Package 'gsignal'

September 5, 2024

Type Package

Title Signal Processing

Version 0.3-6

Date 2024-09-04

Description R implementation of the 'Octave' package 'signal', containing a variety of signal processing tools, such as signal generation and measurement, correlation and convolution, filtering, filter design, filter analysis and conversion, power spectrum analysis, system identification, decimation and sample rate change, and windowing.

Depends R (>= 3.5.0)

LinkingTo Rcpp

Imports pracma, Rcpp, grDevices

License GPL-3

Encoding UTF-8

Language en-US

URL https://github.com/gjmvanboxtel/gsignal

BugReports https://github.com/gjmvanboxtel/gsignal/issues

LazyData true

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat, ggplot2, gridExtra, microbenchmark, covr

VignetteBuilder knitr

NeedsCompilation yes

Author Geert van Boxtel [aut, cre] (Maintainer),
Tom Short [aut] (Author of 'signal' package),
Paul Kienzle [aut] (Majority of the original sources),
Ben Abbott [ctb],
Juan Aguado [ctb],
Muthiah Annamalai [ctb],
Leonardo Araujo [ctb],

William Asquith [ctb],
David Bateman [ctb],
David Billinghurst [ctb],
Juan Pablo Carbajal [ctb],
André Carezia [ctb],
Vincent Cautaerts [ctb],
Eric Chassande-Mottin [ctb],
Luca Citi [ctb],
Dave Cogdell [ctb],
Carlo de Falco [ctb],
Carne Draug [ctb],
Pascal Dupuis [ctb],
John W. Eaton [ctb],
R.G.H Eschauzier [ctb],
Andrew Fitting [ctb],
Alan J. Greenberger [ctb],
Mike Gross [ctb],
Daniel Gunyan [ctb],
Kai Habel [ctb],
Kurt Hornik [ctb],
Jake Janovetz [ctb],
Alexander Klein [ctb],
Peter V. Lanspeary [ctb],
Bill Lash [ctb],
Friedrich Leissh [ctb],
Laurent S. Mazet [ctb],
Mike Miller [ctb],
Petr Mikulik [ctb],
Paolo Neis [ctb],
Georgios Ouzounis [ctb],
Sylvain Pelissier [ctb],
Francesco Potortì [ctb],
Charles Praplan [ctb],
Lukas F. Reichlin [ctb],
Tony Richardson [ctb],
Asbjorn Sabo [ctb],
Thomas Sailer [ctb],
Rolf Schirmacher [ctb],
Rolf Schirmacher [ctb],
Ivan Selesnick [ctb],
Julius O. Smith III [ctb],
Peter L. Soendergaard [ctb],
Quentin Spencer [ctb],
Doug Stewart [ctb],
P. Sudeepam [ctb],
Stefan van der Walt [ctb],
Andreas Weber [ctb],
P. Sudeepam [ctb],

Andreas Weingessel [ctb]

Maintainer Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>

Repository CRAN

Date/Publication 2024-09-04 22:30:02 UTC

Contents

arburg *Autoregressive model coefficients - Burg's method*

Description

Calculate the coefficients of an autoregressive model using the whitening lattice-filter method of Burg (1968)[1].

Usage

```
arburg(x, p, criterion = NULL)
```

Arguments

x	input data, specified as a numeric or complex vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
p	model order; number of poles in the AR model or limit to the number of poles if a valid criterion is provided. Must be < length(x) - 2.
criterion	model-selection criterion. Limits the number of poles so that spurious poles are not added when the whitened data has no more information in it. Recognized values are: **AKICc** approximate corrected Kullback information criterion (recommended) **KIC** Kullback information criterion **AICc** corrected Akaike information criterion **AIC** Akaike information criterion **FPE** final prediction error The default is to NOT use a model-selection criterion (NULL)

Details

The inverse of the autoregressive model is a moving-average filter which reduces x to white noise. The power spectrum of the AR model is an estimate of the maximum entropy power spectrum of the data. The function ar_psd calculates the power spectrum of the AR model.

For data input x(n) and white noise e(n), the autoregressive model is

$$x(n) = \sqrt{v} \cdot e(n) + \sum_{k=1}^{p+1} a(k) \cdot x(n-k)$$

arburg does not remove the mean from the data. You should remove the mean from the data if you want a power spectrum. A non-zero mean can produce large errors in a power-spectrum estimate. See detrend

Value

A `list` containing the following elements:

a vector or matrix containing `(p+1)` autoregression coefficients. If `x` is a matrix, then each row of a corresponds to a column of `x`. `a` has `p + 1` columns.

e white noise input variance, returned as a vector. If `x` is a matrix, then each element of e corresponds to a column of `x`.

k Reflection coefficients defining the lattice-filter embodiment of the model returned as vector or a matrix. If `x` is a matrix, then each column of `k` corresponds to a column of `x`. `k` has `p` rows.

Note

AIC, AICc, KIC and AKICc are based on information theory. They attempt to balance the complexity (or length) of the model against how well the model fits the data. AIC and KIC are biased estimates of the asymmetric and the symmetric Kullback-Leibler divergence, respectively. AICc and AKICc attempt to correct the bias. See reference [2].

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>.

References

[1] Burg, J.P. (1968) A new analysis technique for time series data, NATO advanced study Institute on Signal Processing with Emphasis on Underwater Acoustics, Enschede, Netherlands, Aug. 12-23, 1968.
[2] Seghouane, A. and Bekara, M. (2004). A small sample model selection criterion based on Kullback's symmetric divergence. IEEE Trans. Sign. Proc., 52(12), pp 3314-3323,

See Also

`ar_psd`

Examples

```
A <- Arma(1, c(1, -2.7607, 3.8106, -2.6535, 0.9238))
y <- filter(A, 0.2 * rnorm(1024))
coefs <- arburg(y, 4)
```

Arma *Autoregressive moving average (ARMA) model*

Description

Create an ARMA model representing a filter or system model, or convert other forms to an ARMA model.

Usage

```
Arma(b, a)

as.Arma(x, ...)

## S3 method for class 'Arma'
as.Arma(x, ...)

## S3 method for class 'Ma'
as.Arma(x, ...)

## S3 method for class 'Sos'
as.Arma(x, ...)

## S3 method for class 'Zpg'
as.Arma(x, ...)
```

Arguments

b	moving average (MA) polynomial coefficients.
a	autoregressive (AR) polynomial coefficients.
x	model or filter to be converted to an ARMA representation.
...	additional arguments (ignored).

Details

The ARMA model is defined by:

$$a(L)y(t) = b(L)x(t)$$

The ARMA model can define an analog or digital model. The AR and MA polynomial coefficients follow the convention in 'Matlab' and 'Octave' where the coefficients are in decreasing order of the polynomial (the opposite of the definitions for filterfilter and polyroot). For an analog model,

$$H(s) = (b_1 s^{(m-1)} + b_2 s^{(m-2)} + \ldots + b_m)/(a_1 s^{(n-1)} + a_2 s^{(n-2)} + \ldots + a_n)$$

For a z-plane digital model,

$$H(z) = (b_1 + b_2 z^{-1} + \ldots + b_m z^{(-m+1)})/(a_1 + a_2 z^{-1} + \ldots + a_n z^{(-n+1)})$$

as.Arma converts from other forms, including Zpg and Ma.

Value

A list of class 'Arma' with the following list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Tom Short, <tshort@eprisolutions.com>,
adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>.

See Also

See also Zpg, Ma, filter, and various filter-generation functions like butter and cheby1 that return Arma models.

Examples

```
filt <- Arma(b = c(1, 2, 1)/3, a = c(1, 1))
zplane(filt)
```

---

aryule *Autoregressive model coefficients - Yule-Walker method*

Description

compute autoregressive all-pole model parameters using the Yule-Walker method.

Usage

```
aryule(x, p)
```

Arguments

x	input data, specified as a numeric or complex vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
p	model order; number of poles in the AR model or limit to the number of poles if a valid criterion is provided. Must be smaller than the length of x minus 1.

Details

aryule uses the Levinson-Durbin recursion on the biased estimate of the sample autocorrelation sequence to compute the parameters.

Value

A `list` containing the following elements:

a vector or matrix containing `(p + 1)` autoregression coefficients. If `x` is a matrix, then each row of a corresponds to a column of `x`. `a` has `p + 1` columns.

e white noise input variance, returned as a vector. If `x` is a matrix, then each element of e corresponds to a column of `x`.

k Reflection coefficients defining the lattice-filter embodiment of the model returned as vector or a matrix. If `x` is a matrix, then each column of `k` corresponds to a column of `x`. `k` has `p` rows.

Note

The power spectrum of the resulting filter can be plotted with `pyulear(x, p)`, or you can plot it directly with `ar_psd(a,v,...)`.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>.

See Also

ar_psd, arburg

Examples

```
a <- Arma(1, c(1, -2.7607, 3.8106, -2.6535, 0.9238))
y <- filter(a, rnorm(1024))
coefs <- aryule(y, 4)
```

---

ar_psd *Power spectrum of AR model*

Description

Compute the power spectral density of an autoregressive model.

Usage

```
ar_psd(
  a,
  v = 1,
  freq = 256,
  fs = 1,
  range = ifelse(is.numeric(a), "half", "whole"),
```

```
  method = ifelse(length(freq) == 1 && bitwAnd(freq, freq - 1) == 0, "fft", "poly")
)

## S3 method for class 'ar_psd'
plot(
  x,
  yscale = c("linear", "log", "dB"),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  ...
)

## S3 method for class 'ar_psd'
print(
  x,
  yscale = c("linear", "log", "dB"),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  ...
)
```

Arguments

a	numeric vector of autoregressive model coefficients. The first element is the zero-lag coefficient, which always has a value of 1.
v	square of the moving average coefficient, specified as a positive scalar Default: 1
freq	vector of frequencies at which power spectral density is calculated, or a scalar indicating the number of uniformly distributed frequency values at which spectral density is calculated. Default: 256.
fs	sampling frequency (Hz). Default: 1
range	character string. one of:

`"half"` **or** `"onesided"` frequency range of the spectrum is from zero up to but not including `fs / 2`. Power from negative frequencies is added to the positive side of the spectrum.

`"whole"` **or** `"twosided"` frequency range of the spectrum is `-fs / 2` to `fs / 2`, with negative frequencies stored in "wrap around order" after the positive frequencies; e.g. frequencies for a 10-point `"twosided"` spectrum are 0 0.1 0.2 0.3 0.4 0.5 -0.4 -0.3 -0.2. -0.1.

`"shift"` **or** `"centerdc"` same as `"whole"` but with the first half of the spectrum swapped with second half to put the zero-frequency value in the middle. If `freq` is a vector, `"shift"` is ignored.

Default: If model coefficients `a` are real, the default range is `"half"`, otherwise the default range is `"whole"`.

method	method used to calculate the power spectral density, either "fft" (use the Fast Fourier Transform) or "poly" (calculate the power spectrum as a polynomial). This argument is ignored if the freq argument is a vector. The default is "poly" unless the freq argument is an integer power of 2.
x	object to plot.
yscale	character string specifying scaling of Y-axis; one of "linear", "log", "dB"
xlab, ylab, main	labels passed to plotting function. Default: NULL
...	additional arguments passed to functions

Details

This function calculates the power spectrum of the autoregressive model

$$x(n) = \sqrt{v} \cdot e(n) + \sum_{k=1}^{M} a(k) \cdot x(n-k)$$

where x(n) is the output of the model and e(n) is white noise.

Value

An object of class "ar_psd" , which is a list containing two elements, freq and psd containing the frequency values and the estimates of power-spectral density, respectively.

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>

Examples

```
a <- c(1, -2.7607, 3.8106, -2.6535, 0.9238)
psd <- ar_psd(a)
```

---

barthannwin *Modified Bartlett-Hann window*

Description

Return the filter coefficients of a modified Bartlett-Hann window.

Usage

```
barthannwin(n)
```

Arguments

n	Window length, specified as a positive integer.

Details

Like Bartlett, Hann, and Hamming windows, the Bartlett-Hann window has a mainlobe at the origin and asymptotically decaying sidelobes on both sides. It is a linear combination of weighted Bartlett and Hann windows with near sidelobes lower than both Bartlett and Hann and with far sidelobes lower than both Bartlett and Hamming windows. The mainlobe width of the modified Bartlett-Hann window is not increased relative to either Bartlett or Hann window mainlobes.

Value

Modified Bartlett-Hann window, returned as a vector. If you specify a one-point window (`n = 1`), the value 1 is returned.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>. Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

bartlett, hann, hamming

Examples

```
t <- barthannwin(64)
plot (t, type = "l", xlab = "Samples", ylab =" Amplitude")
```

---

bartlett *Bartlett window*

Description

Return the filter coefficients of a Bartlett (triangular) window.

Usage

```
bartlett(n)
```

Arguments

n	Window length, specified as a positive integer.

Details

The Bartlett window is very similar to a triangular window as returned by the `triang` function. However, the Bartlett window always has zeros at the first and last samples, while the triangular window is nonzero at those points.

Value

Bartlett window, returned as a vector. If you specify a one-point window (`n = 1`), the value 1 is returned.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>. Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`triang`

Examples

```
bw <- bartlett(64)
plot (bw, type = "l", xlab = "Samples", ylab =" Amplitude")
```

---

besselap *Bessel analog low-pass filter prototype*

Description

Return the poles and gain of a Bessel analog low-pass filter prototype.

Usage

```
besselap(n)
```

Arguments

`n`	order of the filter; must be < 25.

Details

The transfer function is

$$H(s) = \frac{k}{(s - p(1))(s - p(2))...(s - p(n))}$$

besselap normalizes the poles and gain so that at low frequency and high frequency the Bessel prototype is asymptotically equivalent to the Butterworth prototype of the same order. The magnitude of the filter is less than $1/\sqrt{2}$ at the unity cutoff frequency $\Omega_c = 1$.

Analog Bessel filters are characterized by a group delay that is maximally flat at zero frequency and almost constant throughout the passband. The group delay at zero frequency is

$$\left(\frac{(2n)!}{2^n n!}\right)^{1/n}$$

Value

List of class Zpg containing poles and gain of the filter

Author(s)

Thomas Sailer, <t.sailer@alumni.ethz.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Bessel_polynomials

Examples

```
## 6th order Bessel low-pass analog filter
zp <- besselap(6)
w <- seq(0, 4, length.out = 128)
freqs(zp, w)
```

---

besself *Bessel analog filter design*

Description

Compute the transfer function coefficients of an analog Bessel filter.

Usage

```
besself(n, w, type = c("low", "high", "stop", "pass"))
```

Arguments

n	filter order.
w	critical frequencies of the filter. w must be a scalar for low-pass and high-pass filters, and w must be a two-element vector c(low, high) specifying the lower and upper bands in radians/second.
type	filter type, one of "low" (default), "high", "stop", or "pass".

Details

Bessel filters are characterized by an almost constant group delay across the entire passband, thus preserving the wave shape of filtered signals in the passband.

Lowpass Bessel filters have a monotonically decreasing magnitude response, as do lowpass Butterworth filters. Compared to the Butterworth, Chebyshev, and elliptic filters, the Bessel filter has the slowest rolloff and requires the highest order to meet an attenuation specification.

Value

List of class 'Zpg' containing poles and gain of the filter.

Note

As the important characteristic of a Bessel filter is its maximally-flat group delay, and not the amplitude response, it is inappropriate to use the bilinear transform to convert the analog Bessel filter into a digital form (since this preserves the amplitude response but not the group delay) [1].

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Doug Stewart, <dastew@sympatico.ca>,
Thomas Sailer, <t.sailer@alumni.ethz.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Bessel_filter

Examples

```
w <- seq(0, 4, length.out = 128)

## 5th order Bessel low-pass analog filter
zp <- besself(5, 1.0)
freqs(zp, w)

## 5th order Bessel high-pass analog filter
zp <- besself(5, 1.0, 'high')
freqs(zp, w)

## 5th order Bessel band-pass analog filter
```

```
zp <- besself(5, c(1, 2), 'pass')
freqs(zp, w)

## 5th order Bessel band-stop analog filter
zp <- besself(5, c(1, 2), 'stop')
freqs(zp, w)
```

`bilinear` *Bilinear transformation*

Description

Transform a s-plane (analog) filter specification into a z-plane (digital) specification.

Usage

```
bilinear(Sz, ...)

## S3 method for class 'Zpg'
bilinear(Sz, T = 2 * tan(1/2), ...)

## S3 method for class 'Arma'
bilinear(Sz, T = 2 * tan(1/2), ...)

## Default S3 method:
bilinear(Sz, Sp, Sg, T = 2 * tan(1/2), ...)
```

Arguments

`Sz`	In the generic case, a model to be transformed. In the default case, a vector containing the zeros in a pole-zero-gain model.
`...`	arguments passed to the generic function.
`T`	the sampling frequency represented in the z plane. Default: `2 * tan(1 / 2)`.
`Sp`	a vector containing the poles in a pole-zero-gain model.
`Sg`	a vector containing the gain in a pole-zero-gain model.

Details

Given a piecewise flat filter design, you can transform it from the s-plane to the z-plane while maintaining the band edges by means of the bilinear transform. This maps the left hand side of the s-plane into the interior of the unit circle. The mapping is highly non-linear, so you must design your filter with band edges in the s-plane positioned at $2/Ttan(wT/2)$ so that they will be positioned at `w` after the bilinear transform is complete.

The bilinear transform is:

$$z = (1 + sT/2)/(1 - sT/2)$$

$$s = (T/2)(z-1)/(z+1)$$

Please note that a pole and a zero at the same place exactly cancel. This is significant since the bilinear transform creates numerous extra poles and zeros, most of which cancel. Those which do not cancel have a "fill-in" effect, extending the shorter of the sets to have the same number of as the longer of the sets of poles and zeros (or at least split the difference in the case of the band pass filter). There may be other opportunistic cancellations, but it will not check for them.

Also note that any pole on the unit circle or beyond will result in an unstable filter. Because of cancellation, this will only happen if the number of poles is smaller than the number of zeros. The analytic design methods all yield more poles than zeros, so this will not be a problem.

Value

For the default case or for bilinear.Zpg, an object of class `'Zpg'`, containing the list elements:

z complex vector of the zeros of the transformed model

p complex vector of the poles of the transformed model

g gain of the transformed model

For bilinear.Arma, an object of class `'Arma'`, containing the list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Paul Kienzle <pkienzle@users.sf.net>. Conversion to R by Tom Short, adapted by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Bilinear_transform

Examples

```
## 6th order Bessel low-pass analog filter
zp <- besselap(6)
w <- seq(0, 4, length.out = 128)
freqs(zp, w)
zzp <- bilinear(zp)
freqz(zzp)
```

`bitrevorder` *Permute input to bit-reversed order*

Description

Reorder the elements of the input vector in bit-reversed order.

Usage

```
bitrevorder(x, index.return = FALSE)
```

Arguments

`x`	input data, specified as a vector. The length of `x` must be an integer power of 2.
`index.return`	logical indicating if the ordering index vector should be returned as well. Default: `FALSE`.

Details

This function is equivalent to calling `digitrevorder(x, 2)`, and is useful for prearranging filter coefficients so that bit-reversed ordering does not have to be performed as part of an fft or ifft computation.

Value

The bit-reversed input vector. If `index.return = TRUE`, then a list containing the bit-reversed input vector (`y`), and the digit-reversed indices (`i`).

Author(s)

Mike Miller.
Port to to by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`digitrevorder`, `fft`, `ifft`

Examples

```
x <- 0:15
v <- bitrevorder(x)
dec2bin <- function(x, l)
  substr(paste(as.integer(rev(intToBits(x))), collapse = ""),
  32 - l + 1, 32)
x_bin <- sapply(x, dec2bin, 4)
v_bin <- sapply(v, dec2bin, 4)
data.frame(x, x_bin, v, v_bin)
```

blackman *Blackman window*

Description

Return the filter coefficients of a Blackman window.

Usage

```
blackman(n, method = c("symmetric", "periodic"))
```

Arguments

n	Window length, specified as a positive integer.
method	Character string. Window sampling method, specified as:

"symmetric" (Default) Use this option when using windows for filter design.

"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When "periodic" is specified, the function computes a window of length n + 1 and returns the first n points.

Details

The Blackman window is a member of the family of cosine sum windows.

Value

Blackman window, returned as a vector.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
h <- blackman(64)
plot (h, type = "l", xlab = "Samples", ylab =" Amplitude")

bs = blackman(64,'symmetric')
bp = blackman(63,'periodic')
plot (bs, type = "l", xlab = "Samples", ylab =" Amplitude")
lines(bp, col="red")
```

`blackmanharris` *Blackman-Harris window*

Description

Return the filter coefficients of a minimum four-term Blackman-Harris window.

Usage

```
blackmanharris(n, method = c("symmetric", "periodic"))
```

Arguments

`n`	Window length, specified as a positive integer.
`method`	Character string. Window sampling method, specified as:
	"symmetric" (Default) Use this option when using windows for filter design.
	"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When "periodic" is specified, the function computes a window of length `n + 1` and returns the first `n` points.

Details

The Blackman window is a member of the family of cosine sum windows. It is a generalization of the Hamming family, produced by adding more shifted sinc functions, meant to minimize side-lobe levels.

Value

Blackman-Harris window, returned as a vector.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
b <- blackmanharris(64)
plot (b, type = "l", xlab = "Samples", ylab =" Amplitude")

bs = blackmanharris(64,'symmetric')
bp = blackmanharris(63,'periodic')
plot (bs, type = "l", xlab = "Samples", ylab =" Amplitude")
lines(bp, col="red")
```

`blackmannuttall` *Blackman-Nuttall window*

Description

Return the filter coefficients of a Blackman-Nuttal window.

Usage

```
blackmannuttall(n, method = c("symmetric", "periodic"))
```

Arguments

`n`	Window length, specified as a positive integer.
`method`	Character string. Window sampling method, specified as:

"symmetric" (Default) Use this option when using windows for filter design.

"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When "periodic" is specified, the function computes a window of length `n + 1` and returns the first `n` points.

Details

The Blackman-Nuttall window is a member of the family of cosine sum windows.

Value

Blackman-Nuttall window, returned as a vector.

Author(s)

Muthiah Annamalai, <muthiah.annamalai@uta.edu>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
b <- blackmannuttall(64)
plot (b, type = "l", xlab = "Samples", ylab =" Amplitude")

bs = blackmannuttall(64,'symmetric')
bp = blackmannuttall(63,'periodic')
plot (bs, type = "l", xlab = "Samples", ylab =" Amplitude")
lines(bp, col="red")
```

bohmanwin *Bohman window*

Description

Return the filter coefficients of a Bohman window.

Usage

```
bohmanwin(n)
```

Arguments

n	Window length, specified as a positive integer.

Details

A Bohman window is the convolution of two half-duration cosine lobes. In the time domain, it is the product of a triangular window and a single cycle of a cosine with a term added to set the first derivative to zero at the boundary.

Value

Bohman window, returned as a vector. If you specify a one-point window (`n = 1`), the value 1 is returned.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

triang

Examples

```
b <- bohmanwin(64)
plot (b, type = "l", xlab = "Samples", ylab =" Amplitude")
```

boxcar *Rectangular window*

Description

Return the filter coefficients of a boxcar (rectangular) window.

Usage

```
boxcar(n)
```

Arguments

n	Window length, specified as a positive integer.

Details

The rectangular window (sometimes known as the boxcar or Dirichlet window) is the simplest window, equivalent to replacing all but n values of a data sequence by zeros, making it appear as though the waveform suddenly turns on and off. Other windows are designed to moderate these sudden changes, which reduces scalloping loss and improves dynamic range.

Value

rectangular window, returned as a vector.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

triang

Examples

```
b <- boxcar(64)
plot (b, type = "l", xlab = "Samples", ylab =" Amplitude")
```

buffer *Buffer signal vector into matrix of data segments*

Description

Partition a signal vector into nonoverlapping, overlapping, or underlapping data segments.

Usage

```
buffer(x, n, p = 0, opt, zopt = FALSE)
```

Arguments

x	The data to be buffered.
n	The number of rows in the produced data buffer. This is an positive integer value and must be supplied.
p	An integer less than n that specifies the under- or overlap between column in the data frame. Default 0.
opt	In the case of an overlap, opt can be either a vector of length p or the string 'nodelay'. If opt is a vector, then the first p entries in y will be filled with these values. If opt is the string 'nodelay', then the first value of y corresponds to the first value of x. In the case of an underlap, opt must be an integer between 0 and -p. The represents the initial underlap of the first y. The default value for opt the vector matrix (0L, 1, p) in the case of an overlap, or 0 otherwise.
zopt	Logical. If TRUE, return values for z and opt in addition to y. Default is FALSE (return only y).

Details

y <- buffer(x, n) partitions a signal vector x of length L into nonoverlapping data segments of length n. Each data segment occupies one column of matrix output y, which has n rows and ceil(L / n) columns. If L is not evenly divisible by n, the last column is zero-padded to length n.

y <- buffer(x, n, p) overlaps or underlaps successive frames in the output matrix by p samples.

- For 0 < p < n (overlap), buffer repeats the final p samples of each segment at the beginning of the following segment. See the example where x = 1:30, n = 7, and an overlap of p = 3. In this case, the first segment starts with p zeros (the default initial condition), and the number of columns in y is ceil(L / (n - p)).
- For p < 0 (underlap), buffer skips p samples between consecutive segments. See the example where x = 1:30, n = 7, and p = -3. The number of columns in y is ceil(L / (n -p)).

In y <- buffer(x, n, p, opt), opt specifies a vector of samples to precede x[1] in an overlapping buffer, or the number of initial samples to skip in an underlapping buffer.

- For `0 < p < n` (overlap), `opt` specifies a vector of length `p` to insert before `x[1]` in the buffer. This vector can be considered an initial condition, which is needed when the current buffering operation is one in a sequence of consecutive buffering operations. To maintain the desired segment overlap from one buffer to the next, `opt` should contain the final `p` samples of the previous buffer in the sequence. Set `opt` to `"nodelay"` to skip the initial condition and begin filling the buffer immediately with `x[1]`. In this case, `L` must be `length(p)` or longer. See the example where `x = 1:30`, `n = 7`, `p = 3`, and `opt = "nodelay"`.
- For `p < 0` (underlap), `opt` is an integer value in the range `0 : -p` specifying the number of initial input samples, `x[1:opt]`, to skip before adding samples to the buffer. The first value in the buffer is therefore `x[opt + 1]`.

The `opt` option is especially useful when the current buffering operation is one in a sequence of consecutive buffering operations. To maintain the desired frame underlap from one buffer to the next, `opt` should equal the difference between the total number of points to skip between frames (`p`) and the number of points that were available to be skipped in the previous input to buffer. If the previous input had fewer than p points that could be skipped after filling the final frame of that buffer, the remaining opt points need to be removed from the first frame of the current buffer. See Continuous Buffering for an example of how this works in practice.

`buf <- buffer(..., zopt = TRUE)` returns the last `p` samples of a overlapping buffer in output `buf$opt`. In an underlapping buffer, `buf$opt` is the difference between the total number of points to skip between frames (`-p`) and the number of points in `x` that were available to be skipped after filling the last frame:

- For `0 < p < n` (overlap), `buf$opt` contains the final `p` samples in the last frame of the buffer. This vector can be used as the initial condition for a subsequent buffering operation in a sequence of consecutive buffering operations. This allows the desired frame overlap to be maintained from one buffer to the next. See Continuous Buffering below.
- For `p < 0` (underlap), `buf$opt` is the difference between the total number of points to skip between frames `(-p)` and the number of points in `x` that were available to be skipped after filling the last frame: `buf$opt = m*(n-p) + opt - L` where `opt` on the right is the input argument to buffer, and `buf$opt` on the left is the output argument. Note that for an underlapping buffer output `buf$opt` is always zero when output `buf$z` contains data.
  The opt output for an underlapping buffer is especially useful when the current buffering operation is one in a sequence of consecutive buffering operations. The `buf$opt` output from each buffering operation specifies the number of samples that need to be skipped at the start of the next buffering operation to maintain the desired frame underlap from one buffer to the next. If fewer than `p` points were available to be skipped after filling the final frame of the current buffer, the remaining opt points need to be removed from the first frame of the next buffer.

In a sequence of buffering operations, the `buf$opt` output from each operation should be used as the `opt` input to the subsequent buffering operation. This ensures that the desired frame overlap or underlap is maintained from buffer to buffer, as well as from frame to frame within the same buffer. See Continuous Buffering below for an example of how this works in practice.

Continuous Buffering

In a continuous buffering operation, the vector input to the buffer function represents one frame in a sequence of frames that make up a discrete signal. These signal frames can originate in a

frame-based data acquisition process, or within a frame-based algorithm like the FFT.
As an example, you might acquire data from an A/D card in frames of 64 samples. In the simplest case, you could rebuffer the data into frames of 16 samples; `buffer` with `n = 16` creates a buffer of four frames from each 64-element input frame. The result is that the signal of frame size 64 has been converted to a signal of frame size 16; no samples were added or removed.
In the general case where the original signal frame size, `L`, is not equally divisible by the new frame size, `n`, the overflow from the last frame needs to be captured and recycled into the following buffer. You can do this by iteratively calling buffer on input `x` with the `zopt` parameter set to `TRUE`. This simply captures any buffer overflow in `buf$z`, and prepends the data to the subsequent input in the next call to buffer.
Note that continuous buffering cannot be done without the `zopt` parameter being set to `TRUE`, because the last frame of `y` (`buf$y` in this case) is zero padded, which adds new samples to the signal. Continuous buffering in the presence of overlap and underlap is handled with the `opt` parameter, which is used as both an input (`opt` and output (`buf$opt`) to buffer. The two examples on this page demonstrate how the `opt` parameter should be used.

Value

If `zopt` equals FALSE (the default), this function returns a single numerical array containing the buffered data (`y`). If `zopt` equals TRUE, then a `list` containing 3 variables is returned: `y`: the buffered data, `z`: the over or underlap (if any), `opt`: the over- or underlap that might be used for a future call to `buffer` to allow continuous buffering.

Author(s)

David Bateman, <adb014@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

Examples

```
## Examples without continuous buffering
y <- buffer(1:10, 5)
y <- buffer(1:10, 4)
y <- buffer(1:30, 7, 3)
y <- buffer(1:30, 7, -3)
y <- buffer(1:30, 7, 3, 'nodelay')

## Continuous buffering examples
# with overlap:
data <- buffer(1:1100, 11)
n <- 4
p <- 1
buf <- list(y = NULL, z = NULL, opt = -5)
for (i in 1:ncol(data)) {
  x <- data[,i]
  buf <- buffer(x = c(buf$z,x), n, p, opt=buf$opt, zopt = TRUE)
}
# with underlap:
data <- buffer(1:1100, 11)
n <- 4
p <- -2
```

```
buf <- list(y = NULL, z = NULL, opt = 1)
for (i in 1:ncol(data)) {
  x <- data[,i]
  buf <- buffer(x = c(buf$z,x), n, p, opt=buf$opt, zopt = TRUE)
}
```

---

buttap *Butterworth filter prototype*

Description

Return the poles and gain of an analog Butterworth lowpass filter prototype.

Usage

```
buttap(n)
```

Arguments

n	Order of the filter.

Details

This function exists for compatibility with 'Matlab' and 'Octave' only, and is equivalent to `butter(n, 1, "low", "s")`.

Value

List of class Zpg containing poles and gain of the filter.

Author(s)

Carne Draug, <carandraug+dev@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## 9th order Butterworth low-pass analog filter
zp <- buttap(9)
w <- seq(0, 4, length.out = 128)
freqs(zp, w)
```

butter *Butterworth filter design*

Description

Compute the transfer function coefficients of a Butterworth filter.

Usage

```
butter(n, ...)

## S3 method for class 'FilterSpecs'
butter(n, ...)

## Default S3 method:
butter(
  n,
  w,
  type = c("low", "high", "stop", "pass"),
  plane = c("z", "s"),
  output = c("Arma", "Zpg", "Sos"),
  ...
)
```

Arguments

n	filter order.
...	additional arguments passed to butter, overriding those given by n of class FilterSpecs.
w	critical frequencies of the filter. w must be a scalar for low-pass and high-pass filters, and w must be a two-element vector c(low, high) specifying the lower and upper bands in radians/second. For digital filters, w must be between 0 and 1 where 1 is the Nyquist frequency.
type	filter type, one of "low", (default) "high", "stop", or "pass".
plane	"z" for a digital filter or "s" for an analog filter.
output	Type of output, one of: **"Arma"** Autoregressive-Moving average (aka numerator/denominator, aka b/a) **"Zpg"** Zero-pole-gain format **"Sos"** Second-order sections Default is "Arma" for compatibility with the 'signal' package and the 'Matlab' and 'Octave' equivalents, but "Sos" should be preferred for general-purpose filtering because of numeric stability.

Details

Butterworth filters have a magnitude response that is maximally flat in the passband and monotonic overall. This smoothness comes at the price of decreased rolloff steepness. Elliptic and Chebyshev filters generally provide steeper rolloff for a given filter order.

Because butter is generic, it can be extended to accept other inputs, using `buttord` to generate filter criteria for example.

Value

Depending on the value of the `output` parameter, a list of class `Arma`, `Zpg`, or `Sos` containing the filter coefficients

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Doug Stewart, <dastew@sympatico.ca>,
Alexander Klein, <alexander.klein@math.uni-giessen.de>,
John W. Eaton.
Conversion to R by Tom Short,
adapted by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Butterworth_filter

See Also

`Arma`, `Zpg`, `Sos`, `filter`, `cheby1`, `ellip`, `buttord`.

Examples

```
## 50 Hz notch filter
fs <- 256
bf <- butter(4, c(48, 52) / (fs / 2), "stop")
freqz(bf, fs = fs)

## EEG alpha rhythm (8 - 12 Hz) bandpass filter
fs <- 128
fpass <- c(8, 12)
wpass <- fpass / (fs / 2)
but <- butter(5, wpass, "pass")
freqz(but, fs = fs)

## filter to remove vocals from songs, 25 dB attenuation in stop band
## (not optimal with a Butterworth filter)
fs <- 44100
specs <- buttord(230/(fs/2), 450/(fs/2), 1, 25)
bf <- butter(specs)
freqz(bf, fs = fs)
zplane(bf)
```

buttord *Butterworth filter order and cutoff frequency*

Description

Compute the minimum filter order of a Butterworth filter with the desired response characteristics.

Usage

```
buttord(Wp, Ws, Rp, Rs, plane = c("z", "s"))
```

Arguments

Wp, Ws	pass-band and stop-band edges. For a low-pass or high-pass filter, Wp and Ws are scalars. For a band-pass or band-rejection filter, both are vectors of length 2. For a low-pass filter, Wp < Ws. For a high-pass filter, Ws > Wp. For a band-pass (Ws[1] < Wp[1] < Wp[2] < Ws[2]) or band-reject (Wp[1] < Ws[1] < Ws[2] < Wp[2]) filter design, Wp gives the edges of the pass band, and Ws gives the edges of the stop band. For digital filters, frequencies are normalized to [0, 1], corresponding to the range [0, fs/2]. In case of an analog filter, all frequencies are specified in radians per second.
Rp	allowable decibels of ripple in the pass band.
Rs	minimum attenuation in the stop band in dB.
plane	"z" for a digital filter or "s" for an analog filter.

Details

Deriving the order and cutoff is based on:

$$|H(W)|^2 = 1/[1 + (W/Wc)^{(2N)}] = 10^{(-R/10)}$$

With some algebra, you can solve simultaneously for Wc and N given Ws, Rs and Wp,Rp. Rounding N to the next greater integer, one can recalculate the allowable range for Wc (filter characteristic touching the pass band edge or the stop band edge).

