and columns stand for time. The objective is to align (time-lock) each signal according to certain events. For each signal, we want to shift the time points by certain amount.

In this case, the shift amount is defined by `shift_amount`, whose length equals to number of signals. `along_margin=2` as we want to shift time points (column, the second dimension) for each signal. `unit_margin=1` because the shift amount is depend on the signal number.

Value

An array with same dimensions as the input `x`, but with index shifted. The missing elements will be filled with NA.

Examples

```
# Set ncores = 2 to comply to CRAN policy. Please don't run this line
ravetools_threads(n_threads = 2L)

x <- matrix(1:10, nrow = 2, byrow = TRUE)
z <- shift_array(x, 2, 1, c(1,2))

y <- NA * x
y[1,1:4] = x[1,2:5]
```

```

y[2,1:3] = x[2,3:5]

# Check if z and y are the same
z = y

# array case
# x is Trial x Frequency x Time
x <- array(1:27, c(3,3,3))

# Shift time for each trial, amount is 1, -1, 0
shift_amount <- c(1,-1,0)
z <- shift_array(x, 3, 1, shift_amount)

oldpar <- par(mfrow = c(3, 2), mai = c(0.8, 0.6, 0.4, 0.1))
for( ii in 1:3 ){
  image(t(x[ii, ,]), ylab = 'Frequency', xlab = 'Time',
        main = paste('Trial', ii))
  image(t(z[ii, ,]), ylab = 'Frequency', xlab = 'Time',
        main = paste('Shifted amount:', shift_amount[ii]))
}
par(oldpar)

```

vcg_isosurface

Create surface mesh from 3D-array

Description

Create surface from 3D-array using marching cubes algorithm

Usage

```

vcg_isosurface(
  volume,
  threshold_lb = 0,
  threshold_ub = NA,
  vox_to_ras = diag(c(-1, -1, 1, 1))
)

```

Arguments

volume	a volume or a mask volume
threshold_lb	lower-bound threshold for creating the surface; default is 0
threshold_ub	upper-bound threshold for creating the surface; default is NA (no upper-bound)
vox_to_ras	a 4x4 'affine' transform matrix indicating the 'voxel'-to-world transform.

Value

A triangular mesh of class 'mesh3d'

Examples

```
if(is_not_cran()) {  
  
  library(ravetools)  
  data("left_hippocampus_mask")  
  
  mesh <- vcg_isosurface(left_hippocampus_mask)  
  
  rgl_view({  
  
    rgl_call("mfrow3d", 1, 2)  
  
    rgl_call("title3d", "Direct ISOSurface")  
    rgl_call("shade3d", mesh, col = 2)  
  
    rgl_call("next3d")  
    rgl_call("title3d", "ISOSurface + Implicit Smooth")  
  
    rgl_call("shade3d",  
            vcg_smooth_implicit(mesh, degree = 2),  
            col = 3)  
  })  
}
```

vcg_mesh_volume

Compute volume for manifold meshes

Description

Compute volume for manifold meshes

Usage

```
vcg_mesh_volume(mesh)
```

Arguments

mesh triangular mesh of class 'mesh3d'

Value

The numeric volume of the mesh

Examples

```
# Initial mesh
mesh <- vcg_sphere()

vcg_mesh_volume(mesh)
```

vcg_smooth

Implicitly smooth a triangular mesh

Description

Applies smoothing algorithms on a triangular mesh.

Usage

```
vcg_smooth_implicit(
  mesh,
  lambda = 0.2,
  use_mass_matrix = TRUE,
  fix_border = FALSE,
  use_cot_weight = FALSE,
  degree = 1L,
  laplacian_weight = 1
)

vcg_smooth_explicit(
  mesh,
  type = c("taubin", "laplace", "HClaplace", "fujiLaplace", "angWeight",
    "surfPreserveLaplace"),
  iteration = 10,
  lambda = 0.5,
  mu = -0.53,
  delta = 0.1
)
```