For other types of filter, before making the above calculation, the requirements must be transformed to lowpass requirements. After the calculation, Wc must be transformed back to the original filter type.

Value

A list of class FilterSpecs with the following list elements:

n filter order

Wc cutoff frequency

type filter type, normally one of "low", "high", "stop", or "pass".

Author(s)

Paul Kienzle,
adapted by Charles Praplan.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

butter, FilterSpecs

Examples

```
## low-pass 30 Hz filter
fs <- 128
butspec <- buttord(30/(fs/2), 40/(fs/2), 0.5, 40)
but <- butter(butspec)
freqz(but, fs = fs)
```

---

cceps *Complex cepstral analysis*

Description

Return the complex cepstrum of the input vector.

Usage

```
cceps(x)
```

Arguments

x	input data, specified as a real vector.

Details

Cepstral analysis is a nonlinear signal processing technique that is applied most commonly in speech and image processing, or as a tool to investigate periodic structures within frequency spectra, for instance resulting from echos/reflections in the signal or to the occurrence of harmonic frequencies (partials, overtones).

The cepstrum is used in many variants. Most important are the power cepstrum, the complex cepstrum, and real cepstrum. The function cceps implements the complex cepstrum by computing the inverse of the log-transformed FFT, i.e.,

$$cceps(x) < -ifft(log(fft(x)))$$

However, because taking the logarithm of a complex number can lead to unexpected results, the phase of fft(x) needs to be unwrapped before taking the log.

Value

Complex cepstrum, returned as a vector.

Note

This function returns slightly different results in comparison with the 'Matlab' and 'Octave' equivalents. The 'Octave' version does not apply phase unwrapping, but has an optional correction procedure in case of zero phase at π radians. The present implementation does apply phase unwrapping so that the correction procedure is unnecessary. The 'Matlab' implementation also applies phase unwrapping, and a circular shift if necessary to avoid zero phase at π radians. The circular shift is not done here. In addition, the 'Octave' version shifts the zero frequency to the center of the series, which neither the 'Matlab' nor the present implementation do.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Cepstrum

See Also

rceps

Examples

```
## Generate a sine of frequency 45 Hz, sampled at 100 Hz.
fs <- 100
t <- seq(0, 1.27, 1/fs)
s1 <- sin(2 * pi * 45 * t)
## Add an echo with half the amplitude and 0.2 s later.
s2 <- s1 + 0.5 * c(rep(0L, 20), s1[1:108])
## Compute the complex cepstrum of the signal. Notice the echo at 0.2 s.
cep <- cceps(s2)
plot(t, cep, type="l")
```

---

cconv *Circular convolution*

Description

Compute the modulo-n circular convolution.

Usage

```
cconv(a, b, n = length(a) + length(b) - 1)
```

Arguments

a, b	Input, coerced to vectors, can be different lengths or data types.
n	Convolution length, specified as a positive integer. Default: `length(a) + length(b) - 1`.

Details

Linear and circular convolution are fundamentally different operations. Linear convolution of an n-point vector x, and an l-point vector y, has length n + l - 1, and can be computed by the function `conv`, which uses `filter`. The circular convolution, by contrast, is equal to the inverse discrete Fourier transform (DFT) of the product of the vectors' DFTs.

For the circular convolution of `x` and `y` to be equivalent to their linear convolution, the vectors must be padded with zeros to length at least `n + l - 1` before taking the DFT. After inverting the product of the DFTs, only the first `n + l - 1` elements should be retained.

For long sequences circular convolution may be more efficient than linear convolution. You can also use `cconv` to compute the circular cross-correlation of two sequences.

Value

Circular convolution of input vectors, returned as a vector.

Author(s)

Leonardo Araujo.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`conv`, `convolve`

Examples

```
a <- c(1, 2, -1, 1)
b <- c(1, 1, 2, 1, 2, 2, 1, 1)
c <- cconv(a, b)       # Circular convolution
cref = conv(a, b)      # Linear convolution
all.equal(max(c - cref), 0)

cconv(a, b, 6)
```

cheb *Chebyshev polynomials*

Description

Return the value of the Chebyshev polynomial at specific points.

Usage

```
cheb(n, x)
```

Arguments

n	Order of the polynomial, specified as a positive integer.
x	Point or points at which to calculate the Chebyshev polynomial

Details

The Chebyshev polynomials are defined by the equations:

$$Tn(x) = cos(n \cdot acos(x), |x| <= 1$$

$$Tn(x) = cosh(n \cdot acosh(x), |x| > 1$$

If x is a vector, the output is a vector of the same size, where each element is calculated as $y(i) = Tn(x(i))$.

Value

Polynomial of order x, evaluated at point(s) x.

Author(s)

André Carezia, <acarezia@uol.com.br>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
cp <- cheb(5, 1)
cp <- cheb(5, c(2,3))
```

cheb1ap *Chebyshev Type I filter prototype*

Description

Return the poles and gain of an analog Chebyshev Type I lowpass filter prototype.

Usage

```
cheb1ap(n, Rp)
```

Arguments

n	Order of the filter.
Rp	dB of pass-band ripple.

Details

This function exists for compatibility with 'Matlab' and 'OCtave' only, and is equivalent to `cheby1(n, Rp, 1, "low", "s")`.

Value

List of class Zpg containing the poles and gain of the filter.

Author(s)

Carne Draug, <carandraug+dev@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## 9th order Chebyshev type I low-pass analog filter
zp <- cheb1ap(9, .1)
w <- seq(0, 4, length.out = 128)
freqs(zp, w)
```

cheb1ord *Chebyshev Type I filter order*

Description

Compute Chebyshev type-I filter order and cutoff for the desired response characteristics.

Usage

```
cheb1ord(Wp, Ws, Rp, Rs, plane = c("z", "s"))
```

Arguments

Wp, Ws	pass-band and stop-band edges. For a low-pass or high-pass filter, Wp and Ws are scalars. For a band-pass or band-rejection filter, both are vectors of length 2. For a low-pass filter, Wp < Ws. For a high-pass filter, Ws > Wp. For a band-pass (Ws[1] < Wp[1] < Wp[2] < Ws[2]) or band-reject (Wp[1] < Ws[1] < Ws[2] < Wp[2]) filter design, Wp gives the edges of the pass band, and Ws gives the edges of the stop band. For digital filters, frequencies are normalized to [0, 1], corresponding to the range [0, fs/2]. In case of an analog filter, all frequencies are specified in radians per second.
Rp	allowable decibels of ripple in the pass band.
Rs	minimum attenuation in the stop band in dB.
plane	"z" for a digital filter or "s" for an analog filter.

Value

A list of class 'FilterSpecs' with the following list elements:

n filter order

Wc cutoff frequency

type filter type, normally one of "low", "high", "stop", or "pass".

Author(s)

Paul Kienzle, Laurent S. Mazet, Charles Praplan.
Conversion to R by Tom Short, adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

cheby1

Examples

```
## low-pass 30 Hz filter
fs <- 128
spec <- cheb1ord(30/(fs/2), 40/(fs/2), 0.5, 40)
cf <- cheby1(spec)
freqz(cf, fs = fs)
```

---

cheb2ap *Chebyshev Type II filter prototype*

Description

Return the poles and gain of an analog Chebyshev Type II lowpass filter prototype.

Usage

```
cheb2ap(n, Rs)
```

Arguments

n	Order of the filter.
Rs	dB of stop-band ripple.

Details

This function exists for compatibility with 'Matlab' and 'Octave' only, and is equivalent to `cheby2(n, Rp, 1, "low", "s")`.

Value

list of class Zpg containing poles and gain of the filter

Author(s)

Carne Draug, <carandraug+dev@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## 9th order Chebyshev type II low-pass analog filter
zp <- cheb2ap(9, 30)
w <- seq(0, 4, length.out = 128)
freqs(zp, w)
```

cheb2ord *Chebyshev Type II filter order*

Description

Compute Chebyshev type-II filter order and cutoff for the desired response characteristics.

Usage

```
cheb2ord(Wp, Ws, Rp, Rs, plane = c("z", "s"))
```

Arguments

Wp, Ws	pass-band and stop-band edges. For a low-pass or high-pass filter, Wp and Ws are scalars. For a band-pass or band-rejection filter, both are vectors of length 2. For a low-pass filter, Wp < Ws. For a high-pass filter, Ws > Wp. For a band-pass (Ws[1] < Wp[1] < Wp[2] < Ws[2]) or band-reject (Wp[1] < Ws[1] < Ws[2] < Wp[2]) filter design, Wp gives the edges of the pass band, and Ws gives the edges of the stop band. For digital filters, frequencies are normalized to [0, 1], corresponding to the range [0, fs / 2]. In case of an analog filter, all frequencies are specified in radians per second.
Rp	allowable decibels of ripple in the pass band.
Rs	minimum attenuation in the stop band in dB.
plane	"z" for a digital filter or "s" for an analog filter.

Value

A list of class 'FilterSpecs' with the following list elements:

n filter order

Wc cutoff frequency

type filter type, normally one of "low", "high", "stop", or "pass".

Author(s)

Paul Kienzle, Charles Praplan.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

cheby1

Examples

```
## low-pass 30 Hz filter
fs <- 128
spec <- cheb2ord(30/(fs/2), 40/(fs/2), 0.5, 40)
cf <- cheby2(spec)
freqz(cf, fs = fs)
```

---

chebwin *Chebyshev window*

Description

Return the filter coefficients of a Dolph-Chebyshev window.

Usage

```
chebwin(n, at = 100)
```

Arguments

n	Window length, specified as a positive integer.
at	Stop-band attenuation in dB. Default: 100.

Details

The window is described in frequency domain by the expression:

$$W(k) = \frac{Cheb(m-1, \beta \cdot cos(\pi \cdot k/m))}{Cheb(m-1, \beta)}$$

with

$$\beta = cosh(1/(m-1) \cdot acosh(10^{(at/20)})$$

and and $Cheb(m, x)$ denoting the m-th order Chebyshev polynomial calculated at the point x.

Note that the denominator in W(k) above is not computed, and after the inverse Fourier transform the window is scaled by making its maximum value unitary.

Value

Chebyshev window, returned as a vector. If you specify a one-point window (`n = 1`), the value 1 is returned.

Author(s)

André Carezia, <acarezia@uol.com.br>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
cw <- chebwin(64)
plot (cw, type = "l", xlab = "Samples", ylab =" Amplitude")
```

---

cheby1 *Chebyshev Type I filter design*

Description

Compute the transfer function coefficients of a Chebyshev Type I filter.

Usage

```
cheby1(n, ...)

## S3 method for class 'FilterSpecs'
cheby1(n, ...)

## Default S3 method:
cheby1(
  n,
  Rp,
  w,
  type = c("low", "high", "stop", "pass"),
  plane = c("z", "s"),
  output = c("Arma", "Zpg", "Sos"),
  ...
)
```

Arguments

n	filter order.
...	additional arguments passed to cheby1, overriding those given by n of class FilterSpecs.
Rp	dB of passband ripple.
w	critical frequencies of the filter. w must be a scalar for low-pass and high-pass filters, and w must be a two-element vector c(low, high) specifying the lower and upper bands in radians/second. For digital filters, W must be between 0 and 1 where 1 is the Nyquist frequency.
type	filter type, one of "low", "high", "stop", or "pass".
plane	"z" for a digital filter or "s" for an analog filter.
output	Type of output, one of:
	"Arma" Autoregressive-Moving average (aka numerator/denominator, aka b/a)

"Zpg" Zero-pole-gain format

"Sos" Second-order sections

Default is "Arma" for compatibility with the 'signal' package and the 'Matlab' and 'Octave' equivalents, but "Sos" should be preferred for general-purpose filtering because of numeric stability.

Details

Chebyshev filters are analog or digital filters having a steeper roll-off than Butterworth filters, and have passband ripple (type I) or stopband ripple (type II).

Because cheby1 is generic, it can be extended to accept other inputs, using cheb1ord to generate filter criteria for example.

Value

Depending on the value of the output parameter, a list of class Arma, Zpg, or Sos containing the filter coefficients

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Doug Stewart, <dastew@sympatico.ca>.
Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Chebyshev_filter

See Also

Arma, filter, butter, ellip, cheb1ord, FilterSpecs

Examples

```
## compare the frequency responses of 5th-order
## Butterworth and Chebyshev filters.
bf <- butter(5, 0.1)
cf <- cheby1(5, 3, 0.1)
bfr <- freqz(bf)
cfr <- freqz(cf)
plot(bfr$w / pi, 20 * log10(abs(bfr$h)), type = "l", ylim = c(-40, 0),
  xlim = c(0, .5), xlab = "Frequency", ylab = c("dB"))
lines(cfr$w / pi, 20 * log10(abs(cfr$h)), col = "red")

# compare type I and type II Chebyshev filters.
c1fr <- freqz(cheby1(5, .5, 0.5))
c2fr <- freqz(cheby2(5, 20, 0.5))
plot(c1fr$w / pi, abs(c1fr$h), type = "l", ylim = c(0, 1),
  xlab = "Frequency", ylab = c("Magnitude"))
```

```
lines(c2fr$w / pi, abs(c2fr$h), col = "red")
```

---

cheby2 *Chebyshev Type II filter design*

Description

Compute the transfer function coefficients of a Chebyshev Type II filter.

Usage

```
cheby2(n, ...)

## S3 method for class 'FilterSpecs'
cheby2(n, ...)

## Default S3 method:
cheby2(
  n,
  Rs,
  w,
  type = c("low", "high", "stop", "pass"),
  plane = c("z", "s"),
  output = c("Arma", "Zpg", "Sos"),
  ...
)
```

Arguments

n	filter order.
...	additional arguments passed to cheby1, overriding those given by n of class FilterSpecs.
Rs	dB of stopband ripple.
w	critical frequencies of the filter. w must be a scalar for low-pass and high-pass filters, and w must be a two-element vector c(low, high) specifying the lower and upper bands in radians/second. For digital filters, W must be between 0 and 1 where 1 is the Nyquist frequency.
type	filter type, one of "low", "high", "stop", or "pass".
plane	"z" for a digital filter or "s" for an analog filter.
output	Type of output, one of: **"Arma"** Autoregressive-Moving average (aka numerator/denominator, aka b/a) **"Zpg"** Zero-pole-gain format **"Sos"** Second-order sections Default is "Arma" compatibility with the 'signal' package and the 'Matlab' and 'Octave' equivalents, but "Sos" should be preferred for general-purpose filtering because of numeric stability.

Details

Chebyshev filters are analog or digital filters having a steeper roll-off than Butterworth filters, and have passband ripple (type I) or stopband ripple (type II).

Because `cheby2` is generic, it can be extended to accept other inputs, using `cheb2ord` to generate filter criteria for example.

Value

Depending on the value of the `output` parameter, a list of class `Arma`, `Zpg`, or `Sos` containing the filter coefficients

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Doug Stewart, <dastew@sympatico.ca>.
Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Chebyshev_filter

See Also

`Arma`, `filter`, `butter`, `ellip`, `cheb2ord`

Examples

```
## compare the frequency responses of 5th-order
## Butterworth and Chebyshev filters.
bf <- butter(5, 0.1)
cf <- cheby2(5, 20, 0.1)
bfr <- freqz(bf)
cfr <- freqz(cf)
plot(bfr$w / pi, 20 * log10(abs(bfr$h)), type = "l", ylim = c(-40, 0),
  xlim = c(0, .5), xlab = "Frequency", ylab = c("dB"))
lines(cfr$w / pi, 20 * log10(abs(cfr$h)), col = "red")

# compare type I and type II Chebyshev filters.
c1fr <- freqz(cheby1(5, .5, 0.5))
c2fr <- freqz(cheby2(5, 20, 0.5))
plot(c1fr$w / pi, abs(c1fr$h), type = "l", ylim = c(0, 1.1),
  xlab = "Frequency", ylab = c("Magnitude"))
lines(c2fr$w / pi, abs(c2fr$h), col = "red")
```

`chirp` *Chirp signal*

Description

Evaluate a chirp signal (frequency swept cosine wave).

Usage

```
chirp(
  t,
  f0,
  t1 = 1,
  f1 = 100,
  shape = c("linear", "quadratic", "logarithmic"),
  phase = 0
)
```

Arguments

`t`	Time array, specified as a vector.
`f0`	Initial instantaneous frequency at time 0, specified as a positive scalar expressed in Hz. Default: 0 Hz for linear and quadratic shapes; 1e-6 for logarithmic shape.
`t1`	Reference time, specified as a positive scalar expressed in seconds. Default: 1 sec.
`f1`	Instantaneous frequency at time t1, specified as a positive scalar expressed in Hz. Default: 100 Hz.
`shape`	Sweep method, specified as `"linear"`, `"quadratic"`, or `"logarithmic"` (see Details). Default: `"linear"`.
`phase`	Initial phase, specified as a positive scalar expressed in degrees. Default: 0.

Details

A chirp is a signal in which the frequency changes with time, commonly used in sonar, radar, and laser. The name is a reference to the chirping sound made by birds.

The chirp can have one of three shapes:

"linear" Specifies an instantaneous frequency sweep $f_i(t)$ given by $f_i(t) = f_0 + \beta t$, where $\beta = (f_1 - f_0)/t_1$ and the default value for f_0 is 0. The coefficient β ensures that the desired frequency breakpoint f_1 at time t_1 is maintained.

"quadratic" Specifies an instantaneous frequency sweep $f_i(t)$ given by $f_i(t) = f_0 + \beta t^2$, where $\beta = (f_1 - f_0)/t_1^2$ and the default value for f_0 is 0. If $f_0 > f_1$ (downsweep), the default shape is convex. If $f_0 < f_1$ (upsweep), the default shape is concave.

"logarithmic" Specifies an instantaneous frequency sweep $f_i(t)$ given by $f_i(t) = f_0 \times \beta t$, where $\beta = \left(\frac{f_1}{f_0}\right)^{\frac{1}{t1}}$ and the default value for f_0 is 10^{-6}.

Value

Chirp signal, returned as an array of the same length as `t`.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
# Shows linear sweep of 100 Hz/sec starting at zero for 5 sec
# since the sample rate is 1000 Hz, this should be a diagonal
# from bottom left to top right.
t <- seq(0, 5, 0.001)
y <- chirp (t)
specgram (y, 256, 1000)

# Shows a quadratic chirp of 400 Hz at t=0 and 100 Hz at t=10
# Time goes from -2 to 15 seconds.
specgram(chirp(seq(-2, 15, by = 0.001), 400, 10, 100, "quadratic"))

# Shows a logarithmic chirp of 200 Hz at t = 0 and 500 Hz at t = 2
# Time goes from 0 to 5 seconds at 8000 Hz.
specgram(chirp(seq(0, 5, by = 1/8000), 200, 2, 500, "logarithmic"),
         fs = 8000)
```

---

cl2bp *Constrained L2 bandpass FIR filter design*

Description

Constrained least square band-pass FIR filter design without specified transition bands.

Usage

```
cl2bp(m = 30, w1, w2, up, lo, L = 2048)
```

Arguments

`m`	degree of cosine polynomial, resulting in a filter of length `2 * m + 1`. Must be an even number. Default: 30.
`w1, w2`	bandpass filter cutoffs in the range `0 <= w1 < w2 <= pi`, where pi is the Nyquist frequency.
`up`	vector of 3 upper bounds for c(stopband1, passband, stopband2).
`lo`	vector of 3 lower bounds for c(stopband1, passband, stopband2).
`L`	search grid size; larger values may improve accuracy, but greatly increase calculation time. Default: 2048, maximum: 1e6.

Details

This is a fast implementation of the algorithm cited below. Compared to `remez`, it offers implicit specification of transition bands, a higher likelihood of convergence, and an error criterion combining features of both L2 and Chebyshev approaches

Value

The FIR filter coefficients, a vector of length `2 * m + 1`, of class `Ma`.

Author(s)

Ivan Selesnick, Rice University, 1995, downloaded from https://www.ece.rice.edu/dsp/software/cl2.shtml.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

Selesnick, I.W., Lang, M., and Burrus, C.S. (1998) A modified algorithm for constrained least square design of multiband FIR filters without specified transition bands. IEEE Trans. on Signal Processing, 46(2), 497-501.
https://www.ece.rice.edu/dsp/software/cl2.shtml

See Also

`Ma`, `filter`, `remez`

Examples

```
w1 <- 0.3 * pi
w2 <- 0.6 * pi
up <- c(0.02, 1.02, 0.02)
lo <- c(-0.02, 0.98, -0.02)
h  <- cl2bp(30, w1, w2, up, lo, 2^11)
freqz(h)
```

`clustersegment` *Cluster Segments*

Description

Calculate boundary indexes of clusters of 1's.

Usage

```
clustersegment(x)
```

Arguments

x	input data, specified as a numeric vector or matrix, coerced to contain only 0's and 1's, i.e., every nonzero element in x will be replaced by 1.

Details

The function calculates the initial index and end index of sequences of 1s rising and falling phases of the signal in x. The clusters are sought in the rows of the array x. The function works by finding the indexes of jumps between consecutive values in the rows of x.

Value

A list of size nr, where nr is the number of rows in x. Each element of the list contains a matrix with two rows. The first row is the initial index of a sequence of 1's and the second row is the end index of that sequence.

Author(s)

Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
(x <- c(0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1))
(ranges <- clustersegment(x))
# The first sequence of 1's in x lies in the interval
(r <- ranges[1,1]:ranges[2,1])

x <- matrix(as.numeric(runif(30) > 0.4), 3, 10)
ranges <- clustersegment(x)

x <- c(0, 1.2, 3, -8, 0)
ranges <- clustersegment(x)
```

---

cmorwavf *Complex Morlet Wavelet*

Description

Compute the complex Morlet wavelet on a grid.

Usage

```
cmorwavf(lb = -8, ub = 8, n = 1000, fb = 5, fc = 1)
```

Arguments

lb, ub	Lower and upper bounds of the interval to evaluate the complex Morlet waveform on. Default: -8 to 8.
n	Number of points on the grid between lb and ub (length of the wavelet). Default: 1000.
fb	Time-decay parameter of the wavelet (bandwidth in the frequency domain). Must be a positive scalar. Default: 5.
fc	Center frequency of the wavelet. Must be a positive scalar. Default: 1.

Details

The Morlet (or Gabor) wavelet is a wavelet composed of a complex exponential (carrier) multiplied by a Gaussian window (envelope). The wavelet exists as a complex version or a purely real-valued version. Some distinguish between the "real Morlet" versus the "complex Morlet". Others consider the complex version to be the "Gabor wavelet", while the real-valued version is the "Morlet wavelet". This function returns the complex Morlet wavelet, with time-decay parameter fb, and center frequency fc. The general expression for the complex Morlet wavelet is

$$\Psi(x) = ((\pi fb)^{-0.5}) \cdot e^{(2\pi i fc x)} \cdot e^{-(x^2)/fb}$$

x is evaluated on an n-point regular grid in the interval (lb, ub).

fb controls the decay in the time domain and the corresponding energy spread (bandwidth) in the frequency domain. fb is the inverse of the variance in the frequency domain. Increasing fb makes the wavelet energy more concentrated around the center frequency and results in slower decay of the wavelet in the time domain. Decreasing fb results in faster decay of the wavelet in the time domain and less energy spread in the frequency domain. The value of fb does not affect the center frequency. When converting from scale to frequency, only the center frequency affects the frequency values. The energy spread or bandwidth parameter affects how localized the wavelet is in the frequency domain. See the examples.

Value

A list containing 2 variables; x, the grid on which the complex Morlet wavelet was evaluated, and psi (Ψ), the evaluated wavelet on the grid x.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## Construct a complex-valued Morlet wavelet with a bandwidth parameter
##  of 1.5 and a center frequency of 1. Set the effective support to [-8,8]
## and the length of the wavelet to 1000.
cmw <- cmorwavf(-8, 8, 1000, 1.5, 1)

# Plot the real and imaginary parts of the wavelet.
```

```
op <- par(mfrow = c(2, 1))
plot(cmw$x, Re(cmw$psi), type = "l", main = "Real Part")
plot(cmw$x, Im(cmw$psi), type = "l", main = "Imaginary Part")
par(op)

## This example shows how the complex Morlet wavelet shape in the frequency
## domain is affected by the value of the bandwidth parameter (fb). Both
## wavelets have a center frequency of 1. One wavelet has an fb value of
## 0.5 and the other wavelet has a value of 8.

op <- par(mfrow = c(2,1))
cmw1 <- cmorwavf(fb = 0.5)
cmw2 <- cmorwavf(fb = 8)

# time domain plot
plot(cmw1$x, Re(cmw1$psi), type = "l", xlab = "Time", ylab = "",
     main = "Time domain, Real part")
lines(cmw2$x, Re(cmw2$psi), col = "red")
legend("topright", legend = c("fb = 0.5", "fb = 8"), lty= 1, col = c(1,2))

# frequency domain plot
f <- seq(-5, 5, .01)
Fc <- 1
Fb1 <- 0.5
Fb2 <- 8
PSI1 <- exp(-pi^2 * Fb1 * (f-Fc)^2)
PSI2 <- exp(-pi^2 * Fb2 * (f-Fc)^2)
plot(f, PSI1, type="l", xlab = "Frequency", ylab = "",
     main = "Frequency domain")
lines(f, PSI2, col = "red")
legend("topright", legend = c("fb = 0.5", "fb = 8"),
       lty= 1, col = c(1,2))
par(op)

## The fb bandwidth parameter for the complex Morlet wavelet is the
## inverse of the variance in frequency. Therefore, increasing Fb results
## in a narrower concentration of energy around the center frequency.

## alternative to the above frequency plot:
fs <- length(cmw1$x) / sum(abs(range(cmw1$x)))
hz  <- seq(0, fs/2, len=floor(length(cmw1$psi)/2)+1)
PSI1 <- fft(cmw1$psi) / length(cmw1$psi)
PSI2 <- fft(cmw2$psi) / length(cmw2$psi)
plot(hz, 2 * abs(PSI1)[1:length(hz)], type="l", xlab = "Frequency",
     ylab = "", main = "Frequency domain", xlim=c(0,5))
lines(hz, 2 * abs(PSI2)[1:length(hz)], col = 2)
legend("topright", legend = c("fb = 0.5", "fb = 8"), lty= 1, col = c(1,2))
```

---

conv *Convolution and polynomial multiplication*

Description

Convolve two vectors a and b.

Usage

```
conv(a, b, shape = c("full", "same", "valid"))
```

Arguments

a, b	Input, coerced to vectors, can be different lengths or data types.
shape	Subsection of convolution, partially matched to "full" (full convolution - default), "same" (central part of the convolution of the same size as a), or "valid" (only those parts of the convolution that are computed without the zero-padded edges)

Details

The convolution of two vectors, a and b, represents the area of overlap under the points as B slides across a. Algebraically, convolution is the same operation as multiplying polynomials whose coefficients are the elements of a and b.

The function conv uses the filter function, NOT fft, which may be faster for large vectors.

Value

Output vector with length equal to length (a) + length (b) - 1. When the parameter shape is set to "valid", the length of the output is max(length(a) - length(b) + 1, 0), except when length(b) is zero. In that case, the length of the output vector equals length(a).

When a and b are the coefficient vectors of two polynomials, the convolution represents the coefficient vector of the product polynomial.

Author(s)

Tony Richardson, <arichard@stark.cc.oh.us>, adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
u <- rep(1L, 3)
v <- c(1, 1, 0, 0, 0, 1, 1)
w <- conv(u, v)

## Create vectors u and v containing the coefficients of the polynomials
## x^2 + 1 and 2x + 7.
u <- c(1, 0, 1)
v <- c(2, 7)
## Use convolution to multiply the polynomials.
w <- conv(u, v)
## w contains the polynomial coefficients for 2x^3 + 7x^2 + 2x + 7.
```

```
## Central part of convolution
u <- c(-1, 2, 3, -2, 0, 1, 2)
v <- c(2, 4, -1, 1)
w <- conv(u, v, 'same')
```

---

conv2 *2-D convolution*

Description

Compute the two-dimensional convolution of two matrices.

Usage

```
conv2(a, b, shape = c("full", "same", "valid"))
```

Arguments

a, b	Input matrices, coerced to numeric.
shape	Subsection of convolution, partially matched to:
	"full" Return the full convolution (default)
	"same" Return the central part of the convolution with the same size as A. The central part of the convolution begins at the indices `floor(c(nrow(b), ncol(b)) / 2 + 1)`
	"valid" Return only the parts which do not include zero-padded edges. The size of the result is `max(nrow(a) - nrow(a) + 1, 0)` by `max(ncol(A) - ncol(B) + 1, 0)`

Value

Convolution of input matrices, returned as a matrix.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

conv, convolve

Examples

```
a <- matrix(1:16, 4, 4)
b <- matrix(1:9, 3,3)
cnv <- conv2(a, b)
cnv <- conv2(a, b, "same")
cnv <- conv2(a, b, "valid")
```

convmtx *Convolution matrix*

Description

Returns the convolution matrix for a filter kernel.

Usage

```
convmtx(h, n)
```

Arguments

h	Input, coerced to a vector, representing the filter kernel
n	Length of vector(s) that h is to be convolved with.

Details

Computing a convolution using conv when the signals are vectors is generally more efficient than using convmtx. For multichannel signals, however, when a large number of vectors are to be convolved with the same filter kernel, convmtx might be more efficient.

The code cm <- convmtx(h, n) computes the convolution matrix of the filter kernel h with a vector of length n. Then, cm x gives the convolution of h and x.

Value

Convolution matrix of input h for a vector of length n. If h is a vector of length m, then the convolution matrix has m + n - 1 rows and n columns.

Author(s)

David Bateman <adb014@gmail.com>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

See Also

conv

Examples

```
N <- 1000
a <- runif(N)
b <- runif(N)
cm <- convmtx(b, N)
d <- cm %*% a

cref = conv(a, b)
all.equal(max(d - cref), 0)
```

cplxpair *Complex conjugate pairs*

Description

Sort complex numbers into complex conjugate pairs ordered by increasing real part.

Usage

```
cplxpair(z, tol = 100 * .Machine$double.eps, MARGIN = 2)
```

Arguments

z	Vector, matrix, or array of complex numbers.
tol	Weighting factor `0 < tol < 1`, which determines the tolerance of matching. Default: `100 * .Machine$double.eps`. (This definition differs from the 'Octave' usage).
MARGIN	Vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names. Default: 2 (columns).

Details

The negative imaginary complex numbers are placed first within each pair. All real numbers (those with `abs(Im(z) / z) < tol`) are placed after the complex pairs.

An error is signaled if some complex numbers could not be paired and if all complex numbers are not exact conjugates (to within `tol`).

Value

Vector, matrix or array containing ordered complex conjugate pairs by increasing real parts.

Note

There is no defined order for pairs with identical real parts but differing imaginary parts.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

cplxreal

Examples

```
r <- rbind(t(cplxpair(exp(2i * pi * 0:4 / 5))),
           t(exp(2i * pi *c(3, 2, 4, 1, 0) / 5)))
```

---

cplxreal *Sort complex conjugate pairs and real*

Description

Sort numbers into into complex-conjugate-valued and real-valued elements.

Usage

```
cplxreal(z, tol = 100 * .Machine$double.eps, MARGIN = 2)
```

Arguments

z	Vector, matrix, or array of complex numbers.
tol	Weighting factor 0 < tol < 1, which determines the tolerance of matching. Default: 100 * .Machine$double.eps.
MARGIN	Vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names. Default: 2 (columns).

Details

An error is signaled if some complex numbers could not be paired and if all complex numbers are not exact conjugates (to within tol). Note that here is no defined order for pairs with identical real parts but differing imaginary parts.

Value

A list containing two variables:

zc Vector, matrix or array containing ordered complex conjugate pairs by increasing real parts. Only the positive imaginary complex numbers of each complex conjugate pair are returned.

zr Vector, matrix or array containing ordered real numbers.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

cplxpair

Examples

```
r <- cplxreal(c(1, 1 + 3i, 2 - 5i, 1-3i, 2 + 5i, 4, 3))
```

---

cpsd *Cross power spectral density*

Description

Estimates the cross power spectral density (CPSD) of discrete-time signals.

Usage

```
cpsd(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none")
)

csd(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none")
)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
window	If window is a vector, each segment has the same length as window and is multiplied by window before (optional) zero-padding and calculation of its periodogram. If window is a scalar, each segment has a length of window and a Hamming window is used. Default: nextpow2(sqrt(length(x))) (the square root of the length of x rounded up to the next power of two). The window length must be larger than 3.
overlap	segment overlap, specified as a numeric value expressed as a multiple of window or segment length. 0 <= overlap < 1. Default: 0.5.
nfft	Length of FFT, specified as an integer scalar. The default is the length of the window vector or has the same value as the scalar window argument. If nfft is larger than the segment length, (seg_len), the data segment is padded nfft -

seg_len zeros. The default is no padding. Nfft values smaller than the length of the data segment (or window) are ignored. Note that the use of padding to increase the frequency resolution of the spectral estimate is controversial.

fs sampling frequency (Hertz), specified as a positive scalar. Default: 1.

detrend character string specifying detrending option; one of:

"long-mean" remove the mean from the data before splitting into segments (default)

"short-mean" remove the mean value of each segment

"long-linear" remove linear trend from the data before splitting into segments

"short-linear" remove linear trend from each segment

"none" no detrending

Details

cpsd estimates the cross power spectral density function using Welch's overlapped averaged periodogram method [1].

Value

A list containing the following elements:

freq vector of frequencies at which the spectral variables are estimated. If x is numeric, power from negative frequencies is added to the positive side of the spectrum, but not at zero or Nyquist (fs/2) frequencies. This keeps power equal in time and spectral domains. If x is complex, then the whole frequency range is returned.

cross NULL for univariate series. For multivariate series, a matrix containing the squared coherence between different series. Column $i+(j-1)*(j-2)/2$ of coh contains the cross-spectral estimates between columns i and j of x, where $i < j$.

Note

The function cpsd (and its deprecated alias csd) is a wrapper for the function pwelch, which is more complete and more flexible.

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] Welch, P.D. (1967). The use of Fast Fourier Transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. IEEE Transactions on Audio and Electroacoustics, AU-15 (2): 70–73.

Examples

```
fs <- 1000
f <- 250
t <- seq(0, 1 - 1/fs, 1/fs)
s1 <- sin(2 * pi * f * t) + runif(length(t))
s2 <- sin(2 * pi * f * t - pi / 3) + runif(length(t))
rv <- cpsd(cbind(s1, s2), fs = fs)
plot(rv$freq, 10 * log10(rv$cross), type="l", xlab = "Frequency",
     ylab = "Cross Spectral Density (dB)")
```

---

czt *Chirp Z-transform*

Description

Compute the Chirp Z-transform along a spiral contour on the z-plane.

Usage

```
czt(x, m = NROW(x), w = exp(complex(real = 0, imaginary = -2 * pi/m)), a = 1)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
m	transform length, specified as a positive integer scalar. Default: `NROW(x)`.
w	ratio between spiral contour points in each step (i.e., radius increases exponentially, and angle increases linearly), specified as a complex scalar. Default: `exp(0-1i * 2 * pi / m)`.
a	initial spiral contour point, specified as a complex scalar. Default: 1.