Arguments

mesh	triangular mesh stored as object of class 'mesh3d'.
lambda	In <code>vcg_smooth_implicit</code> , the amount of smoothness, useful only if <code>use_mass_matrix</code> is TRUE; default is 0.2. In <code>vcg_smooth_explicit</code> , parameter for 'taubin' smoothing.
use_mass_matrix	logical: whether to use mass matrix to keep the mesh close to its original position (weighted per area distributed on vertices); default is TRUE

fix_border	logical: whether to fix the border vertices of the mesh; default is FALSE
use_cot_weight	logical: whether to use cotangent weight; default is FALSE (using uniform 'Laplacian')
degree	integer: degrees of 'Laplacian'; default is 1
laplacian_weight	numeric: weight when use_cot_weight is FALSE; default is 1.0
type	method name of explicit smooth, choices are 'taubin', 'laplace', 'HClaplace', 'fujiLaplace', 'angWeight', 'surfPreserveLaplace'.
iteration	number of iterations
mu	parameter for 'taubin' explicit smoothing.
delta	parameter for scale-dependent 'Laplacian' smoothing or maximum allowed angle (in 'Radian') for deviation between surface preserving 'Laplacian'.

Value

An object of class "mesh3d" with:

vb	vertex coordinates
normals	vertex normal vectors
it	triangular face index

Examples

```

if(is_not_cran()) {

# Prepare mesh with no normals
data("left_hippocampus_mask")

# Grow 2mm on each direction to fill holes
volume <- grow_volume(left_hippocampus_mask, 2)

# Initial mesh
mesh <- vcg_isosurface(volume)

# Start: examples
rgl_view({
  rgl_call("mfrow3d", 2, 4)
  rgl_call("title3d", "Naive ISOSurface")
  rgl_call("shade3d", mesh, col = 2)

  rgl_call("next3d")
  rgl_call("title3d", "Implicit Smooth")
  rgl_call("shade3d", col = 2,
           x = vcg_smooth_implicit(mesh, degree = 2))

  rgl_call("next3d")
  rgl_call("title3d", "Explicit Smooth - taubin")
  rgl_call("shade3d", col = 2,

```

```

        x = vcg_smooth_explicit(mesh, "taubin"))

    rgl_call("next3d")
    rgl_call("title3d", "Explicit Smooth - laplace")
    rgl_call("shade3d", col = 2,
             x = vcg_smooth_explicit(mesh, "laplace"))

    rgl_call("next3d")
    rgl_call("title3d", "Explicit Smooth - angWeight")
    rgl_call("shade3d", col = 2,
             x = vcg_smooth_explicit(mesh, "angWeight"))

    rgl_call("next3d")
    rgl_call("title3d", "Explicit Smooth - HClaplace")
    rgl_call("shade3d", col = 2,
             x = vcg_smooth_explicit(mesh, "HClaplace"))

    rgl_call("next3d")
    rgl_call("title3d", "Explicit Smooth - fujiLaplace")
    rgl_call("shade3d", col = 2,
             x = vcg_smooth_explicit(mesh, "fujiLaplace"))

    rgl_call("next3d")
    rgl_call("title3d", "Explicit Smooth - surfPreserveLaplace")
    rgl_call("shade3d", col = 2,
             x = vcg_smooth_explicit(mesh, "surfPreserveLaplace"))
  })
}

```

vcg_sphere

Simple 3-dimensional sphere mesh

Description

Simple 3-dimensional sphere mesh

Usage

```
vcg_sphere(sub_division = 3L, normals = TRUE)
```

Arguments

sub_division	density of vertex in the resulting mesh
normals	whether the normal vectors should be calculated

Value

A 'mesh3d' object

Examples

```
vcg_sphere()
```

```
vcg_uniform_remesh    Sample a surface mesh uniformly
```

Description

Sample a surface mesh uniformly

Usage

```
vcg_uniform_remesh(  
    x,  
    voxel_size = NULL,  
    offset = 0,  
    discretize = FALSE,  
    multi_sample = FALSE,  
    absolute_distance = FALSE,  
    merge_clost = FALSE,  
    verbose = TRUE  
)
```