Details

The chirp Z-transform (CZT) is a generalization of the discrete Fourier transform (DFT). While the DFT samples the Z plane at uniformly-spaced points along the unit circle, the chirp Z-transform samples along spiral arcs in the Z-plane, corresponding to straight lines in the S plane. The DFT, real DFT, and zoom DFT can be calculated as special cases of the CZT[1]. For the specific case of the DFT, `a = 0`, `m = NCOL(x)`, and `w = 2 * pi / m`[2, p. 656].

Value

Chirp Z-transform, returned as a vector or matrix.

Author(s)

Daniel Gunyan.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Chirp_Z-transform
[2]Oppenheim, A.V., Schafer, R.W., and Buck, J.R. (1999). Discrete-Time Signal Processing, 2nd edition. Prentice-Hall.

Examples

```
fs <- 1000                                          # sampling frequency
secs <- 10                                          # number of seconds
t <- seq(0, secs, 1/fs)                             # time series
x <- sin(100 * 2 * pi * t) + runif(length(t))       # 100 Hz signal + noise
m <- 32                                             # n of points desired
f0 <- 75; f1 <- 175;                                # desired freq range
w <- exp(-1i * 2 * pi * (f1 - f0) / ((m - 1) * fs))  # freq step of f1-f0/m
a <- exp(1i * 2 * pi * f0 / fs);                    # starting at freq f0
y <- czt(x, m, w, a)

# compare DFT and FFT
fs <- 1000
h <- as.numeric(fir1(100, 125/(fs / 2), type = "low"))
m <- 1024
y <- stats::fft(postpad(h, m))

f1 <- 75; f2 <- 175;
w <- exp(-1i * 2 * pi * (f2 - f1) / (m * fs))
a <- exp(1i * 2 * pi * f1 / fs)
z <- czt(h, m, w, a)

fn <- seq(0, m - 1, 1) / m
fy <- fs * fn
fz = (f2 - f1) * fn + f1
plot(fy, 10 * log10(abs(y)), type = "l", xlim = c(50, 200),
  xlab = "Frequency", ylab = "Magnitude (dB")
lines(fz, 10 * log10(abs(z)), col = "red")
legend("topright", legend = c("FFT", "CZT"), col=1:2, lty = 1)
```

`dct` *Discrete Cosine Transform*

Description

Compute the unitary discrete cosine transform of a signal.

Usage

```
dct(x, n = NROW(x))
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
n	transform length, specified as a positive integer scalar. Default: NROW(x).

Details

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. You can often reconstruct a sequence very accurately from only a few DCT coefficients. This property is useful for applications requiring data reduction.

The DCT has four standard variants. This function implements the DCT-II according to the definition in [1], which is the most common variant, and the original variant first proposed for image processing.

Value

Discrete cosine transform, returned as a vector or matrix.

Note

The transform is faster if x is real-valued and has even length.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Discrete_cosine_transform

See Also

idct

Examples

```
x <- matrix(seq_len(100) + 50 * cos(seq_len(100) * 2 * pi / 40))
X <- dct(x)

# Find which cosine coefficients are significant (approx.)
# zero the rest
nsig <- which(abs(X) < 1)
N <- length(X) - length(nsig) + 1
X[nsig] <- 0

# Reconstruct the signal and compare it to the original signal.
xx <- idct(X)
plot(x, type = "l")
lines(xx, col = "red")
```

```
legend("bottomright", legend = c("Original", paste("Reconstructed, N =", N)),
       lty = 1, col = 1:2)
```

---

dct2 *2-D Discrete Cosine Transform*

Description

Compute the two-dimensional discrete cosine transform of a matrix.

Usage

```
dct2(x, m = NROW(x), n = NCOL(x))
```

Arguments

x	2-D numeric matrix
m	Number of rows, specified as a positive integer. dct2 pads or truncates x so that is has m rows. Default: NROW(x).
n	Number of columns, specified as a positive integer. dct2 pads or truncates x so that is has n columns. Default: NCOL(x).

Details

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. It is a separable linear transformation; that is, the two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension.

Value

m-by-n numeric discrete cosine transformed matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

idct2

Examples

```
A <- matrix(runif(100), 10, 10)
B <- dct2(A)
```

dctmtx *Discrete Cosine Transform Matrix*

Description

Compute the discrete cosine transform matrix.

Usage

```
dctmtx(n)
```

Arguments

n	Size of DCT matrix, specified as a positive integer.

Details

A DCT transformation matrix is useful for doing things like JPEG image compression, in which an 8x8 DCT matrix is applied to non-overlapping blocks throughout an image and only a sub-block on the top left of each block is kept. During restoration, the remainder of the block is filled with zeros and the inverse transform is applied to the block.

The two-dimensional DCT of A can be computed as `D %*% A %*% t(D)`. This computation is sometimes faster than using `dct2`, especially if you are computing a large number of small DCTs, because D needs to be determined only once. For example, in JPEG compression, the DCT of each 8-by-8 block is computed. To perform this computation, use `dctmtx` to determine D of input image A, and then calculate each DCT using `D %*% A %*% t(D)` (where A is each 8-by-8 block). This is faster than calling `dct2` for each individual block.

Value

Discrete cosine transform, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

dct, dct2, idct, idct2

Examples

```
D <- dctmtx(8)
```

`decimate` *Decrease sample rate*

Description

Downsample a signal by an integer factor.

Usage

```
decimate(x, q, ftype = c("iir", "fir"), n = ifelse(ftype == "iir", 8, 30))
```

Arguments

`x`	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
`q`	decimation factor, specified as a positive integer.
`ftype`	filter type; either `"fir"`, specifying a FIR filter of length `n` designed with the function `fir1`, or `"iir"` (default), specifying an IIR Chebyshev filter of order 8 using the function `cheby1`.
`n`	Order of the filter used prior to the downsampling, specified as a positive integer. Default: 8 if `ftype` equals `"iir"`; 30 of `ftype` equals `"fir"`.

Value

downsampled signal, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`cheby1`, `fir1`

Examples

```
t <- seq(0, 2, 0.01)
x <- chirp(t, 2, .5, 10, 'quadratic') + sin(2 * pi * t * 0.4)
w <- 1:121
plot(t[w] * 1000, x[w], type = "h", col = "green")
points(t[w] * 1000, x[w], col = "green")
y = decimate(x, 4)
lines(t[seq(1, 121, 4)] * 1000, y[1:31], type = "h", col = "red")
points(t[seq(1, 121, 4)] * 1000, y[1:31], col = "red")
```

detrend *Remove Polynomial Trend*

Description

detrend removes the polynomial trend of order p from the data x.

Usage

```
detrend(x, p = 1)
```

Arguments

x	Input vector or matrix. If x is a matrix, the trend is removed from the columns.
p	Order of the polynomial. Default: 1. The order of the polynomial can also be given as a string, in which case p must be either "constant" (corresponds to p = 0) or "linear" (corresponds to p = 1).

Value

The detrended data, of same type and dimensions as x

Author(s)

Kurt Hornik, <Kurt.Hornik@wu-wien.ac.at>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
t <- 0:20
x <- 3 * sin(t) + t
y <- detrend(x)
plot(t, x, type = "l", ylim = c(-5, 25), xlab = "", ylab = "")
lines(t, y, col = "red")
lines(t, x - y, lty = 2)
legend('topleft', legend = c('Input Data', 'Detrended Data', 'Trend'),
 col = c(1, 2 ,1), lty = c(1, 1, 2))
```

dftmtx *Discrete Fourier Transform Matrix*

Description

Compute the discrete Fourier transform matrix

Usage

```
dftmtx(n)
```

Arguments

n	Size of Fourier transformation matrix, specified as a positive integer.

Details

A discrete Fourier transform matrix is a complex matrix whose matrix product with a vector computes the discrete Fourier transform of the vector. `dftmtx` takes the FFT of the identity matrix to generate the transform matrix. For a column vector `x`, `y <- dftmtx(n) * x` is the same as `y <- fft(x, postpad(x, n)`. The inverse discrete Fourier transform matrix is `inv <- Conj(dftmtx(n)) / n`.

In general this is less efficient than calling the `fft` and `ifft` functions directly.

Value

Fourier transform matrix.

Author(s)

David Bateman, <adb014@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

fft, ifft

Examples

```
x <- seq_len(256)
y1 <- stats::fft(x)
n <- length(x)
y2 <- drop(x %*% dftmtx(n))
mx <- max(abs(y1 - y2))
```

digitrevorder *Permute input to digit-reversed order*

Description

Reorder the elements of the input vector in digit-reversed order.

Usage

```
digitrevorder(x, r, index.return = FALSE)
```

Arguments

x	input data, specified as a vector. The length of x must be an integer power of r.
r	radix base used for the number conversion, which can be any integer from 2 to 36. The elements of x are converted to radix r and reversed.
index.return	logical indicating if the ordering index vector should be returned as well. Default FALSE.

Details

This function is useful for pre-ordering a vector of filter coefficients for use in frequency-domain filtering algorithms, in which the fft and ifft transforms are computed without digit-reversed ordering for improved run-time efficiency.

Value

The digit-reversed input vector. If index.return = TRUE, then a list containing the digit-reversed input vector (y, and the digit-reversed indices (i).

Author(s)

Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

bitrevorder, fft, ifft

Examples

```
res <- digitrevorder(0:8, 3)
```

`diric` *Dirichlet function*

Description

Compute the Dirichlet or periodic sinc function.

Usage

```
diric(x, n)
```

Arguments

`x`	Input array, specified as a real scalar, vector, matrix, or multidimensional array. When `x` is non-scalar, `diric` is an element-wise operation.
`n`	Function degree, specified as a positive integer scalar.

Details

`y <- diric(x, n)` returns the Dirichlet Function of degree `n` evaluated at the elements of the input array `x`.

The Dirichlet function, or periodic sinc function, has period 2π for odd N and period 4π for even N. Its maximum value is 1 for all N, and its minimum value is -1 for even N. The magnitude of the function is 1 / N times the magnitude of the discrete-time Fourier transform of the N-point rectangular window.

Value

Output array, returned as a real-valued scalar, vector, matrix, or multidimensional array of the same size as x.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## Compute and plot the Dirichlet function between -2pi and 2pi for N = 7
## and N = 8. The function has a period of 2pi for odd N and 4pi for even N.
x <- seq(-2*pi, 2*pi, len = 301)
d7 <- diric(x, 7)
d8 <- diric(x, 8)
op <- par(mfrow = c(2,1))
plot(x/pi, d7, type="l", main = "Dirichlet function",
     xlab = "", ylab = "N = 7")
plot(x/pi, d8, type="l", ylab = "N = 8", xlab = expression(x / pi))
par(op)
```

`downsample` *Decrease sample rate*

Description

Downsample a signal by an integer factor.

Usage

```
downsample(x, n, phase = 0)
```

Arguments

`x`	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
`n`	downsampling factor, specified as a positive integer.
`phase`	offset, specified as a positive integer from `0` to `n - 1`. Default: 0.

Details

For most signals you will want to use `decimate` instead since it prefilters the high frequency components of the signal and avoids aliasing effects.

Value

Downsampled signal, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`decimate`, `resample`

Examples

```
x <- seq_len(10)
xd <- downsample(x, 3)     # returns 1  4  7 10
xd <- downsample(x, 3, 2)  # returns 3 6 9

x <- matrix(seq_len(12), 4, 3, byrow = TRUE)
xd <- downsample(x, 3)
```

dst *Discrete Sine Transform*

Description

Compute the discrete sine transform of a signal.

Usage

```
dst(x, n = NROW(x))
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
n	transform length, specified as a positive integer scalar. Default: NROW(x).

Details

The discrete sine transform (DST) is closely related to the discrete Fourier transform. but using a purely real matrix. It is equivalent to the imaginary parts of a DFT of roughly twice the length.

The DST has four standard variants. This function implements the DCT-I according to the definition in [1], which is the most common variant, and the original variant first proposed for image processing.

The 'Matlab' documentation for the DST warns that the use of the function is not recommended. They do not state the reason why, but it is likely that use of the discrete cosine transform (DCT)is preferred for image processing. Because cos(0) is 1, the first coefficient of the DCT (II) is the mean of the values being transformed. This makes the first coefficient of each 8x8 block represent the average tone of its constituent pixels, which is obviously a good start. Subsequent coefficients add increasing levels of detail, starting with sweeping gradients and continuing into increasingly fiddly patterns, and it just so happens that the first few coefficients capture most of the signal in photographic images. Sin(0) is 0, so the DSTs start with an offset of 0.5 or 1, and the first coefficient is a gentle mound rather than a flat plain. That is unlikely to suit ordinary images, and the result is that DSTs require more coefficients than DCTs to encode most blocks. This explanation was provided by Douglas Bagnall on Stackoverflow.

Value

Discrete sine transform, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Discrete_sine_transform

See Also

idst

Examples

```
x <- matrix(seq_len(100) + 50 * cos(seq_len(100) * 2 * pi / 40))
ct <- dct(x)
st <- dst(x)
```

---

dwt *1-D Discrete Wavelet Transform*

Description

Compute the single-level discrete wavelet transform of a signal

Usage

```
dwt(x, wname = "d8", lo = NULL, hi = NULL)

wfilters(wname)
```

Arguments

x	input data, specified as a numeric vector.
wname	analyzing wavelet, specified as a character string consisting of a class name followed by the wavelet length Only two classes of wavelets are supported; Daubechies (denoted by the prefix 'd' of even lengths 2 - 20, and Coiflet (denoted by the prefix ''c' of lengths 6, 12, 18, 24, and 30. The wavelet name 'haar' is the equivalent of 'd2'. Default: d8.
lo	scaling (low-pass) filter, specified as an even-length numeric vector. lo must be the same length as hi. Ignored when wname != NULL.
hi	wavelet (high-pass) filter, specified as an even-length numeric vector. hi must be the same length as lo, Ignored when wname != NULL.

Details

This function is only included because of compatibility with the 'Octave' 'signal' package. Specialized packages exist in R to perform the discrete wavelet transform, e.g., the `wavelets` package [1]. this function recognizes only a few wavelet names, namely those for which scale coefficients are available (Daubechies [2] and Coiflet [3]).

The wavelet and scaling coefficients are returned by the function `wfilters`, which returns the coefficients for reconstruction filters associated with the wavelet `wname`. Decomposition filters are the time reverse of the reconstruction filters (see examples).

Value

A list containing two numeric vectors:

a approximation (average) coefficients, obtained from convolving `x` with the scaling (low-pass) filter `lo`, and then downsampled (keep the even-indexed elements).

d detail (difference) coefficients, obtained from convolving `x` with the wavelet (high-pass) filter `hi`, and then downsampled (keep the even-indexed elements).

Note

The notations `g` and `h` are often used to denote low-pass (scaling) and high-pass (wavelet) coefficients, respectively, but inconsistently. Ref [4] uses it, as does the R `wavelets` package. 'Octave' uses the reverse notation. To avoid confusion, more neutral terms are used here.

There are two naming schemes for wavelet names in use. For instance for Daubechies wavelets (d), dN using the length or number of taps, and dbA referring to the number of vanishing moments. So d4 and db2 are the same wavelet transform. This function uses the formed (dN) notation; 'Matlab' uses the latter (dbA).

Author(s)

Lukas F. Reichlin.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://CRAN.R-project.org/package=wavelets

[2] https://en.wikipedia.org/wiki/Daubechies_wavelet

[3] https://en.wikipedia.org/wiki/Coiflet

[4] https://en.wikipedia.org/wiki/Discrete_wavelet_transform

Examples

```
# get Coiflet 30 coefficients
wv <- wfilters('c30')
lo <- rev(wv$lo)
hi <- rev(wv$hi)

# general time-varying signal
```

```
time <- 1
fs <- 1000
x <- seq(0,time, length.out=time*fs)
y <- c(cos(2*pi*100*x)[1:300], cos(2*pi*50*x)[1:300],
       cos(2*pi*25*x)[1:200], cos(2*pi*10*x)[1:200])
op <- par(mfrow = c(3,1))
plot(x, y, type = "l", xlab = "Time", ylab = "Amplitude",
     main = "Original signal")
wt <- dwt(y, wname = NULL, lo, hi)

x2 <- seq(1, length(x) - length(hi) + 1, 2)
plot(x2, wt$a, type = "h", xlab = "Time", ylab = "",
    main = "Approximation coefficients")
plot(x2, wt$d, type = "h", xlab = "Time", ylab = "",
     main = "Detail coefficients")
par (op)
```

`ellip` *Elliptic filter design*

Description

Compute the transfer function coefficients of an elliptic filter.

Usage

```
ellip(n, ...)

## S3 method for class 'FilterSpecs'
ellip(n, Rp = n$Rp, Rs = n$Rs, w = n$Wc, type = n$type, plane = n$plane, ...)

## Default S3 method:
ellip(
  n,
  Rp,
  Rs,
  w,
  type = c("low", "high", "stop", "pass"),
  plane = c("z", "s"),
  output = c("Arma", "Zpg", "Sos"),
  ...
)
```

Arguments

`n`	filter order.
`...`	additional arguments passed to ellip, overriding those given by `n` of class `FilterSpecs`.

Rp	dB of passband ripple.
Rs	dB of stopband ripple.
w	critical frequencies of the filter. w must be a scalar for low-pass and high-pass filters, and w must be a two-element vector c(low, high) specifying the lower and upper bands in radians/second. For digital filters, w must be between 0 and 1 where 1 is the Nyquist frequency.
type	filter type, one of "low", "high", "stop", or "pass".
plane	"z" for a digital filter or "s" for an analog filter.
output	Type of output, one of: **"Arma"** Autoregressive-Moving average (aka numerator/denominator, aka b/a) **"Zpg"** Zero-pole-gain format **"Sos"** Second-order sections Default is "Arma" for compatibility with the 'signal' package and the 'Matlab' and 'Octave' equivalents, but "Sos" should be preferred for general-purpose filtering because of numeric stability.

Details

An elliptic filter is a filter with equalized ripple (equiripple) behavior in both the passband and the stopband. The amount of ripple in each band is independently adjustable, and no other filter of equal order can have a faster transition in gain between the passband and the stopband, for the given values of ripple.

As the ripple in the stopband approaches zero, the filter becomes a type I Chebyshev filter. As the ripple in the passband approaches zero, the filter becomes a type II Chebyshev filter and finally, as both ripple values approach zero, the filter becomes a Butterworth filter.

Because ellip is generic, it can be extended to accept other inputs, using ellipord to generate filter criteria for example.

Value

Depending on the value of the output parameter, a list of class Arma, Zpg, or Sos containing the filter coefficients

Author(s)

Paulo Neis, <p_neis@yahoo.com.br>,
adapted by Doug Stewart, <dastew@sympatico.ca>.
Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Elliptic_filter

See Also

Arma, filter, butter, cheby1, ellipord

Examples

```
## compare the frequency responses of 5th-order Butterworth
## and elliptic filters.
bf <- butter(5, 0.1)
ef <- ellip(5, 3, 40, 0.1)
bfr <- freqz(bf)
efr <- freqz(ef)
plot(bfr$w, 20 * log10(abs(bfr$h)), type = "l", ylim = c(-80, 0),
     xlab = "Frequency (Rad)", ylab = c("dB"), lwd = 2,
     main = paste("Elliptic versus Butterworth filter",
     "low-pass -3 dB cutoff at 0.1 rad", sep = "\n"))
lines(efr$w, 20 * log10(abs(efr$h)), col = "red", lwd = 2)
legend ("topright", legend = c("Butterworh", "Elliptic"),
        lty = 1, lwd = 2, col = 1:2)
```

---

ellipap *Low-pass analog elliptic filter*

Description

Return the zeros, poles and gain of an analog elliptic low-pass filter prototype.

Usage

```
ellipap(n, Rp, Rs)
```

Arguments

n	Order of the filter.
Rp	dB of passband ripple.
Rs	dB of stopband ripple.

Details

This function exists for compatibility with 'Matlab' and 'OCtave' only, and is equivalent to `ellip(n, Rp, Rs, 1, "low", "s")`.

Value

list of class Zpg containing zeros, poles and gain of the filter.

Author(s)

Carne Draug, <carandraug+dev@gmail.com>. Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## 9th order elliptic low-pass analog filter
zp <- ellipap(9, .1, 40)
w <- seq(0, 4, length.out = 128)
freqs(zp, w)
```

`ellipord` *Elliptic Filter Order*

Description

Compute elliptic filter order and cutoff for the desired response characteristics.

Usage

```
ellipord(Wp, Ws, Rp, Rs, plane = c("z", "s"))
```

Arguments

`Wp`, `Ws`	pass-band and stop-band edges. For a low-pass or high-pass filter, `Wp` and `Ws` are scalars. For a band-pass or band-rejection filter, both are vectors of length 2. For a low-pass filter, `Wp` < `Ws`. For a high-pass filter, `Ws` > `Wp`. For a band-pass (`Ws[1]` < `Wp[1]` < `Wp[2]` < `Ws[2]`) or band-reject (`Wp[1]` < `Ws[1]` < `Ws[2]` < `Wp[2]`) filter design, `Wp` gives the edges of the pass band, and `Ws` gives the edges of the stop band. For digital filters, frequencies are normalized to [0, 1], corresponding to the range [0, fs/2]. In case of an analog filter, all frequencies are specified in radians per second.
`Rp`	allowable decibels of ripple in the pass band.
`Rs`	minimum attenuation in the stop band in dB.
`plane`	"z" for a digital filter or "s" for an analog filter.

Value

A list of class `FilterSpecs` with the following list elements:

n filter order

Wc cutoff frequency

type filter type, one of `"low"`, `"high"`, `"stop"`, or `"pass"`.

Rp dB of passband ripple.

Rs dB of stopband ripple.

Author(s)

Paulo Neis, <p_neis@yahoo.com.br>,
adapted by Charles Praplan.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

buttord, cheb1ord, cheb2ord, ellip

Examples

```
fs <- 10000
spec <- ellipord(1000/(fs/2), 1200/(fs/2), 0.5, 29)
ef <- ellip(spec)
hf <- freqz(ef, fs = fs)
plot(c(0, 1000, 1000, 0, 0), c(0, 0, -0.5, -0.5, 0),
     type = "l", xlab = "Frequency (Hz)", ylab = "Attenuation (dB)",
     col = "red", ylim = c(-35,0), xlim = c(0,2000))
lines(c(5000, 1200, 1200, 5000, 5000), c(-1000, -1000, -29, -29, -1000),
      col = "red")
lines(hf$w, 20*log10(abs(hf$h)))
```

---

fftconv *FFT-based convolution*

Description

Convolve two vectors using the FFT for computation.

Usage

```
fftconv(x, y, n = NULL)
```

Arguments

x, y	input vectors.
n	FFT length, specified as a positive integer. The FFT size must be an even power of 2 and must be greater than or equal to the length of filt. If the specified n does not meet these criteria, it is automatically adjusted to the nearest value that does. If n = NULL (default), then the overlap-add method is not used.

Details

The computation uses the FFT by calling the function fftfilt. If the optional argument n is specified, an n-point overlap-add FFT is used.

Value

Convoluted signal, specified as a a vector of length equal to `length (x) + length (y) - 1`. If `x` and `y` are the coefficient vectors of two polynomials, the returned value is the coefficient vector of the product polynomial.

Author(s)

Kurt Hornik, <Kurt.Hornik@wu-wien.ac.at>,
adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

conv, conv2

Examples

```
u <- rep(1L, 3)
v <- c(1, 1, 0, 0, 0, 1, 1)
w1 <- conv(u, v)              # time-domain convolution
w2 <- fftconv(u, v)           # frequency domain convolution
all.equal(w1, w2)             # same results
```

---

fftfilt *FFT-based FIR filtering*

Description

FFT-based FIR filtering using the overlap-add method.

Usage

```
fftfilt(b, x, n = NULL)

## Default S3 method:
fftfilt(b, x, n = NULL)

## S3 method for class 'Ma'
fftfilt(b, x, n = NULL)
```

Arguments

b	moving average (Ma) coefficients of a FIR filter, specified as a vector.
x	the input signal to be filtered. If x is a matrix, its columns are filtered.
n	FFT length, specified as a positive integer. The FFT size must be an even power of 2 and must be greater than or equal to the length of `filt`. If the specified `n` does not meet these criteria, it is automatically adjusted to the nearest value that does. If `n = NULL` (default), then the overlap-add method is not used.

Details

This function combines two important techniques to speed up filtering of long signals, the overlap-add method, and FFT convolution. The overlap-add method is used to break long signals into smaller segments for easier processing or preventing memory problems. FFT convolution uses the overlap-add method together with the Fast Fourier Transform, allowing signals to be convolved by multiplying their frequency spectra. For filter kernels longer than about 64 points, FFT convolution is faster than standard convolution, while producing exactly the same result.

The overlap-add technique works as follows. When an `N` length signal is convolved with a filter kernel of length `M`, the output signal is `N + M - 1` samples long, i.e., the signal is expanded 'to the right'. The signal is then broken into `k` smaller segments, and the convolution of each segment with the f kernel will have a result of length `N / k + M -1`. The individual segments are then added together. The rightmost `M - 1` samples overlap with the leftmost `M - 1` samples of the next segment. The overlap-add method produces exactly the same output signal as direct convolution.

FFT convolution uses the principle that multiplication in the frequency domain corresponds to convolution in the time domain. The input signal is transformed into the frequency domain using the FFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the inverse FFT. With FFT convolution, the filter kernel can be made very long, with very little penalty in execution time.

Value

The filtered signal, returned as a vector or matrix with the same dimensions as `x`.

Author(s)

Kurt Hornik, <Kurt.Hornik@wu-wien.ac.at>,
adapted by John W. Eaton.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Overlap-add_method.

See Also

filter

Examples

```
t <- seq(0, 1, len = 10000)                          # 1 second sample
x <- sin(2* pi * t * 2.3) + 0.25 * rnorm(length(t))  # 2.3 Hz sinusoid+noise
filt <- rep(0.1, 10)                                 # filter kernel
y1 <- filter(filt, 1, x)                             # use normal convolution
y2 <- fftfilt(filt, x)                               # FFT convolution
plot(t, x, type = "l")
lines(t, y1, col = "red")
lines(t, y2, col = "blue")
```

```
## use 'filter' with different classes
t <- seq(0, 1, len = 10000)                          # 1 second sample
x <- sin(2* pi * t * 2.3) + 0.25 * rnorm(length(t))  # 2.3 Hz sinusoid+noise
ma <- Ma(rep(0.1, 10))                               # filter kernel
y1 <- filter(ma, x)                                  # convulution filter
y2 <- fftfilt(ma, x)                                 # FFT filter
all.equal(y1, y2)                                    # same result
```

`fftshift` *Zero-frequency shift*

Description

Perform a shift in order to move the frequency 0 to the center of the input.

Usage

```
fftshift(x, MARGIN = 2)
```

Arguments

`x`	input data, specified as a vector or matrix.
`MARGIN`	dimension to operate along, 1 = row, 2 = columns (default). Specifying `MARGIN = c(1, 2)` centers along both rows and columns. Ignored when `x` is a vector.

Details

If `x` is a vector of `N` elements corresponding to `N` time samples spaced by `dt`, then `fftshift(x)` corresponds to frequencies `f = c(-seq(ceiling((N-1)/2), 1, -1), 0, (1:floor((N-1)/2))) * df`, where `df = 1 / (N * dt)`. In other words, the left and right halves of `x` are swapped.

If `x` is a matrix, then `fftshift` operates on the rows or columns of `x`, according to the `MARGIN` argument, i.e. it swaps the the upper and lower halves of the matrix `(MARGIN = 1)`, or the left and right halves of the matrix `(MARGIN = 2)`. Specifying `MARGIN = c(1, 2)` swaps along both dimensions, i.e., swaps the first quadrant with the fourth, and the second with the third.

Value

vector or matrix with centered frequency.

Author(s)

Vincent Cautaerts, <vincent@comf5.comm.eng.osaka-u.ac.jp>,
adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`ifftshift`

Examples

```
Xeven <- 1:6
ev <- fftshift(Xeven)   # returns 4 5 6 1 2 3

Xodd <- 1:7
odd <- fftshift(Xodd)   # returns 5 6 7 1 2 3 4

fs <- 100                       # sampling frequency
t <- seq(0, 10 - 1/fs, 1/fs)   # time vector
S <- cos(2 * pi * 15 * t)
n <- length(S)
X <- fft(S)
f <- (0:(n - 1)) * (fs / n);   # frequency range
power <- abs(X)^2 / n          # power
plot(f, power, type="l")
Y <- fftshift(X)
fsh <- ((-n/2):(n/2-1)) * (fs / n)  # zero-centered frequency range
powersh <- abs(Y)^2 / n             # zero-centered power
plot(fsh, powersh, type = "l")
```

---

fht *Fast Hartley Transform*

Description

Compute the (inverse) Hartley transform of a signal using FFT

Usage

```
fht(x, n = NROW(x))

ifht(x, n = NROW(x))
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
n	transform length, specified as a positive integer scalar. Default: `NROW(x)`.

Details

The Hartley transform is an integral transform closely related to the Fourier transform, but which transforms real-valued functions to real-valued functions. Compared to the Fourier transform, the Hartley transform has the advantages of transforming real functions to real functions (as opposed to requiring complex numbers) and of being its own inverse [1].

This function implements the Hartley transform by calculating the difference between the real- and imaginary-valued parts of the Fourier-transformed signal [1]. The forward and inverse Hartley

transforms are the same (except for a scale factor of 1/N for the inverse Hartley transform), but implemented using different functions.

Value

(inverse) Hartley transform, returned as a vector or matrix.

Author(s)

Muthiah Annamalai, <muthiah.annamalai@uta.edu>.\ Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Hartley_transform

See Also

fft

Examples

```
# FHT of a 2.5 Hz signal with offset
fs <- 100
secs <- 10
freq <- 2.5
t <- seq(0, secs - 1 / fs, 1 / fs)
x <- 5 * t + 50 * cos(freq * 2 * pi * t)
X <- fht(x)
op <- par(mfrow = c(2, 1))
plot(t, x, type = "l", xlab = "", ylab = "", main = "Signal")
f <- seq(0, fs - (1 / fs), length.out = length(t))
to <- which(f >= 5)[1]
plot(f[1:to], X[1:to], type = "l", xlab = "", ylab = "",
     main = "Hartley Transform")
par(op)
```

---

filter *Filter a signal*

Description

Apply a 1-D digital filter compatible with 'Matlab' and 'Octave'.

Usage

```
filter(filt, ...)

## Default S3 method:
filter(filt, a, x, zi = NULL, ...)

## S3 method for class 'Arma'
filter(filt, x, ...)

## S3 method for class 'Ma'
filter(filt, x, ...)

## S3 method for class 'Sos'
filter(filt, x, ...)

## S3 method for class 'Zpg'
filter(filt, x, ...)
```

Arguments

filt	For the default case, the moving-average coefficients of an ARMA filter (normally called b), specified as a numeric or complex vector. Generically, filt specifies an arbitrary filter operation.
...	additional arguments (ignored).
a	the autoregressive (recursive) coefficients of an ARMA filter, specified as a numeric or complex vector. If a[1] is not equal to 1, then filter normalizes the filter coefficients by a[1]. Therefore, a[1] must be nonzero.
x	the input signal to be filtered, specified as a numeric or complex vector or matrix. If x is a matrix, each column is filtered.
zi	If zi is provided, it is taken as the initial state of the system and the final state is returned as zf. The state vector is a vector or a matrix (depending on x) whose length or number of rows is equal to the length of the longest coefficient vector b or a minus one. If zi is not supplied (NULL), the initial state vector is set to all zeros. Alternatively, zi may be the character string "zf", which specifies to return the final state vector even though the initial state vector is set to all zeros. Default: NULL.

Details

The filter is a direct form II transposed implementation of the standard linear time-invariant difference equation:

$$\sum_{k=0}^{N} a(k+1)y(n-k) + \sum_{k=0}^{M} b(k+1)x(n-k) = 0; 1 \leq n \leq length(x)$$

where N = length(a) - 1 and M = length(b) - 1.

The initial and final conditions for filter delays can be used to filter data in sections, especially if memory limitations are a consideration. See the examples.

Value

The filtered signal, of the same dimensions as the input signal. In case the `zi` input argument was specified, a list with two elements is returned containing the variables `y`, which represents the output signal, and `zf`, which contains the final state vector or matrix.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`filter_zi`, `sosfilt` (preferred because it avoids numerical problems).

Examples

```
bf <- butter(3, 0.1)                                # 10 Hz low-pass filter
t <- seq(0, 1, len = 100)                           # 1 second sample
x <- sin(2* pi * t * 2.3) + 0.25 * rnorm(length(t))  # 2.3 Hz sinusoid+noise
z <- filter(bf, x)                                  # apply filter
plot(t, x, type = "l")
lines(t, z, col = "red")

## specify initial conditions
## from Python scipy.signal.lfilter() documentation
t <- seq(-1, 1, length.out =  201)
x <- (sin(2 * pi * 0.75 * t * (1 - t) + 2.1)
      + 0.1 * sin(2 * pi * 1.25 * t + 1)
      + 0.18 * cos(2 * pi * 3.85 * t))
h <- butter(3, 0.05)
lab <- max(length(h$b), length(h$a)) - 1
zi <- filtic(h$b, h$a, rep(1, lab), rep(1, lab))
z1 <- filter(h, x)
z2 <- filter(h, x, zi * x[1])
plot(t, x, type = "l")
lines(t, z1, col = "red")
lines(t, z2$y, col = "green")
legend("bottomright", legend = c("Original signal",
        "Filtered without initial conditions",
        "Filtered with initial conditions"),
       lty = 1, col = c("black", "red", "green"))
```

`filter.sgolayFilter` *Savitzky-Golay filtering*

Description

Filter a signal with a Savitzky-Golay FIR filter.

Usage

```
## S3 method for class 'sgolayFilter'
filter(filt, x, ...)

sgolayfilt(x, p = 3, n = p + 3 - p%%2, m = 0, ts = 1)
```

Arguments

`filt`	Filter characteristics, usually the result of a call to `sgolay`
`x`	the input signal to be filtered, specified as a vector or as a matrix. If `x` is a matrix, each column is filtered.
`...`	Additional arguments (ignored)
`p`	Polynomial filter order; must be smaller than `n`.
`n`	Filter length; must a an odd positive integer.
`m`	Return the m-th derivative of the filter coefficients. Default: 0
`ts`	Scaling factor. Default: 1

Details

Savitzky-Golay smoothing filters are typically used to "smooth out" a noisy signal whose frequency span (without noise) is large. They are also called digital smoothing polynomial filters or least-squares smoothing filters. Savitzky-Golay filters perform better in some applications than standard averaging FIR filters, which tend to filter high-frequency content along with the noise. Savitzky-Golay filters are more effective at preserving high frequency signal components but less successful at rejecting noise.

Savitzky-Golay filters are optimal in the sense that they minimize the least-squares error in fitting a polynomial to frames of noisy data.

Value

The filtered signal, of the same dimensions as the input signal.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

sgolay

Examples

```
# Compare a 5 sample averager, an order-5 butterworth lowpass
# filter (cutoff 1/3) and sgolayfilt(x, 3, 5), the best cubic
# estimated from 5 points.
bf <- butter(5, 1/3)
x <- c(rep(0, 15), rep(10, 10), rep(0, 15))
sg <- sgolayfilt(x)
plot(sg, type="l", xlab = "", ylab = "")
lines(filtfilt(rep(1, 5) / 5, 1, x), col = "red") # averaging filter
lines(filtfilt(bf, x), col = "blue")              # butterworth
points(x, pch = "x")                              # original data
legend("topleft", c("sgolay (3,5)", "5 sample average", "order 5
Butterworth", "original data"), lty=c(1, 1, 1, NA),
pch = c(NA, NA, NA, "x"), col = c(1, "red", "blue", 1))
```

---

filter2 *2-D digital filter*

Description

Apply a 2-D digital filter to the data in x.