Arguments

x	surface
voxel_size	'voxel' size for space 'discretization'
offset	offset position shift of the new surface from the input
discretize	whether to use step function (TRUE) instead of linear interpolation (FALSE) to calculate the position of the intersected edge of the marching cube; default is FALSE
multi_sample	whether to calculate multiple samples for more accurate results (at the expense of more computing time) to remove artifacts; default is FALSE
absolute_distance	whether an unsigned distance field should be computed. When set to TRUE, non-zero offsets is to be set, and double-surfaces will be built around the original surface, like a sandwich.
merge_clost	whether to merge close vertices; default is TRUE
verbose	whether to verbose the progress; default is TRUE

Value

A triangular mesh of class 'mesh3d'

Examples

```

sphere <- vcg_sphere()
mesh <- vcg_uniform_remesh(sphere, voxel_size = 0.45)

if(is_not_cran()) {

  rgl_view({

    rgl_call("mfrow3d", 1, 2)

    rgl_call("title3d", "Input")
    rgl_call("wire3d", sphere, col = 2)
    rgl_call("next3d")

    rgl_call("title3d", "Re-meshed to 0.1mm edge distance")
    rgl_call("wire3d", mesh, col = 3)
  })

}

```

vcg_update_normals *Update vertex normal*

Description

Update vertex normal

Usage

```

vcg_update_normals(
  mesh,
  weight = c("area", "angle"),
  pointcloud = c(10, 0),
  verbose = FALSE
)

```

Arguments

mesh	triangular mesh or a point-cloud (matrix of 3 columns)
weight	method to compute per-vertex normal vectors: "area" weighted average of surrounding face normal, or "angle" weighted vertex normal vectors.
pointcloud	integer vector of length 2: containing optional parameters for normal calculation of point clouds; the first entry specifies the number of neighboring points to consider; the second entry specifies the amount of smoothing iterations to be performed.
verbose	whether to verbose the progress

Value

A 'mesh3d' object with normal vectors.

Examples

```
if(is_not_cran()) {  
  
  # Prepare mesh with no normal  
  data("left_hippocampus_mask")  
  mesh <- vcg_isosurface(left_hippocampus_mask)  
  mesh$normals <- NULL  
  
  # Start: examples  
  new_mesh <- vcg_update_normals(mesh, weight = "angle",  
                                pointcloud = c(10, 10))  
  
  rgl_view({  
    rgl_call("mfrow3d", 1, 2)  
    rgl_call("shade3d", mesh, col = 2)  
  
    rgl_call("next3d")  
    rgl_call("shade3d", new_mesh, col = 2)  
  })  
}
```

wavelet

'Morlet' wavelet transform (Discrete)

Description

Transform analog voltage signals with 'Morlet' wavelets: complex wavelet kernels with $\pi/2$ phase differences.

Usage

```
wavelet_kernels(freqs, srate, wave_num)  
  
morlet_wavelet(  
  data,  
  freqs,  
  srate,  
  wave_num,  
  precision = c("float", "double"),  
  trend = c("constant", "linear", "none"),  
  signature = NULL,
```

```

    ...
  )

wavelet_cycles_suggest(
  freqs,
  frequency_range = c(2, 200),
  cycle_range = c(3, 20)
)

```

Arguments

freqs	frequency in which data will be projected on
srate	sample rate, number of time points per second
wave_num	desired number of cycles in wavelet kernels to balance the precision in time and amplitude (control the smoothness); positive integers are strongly suggested
data	numerical vector such as analog voltage signals
precision	the precision of computation; choices are 'float' (default) and 'double'.
trend	choices are 'constant': center the signal at zero; 'linear': remove the linear trend; 'none' do nothing
signature	signature to calculate kernel path to save, internally used
...	further passed to detrend ;
frequency_range	frequency range to calculate, default is 2 to 200
cycle_range	number of cycles corresponding to frequency_range. For default frequency range (2 - 200), the default cycle_range is 3 to 20. That is, 3 wavelet kernel cycles at 2 Hertz, and 20 cycles at 200 Hertz.