Usage

```
filter2(h, x, shape = c("same", "full", "valid"))
```

Arguments

h	transfer function, specified as a matrix.
x	numeric matrix containing the input signal to be filtered.
shape	Subsection of convolution, partially matched to:
	"same" Return the central part of the filtered data; same size as x (Default)
	"full" Return the full 2-D filtered data, with zero-padding on all sides before filtering
	"valid" Return only the parts which do not include zero-padded edges.

Details

The filter2 function filters data by taking the 2-D convolution of the input x and the coefficient matrix h rotated 180 degrees. More specifically, filter2(h, x, shape) is equivalent to conv2(x,rot90(h, 2), shape).

Value

The filtered signal, returned as a matrix

Author(s)

Paul Kienzle. Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

conv2

Examples

```
op <- par(mfcol = c(1, 2))
x <- seq(-10, 10, length.out = 30)
y <- x
f <- function(x, y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
z[is.na(z)] <- 1
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
title( main = "Original")

h <- matrix(c(1, -2, 1, -2, 3, -2, 1, -2, 1), 3, 3)
zf <-filter2(h, z, 'same')
persp(x, y, zf, theta = 30, phi = 30, expand = 0.5, col = "lightgreen")
title( main = "Filtered")
par(op)
```

FilterSpecs *Filter specifications*

Description

Filter specifications, including order, frequency cutoff, type, and possibly others.

Usage

```
FilterSpecs(n, Wc, type, ...)
```

Arguments

n	filter order.
Wc	cutoff frequency.
type	filter type, normally one of "low", "high", "stop", or "pass".
...	other filter description characteristics, possibly including Rp for dB of pass band ripple or Rs for dB of stop band ripple, depending on filter type (Butterworth, Chebyshev, etc.).

Value

A list of class 'FilterSpecs' with the following list elements (repeats of the input arguments):

n filter order

Wc cutoff frequency

type filter type, normally one of "low", "high", "stop", or "pass".

... other filter description characteristics, possibly including Rp for dB of pass band ripple or Rs for dB of stop band ripple, depending on filter type (Butterworth, Chebyshev, etc.).

Author(s)

Tom Short, <tshort@eprisolutions.com>,
renamed and adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>

See Also

filter, butter and buttord, cheby1 and cheb1ord, ellip and ellipord.

Examples

```
filt <- FilterSpecs(3, 0.1, "low")
```

---

filter_zi *Filter initial conditions*

Description

Construct initial conditions for a filter

Usage

```
filter_zi(filt, ...)

## Default S3 method:
filter_zi(filt, a, ...)

## S3 method for class 'Arma'
filter_zi(filt, ...)

## S3 method for class 'Ma'
filter_zi(filt, ...)

## S3 method for class 'Sos'
filter_zi(filt, ...)

## S3 method for class 'Zpg'
filter_zi(filt, ...)
```

Arguments

filt	For the default case, the moving-average coefficients of an ARMA filter (normally called b), specified as a vector.
...	additional arguments (ignored).
a	the autoregressive (recursive) coefficients of an ARMA filter, specified as a vector.

Details

This function computes an initial state for the filter function that corresponds to the steady state of the step response. In other words, it finds the initial condition for which the response to an input of all ones is a constant. Therefore, the results returned by this function can also be obtained using the function filtic by setting x and y to all 1s (see the examples).

A typical use of this function is to set the initial state so that the output of the filter starts at the same value as the first element of the signal to be filtered.

Value

The initial state for the filter, returned as a vector.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>, converted to R from Python scipy.signal.lfilter_zi.

References

Gustafsson, F. (1996). Determining the initial states in forward-backward filtering. IEEE Transactions on Signal Processing, 44(4), 988 - 992.

See Also

filtic

Examples

```
## taken from Python scipy.signal.lfilter_zi documentation

h <- butter(5, 0.25)
zi <- filter_zi(h)
y <- filter(h, rep(1, 10), zi)
## output is all 1, as expected.
y2 <- filter(h, rep(1, 10))
## if the zi argument is not given, the output
## does not return the final conditions

x <- c(0.5, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0)
y <- filter(h, x, zi = zi*x[1])
## Note that the zi argument to filter was computed using
## filter_zi and scaled by x[1]. Then the output y has no
## transient until the input drops from 0.5 to 0.0.
```

```
## obtain the same results with filtic
lab <- max(length(h$b), length(h$a)) - 1
ic <- filtic(h, rep(1, lab), rep(1, lab))
all.equal(zi, ic)
```

`filtfilt` *Zero-phase digital filtering*

Description

Forward and reverse filter the signal.

Usage

```
filtfilt(filt, ...)

## Default S3 method:
filtfilt(filt, a, x, ...)

## S3 method for class 'Arma'
filtfilt(filt, x, ...)

## S3 method for class 'Ma'
filtfilt(filt, x, ...)

## S3 method for class 'Sos'
filtfilt(filt, x, ...)

## S3 method for class 'Zpg'
filtfilt(filt, x, ...)
```

Arguments

`filt`	For the default case, the moving-average coefficients of an ARMA filter (normally called `b`). Generically, `filt` specifies an arbitrary filter operation.
`...`	additional arguments (ignored).
`a`	the autoregressive (recursive) coefficients of an ARMA filter, specified as a vector. If `a[1]` is not equal to 1, then filter normalizes the filter coefficients by `a[1]`. Therefore, `a[1]` must be nonzero.
`x`	the input signal to be filtered. If `x` is a matrix, all colums are filtered.

Details

Forward and reverse filtering the signal corrects for phase distortion introduced by a one-pass filter, though it does square the magnitude response in the process. That's the theory at least. In practice the phase correction is not perfect, and magnitude response is distorted, particularly in the stop band.

Before filtering the input signal is extended with a reflected part of both ends of the signal. The length of this extension is 3 times the filter order. The Gustafsson [1] method is then used to specify the initial conditions used to further handle the edges of the signal.

Value

The filtered signal, normally of the same length of the input signal `x`, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Francesco Potortì, <pot@gnu.org>,
Luca Citi, <lciti@essex.ac.uk>.
Conversion to R and adapted by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

References

[1] Gustafsson, F. (1996). Determining the initial states in forward-backward filtering. IEEE Transactions on Signal Processing, 44(4), 988 - 992.

See Also

`filter`, `filter_zi`, `Arma`, `Sos`, `Zpg`

Examples

```
bf <- butter(3, 0.1)                                # 10 Hz low-pass filter
t <- seq(0, 1, len = 100)                           # 1 second sample
x <- sin(2* pi * t * 2.3) + 0.25 * rnorm(length(t))  # 2.3 Hz sinusoid+noise
z <- filter(bf, x)                                  # apply filter
plot(t, x, type = "l")
lines(t, z, col = "red")
zz <- filtfilt(bf, x)
lines(t, zz, col="blue")
legend("bottomleft", legend = c("original", "filter", "filtfilt"), lty = 1,
 col = c("black", "red", "blue"))
```

filtic *Filter Initial Conditions*

Description

Compute the initial conditions for a filter.

Usage

```
filtic(filt, ...)

## Default S3 method:
filtic(filt, a, y, x = 0, ...)

## S3 method for class 'Arma'
filtic(filt, y, x = 0, ...)

## S3 method for class 'Ma'
filtic(filt, y, x = 0, ...)

## S3 method for class 'Sos'
filtic(filt, y, x = 0, ...)

## S3 method for class 'Zpg'
filtic(filt, y, x = 0, ...)
```

Arguments

filt	For the default case, the moving-average coefficients of an ARMA filter (normally called b), specified as a vector. Generically, filt specifies an arbitrary filter operation.
...	additional arguments (ignored).
a	the autoregressive (recursive) coefficients of an ARMA filter.
y	output vector, with the most recent values first.
x	input vector, with the most recent values first. Default: 0

Details

This function computes the same values that would be obtained from the function filter given past inputs x and outputs y.

The vectors x and y contain the most recent inputs and outputs respectively, with the newest values first:

x = c(x(-1), x(-2), ... x(-nb)); nb = length(b)-1
y = c(y(-1), y(-2), ... y(-na)); na = length(a)-a

If length(x) < nb then it is zero padded. If length(y) < na then it is zero padded.

Value

Initial conditions for filter specified by `filt`, input vector `x`, and output vector `y`, returned as a vector.

Author(s)

David Billinghurst, <David.Billinghurst@riotinto.com>.
Adapted and converted to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

See Also

`filter`, `sosfilt`, `filtfilt`, `filter_zi`

Examples

```
## Simple low pass filter
b <- c(0.25, 0.25)
a <- c(1.0, -0.5)
ic <- filtic(b, a, 1, 1)

## Simple high pass filter
b <- c(0.25, -0.25)
a <- c(1.0, 0.5)
ic <- filtic(b, a, 0, 1)

## Example from Python scipy.signal.lfilter() documentation
t <- seq(-1, 1, length.out =  201)
x <- (sin(2 * pi * 0.75 * t * (1 - t) + 2.1)
      + 0.1 * sin(2 * pi * 1.25 * t + 1)
      + 0.18 * cos(2 * pi * 3.85 * t))
h <- butter(3, 0.05)
l <- max(length(h$b), length(h$a)) - 1
zi <- filtic(h, rep(1, l), rep(1, l))
z <- filter(h, x, zi * x[1])
```

---

findpeaks *Find local extrema*

Description

Return peak values and their locations of the vector `data`.

Usage

```
findpeaks(
  data,
  MinPeakHeight = .Machine$double.eps,
  MinPeakDistance = 1,
```

```
  MinPeakWidth = 1,
  MaxPeakWidth = Inf,
  DoubleSided = FALSE
)
```

Arguments

`data`	the data, expected to be a vector or one-dimensional array.
`MinPeakHeight`	Minimum peak height (non-negative scalar). Only peaks that exceed this value will be returned. For data taking positive and negative values use the option `DoubleSided`. Default: `.Machine$double.eps`.
`MinPeakDistance`	Minimum separation between peaks (positive integer). Peaks separated by less than this distance are considered a single peak. This distance is also used to fit a second order polynomial to the peaks to estimate their width, therefore it acts as a smoothing parameter. The neighborhood size is equal to the value of `MinPeakDistance`. Default: 1.
`MinPeakWidth`	Minimum width of peaks (positive integer). The width of the peaks is estimated using a parabola fitted to the neighborhood of each peak. The width is calculated with the formula $a * (width - x0)^2 = 1$, where a is the the concavity of the parabola and x0 its vertex. Default: 1.
`MaxPeakWidth`	Maximum width of peaks (positive integer). Default: `Inf`.
`DoubleSided`	Tells the function that data takes positive and negative values. The baseline for the peaks is taken as the mean value of the function. This is equivalent as passing the absolute value of the data after removing the mean. Default: FALSE

Details

Peaks of a positive array of `data` are defined as local maxima. For double-sided data, they are maxima of the positive part and minima of the negative part. `data` is expected to be a one-dimensional vector.

Value

A list containing the following elements:

pks The value of data at the peaks.

loc The index indicating the position of the peaks.

parabol A list containing the parabola fitted to each returned peak. The list has two fields, `x` and `pp`. The field `pp` contains the coefficients of the 2nd degree polynomial and `x` the extrema of the interval where it was fitted.

height The estimated height of the returned peaks (in units of data).

baseline The height at which the roots of the returned peaks were calculated (in units of data).

roots The abscissa values (in index units) at which the parabola fitted to each of the returned peaks realizes its width as defined below.

Author(s)

Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
### demo 1
t <- 2 * pi * seq(0, 1,length = 1024)
y <- sin(3.14 * t) + 0.5 * cos(6.09 * t) +
     0.1 * sin(10.11 * t + 1 / 6) + 0.1 * sin(15.3 * t + 1 / 3)

data1 <- abs(y) # Positive values
peaks1 <- findpeaks(data1)

data2 <- y # Double-sided
peaks2 <- findpeaks(data2, DoubleSided = TRUE)
peaks3 <- findpeaks (data2, DoubleSided = TRUE, MinPeakHeight = 0.5)

op <- par(mfrow=c(1,2))
plot(t, data1, type="l", xlab="", ylab="")
points(t[peaks1$loc], peaks1$pks, col = "red", pch = 1)
plot(t, data2, type = "l", xlab = "", ylab = "")
points(t[peaks2$loc], peaks2$pks, col = "red", pch = 1)
points(t[peaks3$loc], peaks3$pks, col = "red", pch = 4)
legend ("topleft", "0: >2*sd, x: >0.5", bty = "n",
        text.col = "red")
par (op)
title("Finding the peaks of smooth data is not a big deal")

## demo 2
t <- 2 * pi * seq(0, 1, length = 1024)
y <- sin(3.14 * t) + 0.5 * cos(6.09 * t) + 0.1 *
     sin(10.11 * t + 1 / 6) + 0.1 * sin(15.3 * t + 1 / 3)
data <- abs(y + 0.1*rnorm(length(y),1))   # Positive values + noise
peaks1 <- findpeaks(data, MinPeakHeight=1)
dt <- t[2]-t[1]
peaks2 <- findpeaks(data, MinPeakHeight=1, MinPeakDistance=round(0.5/dt))
op <- par(mfrow=c(1,2))
plot(t, data, type="l", xlab="", ylab="")
points (t[peaks1$loc],peaks1$pks,col="red", pch=1)
plot(t, data, type="l", xlab="", ylab="")
points (t[peaks2$loc],peaks2$pks,col="red", pch=1)
par (op)
title(paste("Noisy data may need tuning of the parameters.\n",
            "In the 2nd example, MinPeakDistance is used\n",
            "as a smoother of the peaks"))
```

`fir1` *Window-based FIR filter design*

Description

FIR filter coefficients for a filter with the given order and frequency cutoff.

Usage

```
fir1(
  n,
  w,
  type = c("low", "high", "stop", "pass", "DC-0", "DC-1"),
  window = hamming(n + 1),
  scale = TRUE
)
```

Arguments

n	filter order (1 less than the length of the filter).
w	band edges, strictly increasing vector in the range c(0, 1), where 1 is the Nyquist frequency. A scalar for highpass or lowpass filters, a vector pair for bandpass or bandstop, or a vector for an alternating pass/stop filter.
type	character specifying filter type, one of "low" for a low-pass filter, "high" for a high-pass filter, "stop" for a stop-band (band-reject) filter, "pass" for a pass-band filter, "DC-0" for a bandpass as the first band of a multiband filter, or "DC-1" for a bandstop as the first band of a multiband filter. Default: "low".
window	smoothing window. The returned filter is the same shape as the smoothing window. Default: hamming(n + 1).
scale	whether to normalize or not. Use TRUE (default) or "scale" to set the magnitude of the center of the first passband to 1, and FALSE or "noscale" to not normalize.

Value

The FIR filter coefficients, a vector of length n + 1, of class Ma.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>, Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Fir_filter

See Also

Ma, filter, fftfilt, fir2

Examples

```
freqz(fir1(40, 0.3))
freqz(fir1(10, c(0.3, 0.5), "stop"))
freqz(fir1(10, c(0.3, 0.5), "pass"))
```

---

fir2 *Frequency sampling-based FIR filter design*

Description

Produce a FIR filter with arbitrary frequency response over frequency bands.

Usage

```
fir2(n, f, m, grid_n = 512, ramp_n = NULL, window = hamming(n + 1))
```

Arguments

n	filter order (1 less than the length of the filter).
f	vector of frequency points in the range from 0 to 1, where 1 corresponds to the Nyquist frequency. The first point of f must be 0 and the last point must be 1. f must be sorted in increasing order. Duplicate frequency points are allowed and are treated as steps in the frequency response.
m	vector of the same length as f containing the desired magnitude response at each of the points specified in f.
grid_n	length of ideal frequency response function. grid_n defaults to 512, and should be a power of 2 bigger than n.
ramp_n	transition width for jumps in filter response (defaults to grid_n / 20). A wider ramp gives wider transitions but has better stopband characteristics.
window	smoothing window. The returned filter is the same shape as the smoothing window. Default: hamming(n + 1).

Details

The function linearly interpolates the desired frequency response onto a dense grid and then uses the inverse Fourier transform and a Hamming window to obtain the filter coefficients.

Value

The FIR filter coefficients, a vector of length n + 1, of class Ma.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

Ma, filter, fftfilt, fir1

Examples

```
f <- c(0, 0.3, 0.3, 0.6, 0.6, 1)
m <- c(0, 0, 1, 1/2, 0, 0)
fh <- freqz(fir2(100, f, m))
op <- par(mfrow = c(1, 2))
plot(f, m, type = "b", ylab = "magnitude", xlab = "Frequency")
lines(fh$w / pi, abs(fh$h), col = "blue")
# plot in dB:
plot(f, 20*log10(m+1e-5), type = "b", ylab = "dB", xlab = "Frequency")
lines(fh$w / pi, 20*log10(abs(fh$h)), col = "blue")
par(op)
```

---

firls *Least-squares linear-phase FIR filter design*

Description

Produce a linear phase filter such that the integral of the weighted mean squared error in the specified bands is minimized.

Usage

```
firls(n, f, a, w = rep(1L, length(a)/2))
```

Arguments

n	filter order (1 less than the length of the filter). Must be even. If odd, it is incremented by one.
f	vector of frequency points in the range from 0 to 1, where 1 corresponds to the Nyquist frequency. Each band is specified by two frequencies, so the vector must have an even length. .
a	vector of the same length as f containing the desired amplitude at each of the points specified in f.
w	weighting function that contains one value for each band that weights the mean squared error in that band. w must be half the length of f.

Value

The FIR filter coefficients, a vector of length n + 1, of class Ma.

Author(s)

Quentin Spencer, <qspencer@ieee.org>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

Ma, filter, fftfilt, fir1

Examples

```
freqz(firls(255, c(0, 0.25, 0.3, 1), c(1, 1, 0, 0)))
```

---

flattopwin *Flat top window*

Description

Return the filter coefficients of a flat top window.

Usage

```
flattopwin(n, method = c("symmetric", "periodic"))
```

Arguments

n	Window length, specified as a positive integer.
method	Character string. Window sampling method, specified as:
	"symmetric" (Default). Use this option when using windows for filter design.
	"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When 'periodic' is specified, the function computes a window of length n + 1 and returns the first n points.

Details

The Flat Top window is defined by the function:

$$f(w) = 1 - 1.93cos(2\pi w) + 1.29cos(4\pi w) - 0.388cos(6\pi w) + 0.0322cos(8\pi w)$$

where w = i/(n-1) for i=0:n-1 for a symmetric window, or w = i/n for i=0:n-1 for a periodic window. The default is symmetric. The returned window is normalized to a peak of 1 at w = 0.5.

Flat top windows have very low passband ripple (< 0.01 dB) and are used primarily for calibration purposes. Their bandwidth is approximately 2.5 times wider than a Hann window.

Value

Flat top window, returned as a vector.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
ft <- flattopwin(64)
plot (ft, type = "l", xlab = "Samples", ylab =" Amplitude")
```

---

fracshift *Fractional shift*

Description

Shift a signal by a (possibly fractional) number of samples.

Usage

```
fracshift(x, d, h = NULL)
```

Arguments

x	input data, specified as a numeric vector.
d	number of samples to shift x by, specified as a numeric value
h	interpolator impulse response, specified as a numeric vector. If NULL (default), the interpolator is designed by a Kaiser-windowed sinecard.

Details

The function calculates the initial index and end index of the sequences of 1's in the rows of x. The clusters are sought in the rows of the array x. The function works by finding the indexes of jumps between consecutive values in the rows of x.

Value

A list of matrices size nr, where nr is the number of rows in x. Each element of the list contains a matrix with two rows. The first row is the initial index of a sequence of 1s and the second row is the end index of that sequence. If nr == 1 the output is a matrix with two rows.

Author(s)

Eric Chassande-Mottin, <ecm@apc.univ-paris7.fr>,
Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>,
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] A. V. Oppenheim, R. W. Schafer and J. R. Buck, Discrete-time signal processing, Signal processing series, Prentice-Hall, 1999.
[2] T.I. Laakso, V. Valimaki, M. Karjalainen and U.K. Laine Splitting the unit delay, IEEE Signal Processing Magazine, vol. 13, no. 1, pp 30–59 Jan 1996.

Examples

```
N = 1024
t <- seq(0, 1, length.out = N)
x <- exp(-t^2 / 2 / 0.25^2) * sin(2 * pi * 10 * t)
dt <- 0.25
d  <- dt / (t[2] - t[1])
y <- fracshift(x, d)
plot(t, x, type = "l", xlab = "Time", ylab = "Sigfnal")
lines (t, y, col = "red")
legend("topright", legend = c("original", "shifted"), lty = 1, col = 1:2)
```

---

freqs *Frequency response of analog filters*

Description

Compute the s-plane frequency response of an IIR filter.

Usage

```
freqs(filt, ...)

## Default S3 method:
freqs(filt, a, w, ...)

## S3 method for class 'Arma'
freqs(filt, w, ...)

## S3 method for class 'Ma'
freqs(filt, w, ...)

## S3 method for class 'Sos'
freqs(filt, w, ...)

## S3 method for class 'Zpg'
freqs(filt, w, ...)

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

```
## S3 method for class 'freqs'
summary(object, ...)

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

freqs_plot(x, ...)
```

Arguments

filt	for the default case, moving average (MA) polynomial coefficients, specified as a numeric vector or matrix. In case of a matrix, then each row corresponds to an output of the system. The number of columns of b must be less than or equal to the length of a.
...	for methods of freqs, arguments are passed to the default method. For freqs_plot, additional arguments are passed through to plot.
a	autoregressive (AR) polynomial coefficients, specified as a vector.
w	angular frequencies, specified as a positive real vector expressed in rad/second.
x	object to be printed or plotted.
object	object of class "freqs" for summary

Details

The s-plane frequency response of the IIR filter B(s) / A(s) is computed as H = polyval(B, 1i * W) / polyval(A, 1i * W). If called with no output argument, a plot of magnitude and phase are displayed.

Value

For freqs, a list of class 'freqs' with items:

h complex array of frequency responses at frequencies f.

w array of frequencies.

Author(s)

Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Geert van Boxtel <gjmvanboxtel@gmail.com>

Examples

```
b <- c(1, 2); a <- c(1, 1)
w <- seq(0, 4, length.out = 128)
freqs (b, a, w)
```

freqz *Frequency response of digital filter*

Description

Compute the z-plane frequency response of an ARMA model or rational IIR filter.

Usage

```
freqz(filt, ...)

## Default S3 method:
freqz(
  filt,
  a = 1,
  n = 512,
  whole = ifelse((is.numeric(filt) && is.numeric(a)), FALSE, TRUE),
  fs = 2 * pi,
  ...
)

## S3 method for class 'Arma'
freqz(
  filt,
  n = 512,
  whole = ifelse((is.numeric(filt$b) && is.numeric(filt$a)), FALSE, TRUE),
  fs = 2 * pi,
  ...
)

## S3 method for class 'Ma'
freqz(
  filt,
  n = 512,
  whole = ifelse(is.numeric(filt), FALSE, TRUE),
  fs = 2 * pi,
  ...
)

## S3 method for class 'Sos'
freqz(filt, n = 512, whole = FALSE, fs = 2 * pi, ...)

## S3 method for class 'Zpg'
freqz(filt, n = 512, whole = FALSE, fs = 2 * pi, ...)

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

```
## S3 method for class 'freqz'
summary(object, ...)

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

freqz_plot(w, h, ...)
```

Arguments

filt	for the default case, the moving-average coefficients of an ARMA model or filter. Generically, filt specifies an arbitrary model or filter operation.
...	for methods of freqz, arguments are passed to the default method. For freqz_plot, additional arguments are passed through to plot.
a	the autoregressive (recursive) coefficients of an ARMA filter.
n	number of points at which to evaluate the frequency response. If n is a vector with a length greater than 1, then evaluate the frequency response at these points. For fastest computation, n should factor into a small number of small primes. Default: 512.
whole	FALSE (the default) to evaluate around the upper half of the unit circle or TRUE to evaluate around the entire unit circle.
fs	sampling frequency in Hz. If not specified (default = 2 * pi), the frequencies are in radians.
x	object to be printed or plotted.
object	object of class "freqz" for summary
w	vector of frequencies
h	complex frequency response $H(e^{j\omega})$, specified as a vector.

Details

The frequency response of a digital filter can be interpreted as the transfer function evaluated at $z = e^{j\omega}$.

The 'Matlab' and 'Octave' versions of freqz produce magnitude and phase plots. The freqz version in the 'signal' package produces separate plots of magnitude in the pass band (max - 3 dB to max) and stop (total) bands, as well as a phase plot. The current version produces slightly different plots. The magnitude plots are separate for stop and pass bands, but the pass band plot has an absolute lower limit of -3 dB instead of max - 3 dB. In addition a summary method was added that prints out the most important information about the frequency response of the filter.

Value

For freqz, a list of class 'freqz' with items:

h complex array of frequency responses at frequencies f.

w array of frequencies.

u units of (angular) frequency; either rad/s or Hz.

Note

When results of `freqz` are printed, `freqz_plot` will be called to display frequency plots of magnitude and phase. As with lattice plots, automatic printing does not work inside loops and function calls, so explicit calls to print or plot are needed there.

Author(s)

John W. Eaton, Paul Kienzle, <pkienzle@users.sf.net>.
Port to R by Tom Short,
adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>

Examples

```
b <- c(1, 0, -1)
a <- c(1, 0, 0, 0, 0.25)
freqz(b, a)

hw <- freqz(b, a)
summary(hw)
```

`fwhm` *Full width at half maximum*

Description

Compute peak full-width at half maximum or at another level of peak maximum for a vector or matrix.

Usage

```
fwhm(
  x = seq_len(length(y)),
  y,
  ref = c("max", "zero", "middle", "min", "absolute"),
  level = 0.5
)
```

Arguments

`x`	samples at which `y` is measured, specified as a vector. I.e., `y` is sampled as `y[x]`. Default: `seq_len(length(y))`.
`y`	signal to find the width of. If `y` is a matrix, widths of all columns are computed.
`ref`	reference. Compute the width with reference to:
	`"max"` | `"zero"` `max(y)`
	`"middle"` | `"min"` `min(y)` + `max(y)`
	`"absolute"` an absolute level of `y`
`level`	the level at which to compute the width. Default: 0.5.

Value

Full width at half maximum, returned as a vector with a length equal to the number of columns in y, or 1 in case of a vector.

Author(s)

Petr Mikulik.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
x <- seq(-pi, pi, 0.001)
y <- cos(x)
w <- fwhm(x, y)
m <- x[which.max(y)]
f <- m - w/2
t <- m + w/2
plot(x, y, type="l",
     panel.first = {
       usr <- par('usr')
       rect(f, usr[3], t, usr[4], col = rgb(0, 1, 0, 0.4), border = NA)
     })
abline(h = max(y) / 2, lty = 2, col = "gray")
```

---

gauspuls *Gaussian-modulated sinusoidal RF pulse*

Description

Generate a Gaussian modulated sinusoidal pulse sampled at times t.

Usage

```
gauspuls(t, fc = 1000, bw = 0.5)
```

Arguments

t	Vector of time values at which the unit-amplitude Gaussian RF pulse is calculated.
fc	Center frequency of the Gaussian-modulated sinusoidal pulses, specified as a real positive scalar expressed in Hz. Default: 1000
bw	Fractional bandwidth of the Gaussian-modulated sinusoidal pulses, specified as a real positive scalar.

Value

Inphase Gaussian-modulated sinusoidal pulse, returned as a vector of unit amplitude at the times indicated by the time vector t.

Author(s)

Sylvain Pelissier, Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
fs <- 11025    # arbitrary sample rate
t <- seq(-10, 10, 1/fs)
yi1 <- gauspuls(t, 0.1, 1)
yi2 <- gauspuls(t, 0.1, 2)
plot(t, yi1, type="l", xlab = "Time", ylab = "Amplitude")
lines(t, yi2, col = "red")

fs <- 11025  # arbitrary sample rate
f0 <- 100    # pulse train sample rate
x <- pulstran (seq(0, 4/f0, 1/fs), seq(0, 4/f0, 1/f0), "gauspuls")
plot (0:(length(x)-1) * 1000/fs, x, type="l",
      xlab = "Time (ms)", ylab = "Amplitude",
      main = "Gaussian pulse train at 10 ms intervals")
```

---

gaussian *Gaussian convolution window*

Description

Return a Gaussian convolution window of length n.

Usage

```
gaussian(n, a = 1)
```

Arguments

n	Window length, specified as a positive integer.
a	Width factor, specified as a positive real scalar. a is inversely proportional to the width of the window. Default: 1.

Details

The width of the window is inversely proportional to the parameter a. Use larger a for a narrower window. Use larger m for longer tails.

$$w = e^{(-(a*x)^2/2)}$$

for x <- seq(-(n - 1) / 2, (n - 1) / 2, by = n).

Width a is measured in frequency units (sample rate/num samples). It should be f when multiplying in the time domain, but 1/f when multiplying in the frequency domain (for use in convolutions).

Value

Gaussian convolution window, returned as a vector.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
g1 <- gaussian(128, 1)
g2 <- gaussian(128, 0.5)
plot (g1, type = "l", xlab = "Samples", ylab =" Amplitude", ylim = c(0, 1))
lines(g2, col = "red")
```

`gausswin` *Gaussian window*

Description

Return the filter coefficients of a Gaussian window of length `n`.

Usage

```
gausswin(n, a = 2.5)
```

Arguments

`n`	Window length, specified as a positive integer.
`a`	Width factor, specified as a positive real scalar. `a` is inversely proportional to the width of the window. Default: 2.5.

Details

The width of the window is inversely proportional to the parameter `a`. Use larger `a` for a narrower window. Use larger `m` for a smoother curve.

$$w = e^{(-(a*x)^2/2)}$$

for `x <- seq(-(n - 1) / n, (n - 1) / n, by = n)`.

The exact correspondence with the standard deviation of a Gaussian probability density function is $\sigma = (n-1)/(2a)$.

Value

Gaussian convolution window, returned as a vector.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
g1 <- gausswin(64)
g2 <- gausswin(64, 5)
plot (g1, type = "l", xlab = "Samples", ylab =" Amplitude", ylim = c(0, 1))
lines(g2, col = "red")
```

---

gmonopuls *Gaussian monopulse*

Description

Returns samples of the unit-amplitude Gaussian monopulse.

Usage

```
gmonopuls(t, fc = 1000)
```

Arguments

t	Vector of time values at which the unit-amplitude Gaussian monopulse is calculated.
fc	Center frequency of the Gaussian monopulses, specified as a real positive scalar expressed in Hz. Default: 1000

Value

Samples of the Gaussian monopulse, returned as a vector of unit amplitude at the times indicated by the time vector t.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
fs <- 11025    # arbitrary sample rate
t <- seq(-10, 10, 1/fs)
y1 <- gmonopuls(t, 0.1)
y2 <- gmonopuls(t, 0.2)
plot(t, y1, type="l", xlab = "Time", ylab = "Amplitude")
lines(t, y2, col = "red")
```

```
legend("topright", legend = c("fc = 0.1", "fc = 0.2"),
       lty = 1, col = c(1, 2))
```

---

grpdelay *Group delay*

Description

Compute the average delay of a filter (group delay).

Usage

```
grpdelay(filt, ...)

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

## S3 method for class 'grpdelay'
plot(
  x,
  xlab = if (x$HzFlag) "Frequency (Hz)" else "Frequency (rad/sample)",
  ylab = "Group delay (samples)",
  type = "l",
  ...
)

## Default S3 method:
grpdelay(filt, a = 1, n = 512, whole = FALSE, fs = NULL, ...)

## S3 method for class 'Arma'
grpdelay(filt, ...)

## S3 method for class 'Ma'
grpdelay(filt, ...)

## S3 method for class 'Sos'
grpdelay(filt, ...)

## S3 method for class 'Zpg'
grpdelay(filt, ...)
```

Arguments

filt	for the default case, the moving-average coefficients of an ARMA model or filter. Generically, filt specifies an arbitrary model or filter operation.

...	for methods of grpdelay, arguments are passed to the default method. For plot.grpdelay, additional arguments are passed through to plot.
x	object to be plotted.
xlab, ylab, type	as in plot, but with more sensible defaults.
a	the autoregressive (recursive) coefficients of an ARMA filter.
n	number of points at which to evaluate the frequency response. If n is a vector with a length greater than 1, then evaluate the frequency response at these points. For fastest computation, n should factor into a small number of small primes. Default: 512.
whole	FALSE (the default) to evaluate around the upper half of the unit circle or TRUE to evaluate around the entire unit circle.
fs	sampling frequency in Hz. If not specified, the frequencies are in radians.

Details

If the denominator of the computation becomes too small, the group delay is set to zero. (The group delay approaches infinity when there are poles or zeros very close to the unit circle in the z plane.)

Value

A list of class grpdelay with items:

gd the group delay, in units of samples. It can be converted to seconds by multiplying by the sampling period (or dividing by the sampling rate fs).

w frequencies at which the group delay was calculated.

ns number of points at which the group delay was calculated.

Hzflag TRUE for frequencies in Hz, FALSE for frequencies in radians.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>

References

https://ccrma.stanford.edu/~jos/filters/Numerical_Computation_Group_Delay.html
https://en.wikipedia.org/wiki/Group_delay

Examples

```
# Two Zeros and Two Poles
b <- poly(c(1 / 0.9 * exp(1i * pi * 0.2), 0.9 * exp(1i * pi * 0.6)))
a <- poly(c(0.9 * exp(-1i * pi * 0.6), 1 / 0.9 * exp(-1i * pi * 0.2)))
gpd <- grpdelay(b, a, 512, whole = TRUE, fs = 1)
print(gpd)
plot(gpd)
```

`hamming` *Hamming window*

Description

Return the filter coefficients of a Hamming window of length `n`.

Usage

```
hamming(n, method = c("symmetric", "periodic"))
```

Arguments

`n`	Window length, specified as a positive integer.
`method`	Character string. Window sampling method, specified as:
	"symmetric" (Default). Use this option when using windows for filter design.
	"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When `"periodic"` is specified, the function computes a window of length `n + 1` and returns the first `n` points.

Details

The Hamming window is a member of the family of cosine sum windows.

Value

Hamming window, returned as a vector. If you specify a one-point window (`n = 1`), the value 1 is returned.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
h <- hamming(64)
plot (h, type = "l", xlab = "Samples", ylab =" Amplitude")

hs = hamming(64,'symmetric')
hp = hamming(63,'periodic')
plot (hs, type = "l", xlab = "Samples", ylab =" Amplitude")
lines(hp, col="red")
```

hann *Hann window*

Description

Return the filter coefficients of a Hann window of length n.

Usage

```
hann(n, method = c("symmetric", "periodic"))

hanning(n, method = c("symmetric", "periodic"))
```

Arguments

n	Window length, specified as a positive integer.
method	Character string. Window sampling method, specified as:
	"symmetric" (Default). Use this option when using windows for filter design.
	"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When "periodic" is specified, the function computes a window of length n + 1 and returns the first n points.

Details

The Hann window is a member of the family of cosine sum windows. It was named after Julius von Hann, and is sometimes referred to as Hanning, presumably due to its linguistic and formulaic similarities to Hamming window.

Value

Hann window, returned as a vector.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
h <- hann(64)
plot (h, type = "l", xlab = "Samples", ylab =" Amplitude")

hs = hann(64,'symmetric')
hp = hann(63,'periodic')
plot (hs, type = "l", xlab = "Samples", ylab =" Amplitude")
lines(hp, col="red")
```

`hilbert` *Hilbert transform*

Description

Computes the extension of a real valued signal to an analytic signal.

Usage

```
hilbert(x, n = ifelse(is.vector(x), length(x), nrow(x)))
```

Arguments

`x`	Input array, specified as a vector or a matrix. In case of a matrix, the Hilbert transform of all columns is computed.
`n`	use an n-point FFT to compute the Hilbert transform. The input data is zero-padded or truncated to length n, as appropriate.

Details

The function returns returns a complex helical sequence, sometimes called the analytic signal, from a real data sequence. The analytic signal has a real part, which is the original data, and an imaginary part, which contains the Hilbert transform. The imaginary part is a version of the original real sequence with a 90 degrees phase shift. Sines are therefore transformed to cosines, and conversely, cosines are transformed to sines. The Hilbert-transformed series has the same amplitude and frequency content as the original sequence. The transform includes phase information that depends on the phase of the original.

Value

Analytic signal, of length `n`, returned as a complex vector or matrix, the real part of which contains the original signal, and the imaginary part of which contains the Hilbert transform of `x`.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Peter L. Soendergaard.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>

References

https://en.wikipedia.org/wiki/Hilbert_transform, https://en.wikipedia.org/wiki/Analytic_signal

Examples

```
## notice that the imaginary signal is phase-shifted 90 degrees
t <- seq(0, 10, length = 256)
z <- hilbert(sin(2 * pi * 0.5 * t))
plot(t, Re(z), type = "l", col="blue")
lines (t, Im(z), col = "red")
legend('topright', lty = 1, legend = c("Real", "Imag"),
       col = c("blue", "red"))

## the magnitude of the hilbert transform eliminates the carrier
t <- seq(0, 10, length = 1024)
x <- 5 * cos(0.2 * t) * sin(100 * t)
plot(t, x, type = "l", col = "green")
lines (t, abs(hilbert(x)), col = "blue")
legend('topright', lty = 1, legend = c("x", "|hilbert(x)|"),
        col = c("green", "blue"))
```

---

idct *Inverse Discrete Cosine Transform*

Description

Compute the inverse unitary discrete cosine transform of a signal.

Usage

```
idct(x, n = NROW(x))
```

Arguments

x	input discrete cosine transform, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
n	transform length, specified as a positive integer scalar. Default: `NROW(x)`.

Details

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. You can often reconstruct a sequence very accurately from only a few DCT coefficients. This property is useful for applications requiring data reduction.

Value

Inverse discrete cosine transform, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

dct

Examples

```
x <- seq_len(100) + 50 * cos(seq_len(100) * 2 * pi / 40)
X <- dct(x)

# Find which cosine coefficients are significant (approx.)
# zero the rest
nsig <- which(abs(X) < 1)
N <- length(X) - length(nsig) + 1
X[nsig] <- 0

# Reconstruct the signal and compare it to the original signal.
xx <- idct(X)
plot(x, type = "l")
lines(xx, col = "red")
legend("bottomright", legend = c("Original", paste("Reconstructed, N =", N)),
       lty = 1, col = 1:2)
```

---

idct2 *Inverse 2-D Discrete Cosine Transform*

Description

Compute the inverse two-dimensional discrete cosine transform of a matrix.

Usage

```
idct2(x, m = NROW(x), n = NCOL(x))
```

Arguments

x	2-D numeric matrix
m	Number of rows, specified as a positive integer. dct2 pads or truncates x so that it has m rows. Default: NROW(x).
n	Number of columns, specified as a positive integer. dct2 pads or truncates x so that it has n columns. Default: NCOL(x).

Details

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. It is a separable linear transformation; that is, the two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension.

Value

m-by-n numeric discrete cosine transformed matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

dct2

Examples

```
A <- matrix(50 * runif(100), 10, 10)
B <- dct2(A)
B[which(B < 1)] <- 0
AA <- idct2(B)
```

---

idst *Inverse Discrete Sine Transform*

Description

Compute the inverse discrete sine transform of a signal.

Usage

```
idst(x, n = NROW(x))
```

Arguments

x	input discrete cosine transform, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
n	transform length, specified as a positive integer scalar. Default: NROW(x).

Details

The discrete sine transform (DST) is closely related to the discrete Fourier transform. but using a purely real matrix. It is equivalent to the imaginary parts of a DFT of roughly twice the length.

Value

Inverse discrete sine transform, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

dst

Examples

```
x <- seq_len(100) + 50 * cos(seq_len(100) * 2 * pi / 40)
X <- dst(x)
xx <- idst(X)
all.equal(x, xx)
```

---

ifft *Inverse Fast Fourier Transform*

Description

Compute the inverse Fast Fourier Transform compatible with 'Matlab' and 'Octave'.

Usage

```
ifft(x)

imvfft(x)
```

Arguments

x	Real or complex vector, array, or matrix.

Details

The 'fft' function in the 'stats' package can compute the inverse FFT by specifying inverse = TRUE. However, that function does *not* divide the result by length(x), nor does it return real values when appropriate. The present function does both, and is this compatible with 'Matlab' and 'Octave' (and differs from the 'ifft' function in the 'signal' package, which does not return real values).

Value

When x is a vector, the value computed and returned by ifft is the univariate inverse discrete Fourier transform of the sequence of values in x. Specifically, y <- ifft(x) is defined as stats::fft(x, inverse = TRUE) / length(x). The stats::fft function called with inverse = TRUE replaces exp(-2 * pi...) with exp(2 * pi) in the definition of the discrete Fourier transform (see fft).

When x contains an array, ifft computes and returns the normalized inverse multivariate (spatial) transform. By contrast, imvfft takes a real or complex matrix as argument, and returns a similar shaped matrix, but with each column replaced by its normalized inverse discrete Fourier transform. This is useful for analyzing vector-valued series.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

fft

Examples

```
res <- ifft(stats::fft(1:5))
res <- ifft(stats::fft(c(1+5i, 2+3i, 3+2i, 4+6i, 5+2i)))
res <- imvfft(stats::mvfft(matrix(1:20, 4, 5)))
```

---

ifftshift *Inverse zero-frequency shift*

Description

Rearranges a zero-frequency-shifted Fourier transform back to the original.

Usage

```
ifftshift(x, MARGIN = 2)
```

Arguments

x	input data, specified as a vector or matrix.
MARGIN	dimension to operate along, 1 = row, 2 = columns (default). Specifying MARGIN = c(1, 2) centers along both rows and columns. Ignored when x is a vector.

Details

Undo the action of the fftshift function. For even length x, fftshift is its own inverse, but not for odd length input.

Value

back-transformed vector or matrix.

Author(s)

Vincent Cautaerts, <vincent@comf5.comm.eng.osaka-u.ac.jp>,
adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

fftshift

Examples

```
Xeven <- 1:6
res <- fftshift(fftshift(Xeven))

Xodd <- 1:7
res <- fftshift(fftshift(Xodd))
res <- ifftshift(fftshift(Xodd))
```

---

ifwht *Fast Walsh-Hadamard Transform*

Description

Compute the (inverse) Fast Walsh-Hadamard transform of a signal.

Usage

```
ifwht(x, n = NROW(x), ordering = c("sequency", "hadamard", "dyadic"))

fwht(x, n = NROW(x), ordering = c("sequency", "hadamard", "dyadic"))
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal. fwht operates only on signals with length equal to a power of 2. If the length of x is less than a power of 2, its length is padded with zeros to the next greater power of two before processing.
n	transform length, specified as a positive integer scalar. Default: NROW(x).
ordering	order of the Walsh-Hadamard transform coefficients, one of:
	"sequency" (Default) Coefficients in order of increasing sequency value, where each row has an additional zero crossing.

"hadamard" Coefficients in normal Hadamard order

"dyadic" Coefficients in Gray code order, where a single bit change occurs from one coefficient to the next

Value

(Inverse) Fast Walsh Hadamard transform, returned as a vector or matrix.

Author(s)

Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Hadamard_transform

https://en.wikipedia.org/wiki/Fast_Walsh-Hadamard_transform

Examples

```
x <- c(19, -1, 11, -9, -7, 13, -15, 5)
X <- fwht(x)
all.equal(x, ifwht(X))
```

---

iirlp2mb *IIR lowpass filter to IIR multiband*

Description

Transform an IIR lowpass filter prototype to an IIR multiband filter.

Usage

```
iirlp2mb(b, ...)

## S3 method for class 'Arma'
iirlp2mb(b, Wo, Wt, type, ...)

## S3 method for class 'Zpg'
iirlp2mb(b, Wo, Wt, type, ...)

## S3 method for class 'Sos'
iirlp2mb(b, Wo, Wt, type, ...)

## Default S3 method:
iirlp2mb(b, a, Wo, Wt, type = c("pass", "stop"), ...)
```

Arguments

b	numerator polynomial of prototype low pass filter
...	additional arguments (not used)
Wo	(normalized angular frequency)/pi to be transformed
Wt	vector of (norm. angular frequency)/pi transform targets
type	one of "pass" or "stop". Specifies to filter to produce: bandpass (default) or bandstop.
a	denominator polynomial of prototype low pass filter

Details

The utility of a prototype filter comes from the property that all other filters can be derived from it by applying a scaling factor to the components of the prototype. The filter design need thus only be carried out once in full, with other filters being obtained by simply applying a scaling factor. Especially useful is the ability to transform from one bandform to another. In this case, the transform is more than a simple scale factor. Bandform here is meant to indicate the category of passband that the filter possesses. The usual bandforms are lowpass, highpass, bandpass and bandstop, but others are possible. In particular, it is possible for a filter to have multiple passbands. In fact, in some treatments, the bandstop filter is considered to be a type of multiple passband filter having two passbands. Most commonly, the prototype filter is expressed as a lowpass filter, but other techniques are possible[1].

Filters with multiple passbands may be obtained by applying the general transformation described in [2].

Because `iirlp2mb` is generic, it can be extended to accept other inputs.

Value

List of class `Arma` numerator and denominator polynomials of the resulting filter.

Author(s)

Alan J. Greenberger, <alanjg@ptd.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Prototype_filter
[2] https://en.wikipedia.org/wiki/Prototype_filter#Lowpass_to_multi-band

Examples

```
## Design a prototype real IIR lowpass elliptic filter with a gain of about
## -3 dB at 0.5pi rad/sample.
el <- ellip(3, 0.1, 30, 0.409)
## Create a real multiband filter with two passbands.
mb1 <- iirlp2mb(el, 0.5, c(.2, .4, .6, .8), 'pass')
## Create a real multiband filter with two stopbands.
```

```
mb2 <- iirlp2mb(el, 0.5, c(.2, .4, .6, .8), 'stop')
## Compare the magnitude responses of the filters.
hfl <- freqz(el)
hf1 <- freqz(mb1)
hf2 <- freqz(mb2)
plot(hfl$w, 20 * log10(abs(hfl$h)), type = "l",
    xlab = "Normalized frequency (* pi rad/sample)",
    ylab = "Magnitude (dB)")
lines(hf1$w, 20 * log10(abs(hf1$h)), col="red")
lines(hf2$w, 20 * log10(abs(hf2$h)), col="blue")
legend('bottomleft',
       legend = c('Prototype', 'Two passbands', 'Two Stopbands'),
       col=c("black", "red", "blue"), lty = 1)
```

`impinvar` *Impulse invariance method for A/D filter conversion*

Description

Convert analog filter with coefficients b and a to digital, conserving impulse response.

Usage

```
impinvar(b, ...)

## S3 method for class 'Arma'
impinvar(b, ...)

## Default S3 method:
impinvar(b, a, fs = 1, tol = 1e-04, ...)
```

Arguments

`b`	coefficients of numerator polynomial
`...`	additional arguments (not used)
`a`	coefficients of denominator polynomial
`fs`	sampling frequency (Default: 1 Hz)
`tol`	tolerance. Default: 0.0001

Details

Because `impinvar` is generic, it can also accept input of class `Arma`.

Value

A list of class `Arma` containing numerator and denominator polynomial filter coefficients of the A/D converted filter.

Author(s)

Tony Richardson, <arichard@stark.cc.oh.us>,
Ben Abbott, <bpabbott@mac.com>,
adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

See Also

invimpinvar

Examples

```
f <- 2
fs <- 10
but <- butter(6, 2 * pi * f, 'low', 's')
zbut <- impinvar(but, fs)
freqz(zbut, n = 1024, fs = fs)
```

---

impz *Impulse response of digital filter*

Description

Compute the z-plane impulse response of an ARMA model or rational IIR filter. A plot of the impulse and step responses is generated.

Usage

```
impz(filt, ...)

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

## S3 method for class 'Arma'
impz(filt, ...)

## S3 method for class 'Ma'
impz(filt, ...)

## S3 method for class 'Sos'
impz(filt, ...)

## S3 method for class 'Zpg'
impz(filt, ...)

## Default S3 method:
impz(filt, a = 1, n = NULL, fs = 1, ...)
```

Arguments

filt	for the default case, the moving-average coefficients of an ARMA model or filter. Generically, filt specifies an arbitrary model or filter operation.
...	for methods of freqz, arguments are passed to the default method. For plot.impz, additional arguments are passed through to plot.
x	object to be printed or plotted.
a	the autoregressive (recursive) coefficients of an ARMA filter.
n	number of points at which to evaluate the frequency response. If n is a vector with a length greater than 1, then evaluate the frequency response at these points. For fastest computation, n should factor into a small number of small primes. Default: 512.
fs	sampling frequency in Hz. If not specified (default = 2 * pi), the frequencies are in radians.

Value

For impz, a list of class "impz" with items:

x impulse response signal.

t time.

Note

When results of impz are printed, plot will be called to display a plot of the impulse response against frequency. As with lattice plots, automatic printing does not work inside loops and function calls, so explicit calls to print or plot are needed there.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Tom Short;
adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>

Examples

```
## elliptic low-pass filter
elp <- ellip(4, 0.5, 20, 0.4)
impz(elp)

xt <- impz(elp)
```

`interp` *Interpolation*

Description

Increase sample rate by integer factor.

Usage

```
interp(x, q, n = 4, Wc = 0.5)
```

Arguments

`x`	input data, specified as a numeric vector.
`q`	interpolation factor, specified as a positive integer.
`n`	Half the number of input samples used for interpolation, specified as a positive integer. For best results, use `n` no larger than 10. The low-pass interpolation filter has length `2 × n × q + 1`. Default: 4.
`Wc`	Normalized cutoff frequency of the input signal, specified as a positive real scalar not greater than 1 that represents a fraction of the Nyquist frequency. A value of 1 means that the signal occupies the full Nyquist interval. Default: 0.5.

Value

interpolated signal, returned as a vector.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

See Also

`decimate`, `resample`

Examples

```
# Generate a signal
t <- seq(0, 2, 0.01)
x <- chirp(t, 2, .5, 10,'quadratic') + sin(2 * pi * t * 0.4)
w <- seq(1, 121, 4)
plot(t[w] * 1000, x[w], type = "h", xlab = "", ylab = "")
points(t[w] * 1000, x[w])
abline (h = 0)
y <- interp(x[seq(1, length(x), 4)], 4, 4, 1)
lines(t[1:121] * 1000, y[1:121], type = "l", col = "red")
points(t[1:121] * 1000, y[1:121], col = "red", pch = '+')
```

```
legend("topleft", legend = c("original", "interpolated"),
  lty = 1, pch = c(1, 3), col = c(1, 2))
```

---

invfreq *Inverse Frequency Response*

Description

Identify filter parameters from frequency response data.

Usage

```
invfreq(
  h,
  w,
  nb,
  na,
  wt = rep(1, length(w)),
  plane = c("z", "s"),
  method = c("ols", "tls", "qr"),
  norm = TRUE
)

invfreqs(
  h,
  w,
  nb,
  na,
  wt = rep(1, length(w)),
  method = c("ols", "tls", "qr"),
  norm = TRUE
)

invfreqz(
  h,
  w,
  nb,
  na,
  wt = rep(1, length(w)),
  method = c("ols", "tls", "qr"),
  norm = TRUE
)
```

Arguments

h	Frequency response, specified as a vector

w	Angular frequencies at which h is computed, specified as a vector
nb, na	Desired order of the numerator and denominator polynomials, specified as positive integers.
wt	Weighting factors, specified as a vector of the same length as w. Default: rep(1, length(w))
plane	"z" (default) for discrete-time spectra; "s" for continuous-time spectra
method	minimization method used to solve the normal equations, one of: **"ols"** ordinary least squares (default) **"tls"** total least squares **"qr"** QR decomposition
norm	logical indicating whether frequencies must be normalized to avoid matrices with rank deficiency. Default: TRUE

Details

Given a desired (one-sided, complex) spectrum h(w) at equally spaced angular frequencies $w = (2\pi k)/N$, k = 0, ... N-1, this function finds the filter B(z)/A(z) or B(s)/A(s) with nb zeroes and na poles. Optionally, the fit-errors can be weighted with respect to frequency according to the weights wt.

Value

A list of class 'Arma' with the following list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Julius O. Smith III, Rolf Schirmacher, Andrew Fitting, Pascal Dupuis.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://ccrma.stanford.edu/~jos/filters/FFT_Based_Equation_Error_Method.html

Examples

```
order <- 6  # order of test filter
fc <- 1/2   # sampling rate / 4
n <- 128    # frequency grid size
ba <- butter(order, fc)
hw <- freqz(ba, n)
BA = invfreq(hw$h, hw$w, order, order)
HW = freqz(BA, n)
plot(hw$w, abs(hw$h), type = "l", xlab = "Frequency (rad/sample)",
  ylab = "Magnitude")
lines(HW$w, abs(HW$h), col = "red")
```

```
legend("topright", legend = c("Original", "Measured"), lty  = 1, col = 1:2)
err <- norm(hw$h - HW$h, type = "2")
title(paste('L2 norm of frequency response error =', err))
```

`invimpinvar` *Inverse impulse invariance method*

Description

Convert digital filter with coefficients b and a to analog, conserving impulse response.

Usage

```
invimpinvar(b, ...)

## S3 method for class 'Arma'
invimpinvar(b, ...)

## Default S3 method:
invimpinvar(b, a, fs = 1, tol = 1e-04, ...)
```

Arguments

`b`	coefficients of numerator polynomial
`...`	additional arguments (not used)
`a`	coefficients of denominator polynomial
`fs`	sampling frequency (Default: 1 Hz)
`tol`	tolerance. Default: 0.0001

Details

Because `invimpinvar` is generic, it can also accept input of class `Arma`.

Value

A list of class `Arma` containing numerator and denominator polynomial filter coefficients of the A/D converted filter.

Author(s)

R.G.H. Eschauzier, <reschauzier@yahoo.com>,
Carne Draug, <carandraug+dev@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

References

Thomas J. Cavicchi (1996) Impulse invariance and multiple-order poles. IEEE transactions on signal processing, Vol 40 (9): 2344–2347.

See Also

impinvar

Examples

```
f <- 2
fs <- 10
but <- butter(6, 2 * pi * f, 'low', 's')
zbut <- impinvar(but, fs)
sbut <- invimpinvar(zbut, fs)
all.equal(but, sbut, tolerance = 1e-7)
```

---

kaiser *Kaiser window*

Description

Return the filter coefficients of a kaiser window of length n.

Usage

```
kaiser(n, beta = 0.5)
```

Arguments

n	Window length, specified as a positive integer.
beta	Shape factor, specified as a positive real scalar. The parameter beta affects the side lobe attenuation of the Fourier transform of the window. Default: 0.5

Details

The Kaiser, or Kaiser-Bessel, window is a simple approximation of the DPSS window using Bessel functions, discovered by James Kaiser.

$$w(x) = \frac{besselI(0, \beta \cdot \sqrt{(1 - (2 * x/m)^2)})}{besselI(0, \beta)}; -m/2 <= x <= m/2$$

The variable parameter β determines the trade-off between main lobe width and side lobe levels of the spectral leakage pattern. Increasing β widens the main lobe and decreases the amplitude of the side lobes (i.e., increases the attenuation).

Value

Kaiser window, returned as a vector.

Author(s)

Kurt Hornik, <Kurt.Hornik@ci.tuwien.ac.at>,
Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
k <- kaiser(200, 2.5)
plot (k, type = "l", xlab = "Samples", ylab =" Amplitude")
```

---

kaiserord *Kaiser filter order and cutoff frequency*

Description

Return the parameters needed to produce a FIR filter of the desired specification from a Kaiser window.

Usage

```
kaiserord(f, m, dev, fs = 2)
```

Arguments

f	frequency bands, given as pairs, with the first half of the first pair assumed to start at 0 and the last half of the last pair assumed to end at 1. It is important to separate the band edges, since narrow transition regions require large order filters.
m	magnitude within each band. Should be non-zero for pass band and zero for stop band. All passbands must have the same magnitude, or you will get the error that pass and stop bands must be strictly alternating.
dev	deviation within each band. Since all bands in the resulting filter have the same deviation, only the minimum deviation is used. In this version, a single scalar will work just as well.
fs	sampling rate. Used to convert the frequency specification into the c(0, 1) range, where 1 corresponds to the Nyquist frequency, fs / 2.

Details

Given a set of specifications in the frequency domain, `kaiserord` estimates the minimum FIR filter order that will approximately meet the specifications. `kaiserord` converts the given filter specifications into passband and stopband ripples and converts cutoff frequencies into the form needed for windowed FIR filter design.

`kaiserord` uses empirically derived formulas for estimating the orders of lowpass filters, as well as differentiators and Hilbert transformers. Estimates for multiband filters (such as band-pass filters) are derived from the low-pass design formulas.

The design formulas that underlie the Kaiser window and its application to FIR filter design are

$$\beta =$$

$$0.1102(\alpha - 8.7), \alpha > 50$$
$$0.5842(\alpha - 21)^{0.4} + 0.07886(\alpha - 21), 21 \leq \alpha \leq 50$$
$$0, \alpha < 21$$

where $\alpha = -20log_{10}(\delta)$ is the stopband attenuation expressed in decibels, $n = (\alpha - 8)/2.285(\Delta\omega)$, where n is the filter order and $\Delta\omega$ is the width of the smallest transition region.

Value

A list of class `FilterSpecs` with the following list elements:

n filter order

Wc cutoff frequency

type filter type, one of "low", "high", "stop", "pass", "DC-0", or "DC-1".

beta shape parameter

Author(s)

Paul Kienzle.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`hamming`, `kaiser`

Examples

```
fs <- 11025
op <- par(mfrow = c(2, 2), mar = c(3, 3, 1, 1))
for (i in 1:4) {
  if (i == 1) {
    bands <- c(1200, 1500)
    mag <- c(1, 0)
    dev <- c(0.1, 0.1)
  }
```

```
  if (i == 2) {
    bands <- c(1000, 1500)
    mag <- c(0, 1)
    dev <- c(0.1, 0.1)
  }
  if (i == 3) {
    bands <- c(1000, 1200, 3000, 3500)
    mag <- c(0, 1, 0)
    dev <- 0.1
  }
  if (i == 4) {
    bands <- 100 * c(10, 13, 15, 20, 30, 33, 35, 40)
    mag <- c(1, 0, 1, 0, 1)
    dev <- 0.05
  }
  kaisprm <- kaiserord(bands, mag, dev, fs)
  d <- max(1, trunc(kaisprm$n / 10))
  if (mag[length(mag)] == 1 && (d %% 2) == 1) {
     d <- d + 1
  }
  f1 <- freqz(fir1(kaisprm$n, kaisprm$Wc, kaisprm$type,
                   kaiser(kaisprm$n + 1, kaisprm$beta),
                   scale = FALSE),
              fs = fs)
  f2 <- freqz(fir1(kaisprm$n - d, kaisprm$Wc, kaisprm$type,
                   kaiser(kaisprm$n - d + 1, kaisprm$beta),
                   scale = FALSE),
              fs = fs)
  plot(f1$w, abs(f1$h), col = "blue", type = "l",  xlab = "", ylab = "")
  lines(f2$w, abs(f2$h), col = "red")
  legend("right", paste("order", c(kaisprm$n-d, kaisprm$n)),
         col = c("red", "blue"), lty = 1, bty = "n")
  b <- c(0, bands, fs/2)
  for (i in seq(2, length(b), by=2)) {
    hi <- mag[i/2] + dev[1]
    lo <- max(mag[i/2] - dev[1], 0)
    lines(c(b[i-1], b[i], b[i], b[i-1], b[i-1]), c(hi, hi, lo, lo, hi))
  }
}
par(op)
```

`levinson` *Durbin-Levinson Recursion*

Description

Use the Durbin-Levinson algorithm to compute the coefficients of an autoregressive linear process.

Usage

```
levinson(acf, p = NROW(acf))
```

Arguments

acf	autocorrelation function for lags 0 to p, specified as a vector or matrix. If r is a matrix, the function finds the coefficients for each column of acf and returns them in the rows of a.
p	model order, specified as a positive integer. Default: NROW(acf) - 1.

Details

levinson uses the Durbin-Levinson algorithm to solve:

$$toeplitz(acf(1:p)) * x = -acf(2:p+1)$$

The solution c(1, x) is the denominator of an all pole filter approximation to the signal x which generated the autocorrelation function acf.

From ref [2]: Levinson recursion or Levinson–Durbin recursion is a procedure in linear algebra to recursively calculate the solution to an equation involving a Toeplitz matrix. Other methods to process data include Schur decomposition and Cholesky decomposition. In comparison to these, Levinson recursion (particularly split Levinson recursion) tends to be faster computationally, but more sensitive to computational inaccuracies like round-off errors.

Value

A list containing the following elements:

a vector or matrix containing (p+1) autoregression coefficients. If x is a matrix, then each row of a corresponds to a column of x. a has p + 1 columns.

e white noise input variance, returned as a vector. If x is a matrix, then each element of e corresponds to a column of x.

k Reflection coefficients defining the lattice-filter embodiment of the model returned as vector or a matrix. If x is a matrix, then each column of k corresponds to a column of x. k has p rows.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] Steven M. Kay and Stanley Lawrence Marple Jr. (1981). Spectrum analysis – a modern perspective. Proceedings of the IEEE, Vol 69, 1380-1419.
[2] https://en.wikipedia.org/wiki/Levinson_recursion

Examples

```
## Estimate the coefficients of an autoregressive process given by
## x(n) = 0.1x(n-1) - 0.8x(n-2) - 0.27x(n-3) + w(n).
a <- c(1, 0.1, -0.8, -0.27)
v <- 0.4
```

```
w <- sqrt(v) * rnorm(15000)
x <- filter(1, a, w)
xc <- xcorr(x, scale = 'biased')
acf <- xc$R[-which(xc$lags < 0)]
lev <- levinson(acf, length(a) - 1)
```

Ma *Moving average (MA) model*

Description

Create an MA model representing a filter or system model

Usage

```
Ma(b)
```

Arguments

b	moving average (MA) polynomial coefficients.

Value

A list of class Ma with the polynomial coefficients

Author(s)

Tom Short, <tshort@eprisolutions.com>

See Also

See also Arma

Examples

```
f <- Ma(b = c(1, 2, 1) / 3)
freqz(f)
zplane(f)
```

`marcumq` *Marcum Q function*

Description

Compute the generalized Marcum Q function

Usage

```
marcumq(a, b, m = 1)
```

Arguments

`a, b`	input arguments, specified as non-negative real numbers.
`m`	order, specified as a positive integer

Details

The code for this function was taken from the help file of the `cdfkmu` function in the `lmomco` package, based on a suggestion of Daniel Wollschlaeger.

Value

Marcum Q function.

Author(s)

William Asquith, <william.asquith@ttu.edu>.

References

https://cran.r-project.org/package=lmomco

Examples

```
mq <- marcumq(12.4, 12.5)
```

`medfilt1` *1-D median filtering*

Description

Apply a running median of odd span to the input `x`

Usage

```
medfilt1(x, n = 3, MARGIN = 2, na.omit = FALSE, ...)
```

Arguments

`x`	Input signal, specified as a numeric vector, matrix or array.
`n`	positive integer width of the median window; must be odd. Default: 3
`MARGIN`	Vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names. Default: 2 (columns).
`na.omit`	logical indicating whether to omit missing values, or interpolate then using a cubic spline function (`splinefun`). Default: FALSE
`...`	other arguments passed to `runmed`

Details

This function computes a running median over the input `x`, using the `runmed` function. Because of that, it works a little differently than the 'Matlab' or 'Octave' versions (i.e., it does not produce exactly the same values).

missing values The 'Mablab' and 'Octave' functions have a `'nanflag'` option that allows to include or remove missing values. If inclusion is specifies, then the function returns a signal so that the median of any segment containing NAs is also NA. Because the `'runmed'` function does not include an `na.omit` option, implementing this functionality would lead to a considerable speed loss. Instead, a `na.omit` parameter was implemented that allows either omitting NAs or interpolating them with a spline function.

endpoint filtering Instead of the `'zeropad'` and `'truncate'` options to the `'padding'` argument in the 'Matlab' and 'Octave' functions, the present version uses the standard `endrule` parameter of the `'runmed'` function, with options `keep`, `constant`, or `median`.

Value

Filtered signal, returned as a numeric vector, matrix, or array, of the same size as `x`.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

runmed, splinefun

Examples

```
## noise suppression
fs <- 100
t <- seq(0, 1, 1/fs)
x <- sin(2 * pi * t * 3) + 0.25 * sin(2 * pi * t * 40)
plot(t, x, type = "l", xlab = "", ylab = "")
y <- medfilt1(x, 11)
lines (t, y, col = "red")
legend("topright", c("Original", "Filtered"), lty = 1, col = 1:2)
```

---

mexihat *Mexicat Hat*

Description

Generate a Mexican Hat (Ricker) wavelet sampled on a regular grid.

Usage

```
mexihat(lb = -5, ub = 5, n = 1000)
```

Arguments

lb, ub	Lower and upper bounds of the interval to evaluate the wavelet on. Default: -5 to 5.
n	Number of points on the grid between lb and ub (length of the wavelet). Default: 1000.

Details

The Mexican Hat or Ricker wavelet is the negative normalized second derivative of a Gaussian function, i.e., up to scale and normalization, the second Hermite function. It is a special case of the family of continuous wavelets (wavelets used in a continuous wavelet transform) known as Hermitian wavelets. The Ricker wavelet is frequently employed to model seismic data, and as a broad spectrum source term in computational electrodynamics. It is usually only referred to as the Mexican hat wavelet in the Americas, due to taking the shape of a sombrero when used as a 2D image processing kernel. It is also known as the Marr wavelet (source: Wikipedia)

Value

A list containing 2 variables; x, the grid on which the complex Mexican Hat wavelet was evaluated, and psi (Ψ), the evaluated wavelet on the grid x.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
mh <- mexihat(-5, 5, 1000)
plot(mh$x, mh$psi, type="l", main = "Mexican Hat Wavelet",
     xlab = "", ylab = "")
```

---

meyeraux *Meyer wavelet auxiliary function*

Description

Compute the Meyer wavelet auxiliary function.

Usage

```
meyeraux(x)
```

Arguments

x	Input array, specified as a real scalar, vector, matrix, or multidimensional array.

Details

The code y = meyeraux(x) returns values of the auxiliary function used for Meyer wavelet generation evaluated at the elements of x. The input x is a vector or matrix of real values. The function is

$$y = 35x^4 - 84x^5 + 70x^6 - 20x^7.$$

x and y have the same dimensions. The range of meyeraux is the closed interval c(0, 1).

Value

Output array, returned as a real-valued scalar, vector, matrix, or multidimensional array of the same size as x.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
x <- seq(0, 1, length.out = 100)
y <- meyeraux(x)
plot(x, y, type="l", main = "Meyer wavelet auxiliary function",
     xlab = "", ylab = "")
```

`morlet` *Morlet Wavelet*

Description

Compute the Morlet wavelet on a regular grid.

Usage

```
morlet(lb = -4, ub = 4, n = 1000)
```

Arguments

`lb`, `ub`	Lower and upper bounds of the interval to evaluate the wavelet on. Default: -4 to 4.
`n`	Number of points on the grid between `lb` and `ub` (length of the wavelet). Default: 1000.

Details

The code `m <- morlet(lb, ub, n)` returns values of the Morlet wavelet on an `n`-point regular grid in the interval `c(lb, ub)`.

The Morlet waveform is defined as

$$\psi(x) = e^{-x^2/2} cos(5x)$$

Value

A list containing 2 variables; `x`, the grid on which the Morlet wavelet was evaluated, and `psi` (Ψ), the evaluated wavelet on the grid `x`.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
m <- morlet(-4, 4, 1000)
plot(m$x, m$psi, type="l", main = "Morlet Wavelet", xlab = "", ylab = "")
```

`movingrms` *Moving Root Mean Square*

Description

Compute the moving root mean square (RMS) of the input signal.

Usage

```
movingrms(x, width = 0.1, rc = 0.001, fs = 1)
```

Arguments

`x`	Input signal, specified as a numeric vector or matrix. In case of a matrix, the function operates along the columns
`width`	width of the sigmoid window, in units relative to `fs`. Default: 0.1
`rc`	Rise time (time constant) of the sigmoid window, in units relative to `fs`. Default: 1e-3
`fs`	Sampling frequency. Default: 1

Details

The signal is convoluted against a sigmoid window of width `w` and risetime `rc`. The units of these parameters are relative to the value of the sampling frequency given in `fs`.

Value

A `list` containing 2 variables:

rmsx Output signal with the same dimensions as `x`

w Window, returned as a vector

Author(s)

Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`sigmoid_train`

Examples

```
N <- 128
fs <- 5
t <- seq(0, 1, length.out = N)
x <- sin(2 * pi * fs * t) + runif(N)
y <- movingrms(x, 5)
```

mpoles *Multiplicity of poles*

Description

Identify unique poles and their associated multiplicity.

Usage

```
mpoles(p, tol = 0.001, reorder = TRUE, index.return = FALSE)
```

Arguments

p	vector of poles.
tol	tolerance. If the relative difference of two poles is less than tol then they are considered to be multiples. The default value for tol is 0.001.
reorder	logical. If TRUE, (default), the output is ordered from largest pole to smallest pole.
index.return	logical indicating if index vector should be returned as well. See examples. Default: FALSE.

Value

If index.return = TRUE, a list consisting of two vectors:

m vector specifying the multiplicity of the poles

n index

If index.return = FALSE, only m is returned (as a vector).

Author(s)

Ben Abbott, <bpabbott@mac.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

See Also

poly, residue

Examples

```
p <- c(2, 3, 1, 1, 2)
ret <- mpoles(p, index = TRUE)
```

`mscohere` *Magnitude-squared coherence*

Description

Compute the magnitude-squared coherence estimates of input signals.