Value

wavelet_kernels returns wavelet kernels to be used for wavelet function; morlet_wavelet returns a file-based array if precision is 'float', or a list of real and imaginary arrays if precision is 'double'

Examples

```

# generate sine waves
time <- seq(0, 3, by = 0.01)
x <- sin(time * 20*pi) + exp(-time^2) * cos(time * 10*pi)

plot(time, x, type = 'l')

# freq from 1 - 15 Hz; wavelet using float precision
freq <- seq(1, 15, 0.2)
coef <- morlet_wavelet(x, freq, 100, c(2,3))

```

```
# to get coefficients in complex number from 1-10 time points
coef[1:10, ]

# power
power <- Mod(coef[])^2

# Power peaks at 5Hz and 10Hz at early stages
# After 1.0 second, 5Hz component fade away
image(power, x = time, y = freq, ylab = "frequency")

# wavelet using double precision
coef2 <- morlet_wavelet(x, freq, 100, c(2,3), precision = "double")
power2 <- (coef2$real[])^2 + (coef2$imag[])^2

image(power2, x = time, y = freq, ylab = "frequency")

# The maximum relative change of power with different precisions
max(abs(power/power2 - 1))

# display kernels
freq <- seq(1, 15, 1)
kern <- wavelet_kernels(freq, 100, c(2,3))
print(kern)

plot(kern)
```

Index

* datasets

- left_hippocampus_mask, 24
- as_matrix4 (new_matrix4), 28
- as_quaternion (new_quaternion), 28
- as_vector3 (new_vector3), 29
- band_pass, 3
- band_pass1 (band_pass), 3
- band_pass2 (band_pass), 3
- baseline_array, 5
- collapse, 8
- convolve, 10
- convolve_image (convolve), 10
- convolve_signal (convolve), 10
- convolve_volume (convolve), 10
- cov, 15
- decimate, 11
- detect_threads (parallel-options), 31
- detrend, 12, 50
- diagnose_channel, 13
- fast_cov, 15
- fast_median (fast_quantile), 16
- fast_mvmedian (fast_quantile), 16
- fast_mvquantile (fast_quantile), 16
- fast_quantile, 16
- fill_surface, 18
- filter_signal, 19
- filtfilt, 20
- get0, 22
- grow_volume, 21
- hist, 13, 14
- internal_rave_function, 22
- interpolate_stimulation, 22
- left_hippocampus_mask, 24
- matlab_palette, 24
- mesh_from_volume, 24
- morlet_wavelet (wavelet), 49
- multitaper, 26
- multitaper_config (multitaper), 26
- mv_pwelch (pwelch), 33
- new_matrix4, 28, 29
- new_quaternion, 28, 28, 29
- new_vector3, 28, 29, 29
- niftyreg, 39
- notch_filter, 30
- par, 14, 33
- parallel-options, 31
- plot, 33
- plot.default, 35
- plot.pwelch (pwelch), 33
- plot_signals, 32
- print.pwelch (pwelch), 33
- pwelch, 13, 14, 33
- ravetools_threads (parallel-options), 31
- raw-to-sexp, 36
- raw_to_float (raw-to-sexp), 36
- raw_to_int16 (raw-to-sexp), 36
- raw_to_int32 (raw-to-sexp), 36
- raw_to_int64 (raw-to-sexp), 36
- raw_to_int8 (raw-to-sexp), 36
- raw_to_string (raw-to-sexp), 36
- raw_to_uint16 (raw-to-sexp), 36
- raw_to_uint32 (raw-to-sexp), 36
- raw_to_uint8 (raw-to-sexp), 36
- rawToChar, 37
- register_volume, 38
- rgl-call, 40
- rgl_call (rgl-call), 40
- rgl_plot_normals (rgl-call), 40
- rgl_view (rgl-call), 40
- shift_array, 41

vcg_isosurface, [42](#)
vcg_mesh_volume, [24](#), [43](#)
vcg_smooth, [44](#)
vcg_smooth_explicit, [24](#), [25](#)
vcg_smooth_explicit (vcg_smooth), [44](#)
vcg_smooth_implicit, [24](#)
vcg_smooth_implicit (vcg_smooth), [44](#)
vcg_sphere, [46](#)
vcg_uniform_remesh, [24](#), [25](#), [47](#)
vcg_update_normals, [48](#)

wavelet, [49](#)
wavelet_cycles_suggest (wavelet), [49](#)
wavelet_kernels (wavelet), [49](#)