Usage

```
mscohere(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none")
)

cohere(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none")
)
```

Arguments

`x`	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
`window`	If `window` is a vector, each segment has the same length as `window` and is multiplied by `window` before (optional) zero-padding and calculation of its periodogram. If `window` is a scalar, each segment has a length of `window` and a Hamming window is used. Default: `nextpow2(sqrt(length(x)))` (the square root of the length of `x` rounded up to the next power of two). The window length must be larger than 3.
`overlap`	segment overlap, specified as a numeric value expressed as a multiple of window or segment length. 0 <= overlap < 1. Default: 0.5.
`nfft`	Length of FFT, specified as an integer scalar. The default is the length of the `window` vector or has the same value as the scalar `window` argument. If `nfft` is larger than the segment length, (seg_len), the data segment is padded `nfft - seg_len` zeros. The default is no padding. Nfft values smaller than the length of the data segment (or window) are ignored. Note that the use of padding to increase the frequency resolution of the spectral estimate is controversial.
`fs`	sampling frequency (Hertz), specified as a positive scalar. Default: 1.

detrend character string specifying detrending option; one of:

- long-mean remove the mean from the data before splitting into segments (default)
- short-mean remove the mean value of each segment
- long-linear remove linear trend from the data before splitting into segments
- short-linear remove linear trend from each segment
- none no detrending

Details

mscohere estimates the magnitude-squared coherence function using Welch's overlapped averaged periodogram method [1]

Value

A list containing the following elements:

freq vector of frequencies at which the spectral variables are estimated. If x is numeric, power from negative frequencies is added to the positive side of the spectrum, but not at zero or Nyquist (fs/2) frequencies. This keeps power equal in time and spectral domains. If x is complex, then the whole frequency range is returned.

coh NULL for univariate series. For multivariate series, a matrix containing the squared coherence between different series. Column $i + (j-1) * (j-2)/2$ of coh contains the cross-spectral estimates between columns i and j of x, where $i < j$.

Note

The function mscohere (and its deprecated alias cohere) is a wrapper for the function pwelch, which is more complete and more flexible.

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] Welch, P.D. (1967). The use of Fast Fourier Transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. IEEE Transactions on Audio and Electroacoustics, AU-15 (2): 70–73.

Examples

```
fs <- 1000
f <- 250
t <- seq(0, 1 - 1/fs, 1/fs)
s1 <- sin(2 * pi * f * t) + runif(length(t))
s2 <- sin(2 * pi * f * t - pi / 3) + runif(length(t))
```

```
rv <- mscohere(cbind(s1, s2), fs = fs)
plot(rv$freq, rv$coh, type="l", xlab = "Frequency", ylab = "Coherence")
```

---

ncauer *ncauer analog filter design*

Description

Compute the transfer function coefficients of a Cauer analog filter.

Usage

```
ncauer(Rp, Rs, n)
```

Arguments

Rp	dB of passband ripple.
Rs	dB of stopband ripple.
n	filter order.

Details

Cauer filters have equal maximum ripple in the passband and the stopband. The Cauer filter has a faster transition from the passband to the stopband than any other class of network synthesis filter. The term Cauer filter can be used interchangeably with elliptical filter, but the general case of elliptical filters can have unequal ripples in the passband and stopband. An elliptical filter in the limit of zero ripple in the passband is identical to a Chebyshev Type 2 filter. An elliptical filter in the limit of zero ripple in the stopband is identical to a Chebyshev Type 1 filter. An elliptical filter in the limit of zero ripple in both passbands is identical to a Butterworth filter. The filter is named after Wilhelm Cauer and the transfer function is based on elliptic rational functions.Cauer-type filters use generalized continued fractions.[1]

Value

A list of class Zpg with the following list elements:

zero complex vector of the zeros of the model

pole complex vector of the poles of the model

gain gain of the model

Author(s)

Paulo Neis, <p_neis@yahoo.com.br>.
Conversion to R Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] https://en.wikipedia.org/wiki/Network_synthesis_filters#Cauer_filter

See Also

Zpg, filter, ellip

Examples

```
zpg <- ncauer(1, 40, 5)
freqz(zpg)
zplane(zpg)
```

---

nuttallwin *Nuttall-defined minimum 4-term Blackman-Harris window*

Description

Return the filter coefficients of a Blackman-Harris window defined by Nuttall of length n.

Usage

```
nuttallwin(n, method = c("symmetric", "periodic"))
```

Arguments

n	Window length, specified as a positive integer.
method	Character string. Window sampling method, specified as:

"symmetric" (Default). Use this option when using windows for filter design.

"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When periodic is specified, the function computes a window of length n + 1 and returns the first n points.

Details

The window is minimum in the sense that its maximum sidelobes are minimized. The coefficients for this window differ from the Blackman-Harris window coefficients computed with blackmanharris and produce slightly lower sidelobes.

Value

Nuttall-defined Blackman-Harris window, returned as a vector.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

blackman, blackmanharris

Examples

```
n <- nuttallwin(64)
plot (n, type = "l", xlab = "Samples", ylab =" Amplitude")
```

---

pad *Pad data*

Description

Pre- or postpad the data object x with the value c until it is of length l.

Usage

```
pad(x, l, c = 0, MARGIN = 2, direction = c("both", "pre", "post"))

prepad(x, l, c = 0, MARGIN = 2)

postpad(x, l, c = 0, MARGIN = 2)
```

Arguments

x	Vector or matrix to be padded
l	Length of output data along the padding dimension. If length (x) > l, elements from the beginning (dimension = "pre") or the end (direction = "post") of x are removed until a vector of length l is obtained. If direction = "both", values are removed from both ends, and in case of an uneven length the smallest number of elements is removed from the beginning of vector.
c	Value to be used for the padding (scalar). Must be of the same type as the elements in x. Default: 0
MARGIN	A vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where x has named dimnames, it can be a character vector selecting dimension names. If MARGIN is larger than the dimensions of x, the result will have MARGIN dimensions. Default: 2 (columns).
direction	Where to pad the array along each dimension. One of the following: **"pre"** Before the first element **"post"** After the last element **"both"** (default) Before the first and after the last element

Value

Padded data, returned as a vector or matrix.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
v <- 1:24
res <- postpad(v, 30)
res <- postpad(v, 20)
res <- prepad(v, 30)
res <- prepad(v, 20)

m <- matrix(1:24, 4, 6)
res <- postpad(m, 8, 100)
res <- postpad(m, 8, 100, MARGIN = 1)
res <- prepad(m, 8, 100)
res <- prepad(m, 8, 100, MARGIN = 1)

res <- postpad(m, 2)
res <- postpad(m, 2, MARGIN = 1)
res <- prepad(m, 2)
res <- prepad(m, 2, MARGIN = 1)
```

---

parzenwin *Parzen (de la Vallée Poussin) window*

Description

Return the filter coefficients of a Parzen window of length n.

Usage

```
parzenwin(n)
```

Arguments

n	Window length, specified as a positive integer.

Details

Parzen windows are piecewise-cubic approximations of Gaussian windows.

Value

Parzen window, returned as a vector.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
p <- parzenwin(64)
g <- gausswin(64)
plot (p, type = "l", xlab = "Samples", ylab =" Amplitude", ylim = c(0, 1))
lines(g, col = "red")
```

---

pburg *Autoregressive PSD estimate - Burg's method*

Description

Calculate Burg maximum-entropy power spectral density.

Usage

```
pburg(
  x,
  p,
  criterion = NULL,
  freq = 256,
  fs = 1,
  range = NULL,
  method = if (length(freq) == 1 && bitwAnd(freq, freq - 1) == 0) "fft" else "poly"
)
```

Arguments

x	input data, specified as a numeric or complex vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
p	model order; number of poles in the AR model or limit to the number of poles if a valid criterion is provided. Must be < length(x) - 2.
criterion	model-selection criterion. Limits the number of poles so that spurious poles are not added when the whitened data has no more information in it. Recognized values are:
	"AKICc" approximate corrected Kullback information criterion (recommended)
	"KIC" Kullback information criterion
	"AICc" corrected Akaike information criterion
	"AIC" Akaike information criterion
	"FPE" final prediction error

The default is to NOT use a model-selection criterion (NULL)

`freq` vector of frequencies at which power spectral density is calculated, or a scalar indicating the number of uniformly distributed frequency values at which spectral density is calculated. Default: 256.

`fs` sampling frequency (Hz). Default: 1

`range` character string. one of:

`"half"` **or** `"onesided"` frequency range of the spectrum is from zero up to but not including `fs / 2`. Power from negative frequencies is added to the positive side of the spectrum.

`"whole"` **or** `"twosided"` frequency range of the spectrum is `-fs / 2` to `fs / 2`, with negative frequencies stored in "wrap around order" after the positive frequencies; e.g. frequencies for a 10-point `"twosided"` spectrum are 0 0.1 0.2 0.3 0.4 0.5 -0.4 -0.3 -0.2. -0.1.

`"shift"` **or** `"centerdc"` same as `"whole"` but with the first half of the spectrum swapped with second half to put the zero-frequency value in the middle. If `freq` is a vector, `"shift"` is ignored.

Default: If model coefficients `a` are real, the default range is `"half"`, otherwise the default range is `"whole"`.

`method` method used to calculate the power spectral density, either `"fft"` (use the Fast Fourier Transform) or `"poly"` (calculate the power spectrum as a polynomial). This argument is ignored if the `freq` argument is a vector. The default is `"poly"` unless the `freq` argument is an integer power of 2.

Value

An object of class "ar_psd" , which is a list containing two elements, `freq` and `psd` containing the frequency values and the estimates of power-spectral density, respectively.

Note

This function is a wrapper for `arburg` and `ar_psd`.

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>

See Also

`ar_psd`, `arburg`

Examples

```
A <- Arma(1, c(1, -2.7607, 3.8106, -2.6535, 0.9238))
y <- filter(A, 0.2 * rnorm(1024))
plot(pb <- pburg(y, 4))
```

peak2peak *Maximum-to-minimum difference*

Description

Compute the maximum-to-minimum difference of the input data x.

Usage

```
peak2peak(x, MARGIN = 2)
```

Arguments

x	the data, expected to be a vector, a matrix, an array.
MARGIN	a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where x has named dimnames, it can be a character vector selecting dimension names. Default: 2 (columns)

Details

The input x can be a vector, a matrix or an array. If the input is a vector, a single value is returned representing the maximum-to-minimum difference of the vector. If the input is a matrix or an array, a vector or an array of values is returned representing the maximum-to-minimum differences of the dimensions of x indicated by the MARGIN argument.

Support for complex valued input is provided. In this case, the function peak2peak identifies the maximum and minimum in complex magnitude, and then subtracts the complex number with the minimum modulus from the complex number with the maximum modulus.

Value

Vector or array of values containing the maximum-to-minimum differences of the specified MARGIN of x.

Author(s)

Georgios Ouzounis, <ouzounis_georgios@hotmail.com>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## numeric vector
x <- c(1:5)
pp <- peak2peak(x)

## numeric matrix
x <- matrix(c(1,2,3, 100, 150, 200, 1000, 1500, 2000), 3, 3)
pp <- peak2peak(x)
```

```
pp <- peak2peak(x, 1)

## numeric array
x <- array(c(1, 1.5, 2, 100, 150, 200, 1000, 1500, 2000,
             10000, 15000, 20000), c(2,3,2))
pp <- peak2peak(x, 1)
pp <- peak2peak(x, 2)
pp <- peak2peak(x, 3)

## complex input
x <- c(1+1i, 2+3i, 3+5i, 4+7i, 5+9i)
pp <- peak2peak(x)
```

---

peak2rms *Peak-magnitude-to-RMS ratio*

Description

Compute the ratio of the largest absolute value to the root-mean-square (RMS) value of the object x.

Usage

```
peak2rms(x, MARGIN = 2)
```

Arguments

x	the data, expected to be a vector, a matrix, an array.
MARGIN	a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where x has named dimnames, it can be a character vector selecting dimension names. Default: 2 (usually columns)

Details

The input x can be a vector, a matrix or an array. If the input is a vector, a single value is returned representing the peak-magnitude-to-RMS ratio of the vector. If the input is a matrix or an array, a vector or an array of values is returned representing the peak-magnitude-to-RMS ratios of the dimensions of x indicated by the MARGIN argument.

Support for complex valued input is provided.

Value

Vector or array of values containing the peak-magnitude-to-RMS ratios of the specified MARGIN of x.

Author(s)

Andreas Weber, <octave@tech-chat.de>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## numeric vector
x <- c(1:5)
p <- peak2rms(x)

## numeric matrix
x <- matrix(c(1,2,3, 100, 150, 200, 1000, 1500, 2000), 3, 3)
p <- peak2rms(x)
p <- peak2rms(x, 1)

## numeric array
x <- array(c(1, 1.5, 2, 100, 150, 200, 1000, 1500, 2000,
           10000, 15000, 20000), c(2,3,2))
p <- peak2rms(x, 1)
p <- peak2rms(x, 2)
p <- peak2rms(x, 3)

## complex input
x <- c(1+1i, 2+3i, 3+5i, 4+7i, 5+9i)
p <- peak2rms(x)
```

---

pei_tseng_notch *Pei-Tseng notch filter*

Description

Compute the transfer function coefficients of an IIR narrow-band notch filter.

Usage

```
pei_tseng_notch(w, bw)
```

Arguments

w	vector of critical frequencies of the filter. Must be between 0 and 1 where 1 is the Nyquist frequency.
bw	vector of bandwidths. Bw should be of the same length as w.

Details

The filter construction is based on an all-pass which performs a reversal of phase at the filter frequencies. Thus, the mean of the phase-distorted and the original signal has the respective frequencies removed.

Value

List of class Arma with list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Alexander Klein, <alexander.klein@math.uni-giessen.de>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

Pei, Soo-Chang, and Tseng, Chien-Cheng "IIR Multiple Notch Filter Design Based on Allpass Filter"; 1996 IEEE Tencon, doi: doi:10.1109/TENCON.1996.608814

See Also

Arma, filter

Examples

```
## 50 Hz notch filter
fs <- 256
nyq <- fs / 2
notch <- pei_tseng_notch(50 / nyq, 2 / nyq)
freqz(notch, fs = fs)
```

---

poly *Polynomial with specified roots*

Description

Compute the coefficients of a polynomial when the roots are given, or the characteristic polynomial of a matrix.

Usage

```
poly(x)
```

Arguments

x	Real or complex vector, or square matrix.

Details

If a vector is passed as an argument, then `poly(x)` is a vector of the coefficients of the polynomial whose roots are the elements of `x`.

If an NxN square matrix is given, `poly(x)` is the row vector of the coefficients of `det(z * diag(N) - x)`, which is the characteristic polynomial of `x`.

Value

A vector of the coefficients of the polynomial in order from highest to lowest polynomial power.

Author(s)

Kurt Hornik.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

See Also

roots

Examples

```
p <- poly(c(1, -1))
p <- poly(pracma::roots(1:3))
p <- poly(matrix(1:9, 3, 3))
```

---

polyreduce *Reduce polynomial*

Description

Reduce a polynomial coefficient vector to a minimum number of terms by stripping off any leading zeros.

Usage

```
polyreduce(pc)
```

Arguments

pc	vector of polynomial coefficients

Value

Vector of reduced polynomial coefficients.

Author(s)

Tony Richardson, <arichard@stark.cc.oh.us>,
adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

Examples

```
p <- polyreduce(c(0, 0, 1, 2, 3))
```

---

polystab *Stabilize polynomial*

Description

Stabilize the polynomial transfer function by replacing all roots outside the unit circle with their reflection inside the unit circle.

Usage

```
polystab(a)
```

Arguments

a	vector of polynomial coefficients, normally in the z-domain

Value

Vector of stabilized polynomial coefficients.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

Examples

```
unstable <- c(-0.5, 1)
zplane(unstable, 1)
stable <- polystab(unstable)
zplane(stable, 1)
```

pow2db *Power - decibel conversion*

Description

Convert power to decibel and decibel to power.

Usage

```
pow2db(x)

db2pow(x)
```

Arguments

x	input data, specified as a numeric vector, matrix, or multidimensional array. Must be non-negative for numeric x.

Value

Converted data, same type and dimensions as x.

Author(s)

P. Sudeepam
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
db <- pow2db(c(0, 10, 100))
pow <- db2pow(c(-10, 0, 10))
```

primitive *Primitive*

Description

Calculate the indefinitive integral of a function.

Usage

```
primitive(FUN, t, C = 0)
```

Arguments

FUN	the function to calculate the primitive of.
t	points at which the function FUN is evaluated, specified as a vector of ascending values
C	constant of integration. Default: 0

Details

This function is a fancy way of calculating the cumulative sum.

Value

Vector of integrated function values.

Author(s)

Juan Pablo Carbajal.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

See Also

cumsum

Examples

```
f <- function(t) sin(2 * pi * 3 * t)
t <- c(0, sort(runif(100)))
F <- primitive (f, t, 0)
t_true <- seq(0, 1, length.out = 1e3)
F_true <- (1 - cos(2 * pi * 3 * t_true)) / (2 * pi * 3)
plot (t, F, xlab = "", ylab = "")
lines (t_true, F_true, col = "red")
legend("topright", legend = c("Numerical primitive", "True primitive"),
  lty = c(0, 1), pch = c(1, NA), col = 1:2)
```

---

pulstran *Pulse train*

Description

Generate a train of pulses based on samples of a continuous function.

Usage

```
pulstran(
  t,
  d,
  func,
  fs = 1,
  method = c("linear", "nearest", "cubic", "spline"),
  ...
)
```

Arguments

t	Time values at which func is evaluated, specified as a vector.
d	Offset removed from the values of the array t, specified as a real vector, matrix, or array. You can apply an optional gain factor to each delayed evaluation by specifying d as a two-column matrix, with offset defined in column 1 and associated gain in column 2. If you specify d as a vector, the values are interpreted as delays only.
func	Continuous function used to generate a pulse train based on its samples, specified as 'rectpuls', 'gauspuls', 'tripuls', or a function handle. If you use func as a function handle, you can pass the function parameters as follows: y <- pulstran(t, d, 'gauspuls',10e3, bw = 0.5). This creates a pulse train using a 10 kHz Gaussian pulse with 50% bandwidth. Alternatively, func can be a prototype function, specified as a vector. The interval of the function 0 to (length(p) - 1) / fs, and its samples are identically zero outside this interval. By default, linear interpolation is used for generating delays.
fs	Sample rate in Hz, specified as a real scalar.
method	Interpolation method, specified as one of the following options: **"linear" (default)** Linear interpolation. The interpolated value at a query point is based on linear interpolation of the values at neighboring grid points in each respective dimension. This is the default interpolation method. **"nearest"** Nearest neighbor interpolation. The interpolated value at a query point is the value at the nearest sample grid point. **"cubic"** Shape-preserving piecewise cubic interpolation. The interpolated value at a query point is based on a shape-preserving piecewise cubic interpolation of the values at neighboring grid points. **"spline"** Spline interpolation using not-a-knot end conditions. The interpolated value at a query point is based on a cubic interpolation of the values at neighboring grid points in each respective dimension. Interpolation is performed by the function 'interp1' function in the library 'pracma', and any interpolation method accepted by the function 'interp1' can be specified here.
...	Further arguments passed to func.

Details

Generate the signal `y <- sum(func(t + d, ...))` for each `d`. If `d` is a matrix of two columns, the first column is the delay `d` and the second column is the amplitude `a`, and `y <- sum(a * func(t + d))` for each `d, a`. Clearly, `func` must be a function which accepts a vector of times. Any extra arguments needed for the function must be tagged on the end.

If instead of a function name you supply a pulse shape sampled at frequency `fs` (default 1 Hz), an interpolated version of the pulse is added at each delay `d`. The interpolation stays within the the time range of the delayed pulse. The interpolation method defaults to linear, but it can be any interpolation method accepted by the function `interp1`

Value

Pulse train generated by the function, returned as a vector.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## periodic rectangular pulse
t <- seq(0, 60, 1/1e3)
d <- cbind(seq(0, 60, 2), sin(2 * pi * 0.05 * seq(0, 60, 2)))
y <- pulstran(t, d, 'rectpuls')
plot(t, y, type = "l", xlab = "Time (s)", ylab = "Waveform",
     main = "Periodic rectangular pulse")

## assymetric sawtooth waveform
fs <- 1e3
t <- seq(0, 1, 1/fs)
d <- seq(0, 1, 1/3)
x <- tripuls(t, 0.2, -1)
y <- pulstran(t, d, x, fs)
plot(t, y, type = "l", xlab = "Time (s)", ylab = "Waveform",
     main = "Asymmetric sawtooth waveform")

## Periodic Gaussian waveform
fs <- 1e7
tc <- 0.00025
t <- seq(-tc, tc, 1/fs)
x <- gauspuls(t, 10e3, 0.5)
plot(t, x, type="l", xlab = "Time (s)", ylab = "Waveform",
     main = "Gaussian pulse")
ts <- seq(0, 0.025, 1/50e3)
d <- cbind(seq(0, 0.025, 1/1e3), sin(2 * pi * 0.1 * (0:25)))
y <- pulstran(ts, d, x, fs)
plot(ts, y, type = "l", xlab = "Time (s)", ylab = "Waveform",
     main = "Gaussian pulse train")

# Custom pulse trains
```

```
fnx <- function(x, fn) sin(2 * pi * fn * x) * exp(-fn * abs(x))
ffs <- 1000
tp <- seq(0, 1, 1/ffs)
pp <- fnx(tp, 30)
plot(tp, pp, type = "l",xlab = 'Time (s)', ylab = 'Waveform',
     main = "Custom pulse")
fs <- 2e3
t <- seq(0, 1.2, 1/fs)
d <- seq(0, 1, 1/3)
dd <- cbind(d, 4^-d)
z <- pulstran(t, dd, pp, ffs)
plot(t, z, type = "l", xlab = "Time (s)", ylab = "Waveform",
     main = "Custom pulse train")
```

---

pwelch *Welch's power spectral density estimate*

Description

Compute power spectral density (PSD) using Welch's method.

Usage

```
pwelch(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = if (isScalar(window)) window else length(window),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none"),
  range = if (is.numeric(x)) "half" else "whole"
)

## S3 method for class 'pwelch'
plot(
  x,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  plot.type = c("spectrum", "cross-spectrum", "phase", "coherence", "transfer"),
  yscale = c("linear", "log", "dB"),
  ...
)

## S3 method for class 'pwelch'
print(
  x,
```

```
  plot.type = c("spectrum", "cross-spectrum", "phase", "coherence", "transfer"),
  yscale = c("linear", "log", "dB"),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  ...
)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
window	If window is a vector, each segment has the same length as window and is multiplied by window before (optional) zero-padding and calculation of its periodogram. If window is a scalar, each segment has a length of window and a Hamming window is used. Default: nextpow2(sqrt(length(x))) (the square root of the length of x rounded up to the next power of two). The window length must be larger than 3.
overlap	segment overlap, specified as a numeric value expressed as a multiple of window or segment length. 0 <= overlap < 1. Default: 0.5.
nfft	Length of FFT, specified as an integer scalar. The default is the length of the window vector or has the same value as the scalar window argument. If nfft is larger than the segment length, (seg_len), the data segment is padded nfft - seg_len zeros. The default is no padding. Nfft values smaller than the length of the data segment (or window) are ignored. Note that the use of padding to increase the frequency resolution of the spectral estimate is controversial.
fs	sampling frequency (Hertz), specified as a positive scalar. Default: 1.
detrend	character string specifying detrending option; one of: long-mean remove the mean from the data before splitting into segments (default) short-mean remove the mean value of each segment long-linear remove linear trend from the data before splitting into segments short-linear remove linear trend from each segment none no detrending
range	character string. one of: "half" **or** "onesided" frequency range of the spectrum is from zero up to but not including fs / 2. Power from negative frequencies is added to the positive side of the spectrum. "whole" **or** "twosided" frequency range of the spectrum is -fs / 2 to fs / 2, with negative frequencies stored in "wrap around order" after the positive frequencies; e.g. frequencies for a 10-point "twosided" spectrum are 0 0.1 0.2 0.3 0.4 0.5 -0.4 -0.3 -0.2. -0.1. "shift" **or** "centerdc" same as "whole" but with the first half of the spectrum swapped with second half to put the zero-frequency value in the middle.

	Default: If x are real, the default range is "half", otherwise the default range is "whole".
xlab, ylab, main	labels passed to plotting function. Default: NULL
plot.type	character string specifying which plot to produce; one of "spectrum", "cross-spectrum", "phase", "coherence", "transfer"
yscale	character string specifying scaling of Y-axis; one of "linear", "log", "dB"
...	additional arguments passed to functions

Details

The Welch method [1] reduces the variance of the periodogram estimate to the PSD by splitting the signal into (usually) overlapping segments and windowing each segment, for instance by a Hamming window. The periodogram is then computed for each segment, and the squared magnitude is computed, which is then averaged for all segments. See also [2].

The spectral density is the mean of the modified periodograms, scaled so that area under the spectrum is the same as the mean square of the data. This equivalence is supposed to be exact, but in practice there is a mismatch of up to 0.5 the data.

In case of multivariate signals, Cross-spectral density, phase, and coherence are also returned. The input data can be demeaned or detrended, overall or for each segment separately.

Value

An object of class "pwelch", which is a list containing the following elements:

freq vector of frequencies at which the spectral variables are estimated. If x is numeric, power from negative frequencies is added to the positive side of the spectrum, but not at zero or Nyquist (fs/2) frequencies. This keeps power equal in time and spectral domains. If x is complex, then the whole frequency range is returned.

spec Vector (for univariate series) or matrix (for multivariate series) of estimates of the spectral density at frequencies corresponding to freq.

cross NULL for univariate series. For multivariateseries, a matrix containing the cross-spectral density estimates between different series. Column $i + (j-1) * (j-2)/2$ of contains the cross-spectral estimates between columns i and j of x, where $i < j$.

phase NULL for univariate series. For multivariate series, a matrix containing the cross-spectrum phase between different series. The format is the same as cross.

coh NULL for univariate series. For multivariate series, a matrix containing the squared coherence between different series. The format is the same as cross.

trans NULL for univariate series. For multivariate series, a matrix containing estimates of the transfer function between different series. The format is the same as cross.

x_len The length of the input series.

seg_len The length of each segment making up the averages.

psd_len The number of frequencies. See freq

nseries The number of series

series The name of the series

`snames` For multivariate input, the names of the individual series

`window` The window used to compute the modified periodogram

`fs` The sampling frequency

`detrend` Character string specifying detrending option

Note

Unlike the 'Octave' function 'pwelch', the current implementation does not compute confidence intervals because they can be inaccurate in case of overlapping segments.

Author(s)

Peter V. Lanspeary <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] Welch, P.D. (1967). The use of Fast Fourier Transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. IEEE Transactions on Audio and Electroacoustics, AU-15 (2): 70–73.

[2] https://en.wikipedia.org/wiki/Welch%27s_method

Examples

```
fs <- 256
secs <- 10
freq <- 30
ampl <- 1
t <- seq(0, secs, length.out = fs * secs)

x <- ampl * cos(freq * 2 * pi * t) + runif(length(t))
Pxx <- pwelch(x, fs = fs)              # no plot
pwelch(x, fs = fs)                     # plot

# 90 degrees phase shift with with respect to x
y <- ampl * sin(freq * 2 * pi * t) + runif(length(t))
Pxy <- pwelch(cbind(x, y), fs = fs)
plot(Pxy, yscale = "dB")
plot(Pxy, plot.type = "phase")
# note the phase shift around 30 Hz is pi/2
plot(Pxy, plot.type = "coherence")

# Transfer function estimate example
fs <- 1000                 # Sampling frequency
t <- (0:fs) / fs           # One second worth of samples
A <- c(1, 2)               # Sinusoid amplitudes
f <- c(150, 140)           # Sinusoid frequencies
xn <- A[1] * sin(2 * pi * f[1] * t) +
      A[2] * sin(2 * pi * f[2] * t) +  0.1 * runif(length(t))
```

```
h <- Ma(rep(1L, 10) / 10)      # Moving average filter
yn <- filter(h, xn)
atfm <- freqz(h, fs = fs)
etfm <- pwelch(cbind(xn, yn), fs = fs)
op <- par(mfrow = c(2, 1))
xl <- "Frequency (Hz)"; yl <- "Magnitude"
plot(atfm$w, abs(atfm$h), type = "l", main = "Actual", xlab = xl, ylab = yl)
plot(etfm$freq, abs(etfm$trans), type = "l", main = "Estimated",
     xlab = xl, ylab = yl)
par(op)
```

`pyulear` *Autoregressive PSD estimate - Yule-Walker method*

Description

Calculate Yule-Walker autoregressive power spectral density.

Usage

```
pyulear(
  x,
  p,
  freq = 256,
  fs = 1,
  range = NULL,
 method = if (length(freq) == 1 && bitwAnd(freq, freq - 1) == 0) "fft" else "poly"
)
```

Arguments

`x`	input data, specified as a numeric or complex vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
`p`	model order; number of poles in the AR model or limit to the number of poles if a valid criterion is provided. Must be < length(x) - 2.
`freq`	vector of frequencies at which power spectral density is calculated, or a scalar indicating the number of uniformly distributed frequency values at which spectral density is calculated. Default: 256.
`fs`	sampling frequency (Hz). Default: 1
`range`	character string. one of:
	`"half"` **or** `"onesided"` frequency range of the spectrum is from zero up to but not including `fs / 2`. Power from negative frequencies is added to the positive side of the spectrum.

"whole" **or** "twosided" frequency range of the spectrum is -fs / 2 to fs / 2, with negative frequencies stored in "wrap around order" after the positive frequencies; e.g. frequencies for a 10-point 'twosided' spectrum are 0 0.1 0.2 0.3 0.4 0.5 -0.4 -0.3 -0.2. -0.1.

"shift" **or** "centerdc" same as "whole" but with the first half of the spectrum swapped with second half to put the zero-frequency value in the middle. If freq is vector, "shift" is ignored.

Default: If model coefficients a are real, the default range is "half", otherwise the default range is "whole".

method — method used to calculate the power spectral density, either "fft" (use the Fast Fourier Transform) or "poly" (calculate the power spectrum as a polynomial). This argument is ignored if the freq argument is a vector. The default is "poly" unless the freq argument is an integer power of 2.

Value

An object of class "ar_psd" , which is a list containing two elements, freq and psd containing the frequency values and the estimates of power-spectral density, respectively.

Note

This function is a wrapper for arburg and ar_psd.

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>

See Also

ar_psd, arburg

Examples

```
A <- Arma(1, c(1, -2.7607, 3.8106, -2.6535, 0.9238))
y <- filter(A, 0.2 * rnorm(1024))
py <- pyulear(y, 4)
```

---

qp_kaiser *Kaiser FIR filter design*

Description

Compute FIR filter for use with a quasi-perfect reconstruction polyphase-network filter bank.

Usage

```
qp_kaiser(nb, at, linear = FALSE)
```

Arguments

nb	number of frequency bands, specified as a scalar
at	attenuation (in dB) in the stop band.
linear	logical, indicating linear scaling. If FALSE (default), the Kaiser window is multiplied by the ideal impulse response $h(n) = asinc(an)$ and converted to its minimum-phase version by means of a Hilbert transform.

Value

The FIR filter coefficients, of class Ma.

Author(s)

André Carezia, <andre@carezia.eng.br>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

Ma, filter, fftfilt, fir2

Examples

```
freqz(qp_kaiser(1, 20))
freqz(qp_kaiser(1, 40))
```

---

rceps *Real cepstrum*

Description

Return the real cepstrum and minimum-phase reconstruction of a signal

Usage

```
rceps(x, minphase = FALSE)
```

Arguments

x	input data, specified as a real vector.
minphase	logical (default: FALSE) indication whether to compute minimum-phase reconstructed signal

Details

Cepstral analysis is a nonlinear signal processing technique that is applied most commonly in speech and image processing, or as a tool to investigate periodic structures within frequency spectra, for instance resulting from echos/reflections in the signal or to the occurrence of harmonic frequencies (partials, overtones).

The cepstrum is used in many variants. Most important are the power cepstrum, the complex cepstrum, and real cepstrum. The function `rceps` implements the real cepstrum by computing the inverse of the log-transformed FFT while discarding phase, i.e.,

$$rceps(x) < -ifft(log(Mag(fft(x))))$$

The real cepstrum is related to the power spectrum by the relation $pceps = 4 * rceps^2$.

The function `rceps()` can also return a minimum-phase reconstruction of the original signal. The concept of minimum phase originates from filtering theory, and denotes a filter transfer function with all of its poles and zeroes in the Z-transform domain lie inside the unit circle on the complex plane. Such a transfer function represents a stable filter.

A minimum-phase signal is a signal that has its energy concentrated near the front of the signal (near time 0). Such signals have many applications, e.g. in seismology and speech analysis.

Value

If `minphase` equals `FALSE`, the real cepstrum is returned as a vector. If `minphase` equals `TRUE`, a list is returned containing two vectors; `y` containing the real cepstrum, and `ym` containing the minimum-phase reconstructed signal

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Minimum_phase

See Also

cceps

Examples

```
## Simulate a speech signal with a 70 Hz glottal wave
f0 <- 70; fs = 10000          # 100 Hz fundamental, 10 kHz sampling rate
a <- Re(poly(0.985 * exp(1i * pi * c(0.1, -0.1, 0.3, -0.3))))
s <- 0.05 * runif(1024)
s[floor(seq(1, length(s), fs / f0))] <- 1
x <- filter(1, a, s)

## compute real cepstrum and min-phase of x
```

```
cep <- rceps(x, TRUE)
hx <- freqz(x, fs = fs)
hxm <- freqz (cep$ym, fs = fs)
len <- 1000 * trunc(min(length(x), length(cep$ym)) / 1000)
time <- 0:(len-1) * 1000 / fs

op <- par(mfcol = c(2, 2))
plot(time, x[1:len], type = "l", ylim = c(-10, 10),
  xlab = "Time (ms)", ylab = "Amplitude",
  main = "Original and reconstructed signals")
lines(time, cep$ym[1:len], col = "red")
legend("topright", legend = c("original", "reconstructed"),
  lty = 1, col = c(1, 2))

plot(time, cep$y[1:len], type = "l",
  xlab = "Quefrency (ms)", ylab = "Amplitude",
  main = "Real cepstrum")

plot (hx$w, log(abs(hx$h)), type = "l",
  xlab = "Frequency (Hz)", ylab = "Magnitude",
  main = "Magnitudes are identical")
lines(hxm$w, log(abs(hxm$h)), col = "red")
legend("topright", legend = c("original", "reconstructed"),
  lty = 1, col = c(1, 2))

phx <- unwrap(Arg(hx$h))
phym <- unwrap(Arg(hxm$h))
range <- c(round(min(phx, phym)), round(max(phx, phym)))
plot (hx$w, phx, type = "l", ylim = range,
  xlab = "Frequency (Hz)", ylab = "Phase",
  main = "Unwrapped phase")
lines(hxm$w, phym, col = "red")
legend("bottomright", legend = c("original", "reconstructed"),
  lty = 1, col = c(1, 2))
par(op)

## confirm the magnitude spectrum is identical in the signal
## and the reconstruction and that there are peaks in the
## cepstrum at 14 ms intervals corresponding to an F0 of 70 Hz.
```

`rectpuls` *Rectangular pulse*

Description

Return samples of the unit-amplitude rectangular pulse at the times indicated by `t`.

Usage

```
rectpuls(t, w = 1)
```

Arguments

t	Sample times of unit rectangular pulse, specified by a vector.
w	Rectangle width, specified by a positive number. Default: 1

Details

y <- rectpuls(t) returns a continuous, aperiodic, unit-height rectangular pulse at the sample times indicated in array t, centered about t = 0.

y <- rectpuls(t, w) generates a rectangular pulse over the interval from -w/2 to w/2, sampled at times t. This is useful with the function pulstran for generating a series of pulses.

Value

Rectangular pulse of unit amplitude, returned as a vector.

Author(s)

Paul Kienzle, Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

pulstran

Examples

```
fs <- 10e3
t <- seq(-0.1, 0.1, 1/fs)
w <- 20e-3
y <- rectpuls(t, w)
plot(t, y, type="l", xlab = "Time", ylab = "Amplitude")

fs <- 11025  # arbitrary sample rate
f0 <- 100    # pulse train sample rate
w <- 0.3/f0  # pulse width 1/10th the distance between pulses
y <- pulstran (seq(0, 4/f0, 1/fs), seq(0, 4/f0, 1/f0), 'rectpuls', w = w)
plot (seq(0, length(y)-1) * 1000/fs, y, type ="l", xlab = "Time (ms)",
      ylab = "Amplitude",
      main = "Rectangular pulse train of 3 ms pulses at 10 ms intervals")
```

rectwin *Rectangular window*

Description

Return the filter coefficients of a rectangular window of length n.

Usage

```
rectwin(n)
```

Arguments

n	Window length, specified as a positive integer.

Details

The output of the rectwin function with input n can also be created using the rep function: w <- rep(1L, n)

Value

rectangular window, returned as a vector.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

boxcar

Examples

```
r <- rectwin(64)
plot (r, type = "l", xlab = "Samples", ylab =" Amplitude", ylim = c(0, 1))
```

remez *Parks-McClellan optimal FIR filter design*

Description

Parks-McClellan optimal FIR filter design using the Remez exchange algorithm.

Usage

```
remez(
  n,
  f,
  a,
  w = rep(1, length(f)/2),
  ftype = c("bandpass", "differentiator", "hilbert"),
  density = 16
)
```

Arguments

n	filter order (1 less than the length of the filter).
f	normalized frequency points, strictly increasing vector in the range [0, 1], where 1 is the Nyquist frequency. The number of elements in the vector is always a multiple of 2.
a	vector of desired amplitudes at the points specified in f. f and a must be the same length. The length must be an even number.
w	vector of weights used to adjust the fit in each frequency band. The length of w is half the length of f and a, so there is exactly one weight per band. Default: 1.
ftype	filter type, matched to one of "bandpass" (default), "differentiator", or "hilbert".
density	determines how accurately the filter will be constructed. The minimum value is 16 (default), but higher numbers are slower to compute.

Value

The FIR filter coefficients, a vector of length n + 1, of class Ma

Author(s)

Jake Janovetz, <janovetz@uiuc.edu>,
Paul Kienzle, <pkienzle@users.sf.net>,
Kai Habel, <kahacjde@linux.zrz.tu-berlin.de>.
Conversion to R Tom Short
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

https://en.wikipedia.org/wiki/Fir_filter

Rabiner, L.R., McClellan, J.H., and Parks, T.W. (1975). FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximations, IEEE Proceedings, vol. 63, pp. 595 - 610.
https://en.wikipedia.org/wiki/Parks-McClellan_filter_design_algorithm

See Also

Ma, filter, fftfilt, fir1

Examples

```
## low pass filter
f1 <- remez(15, c(0, 0.3, 0.4, 1), c(1, 1, 0, 0))
freqz(f1)

## band pass
f <- c(0, 0.3, 0.4, 0.6, 0.7, 1)
a <- c(0, 0, 1, 1, 0, 0)
b <- remez(17, f, a)
hw <- freqz(b, 512)
plot(f, a, type = "l", xlab = "Radian Frequency (w / pi)",
     ylab = "Magnitude")
lines(hw$w/pi, abs(hw$h), col = "red")
legend("topright", legend = c("Ideal", "Remez"), lty = 1,
       col = c("black", "red"))
```

---

resample *Change sampling rate*

Description

Resample using a polyphase algorithm.

Usage

```
resample(x, p, q, h)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
p, q	resampling factors, specified as positive integers. p / q is the resampling factor.
h	Impulse response of the FIR filter specified as a numeric vector or matrix. If it is a vector, then it represents one FIR filter to may be applied to multiple signals in x; if it is a matrix, then each column is a separate FIR impulse response. If not specified, a FIR filter based on a Kaiser window is designed.

Details

If h is not specified, this function will design an optimal FIR filter using a Kaiser-Bessel window. The filter length and the parameter β are computed based on ref [2], Chapter 7, Eq. 7.63 (p. 476), and Eq. 7.62 (p. 474), respectively.

Value

output signal, returned as a vector or matrix. Each column has length `ceiling(((length(x) - 1) * p + length(h)) / q)`..

Author(s)

Eric Chassande-Mottin, <ecm@apc.univ-paris7.fr>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

[1] Proakis, J.G., and Manolakis, D.G. (2007). Digital Signal Processing: Principles, Algorithms, and Applications, 4th ed., Prentice Hall, Chap. 6.
[2] Oppenheim, A.V., Schafer, R.W., and Buck, J.R. (1999). Discrete-time signal processing, Signal processing series, Prentice-Hall.

See Also

kaiser

Examples

```
lx <- 60
tx <- seq(0, 360, length.out = lx)
x <- sin(2 * pi * tx / 120)

# upsample
p <- 3; q <- 2
ty <- seq(0, 360, length.out = lx * p / q)
y <- resample(x, p, q)

# downsample
p <- 2; q <- 3
tz <- seq(0, 360, length.out = lx * p / q)
z <- resample(x, p, q)

# plot
plot(tx, x, type = "b", col = 1, pch = 1,
 xlab = "", ylab = "")
points(ty, y, col = 2, pch = 2)
points(tz, z, col = 3, pch = 3)
legend("bottomleft", legend = c("original", "upsampled", "downsampled"),
  lty = 1, pch = 1:3, col = 1:3)
```

residue *Partial fraction expansion*

Description

Finds the residues, poles, and direct term of a Partial Fraction Expansion of the ratio of two polynomials.

Usage

```
residue(b, a, tol = 0.001)

rresidue(r, p, k, tol = 0.001)
```

Arguments

b	coefficients of numerator polynomial
a	coefficients of denominator polynomial
tol	tolerance. Default: 0.001
r	residues of partial fraction expansion
p	poles of partial fraction expansion
k	direct term

Details

The call `res <- residue(b, a)` computes the partial fraction expansion for the quotient of the polynomials, b and a.

The call `res <- rresidue(r, p, k)` performs the inverse operation and computes the reconstituted quotient of polynomials, b(s) / a(s), from the partial fraction expansion; represented by the residues, poles, and a direct polynomial specified by r, p and k, and the pole multiplicity e.

Value

For residue, a list containing r, p and k. For rresidue, a list containing b and a.

Author(s)

Tony Richardson, <arichard@stark.cc.oh.us>,
Ben Abbott, <bpabbott@mac.com>,
adapted by John W. Eaton.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

Examples

```
b <- c(-4, 8)
a <- c(1, 6, 8)
rpk <- residue(b, a)
ba <- rresidue(rpk$r, rpk$p, rpk$k)
```

`residued` *delayed z-transform partial fraction expansion*

Description

Finds the residues, poles, and direct term of a Partial Fraction Expansion of the ratio of two polynomials.

Usage

```
residued(b, a)
```

Arguments

`b`	coefficients of numerator polynomial
`a`	coefficients of denominator polynomial

Details

In the usual PFE function `residuez`, the IIR part (poles `p` and residues `r`) is driven in parallel with the FIR part (`f`). In this variant, the IIR part is driven by the output of the FIR part. This structure can be more accurate in signal modeling applications.

Value

A `list` containing

r vector of filter pole residues of the partial fraction

p vector of partial fraction poles

k vector containing FIR part, if any (empty if `length(b) < length(a)`)

Author(s)

Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

References

https://ccrma.stanford.edu/~jos/filters/residued.html

See Also

residue, residuez

Examples

```
b <- c(2, 6, 6, 2)
a <- c(1, -2, 1)
resd <- residued(b, a)
resz <- residuez(b, a)
```

---

residuez *Z-transform partial fraction expansion*

Description

Finds the residues, poles, and direct term of a Partial Fraction Expansion of the ratio of two polynomials.

Usage

```
residuez(b, a)
```

Arguments

b	coefficients of numerator polynomial
a	coefficients of denominator polynomial

Details

residuez converts a discrete time system, expressed as the ratio of two polynomials, to partial fraction expansion, or residue, form.

Value

A list containing

r vector of filter pole residues of the partial fraction

p vector of partial fraction poles

k vector containing FIR part, if any (empty if length(b) < length(a))

Author(s)

Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

See Also

residue, residued

Examples

```
b0 <- 0.05634
b1 <- c(1,  1)
b2 <- c(1, -1.0166, 1)
a1 <- c(1, -0.683)
a2 <- c(1, -1.4461, 0.7957)
b <- b0 * conv(b1, b2)
a <- conv(a1, a2)
res <- residuez(b, a)
```

---

rms *Root-mean-square*

Description

Compute the root-mean-square (RMS) of the object x.

Usage

```
rms(x, MARGIN = 2)
```

Arguments

x	the data, expected to be a vector, a matrix, an array.
MARGIN	a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where x has named dimnames, it can be a character vector selecting dimension names. Default: 2 (columns)

Details

The input x can be a vector, a matrix or an array. If the input is a vector, a single value is returned representing the root-mean-square of the vector. If the input is a matrix or an array, a vector or an array of values is returned representing the root-mean-square of the dimensions of x indicated by the MARGIN argument.

Support for complex valued input is provided. The sum of squares of complex numbers is defined by sum(x * Conj(x))

Value

Vector or array of values containing the root-mean-squares of the specified MARGIN of x.

Author(s)

Andreas Weber, <octave@tech-chat.de>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## numeric vector
x <- c(1:5)
r <- rms(x)

## numeric matrix
x <- matrix(c(1,2,3, 100, 150, 200, 1000, 1500, 2000), 3, 3)
p <- rms(x)
p <- rms(x, 1)

## numeric array
x <- array(c(1, 1.5, 2, 100, 150, 200, 1000, 1500,
           2000, 10000, 15000, 20000), c(2,3,2))
p <- rms(x, 1)
p <- rms(x, 2)
p <- rms(x, 3)

## complex input
x <- c(1+1i, 2+3i, 3+5i, 4+7i, 5+9i)
p <- rms(x)
```

---

rssq *Root-sum-of-squares*

Description

Compute the root-sum-of-squares (SSQ) of the object x.

Usage

```
rssq(x, MARGIN = 2)
```

Arguments

x	the data, expected to be a vector, a matrix, an array.
MARGIN	a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where x has named dimnames, it can be a character vector selecting dimension names. Default: 2 (usually columns)

Details

The input x can be a vector, a matrix or an array. If the input is a vector, a single value is returned representing the root-sum-of-squares of the vector. If the input is a matrix or an array, a vector or an array of values is returned representing the root-sum-of-squares of the dimensions of x indicated by the MARGIN argument.

Support for complex valued input is provided. The sum of squares of complex numbers is defined by sum(x * Conj(x))

Value

Vector or array of values containing the root-sum-of-squares of the specified MARGIN of x.

Author(s)

Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## numeric vector
x <- c(1:5)
p <- rssq(x)

## numeric matrix
x <- matrix(c(1,2,3, 100, 150, 200, 1000, 1500, 2000), 3, 3)
p <- rssq(x)
p <- rssq(x, 1)

## numeric array
x <- array(c(1, 1.5, 2, 100, 150, 200, 1000, 1500,
            2000, 10000, 15000, 20000), c(2,3,2))
p <- rssq(x, 1)
p <- rssq(x, 2)
p <- rssq(x, 3)

## complex input
x <- c(1+1i, 2+3i, 3+5i, 4+7i, 5+9i)
p <- rssq(x)
```

---

sampled2continuous *Signal reconstruction*

Description

Analog signal reconstruction from discrete samples.

Usage

```
sampled2continuous(xn, fs, t)
```

Arguments

xn	the sampled input signal, specified as a vector
fs	sampling frequency in Hz used in collecting x, specified as a positive scalar value. Default: 1
t	time points at which data is to be reconstructed, specified as a vector relative to x[0] (not real time).

Details

Given a discrete signal x[n] sampled with a frequency of fs Hz, this function reconstruct the original analog signal x(t) at time points t. The function can be used, for instance, to calculate sampling rate effects on aliasing.

Value

Reconstructed signal x(t), returned as a vector.

Author(s)

Muthiah Annamalai, <muthiah.annamalai@uta.edu>. Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
# 'analog' signal: 3 Hz cosine
t <- seq(0, 1, length.out = 100)
xt <- cos(3 * 2 * pi * t)
plot(t, xt, type = "l", xlab = "", ylab = "", ylim = c(-1, 1.2))

# 'sample' it at 4 Hz to simulate aliasing
fs <- 4
n <- ceiling(length(t) / fs)
xn <- xt[seq(ceiling(n / 2), length(t), n)]
s4 <- sampled2continuous(xn, fs, t)
lines(t, s4, col = "red")

# 'sample' it > 6 Hz to avoid aliasing
fs <- 7
n <- ceiling(length(t) / fs)
xn <- xt[seq(ceiling(n / 2), length(t), n)]
s7 <- sampled2continuous(xn, fs, t)
lines(t, s7, col = "green")
legend("topright", legend = c("original", "aliased", "non-aliased"),
  lty = 1, col = c("black", "red", "green"))
```

sawtooth *Sawtooth or triangle wave*

Description

Returns samples of the sawtooth function at the times indicated by `t`.

Usage

```
sawtooth(t, width = 1)
```

Arguments

`t`	Sample times of unit sawtooth wave specified by a vector.
`width`	Real number between 0 and 1 which specifies the point between 0 and 2π where the maximum is. The function increases linearly from -1 to 1 in the interval from 0 to $2 * \pi * width$, and decreases linearly from 1 to -1 in the interval from $2 * \pi * width$ to $2 * \pi$. Default: 1 (standard sawtooth).

Details

The code `y <- sawtooth(t)` generates a sawtooth wave with period 2π for the elements of the time array `t`. `sawtooth()` is similar to the sine function but creates a sawtooth wave with peaks of –1 and 1. The sawtooth wave is defined to be –1 at multiples of 2π and to increase linearly with time with a slope of $1/\pi$ at all other times.

`y <- sawtooth(t, width)` generates a modified triangle wave with the maximum location at each period controlled by `width`. Set `width` to 0.5 to generate a standard triangle wave.

Value

Sawtooth wave, returned as a vector.

Author(s)

Juan Aguado.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
T <- 10 * (1 / 50)
fs <- 1000
t <- seq(0, T-1/fs, 1/fs)
y <- sawtooth(2 * pi * 50 *t)
plot(t, y, type="l", xlab = "", ylab = "", main = "50 Hz sawtooth wave")

T <- 10 * (1 / 50)
fs <- 1000
t <- seq(0, T-1/fs, 1/fs)
```

```
y <- sawtooth(2 * pi * 50 * t, 1/2)
plot(t, y, type="l", xlab = "", ylab = "", main = "50 Hz triangle wave")
```

---

schtrig *Schmitt Trigger*

Description

Multisignal Schmitt trigger with levels.

Usage

```
schtrig(x, lvl, st = NULL)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
lvl	threshold levels against which x is compared, specified as a vector. If this is a scalar, the thresholds are symmetric around 0, i.e. c(-lvl, lvl).
st	trigger state, specified as a vector of length ncol(x. The trigger state is returned in the output list and may be passed again to a subsequent call to schtrig. Default: NULL.

Details

The trigger works compares each column in x to the levels in lvl, when the value is higher than max(lvl), the output v is high (i.e. 1); when the value is below min(lvl) the output is low (i.e. 0); and when the value is between the two levels the output retains its value.

Value

a list containing the following variables:

v vector or matrix of 0's and 1's, according to whether x is above or below lvl, or the value of x if indeterminate

rng ranges in which the output is high, so the indexes rng[1,i]:rng[2,i] point to the i-th segment of 1s in v. See clustersegment for a detailed explanation.

st trigger state, returned as a vector with a length of the number of columns in x.

Author(s)

Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

clustersegment

Examples

```
t <- seq(0, 1, length.out = 100)
x <- sin(2 * pi * 2 * t) + sin(2 * pi * 5 * t) %*% matrix(c(0.8, 0.3), 1, 2)
lvl <- c(0.8, 0.25)
trig  <- schtrig (x, lvl)

op <- par(mfrow = c(2, 1))
plot(t, x[, 1], type = "l", xlab = "", ylab = "")
abline(h = lvl, col = "blue")
lines(t, trig$v[, 1], col = "red", lwd = 2)
plot(t, x[, 2], type = "l", xlab = "", ylab = "")
abline(h = lvl, col = "blue")
lines(t, trig$v[, 2], col = "red", lwd = 2)
par(op)
```

---

sftrans *Transform filter band edges*

Description

Transform band edges of a generic lowpass filter to a filter with different band edges and to other filter types (high pass, band pass, or band stop).

Usage

```
sftrans(Sz, ...)

## S3 method for class 'Zpg'
sftrans(Sz, w, stop = FALSE, ...)

## S3 method for class 'Arma'
sftrans(Sz, w, stop = FALSE, ...)

## Default S3 method:
sftrans(Sz, Sp, Sg, w, stop = FALSE, ...)
```

Arguments

Sz	In the generic case, a model to be transformed. In the default case, a vector containing the zeros in a pole-zero-gain model.
...	arguments passed to the generic function.

w	critical frequencies of the target filter specified in radians. w must be a scalar for low-pass and high-pass filters, and w must be a two-element vector c(low, high) specifying the lower and upper bands in radians.
stop	FALSE for a low-pass or band-pass filter, TRUE for a high-pass or band-stop filter.
Sp	a vector containing the poles in a pole-zero-gain model.
Sg	a vector containing the gain in a pole-zero-gain model.

Details

Given a low pass filter represented by poles and zeros in the splane, you can convert it to a low pass, high pass, band pass or band stop by transforming each of the poles and zeros individually. The following summarizes the transformations:

Transform	**Zero at x**	**Pole at x**
Low-Pass $S \rightarrow CS/Fc$	zero: Fcx/C gain: C/Fc	pole: Fcx/C gain: Fc/C
High Pass $S \rightarrow CFc/S$	zero: FcC/x pole: 0 gain: $-x$	pole: FcC/x zero: 0 gain: $-1/x$
Band Pass S -> $C\frac{S^2+FhFl}{S(Fh-Fl)}$	zero: $b + -\sqrt{(b^2 - FhFl)}$ pole: 0 gain: $C/(Fh - Fl)$ $b = x/C(Fh - Fl)/2$	pole: $b \pm \sqrt{(b^2 - FhFl)}$ zero: 0 gain: $(Fh - Fl)/C$ $b = x/C(Fh - Fl)/2$
Band Stop S -> $C\frac{S(Fh-Fl)}{S^2+FhFl}$	zero: $b \pm \sqrt{(b^2 - FhFl)}$ pole: $\pm\sqrt{(-FhFl)}$ gain: $-x$ $b = C/x(Fh - Fl)/2$	pole: $b + -\sqrt{(b^2 - FhFl)}$ zero: $\pm\sqrt{(-FhFl)}$ gain: $-1/x$ $b = C/x(Fh - Fl)/2$
Bilinear $S \rightarrow \frac{2z-1}{Tz+1}$	zero: $(2 + xT)/(2 - xT)$ pole: -1 gain: $(2 - xT)/T$	pole: $(2 + xT)/(2 - xT)$ zero: -1 gain: $(2 - xT)/T$

where C is the cutoff frequency of the initial lowpass filter, F_c is the edge of the target low/high pass filter and [F_l,F_h] are the edges of the target band pass/stop filter. With abundant tedious algebra, you can derive the above formulae yourself by substituting the transform for S into $H(S) = S - x$ for a zero at x or $H(S) = 1/(S - x)$ for a pole at x, and converting the result into the form:

$$gprod(S - Xi)/prod(S - Xj)$$

Please note that a pole and a zero at the same place exactly cancel. This is significant for High Pass, Band Pass and Band Stop filters which create numerous extra poles and zeros, most of which cancel. Those which do not cancel have a fill-in effect, extending the shorter of the sets to have the same number of as the longer of the sets of poles and zeros (or at least split the difference in the case of the band pass filter). There may be other opportunistic cancellations, but it does not check for them.

Also note that any pole on the unit circle or beyond will result in an unstable filter. Because of cancellation, this will only happen if the number of poles is smaller than the number of zeros and the filter is high pass or band pass. The analytic design methods all yield more poles than zeros, so this will not be a problem.

Value

For the default case or for sftrans.Zpg, an object of class "Zpg", containing the list elements:

z complex vector of the zeros of the transformed model

p complex vector of the poles of the transformed model

g gain of the transformed model

For sftrans.Arma, an object of class "Arma", containing the list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

Proakis & Manolakis (1992). *Digital Signal Processing*. New York: Macmillan Publishing Company.

Examples

```
## 6th order Bessel bandpass
zpg <- besselap(6)
bp <- sftrans(zpg, c(2, 3), stop = TRUE)
freqs(bp, seq(0, 4, length.out = 128))
bp <- sftrans(zpg, c(0.1,0.3), stop = FALSE)
freqs(bp, seq(0, 4, length.out = 128))
```

sgolay *Savitzky-Golay filter design*

Description

Compute the filter coefficients for all Savitzky-Golay FIR smoothing filters.

Usage

```
sgolay(p, n, m = 0, ts = 1)
```

Arguments

p	Polynomial filter order; must be smaller than n.
n	Filter length; must a an odd positive integer.
m	Return the m-th derivative of the filter coefficients. Default: 0
ts	Scaling factor. Default: 1

Details

The early rows of the resulting filter smooth based on future values and later rows smooth based on past values, with the middle row using half future and half past. In particular, you can use row `i` to estimate `x(k)` based on the `i-1` preceding values and the `n-i` following values of `x` values as `y(k) = F[i, ] * x[(k - i + 1):(k + n -i)]`.

Normally, you would apply the first `(n-1)/2` rows to the first `k` points of the vector, the last `k` rows to the last `k` points of the vector and middle row to the remainder, but for example if you were running on a real-time system where you wanted to smooth based on the all the data collected up to the current time, with a lag of five samples, you could apply just the filter on row `n - 5` to your window of length `n` each time you added a new sample.

Value

An square matrix with dimensions `length(n)` that is of class `"sgolayFilter"`, so it can be used with `filter`.

Author(s)

Paul Kienzle <pkienzle@users.sf.net>,
Pascal Dupuis, <Pascal.Dupuis@esat.kuleuven.ac.be>.
Conversion to R Tom Short,
adapted by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

See Also

sgolayfilt

Examples

```
## Generate a signal that consists of a 0.2 Hz sinusoid embedded
## in white Gaussian noise and sampled five times a second for 200 seconds.
dt <- 1 / 5
t <- seq(0, 200 - dt, dt)
x <- 5 * sin(2 * pi * 0.2 * t) + rnorm(length(t))
## Use sgolay to smooth the signal.
## Use 21-sample frames and fourth order polynomials.
p <- 4
n <- 21
sg <- sgolay(p, n)
## Compute the steady-state portion of the signal by convolving it
## with the center row of b.
ycenter <- conv(x, sg[(n + 1)/2, ], 'valid')
## Compute the transients. Use the last rows of b for the startup
## and the first rows of b for the terminal.
ybegin <- sg[seq(nrow(sg), (n + 3) / 2, -1), ] %*% x[seq(n, 1, -1)]
yend <- sg[seq((n - 1)/2, 1, -1), ] %*%
        x[seq(length(x), (length(x) - (n - 1)), -1)]
## Concatenate the transients and the steady-state portion to
## generate the complete smoothed signal.
## Plot the original signal and the Savitzky-Golay estimate.
y = c(ybegin, ycenter, yend)
plot(t, x, type = "l", xlab = "", ylab = "", ylim = c(-8, 10))
lines(t, y, col = 2)
legend("topright", c('Noisy Sinusoid','S-G smoothed sinusoid'),
  lty = 1, col = c(1,2))
```

---

shanwavf *Complex Shannon Wavelet*

Description

Compute the Complex Shannon wavelet.

Usage

```
shanwavf(lb = -8, ub = 8, n = 1000, fb = 5, fc = 1)
```

Arguments

lb, ub	Lower and upper bounds of the interval to evaluate the waveform on. Default: -8 to 8.
n	Number of points on the grid between lb and ub (length of the wavelet). Default: 1000.
fb	Time-decay parameter of the wavelet (bandwidth in the frequency domain). Must be a positive scalar. Default: 5.
fc	Center frequency of the wavelet. Must be a positive scalar. Default: 1.

Details

The complex Shannon wavelet is defined by a bandwidth parameter fb, a wavelet center frequency fc, and the expression

$$\psi(x) = (fb^{0.5} * (sinc(fb * x) * e^{2*1i*pi*fc*x}))$$

on an n-point regular grid in the interval of lb to ub.

Value

A list containing 2 variables; x, the grid on which the complex Shannon wavelet was evaluated, and psi (Ψ), the evaluated wavelet on the grid x.

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

Examples

```
fb <- 1
fc <- 1.5
lb <- -20
ub <- 20
n <- 1000
sw <- shanwavf(lb, ub, n, fb, fc)
op <- par(mfrow = c(2,1))
plot(sw$x, Re(sw$psi), type="l", main = "Complex Shannon Wavelet",
     xlab = "real part", ylab = "")
plot(sw$x, Im(sw$psi), type="l", xlab = "imaginary part", ylab = "")
par(op)
```

---

shiftdata *Shift data to operate on specified dimension*

Description

Shift data in to permute the dimension dimx to the first column.

Usage

```
shiftdata(x, dimx)
```

Arguments

x	The data to be shifted. Can be of any type.
dimx	Dimension of x to be shifted to the first column. Named "dimx" instead of "dim" to avoid confusion with R's dim() function. Default: NULL (shift the first nonsingleton dimension)

Details

shiftdata(x, dimx) shifts data x to permute dimension dimx to the first column using the same permutation as the built-in filter function. The vector perm in the output list returns the permutation vector that is used.

If dimx is missing or empty, then the first nonsingleton dimension is shifted to the first column, and the number of shifts is returned in nshifts.

shiftdata is meant to be used in tandem with unshiftdata, which shifts the data back to its original shape. These functions are useful for creating functions that work along a certain dimension, like filter, sgolayfilt, and sosfilt.

Value

A list containing 3 variables; x, the shifted data, perm, the permutation vector, and nshifts, the number of shifts

Author(s)

Georgios Ouzounis, <ouzounis_georgios@hotmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

unshiftdata

Examples

```
## create a 3x3 magic square
x <- pracma::magic(3)
## Shift the matrix x to work along the second dimension.
## The permutation vector, perm, and the number of shifts, nshifts,
## are returned along with the shifted matrix.
sd <- shiftdata(x, 2)

## Shift the matrix back to its original shape.
y <- unshiftdata(sd)

## Rearrange Array to Operate on First nonsingleton Dimension
x <- 1:5
sd <- shiftdata(x)
y <- unshiftdata(sd)
```

sigmoid_train *Sigmoid Train*

Description

Evaluate a train of sigmoid functions at t.

Usage

```
sigmoid_train(t, ranges, rc)
```

Arguments

t	Vector (or coerced to a vector) of time values at which the sigmoids are calculated.
ranges	Matrix or array with 2 columns containing the time values within t at which each sigmoid is evaluated. The number of sigmoids is determined by the number of rows in ranges.
rc	Time constant. Either a scalar or a matrix or array with 2 columns containing the rising and falling time constants of each sigmoid. If a matrix or array is passed in rc, its size must equal the size of ranges. If a single scalar is passed in rc, then all sigmoids have the same time constant and are symmetrical.

Details

The number and duration of each sigmoid is determined from ranges. Each row of ranges represents a real interval, e.g. if sigmoid i starts at t = 0.1 and ends at t = 0.5, then ranges[i,] = c(0.1,0.5). The input rc is an array that defines the rising and falling time constants of each sigmoid. Its size must equal the size of ranges.

The individual sigmoids are returned in s. The combined sigmoid train is returned in the vector y of length equal to t, and such that y = max(s).

Value

A list consisting two variables; y the combined sigmoid train (length identical to t), and s, the individual sigmoids (number of rows equal to number of rows in ranges and rc.

Author(s)

Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
t <- seq(0, 2, length.out = 500)
ranges <- rbind(c(0.1, 0.4), c(0.6, 0.8), c(1, 2))
rc <- rbind(c(1e-2, 1e-3), c(1e-3, 2e-2), c(2e-2, 1e-2))
st <- sigmoid_train (t, ranges, rc)
plot(t, st$y[1,], type="n", xlab = "Time(s)", ylab = "S(t)",
     main = "Vectorized use of sigmoid train")
for (i in 1:3) rect(ranges[i, 1], 0, ranges[i, 2], 1,
                    border = NA, col="pink")
for (i in 1:3) lines(t, st$y[i,])
# The colored regions show the limits defined in range.

t <- seq(0, 2, length.out = 500)
ranges <- rbind(c(0.1, 0.4), c(0.6, 0.8), c(1, 2))
rc <- rbind(c(1e-2, 1e-3), c(1e-3, 2e-2), c(2e-2, 1e-2))
amp <- c(4, 2, 3)
st <- sigmoid_train (t, ranges, rc)
y <- amp %*% st$y
plot(t, y[1,], type="l", xlab = 'time', ylab = 'signal',
     main = 'Varying amplitude sigmoid train', col="blue")
lines(t, st$s, col = "orange")
legend("topright", legend = c("Sigmoid train", "Components"),
       lty = 1, col = c("blue", "orange"))
```

---

signals *signals*

Description

Sample EEG and ECG data.

Usage

```
signals
```

Format

A `data.frame` containing 10 seconds of data electrophysiological data, sampled at 256 Hz with a 24 bit A/D converter, measured in microVolts. The data frame consists of 2 columns (channels):

eeg electroencephalogram (EEG) data measured from electrode Pz according to the 10-20 system, referred to algebraically linked mastoids (the brain's alpha rhythm is clearly visible).

ecg electrocardiogram (ECG) data, recorded bipolarly with a V6 versus V1 chest lead (this lead maximizes the R wave of the ECG with respect to the P, Q, S, T and U waves of the cardiac cycle).

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>

Examples

```
data(signals)
time <- seq(0, 10, length.out = nrow(signals))
op <- par(mfcol = c(2, 1))
plot(time, signals[, 1], type = "l", xlab = "Time", ylab = "EEG (uV)")
plot(time, signals[, 2], type = "l", xlab = "Time", ylab = "ECG (uV)")
par(op)
```

---

sinetone *Sine tone*

Description

Generate discrete sine tone.

Usage

```
sinetone(freq, rate = 8000, sec = 1, ampl = 64)
```

Arguments

`freq`	frequency of the tone, specified as a vector of positive numeric values. The length of `freq` should equal the length of the `ampl` vector; the shorter of the two is recycled to the longer vector.
`rate`	sampling frequency, specified as a positive scalar. Default: 8000.
`sec`	length of the generated tone in seconds. Default: 1
`ampl`	amplitude of the tone, specified as a vector of positive numeric values. The length of `ampl` should equal the length of the `freq` vector; the shorter of the two is recycled to the longer vector. Default: 64.

Value

Sine tone, returned as a vector of length `rate * sec`, or as a matrix with `rate * sec` columns and `max(length(freq), length(ampl))` columns.

Author(s)

Friedrich Leisch.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
fs <- 1000
sec <- 2
y <- sinetone(10, fs, sec, 1)
plot(seq(0, sec, length.out = sec * fs), y, type = "l", xlab = "", ylab = "")

y <- sinetone(c(10, 15), fs, sec, c(1, 2))
matplot(seq(0, sec, length.out = sec * fs), y, type = "l",
        xlab = "", ylab = "")
```

---

sinewave *Sine wave*

Description

Generate a discrete sine wave.

Usage

```
sinewave(m, n = m, d = 0)
```

Arguments

m	desired length of the generated series, specified as a positive integer.
n	rate, of the generated series, specified as a positive integer. Default: m.
d	delay, specified as a positive integer. Default: 0.

Value

Sine wave, returned as a vector of length m.

Author(s)

Friedrich Leisch.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
plot(sinewave(100, 10), type = "l")
```

`Sos` *Second-order sections*

Description

Create or convert filter models to second-order sections form.

Usage

```
Sos(sos, g = 1)

as.Sos(x, ...)

## S3 method for class 'Arma'
as.Sos(x, ...)

## S3 method for class 'Ma'
as.Sos(x, ...)

## S3 method for class 'Sos'
as.Sos(x, ...)

## S3 method for class 'Zpg'
as.Sos(x, ...)
```

Arguments

`sos`	second-order sections representation of the model
`g`	overall gain factor
`x`	model to be converted.
`...`	additional arguments (ignored).

Details

`as.Sos` converts from other forms, including `Arma`, `Ma`, and `Zpg`.

Value

A list of class `Sos` with the following list elements:

sos second-order section representation of the model, returned as an `L x 6` matrix, one row for each section `1:L`. Each row consists of an `[B, A]`, pair, where `B = c(b0, b1, b2)`, and `A = c(1, a1, a2)`, the filer coefficients for each section. Each `b0` entry must be nonzero for each section.

g overall gain factor that scales any one of the B_i vectors. Default: 1

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

Arma, Ma, Zpg

Examples

```
ba <- butter(3, 0.2)
sos <- as.Sos(ba)
```

`sos2tf` *Sos to transfer function*

Description

Convert digital filter second-order section data to transfer function form.

Usage

```
sos2tf(sos, g = 1)
```

Arguments

`sos`	Second-order section representation, specified as an nrow-by-6 matrix, whose rows contain the numerator and denominator coefficients of the second-order sections: `sos <- rbind(cbind(B1, A1), cbind(...),cbind(Bn, An))`, where `B1 <- c(b0, b1, b2)`, and `A1 <- c(a0,a1, a2)` for section 1, etc. The b0 entry must be nonzero for each section.
`g`	Overall gain factor that effectively scales the output `b` vector (or any one of the input `Bi` vectors). Default: 1.

Value

An object of class "Arma" with the following list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>.

See Also

as.Arma, filter

Examples

```
sos <- rbind(c(1, 1, 1, 1, 0, -1), c(-2, 3, 1, 1, 10, 1))
ba <- sos2tf(sos)
```

---

sos2zp *Sos to zero-pole-gain*

Description

Convert digital filter second-order section data to zero-pole-gain form.

Usage

```
sos2zp(sos, g = 1)
```

Arguments

sos	Second-order section representation, specified as an nrow-by-6 matrix, whose rows contain the numerator and denominator coefficients of the second-order sections: `sos <- rbind(cbind(B1, A1), cbind(...),cbind(Bn, An))`, where `B1 <- c(b0, b1, b2)`, and `A1 <- c(a0,a1, a2)` for section 1, etc. The b0 entry must be nonzero for each section.
g	Overall gain factor that effectively scales the output `b` vector (or any one of the input `B_i` vectors). Default: 1.

Value

A list of class "Zpg" with the following list elements:

z complex vector of the zeros of the model (roots of `B(z)`)

p complex vector of the poles of the model (roots of `A(z)`)

k overall gain (`B(Inf)`)

Author(s)

Julius O. Smith III <jos@ccrma.stanford.edu>.
Conversion to R by, Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>

See Also

filter

Examples

```
sos <- rbind(c(1, 0, 1, 1, 0, -0.81), c(1, 0, 0, 1, 0, 0.49))
zpk <- sos2zp(sos)
```

`sosfilt` *Second-order sections filtering*

Description

One-dimensional second-order (biquadratic) sections IIR digital filtering.

Usage

```
sosfilt(sos, x, zi = NULL)
```

Arguments

`sos`	Second-order section representation, specified as an nrow-by-6 matrix, whose rows contain the numerator and denominator coefficients of the second-order sections: `sos <- rbind(cbind(B1, A1), cbind(...),cbind(Bn, An))`, where `B1 <- c(b0, b1, b2)`, and `A1 <- c(a0,a1, a2)` for section 1, etc. The b0 entry must be nonzero for each section.
`x`	the input signal to be filtered, specified as a numeric or complex vector or matrix. If `x` is a matrix, each column is filtered.
`zi`	If `zi` is provided, it is taken as the initial state of the system and the final state is returned as zf. If `x` is a vector, `zi` must be a matrix with `nrow(sos)` rows and 2 columns. If `x` is a matrix, then `zi` must be a 3-dimensional array of size `(nrow(sos), 2, ncol(x))`. Alternatively, `zi` may be the character string `"zf"`, which specifies to return the final state vector even though the initial state vector is set to all zeros. Default: NULL.

Details

The filter function is implemented as a series of second-order filters with direct-form II transposed structure. It is designed to minimize numerical precision errors for high-order filters [1].

Value

The filtered signal, of the same dimensions as the input signal. In case the `zi` input argument was specified, a list with two elements is returned containing the variables `y`, which represents the output signal, and `zf`, which contains the final state vector or matrix.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

References

Smith III, J.O. (2012). Introduction to digital filters, with audio applications (3rd Ed.). W3K Publishing.

See Also

filter, filtfilt, Sos

Examples

```
fs <- 1000
t <- seq(0, 1, 1/fs)
s <- sin(2* pi * t * 6)
x <- s + rnorm(length(t))
plot(t, x, type = "l", col="light gray")
lines(t, s, col="black")
sosg <- butter(3, 0.02, output = "Sos")
sos <- sosg$sos
sos[1, 1:3] <- sos[1, 1:3] * sosg$g
y <- sosfilt(matrix(sos, ncol=6), x)
lines(t, y, col="red")

## using 'filter' will handle the gain for you
y2 <- filter(sosg, x)
all.equal(y, y2)

## The following example is from Python scipy.signal.sosfilt
## It shows the instability that results from trying to do a
## 13th-order filter in a single stage (the numerical error
## pushes some poles outside of the unit circle)
arma <- ellip(13, 0.009, 80, 0.05, output='Arma')
sos <- ellip(13, 0.009, 80, 0.05, output='Sos')
x <- rep(0, 700); x[1] <- 1
y_arma <- filter(arma, x)
y_sos <- filter(sos, x)
plot(y_arma, type ="l")
lines (y_sos, col = 2)
legend("topleft", legend = c("Arma", "Sos"), lty = 1, col = 1:2)
```

---

specgram *Spectrogram*

Description

Spectrogram using short-time Fourier transform.

Usage

```
specgram(
  x,
  n = min(256, length(x)),
  fs = 2,
  window = hanning(n),
```

```
  overlap = ceiling(n/2)
)

## S3 method for class 'specgram'
plot(
  x,
  col = grDevices::gray(0:512/512),
  xlab = "Time",
  ylab = "Frequency",
  ...
)

## S3 method for class 'specgram'
print(
  x,
  col = grDevices::gray(0:512/512),
  xlab = "Time",
  ylab = "Frequency",
  ...
)
```

Arguments

x	Input signal, specified as a vector.
n	Size of the FFT window. Default: 256 (or less if x is shorter).
fs	Sample rate in Hz. Default: 2
window	Either an integer indicating the length of a Hanning window, or a vector of values representing the shape of the FFT tapering window. Default: hanning(n)
overlap	Overlap with previous window. Default: half the window length
col	Colormap to use for plotting. Default: grDevices::gray(0:512 / 512)
xlab	Label for x-axis of plot. Default: "Time"
ylab	Label for y-axis of plot. Default: "Frequency"
...	Additional arguments passed to the image plotting function

Details

Generate a spectrogram for the signal x. The signal is chopped into overlapping segments of length n, and each segment is windowed and transformed into the frequency domain using the FFT. The default segment size is 256. If fs is given, it specifies the sampling rate of the input signal. The argument window specifies an alternate window to apply rather than the default of hanning(n). The argument overlap specifies the number of samples overlap between successive segments of the input signal. The default overlap is length (window)/2.

When results of specgram are printed, a spectrogram will be plotted. As with lattice plots, automatic printing does not work inside loops and function calls, so explicit calls to print or plot are needed there.

The choice of window defines the time-frequency resolution. In speech for example, a wide window shows more harmonic detail while a narrow window averages over the harmonic detail and shows more formant structure. The shape of the window is not so critical so long as it goes gradually to zero on the ends.

Step size (which is window length minus overlap) controls the horizontal scale of the spectrogram. Decrease it to stretch, or increase it to compress. Increasing step size will reduce time resolution, but decreasing it will not improve it much beyond the limits imposed by the window size (you do gain a little bit, depending on the shape of your window, as the peak of the window slides over peaks in the signal energy). The range 1-5 msec is good for speech.

FFT length controls the vertical scale. Selecting an FFT length greater than the window length does not add any information to the spectrum, but it is a good way to interpolate between frequency points which can make for prettier spectrograms.

AFTER you have generated the spectral slices, there are a number of decisions for displaying them. First the phase information is discarded and the energy normalized:

```
S <- abs(S); S <- S / max(S)
```

Then the dynamic range of the signal is chosen. Since information in speech is well above the noise floor, it makes sense to eliminate any dynamic range at the bottom end. This is done by taking the max of the magnitude and some minimum energy such as minE = -40dB. Similarly, there is not much information in the very top of the range, so clipping to a maximum energy such as maxE = -3dB makes sense:

```
S <- max(S, 10^(minE / 10)); S <- min(S, 10^(maxE / 10))
```

The frequency range of the FFT is from 0 to the Nyquist frequency of one half the sampling rate. If the signal of interest is band limited, you do not need to display the entire frequency range. In speech for example, most of the signal is below 4 kHz, so there is no reason to display up to the Nyquist frequency of 10 kHz for a 20 kHz sampling rate. In this case you will want to keep only the first 40 More generally, to display the frequency range from minF to maxF, you could use the following row index:

```
idx <- (f >= minF & f <= maxF)
```

Then there is the choice of colormap. A brightness varying colormap such as copper or bone gives good shape to the ridges and valleys. A hue varying colormap such as jet or hsv gives an indication of the steepness of the slopes. In the field that I am working in (neuroscience / electrophysiology) rainbow color palettes such as jet are very often used. This is an unfortunate choice mainly because (a) colors do not have a natural order, and (b) rainbow palettes are not perceptually linear. It would be better to use a grayscale palette or the 'cool-to-warm' scheme. The examples show how to do this in R.

The final spectrogram is displayed in log energy scale and by convention has low frequencies on the bottom of the image.

Value

A list of class `specgram` consisting of the following elements:

S the complex output of the FFT, one row per slice

f the frequency indices corresponding to the rows of S

t the time indices corresponding to the columns of S

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Tom Short
This conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
sp <- specgram(chirp(seq(-2, 15, by = 0.001), 400, 10, 100, 'quadratic'))
specgram(chirp(seq(0, 5, by = 1/8000), 200, 2, 500,
        "logarithmic"), fs = 8000)

# use other color palettes than grayscale
jet <- grDevices::colorRampPalette(
         c("#00007F", "blue", "#007FFF", "cyan", "#7FFF7F",
           "yellow", "#FF7F00", "red", "#7F0000"))
plot(specgram(chirp(seq(0, 5, by = 1/8000), 200, 2, 500, "logarithmic"),
         fs = 8000), col = jet(20))
c2w <- grDevices::colorRampPalette(colors = c("red", "white", "blue"))
plot(specgram(chirp(seq(0, 5, by = 1/8000), 200, 2, 500, "logarithmic"),
         fs = 8000), col = c2w(50))
```

---

square *Square wave*

Description

Generate a square wave of period 2π with limits +1 and -1.

Usage

```
square(t, duty = 50)
```

Arguments

t	Time array, specified as a vector.
duty	Duty cycle, specified as a real scalar from 0 to 100. Default: 50.

Details

`y <- square(t)` generates a square wave with period 2π for the elements of the time array `t`. `square` is similar to the sine function but creates a square wave with values of –1 and 1.

`y <- square(t, duty)` generates a square wave with specified duty cycle `duty`. The duty cycle is the percent of the signal period in which the square wave is positive.

$$dutycycle = \frac{ontime * 100}{ontime + offtime}$$

Value

Square wave, returned as a vector.

Author(s)

Paul Kienzle.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## Create a vector of 100 equally spaced numbers from 0 to 3pi.
## Generate a square wave with a period of 2pi.
t <- seq(0, 3*pi, length.out = 100)
y <- square(t)
plot(t/pi, y, type="l", xlab = expression(t/pi), ylab = "")
lines (t/pi, sin(t), col = "red")

## Generate a 30 Hz square wave sampled at 1 kHz for 70 ms.
## Specify a duty cycle of 37%.
## Add white Gaussian noise with a variance of 1/100.
t <- seq(0, 0.07, 1/1e3)
y <- square(2 * pi * 30 * t, 37) + rnorm(length(t)) / 10
plot(t, y, type="l", xlab = "", ylab = "")
```

---

stft *Short-Term Fourier Transform*

Description

Compute the short-term Fourier transform of a vector or matrix.

Usage

```
stft(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.75,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1
)
```

Arguments

x	input data, specified as a numeric or complex vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.

`window` If `window` is a vector, each segment has the same length as `window` and is multiplied by `window` before (optional) zero-padding and calculation of its periodogram. If `window` is a scalar, each segment has a length of `window` and a Hamming window is used. Default: `nextpow2(sqrt(NROW(x)))` (the square root of the length of `x` rounded up to the next power of two). The window length must be larger than 3.

`overlap` segment overlap, specified as a numeric value expressed as a multiple of window or segment length. 0 <= overlap < 1. Default: 0.75.

`nfft` Length of FFT, specified as an integer scalar. The default is the length of the `window` vector or has the same value as the scalar `window` argument. If `nfft` is larger than the segment length, (seg_len), the data segment is padded `nfft - seg_len` zeros. The default is no padding. Nfft values smaller than the length of the data segment (or window) are ignored. Note that the use of padding to increase the frequency resolution of the spectral estimate is controversial.

`fs` sampling frequency (Hertz), specified as a positive scalar. Default: 1.

Value

A list containing the following elements:

`f` vector of frequencies at which the STFT is estimated. If `x` is numeric, power from negative frequencies is added to the positive side of the spectrum, but not at zero or Nyquist (fs/2) frequencies. This keeps power equal in time and spectral domains. If `x` is complex, then the whole frequency range is returned.

`t` vector of time points at which the STFT is estimated.

`s` Short-time Fourier transform, returned as a matrix or a 3-D array. Time increases across the columns of `s` and frequency increases down the rows. The third dimension, if present, corresponds to the input channels.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
fs <- 8000
y <- chirp(seq(0, 5 - 1/fs, by = 1/fs), 200, 2, 500, "logarithmic")
ft <- stft (y, fs = fs)
filled.contour(ft$t, ft$f, t(ft$s), xlab = "Time (s)",
               ylab = "Frequency (Hz)")
```

tf2sos *Transfer function to second-order sections form*

Description

Convert digital filter transfer function data to second-order section form.

Usage

```
tf2sos(b, a)
```

Arguments

b	moving average (MA) polynomial coefficients
a	autoregressive (AR) polynomial coefficients

Value

A list with the following list elements:

sos Second-order section representation, specified as an nrow-by-6 matrix, whose rows contain the numerator and denominator coefficients of the second-order sections: `sos <- rbind(cbind(B1, A1), cbind(...), cbind(Bn, An))`, where `B1 <- c(b0, b1, b2)`, and `A1 <- c(a0, a1, a2)` for section 1, etc. The b0 entry must be nonzero for each section.

g Overall gain factor that effectively scales the output `b` vector (or any one of the input `Bi` vectors).

Author(s)

Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>.

See Also

See also `filter`

Examples

```
b <- c(1, 0, 0, 0, 0, 1)
a <- c(1, 0, 0, 0, 0, .9)
sosg <- tf2sos (b, a)
```

`tf2zp` *Transfer function to zero-pole-gain form*

Description

Convert digital filter transfer function parameters to zero-pole-gain form.

Usage

```
tf2zp(b, a)
```

Arguments

`b`	moving average (MA) polynomial coefficients, specified as a numeric vector or matrix. In case of a matrix, then each row corresponds to an output of the system. The number of columns of `b` must be less than or equal to the length of `a`.
`a`	autoregressive (AR) polynomial coefficients, specified as a vector.

Value

A list of class Zpg with the following list elements:

z complex vector of the zeros of the model (roots of `B(z)`)

p complex vector of the poles of the model (roots of `A(z)`)

g overall gain (`B(Inf)`)

Author(s)

Geert van Boxtel, <gjmvanboxtel@gmail.com>

See Also

`filter`

Examples

```
b <- c(2, 3)
a <- c(1, 1/sqrt(2), 1/4)
zpk <- tf2zp(b, a)
```

`tfestimate` *Transfer Function Estimate*

Description

Finds a transfer function estimate for signals.

Usage

```
tfestimate(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none")
)

tfe(
  x,
  window = nextpow2(sqrt(NROW(x))),
  overlap = 0.5,
  nfft = ifelse(isScalar(window), window, length(window)),
  fs = 1,
  detrend = c("long-mean", "short-mean", "long-linear", "short-linear", "none")
)
```

Arguments

`x`	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
`window`	If `window` is a vector, each segment has the same length as `window` and is multiplied by `window` before (optional) zero-padding and calculation of its periodogram. If `window` is a scalar, each segment has a length of `window` and a Hamming window is used. Default: `nextpow2(sqrt(length(x)))` (the square root of the length of `x` rounded up to the next power of two). The window length must be larger than 3.
`overlap`	segment overlap, specified as a numeric value expressed as a multiple of window or segment length. 0 <= overlap < 1. Default: 0.5.
`nfft`	Length of FFT, specified as an integer scalar. The default is the length of the `window` vector or has the same value as the scalar `window` argument. If `nfft` is larger than the segment length, (seg_len), the data segment is padded `nfft - seg_len` zeros. The default is no padding. Nfft values smaller than the length of the data segment (or window) are ignored. Note that the use of padding to increase the frequency resolution of the spectral estimate is controversial.
`fs`	sampling frequency (Hertz), specified as a positive scalar. Default: 1.

detrend character string specifying detrending option; one of:

"long-mean" remove the mean from the data before splitting into segments (default)

"short-mean" remove the mean value of each segment

"long-linear" remove linear trend from the data before splitting into segments

"short-linear" remove linear trend from each segment

"none" no detrending

Details

tfestimate uses Welch's averaged periodogram method.

Value

A list containing the following elements:

freq vector of frequencies at which the spectral variables are estimated. If x is numeric, power from negative frequencies is added to the positive side of the spectrum, but not at zero or Nyquist (fs/2) frequencies. This keeps power equal in time and spectral domains. If x is complex, then the whole frequency range is returned.

trans NULL for univariate series. For multivariate series, a matrix containing the transfer function estimates between different series. Column $i + (j-1) * (j-2)/2$ of coh contains the cross-spectral estimates between columns i and j of x, where $i < j$.

Note

The function tfestimate (and its deprecated alias tfe) is a wrapper for the function pwelch, which is more complete and more flexible.

Author(s)

Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

pwelch

Examples

```
fs <- 1000
f <- 250
t <- seq(0, 1 - 1/fs, 1/fs)
s1 <- sin(2 * pi * f * t) + runif(length(t))
s2 <- sin(2 * pi * f * t - pi / 3) + runif(length(t))
rv <- tfestimate(cbind(s1, s2), fs = fs)
plot(rv$freq, 10*log10(abs(rv$trans)), type="l", xlab = "Frequency",
 ylab = "Tranfer Function Estimate (dB)", main = colnames((rv$trans)))
```

```
h <- fir1(30, 0.2, window = rectwin(31))
x <- rnorm(16384)
y <- filter(h, x)
tfe <- tfestimate(cbind(x, y), 1024, fs = 500)
plot(tfe$freq, 10*log10(abs(tfe$trans)), type="l", xlab = "Frequency",
  ylab = "Tranfer Function Estimate (dB)", main = colnames((tfe$trans)))
```

`triang` *Triangular window*

Description

Return the filter coefficients of a triangular window of length `n`.

Usage

```
triang(n)
```

Arguments

`n`	Window length, specified as a positive integer.

Details

Unlike the Bartlett window, `triang` does not go to zero at the edges of the window. For odd `n`, `triang(n)` is equal to `bartlett(m + 2)` except for the zeros at the edges of the window.

Value

triangular window, returned as a vector. If you specify a one-point window (`n = 1`), the value 1 is returned.

Author(s)

Andreas Weingessel, <Andreas.Weingessel@ci.tuwien.ac.at>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

`bartlett`

Examples

```
t <- triang(64)
plot (t, type = "l", xlab = "Samples", ylab =" Amplitude")
```

tripuls *Sampled aperiodic triangle*

Description

Generate a triangular pulse over the interval `-w / 2` to `w / 2`, sampled at times `t`.

Usage

```
tripuls(t, w = 1, skew = 0)
```

Arguments

`t`	Sample times of triangle wave specified by a vector.
`w`	Width of the triangular pulse to be generated. Default: 1.
`skew`	Skew, a value between -1 and 1, indicating the relative placement of the peak within the width. -1 indicates that the peak should be at `-w / 2`, and 1 indicates that the peak should be at `w / 2`. Default: 0 (no skew).

Details

`y <- tripuls(t)` returns a continuous, aperiodic, symmetric, unity-height triangular pulse at the times indicated in array `t`, centered about `t = 0` and with a default width of 1.

`y <- tripuls(t, w)` generates a triangular pulse of width `w`.

`y <- tripuls(t, w, skew)` generates a triangular pulse with skew `skew`, where $-1 \leq skew \leq 1$. When `skew` is 0, a symmetric triangular pulse is generated.

Value

Triangular pulse, returned as a vector.

Author(s)

Paul Kienzle, Mike Miller.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
fs <- 10e3
t <- seq(-0.1, 0.1, 1/fs)
w <- 40e-3
y <- tripuls(t, w)
plot(t, y, type="l", xlab = "", ylab = "",
     main = "Symmetric triangular pulse")

## displace into paste and future
tpast <- -45e-3
```

```
spast <- -0.45
ypast <- tripuls(t-tpast, w, spast)
tfutr <- 60e-3
sfutr <- 1
yfutr <- tripuls(t-tfutr, w/2, sfutr)
plot (t, y, type = "l", xlab = "", ylab = "", ylim = c(0, 1))
lines(t, ypast, col = "red")
lines(t, yfutr, col = "blue")
```

`tukeywin` *Tukey (tapered cosine) window*

Description

Return the filter coefficients of a Tukey window (also known as the cosine-tapered window) of length `n`.

Usage

```
tukeywin(n, r = 1/2)
```

Arguments

`n`	Window length, specified as a positive integer.
`r`	Cosine fraction, specified as a real scalar. The Tukey window is a rectangular window with the first and last `r / 2` percent of the samples equal to parts of a cosine. For example, setting `r = 0.5` (default) produces a Tukey window where 1/2 of the entire window length consists of segments of a phase-shifted cosine with period 2r = 1. If you specify r <= 0, an n-point rectangular window is returned. If you specify r >= 1, an n-point von Hann window is returned.

Details

The Tukey window, also known as the tapered cosine window, can be regarded as a cosine lobe that is convolved with a rectangular window. `r` defines the ratio between the constant section and and the cosine section. It has to be between 0 and 1. The function returns a Hann window for `r` equal to 1 and a rectangular window for `r` equal to 0.

Value

Tukey window, returned as a vector.

Author(s)

Laurent Mazet, <mazet@crm.mot.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
n <- 128
t0 <- tukeywin(n, 0)        # Equivalent to a rectangular window
t25 <- tukeywin(n, 0.25)
t5 <- tukeywin(n)           # default r = 0.5
t75 <- tukeywin(n, 0.75)
t1 <- tukeywin(n, 1)         # Equivalent to a Hann window
plot(t0, type = "l", xlab = "Samples", ylab =" Amplitude", ylim=c(0,1.2))
lines(t25, col = 2)
lines(t5, col = 3)
lines(t75, col = 4)
lines(t1, col = 5)
```

---

udecode *Uniform decoder*

Description

Decode 2^n-level quantized integer inputs to floating-point outputs.

Usage

```
udecode(u, n, v = 1, saturate = TRUE)
```

Arguments

u	Input, a multidimensional array of integer numbers (can be complex).
n	Number of levels used in 2^n-level quantization. n must be between 2 and 32
v	Limit on the range of u to the range from -v to v before saturating them. Default 1.
saturate	Logical indicating to saturate (TRUE, default) or to wrap (FALSE) overflows. See Details.

Details

y <- udecode(u, n) inverts the operation of uencode and reconstructs quantized floating-point values from an encoded multidimensional array of integers u. The input argument n must be an integer between 2 and 32. The integer n specifies that there are 2^n quantization levels for the inputs, so that entries in u must be either:

- Signed integers in the range $-2^n/2$ to $(2^n/2) - 1$
- Unsigned integers in the range 0 to $2^n - 1$

Inputs can be real or complex values of any integer data type. Overflows (entries in u outside of the ranges specified above) are saturated to the endpoints of the range interval. The output has the same dimensions as the input u. Its entries have values in the range -1 to 1.

`y <- udecode(u, n, v)` decodes u such that the output has values in the range `-v` to `v`, where the default value for `v` is 1.

`y <- udecode(u, n, v, saturate)` decodes u and treats input overflows (entries in u outside of the range `-v` to `v` according to `saturate`, which can be set to one of the following:

- TRUE (default). Saturate overflows.
  - Entries in signed inputs u whose values are outside of the range $-2^n/2$ to $(2^n/2)\breve{}1$ are assigned the value determined by the closest endpoint of this interval.
  - Entries in unsigned inputs u whose values are outside of the range 0 to $2^n - 1$ are assigned the value determined by the closest endpoint of this interval.
- FALSE Wrap all overflows according to the following:
  - Entries in signed inputs u whose values are outside of the range $-2^n/2$ to $(2^n/2)\breve{}1$ are wrapped back into that range using modulo 2^n arithmetic (calculated using $u = mod(u + 2^n/2, 2^n) - (2^n/2)$).
  - Entries in unsigned inputs u whose values are outside of the range 0 to $2^n - 1$ are wrapped back into the required range before decoding using modulo 2^n arithmetic (calculated using $u = mod(u, 2^n)$).

Value

Multidimensional array of the same size as u containing floating point numbers.

Note

The real and imaginary components of complex inputs are decoded independently.

Author(s)

Georgios Ouzounis, <ouzounis_georgios@hotmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
u <- c(-1, 1, 2, -5)
ysat <- udecode(u, 3)

# Notice the last entry in u saturates to 1, the default peak input
# magnitude. Change the peak input magnitude to 6.
ysatv <- udecode(u, 3, 6)

# The last input entry still saturates. Wrap the overflows.
ywrap = udecode(u, 3, 6, FALSE)

# Add more quantization levels.
yprec <- udecode(u, 5)
```

uencode *Uniform encoder*

Description

Quantize and encode floating-point inputs to integer outputs.

Usage

```
uencode(u, n, v = 1, signed = FALSE)
```

Arguments

u	Input, a multidimensional array of numbers, real or complex, single or double precision.
n	Number of levels used in 2^n-level quantization. n must be between 2 and 32
v	Limit on the range of u to the range from -v to v before saturating them. Default 1.
signed	Logical indicating signed or unsigned output. See Details. Default: FALSE.

Details

y <- uencode(u, n) quantizes the entries in a multidimensional array of floating-point numbers u and encodes them as integers using 2^n-level quantization. n must be an integer between 2 and 32 (inclusive). Inputs can be real or complex, double- or single-precision. The output y and the input u are arrays of the same size. The elements of the output y are unsigned integers with magnitudes in the range 0 to $2^n - 1$. Elements of the input u outside of the range -1 to 1 are treated as overflows and are saturated.

- For entries in the input u that are less than -1, the value of the output of uencode is 0.
- For entries in the input u that are greater than 1, the value of the output of uencode is $2^n - 1$.

y <- uencode(u, n, v) allows the input u to have entries with floating-point values in the range -v to v before saturating them (the default value for v is 1). Elements of the input u outside of the range -v to v are treated as overflows and are saturated:

- For input entries less than -v, the value of the output of uencode is 0.
- For input entries greater than v, the value of the output of uencode is $2^n - 1$.

y <- uencode(u, n, v, signed) maps entries in a multidimensional array of floating-point numbers u whose entries have values in the range -v to v to an integer output y. Input entries outside this range are saturated. The integer type of the output depends on the number of quantization levels 2^n and the value of signed, which can be one of the following:

- TRUE: Outputs are signed integers with magnitudes in the range $-2^n/2$ to $(2^n/2) - 1$.
- FALSE (default): Outputs are unsigned integers with magnitudes in the range 0 to $2^n - 1$.

Value

Multidimensional array of the same size as u containing signed or unsigned integers.

Author(s)

Georgios Ouzounis, <ouzounis_georgios@hotmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
u <- seq(-1, 1, 0.01)
y <- uencode(u, 3)
plot(u, y)
```

---

ultrwin *Ultraspherical window*

Description

Return the coefficients of an ultraspherical window

Usage

```
ultrwin(n, mu = 3, xmu = 1)
```

Arguments

n	Window length, specified as a positive integer.
mu	parameter that controls the side-lobe roll-off ratio. Default: 3.
xmu	parameters that provides a trade-off between the ripple ratio and a width characteristic. Default: 1

Value

ultraspherical window, returned as a vector.

Note

The Dolph-Chebyshev and Saramaki windows are special cases of the Ultraspherical window, with mu set to 0 and 1, respectively.

Author(s)

Geert van Boxtel <G.J.M.vanBoxtel@gmail.com>.

References

[1] Bergen, S.W.A., and Antoniou, A. Design of Ultraspherical Window Functions with Prescribed Spectral Characteristics. EURASIP Journal on Applied Signal Processing 2004:13, 2053–2065.

Examples

```
w <- ultrwin(101, 3, 1)
plot (w, type = "l", xlab = "Samples", ylab =" Amplitude")
freqz(w)

w2 <- ultrwin(101, 2, 1)
f2 <- freqz(w2)
w3 <- ultrwin(101, 3, 1)
f3 <- freqz(w3)
w4 <- ultrwin(101, 4, 1)
f4 <- freqz(w4)
op <- par(mfrow = c(2, 1))
plot(w2, type = "l", col = "black", xlab = "", ylab = "")
lines(w3, col = "red")
lines(w4, col = "blue")
legend("topright", legend = 2:4, col = c("black", "red", "blue"), lty = 1)
plot (f2$w, 20 * log10(abs(f2$h)), type = "l", col = "black",
      xlab = "", ylab = "", ylim = c(-100, 50))
lines(f3$w, 20 * log10(abs(f3$h)), col = "red")
lines(f4$w, 20 * log10(abs(f4$h)), col = "blue")
legend("topright", legend = 2:4, col = c("black", "red", "blue"), lty = 1)
par(op)
title(main = "Effect of increasing the values of mu (xmu = 1)")

w1 <- ultrwin(101, 2, 1)
f1 <- freqz(w1)
w2 <- ultrwin(101, 2, 1.001)
f2 <- freqz(w2)
w3 <- ultrwin(101, 2, 1.002)
f3 <- freqz(w3)
op <- par(mfrow = c(2, 1))
plot(w1, type = "l", col = "black", xlab = "", ylab = "")
lines(w2, col = "red")
lines(w3, col = "blue")
legend("topright", legend = 2:4, col = c("black", "red", "blue"), lty = 1)
plot (f1$w, 20 * log10(abs(f1$h)), type = "l", col = "black",
      xlab = "", ylab = "", ylim = c(-100, 50))
lines(f2$w, 20 * log10(abs(f2$h)), col = "red")
lines(f3$w, 20 * log10(abs(f3$h)), col = "blue")
legend("topright", legend = c(1, 1.001, 1.002),
       col = c("black", "red", "blue"), lty = 1)
par(op)
title(main = "Effect of increasing the values of xmu (mu = 2)")
```

unshiftdata *Inverse of shiftdata*

Description

Reverse what has been done by shiftdata().

Usage

```
unshiftdata(sd)
```

Arguments

sd	A list of objects named x, perm, and nshifts, as returned by shiftdata()

Details

unshiftdata restores the orientation of the data that was shifted with shiftdata. The permutation vector is given by perm, and nshifts is the number of shifts that was returned from shiftdata().

unshiftdata is meant to be used in tandem with shiftdata. These functions are useful for creating functions that work along a certain dimension, like filter, goertzel, sgolayfilt, and sosfilt. These functions are useful for creating functions that work along a certain dimension, like filter, sgolayfilt, and sosfilt.

Value

Array with the same values and dimensions as passed to a previous call to shiftdata.

Author(s)

Georgios Ouzounis, <ouzounis_georgios@hotmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

shiftdata

Examples

```
## create a 3x3 magic square
x <- pracma::magic(3)
## Shift the matrix x to work along the second dimension.
## The permutation vector, perm, and the number of shifts, nshifts,
## are returned along with the shifted matrix.
sd <- shiftdata(x, 2)

## Shift the matrix back to its original shape.
y <- unshiftdata(sd)
```

```
## Rearrange Array to Operate on First Nonsingleton Dimension
x <- 1:5
sd <- shiftdata(x)
y <- unshiftdata(sd)
```

---

unwrap *Unwrap phase angles*

Description

Unwrap radian phases by adding or subtracting multiples of `2 * pi`.

Usage

```
unwrap(x, tol = pi)
```

Arguments

`x`	Input array, specified as a vector or a matrix. If `x` is a matrix, unwrapping along the columns of `x` is applied.
`tol`	Jump threshold to apply phase shift, specified as a scalar. A jump threshold less than pi has the same effect as the threshold pi. Default: $$pi$$.

Value

Unwrapped phase angle, returned as a vector, matrix, or multidimensional array.

Author(s)

Bill Lash.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>

Examples

```
## Define spiral shape.
t <- seq(0, 6 * pi, length.out = 201)
x <- t / pi * cos(t)
y <- t / pi * sin(t)
plot(x, y, type = "l")
## find phase angle
p = atan2(y, x)
plot(t, p, type="l")
## unwrap it
```

```
q = unwrap(p)
plot(t, q, type ="l")
```

---

upfirdn *Upsample, apply FIR filter, downsample*

Description

Filter and resample a signal using polyphase interpolation.

Usage

```
upfirdn(x, h, p = 1, q = 1)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
h	Impulse response of the FIR filter specified as a numeric vector or matrix. If it is a vector, then it represents one FIR filter to may be applied to multiple signals in x; if it is a matrix, then each column is a separate FIR impulse response.
p	Upsampling factor, specified as a positive integer (default: 1).
q	downsampling factor, specified as a positive integer (default: 1).

Details

upfirdn performs a cascade of three operations:

1. Upsample the input data in the matrix x by a factor of the integer p (inserting zeros)
2. FIR filter the upsampled signal data with the impulse response sequence given in the vector or matrix h
3. Downsample the result by a factor of the integer q (throwing away samples)

The FIR filter is usually a lowpass filter, which you must design using another function such as `fir1`.

Value

output signal, returned as a vector or matrix. Each column has length `ceiling(((length(x) - 1) * p + length(h)) / q)`.

Note

This function uses a polyphase implementation, which is generally faster than using `filter` by a factor equal to the downsampling factor, since it only calculates the needed outputs.

Author(s)

Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

fir1

Examples

```
x <- c(1, 1, 1)
h <- c(1, 1)

## FIR filter
y <- upfirdn(x, h)

## FIR filter + upsampling
y <- upfirdn(x, h, 5)

## FIR filter + downsampling
y <- upfirdn(x, h, 1, 2)

## FIR filter + up/downsampling
y <- upfirdn(x, h, 5, 2)
```

---

upsample *Increase sample rate*

Description

Upsample a signal by an integer factor.

Usage

```
upsample(x, n, phase = 0)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
n	upsampling factor, specified as a positive integer. The signal is upsampled by inserting n - 1 zeros between samples.
phase	offset, specified as a positive integer from 0 to n - 1. Default: 0.

Value

Upsampled signal, returned as a vector or matrix.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

downsample, interp, decimate, resample

Examples

```
x <- seq_len(4)
u <- upsample(x, 3)
u <- upsample(x, 3, 2)

x <- matrix(seq_len(6), 3, byrow = TRUE)
u <- upsample(x, 3)
```

---

upsamplefill *Upsample and Fill*

Description

Upsample and fill with given values or copies of the vector elements.

Usage

```
upsamplefill(x, v, copy = FALSE)
```

Arguments

x	input data, specified as a numeric vector or matrix. In case of a vector it represents a single signal; in case of a matrix each column is a signal.
v	vector of values to be placed between the elements of x.
copy	logical. If TRUE then v should be a scalar (length(v) == 1) and each value in x are repeated v times. If FALSE (default), the values in the vector v are placed between the elements of x.

Value

upsampled vector or matrix

Author(s)

Juan Pablo Carbajal, <carbajal@ifi.uzh.ch>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

upsample

Examples

```
u <- upsamplefill(diag(2), 2, TRUE)
u <- upsamplefill(diag(2), rep(-1, 3))
```

---

wconv *1-D or 2-D convolution*

Description

Compute the one- or two-dimensional convolution of two vectors or matrices.

Usage

```
wconv(
  type = c("1d", "2d", "row", "column"),
  a,
  b,
  shape = c("full", "same", "valid")
)
```

Arguments

type	Numeric or character, specifies the type of convolution to perform: **"1d"** For a and b as (coerced to) vectors, perform 1-D convolution of a and b; **"2d** For a and b as (coerced to) matrices, perform 2-D convolution of a and b; **"row"** For a as (coerced to) a matrix, and b (coerced to) a vector, perform the 1-D convolution of the rows of a and b; **"column"** For a as (coerced to) a matrix, and b (coerced to) a vector, perform the 1-D convolution of the colums of a and b;
a, b	Input vectors or matrices, coerced to numeric.
shape	Subsection of convolution, partially matched to: **"full"** Return the full convolution (default) **"same"** Return the central part of the convolution with the same size as A. The central part of the convolution begins at the indices `floor(c(nrow(b), ncol(b)) / 2 + 1)` **"valid"** Return only the parts which do not include zero-padded edges. The size of the result is `max(c(nrow(a), ncol(b)) - c(nrow(b), ncol(b)) + 1, 0)`

Value

Convolution of input matrices, returned as a matrix or a vector.

Author(s)

Lukas Reichlin, <lukas.reichlin@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

conv

Examples

```
a <- matrix(1:16, 4, 4)
b <- matrix(1:9, 3,3)
w <- wconv('2', a, b)
w <- wconv('1', a, b, 'same')
w <- wconv('r', a, b)
w <- wconv('r', a, c(0,1), 'same')
w <- wconv('c', a, c(0,1), 'valid')
```

---

welchwin *Welch window*

Description

Return the filter coefficients of a Welch window of length n.

Usage

```
welchwin(n, method = c("symmetric", "periodic"))
```

Arguments

n	Window length, specified as a positive integer.
method	Character string. Window sampling method, specified as:
	"symmetric" (Default). Use this option when using windows for filter design.
	"periodic" This option is useful for spectral analysis because it enables a windowed signal to have the perfect periodic extension implicit in the discrete Fourier transform. When 'periodic' is specified, the function computes a window of length n + 1 and returns the first n points.

Details

The Welch window is a polynomial window consisting of a single parabolic section:

$$w(k) = 1 - (k/N - 1)^2, n = 0, 1, ...n - 1$$

. The optional argument specifies a "symmetric" window (the default) or a "periodic" window. A symmetric window has zero at each end and maximum in the middle, and the length must be an integer greater than 2. The variable `N` in the formula above is `(n-1)/2`. A periodic window wraps around the cyclic interval `0,1, ... m-1`, and is intended for use with the DFT. The length must be an integer greater than 1. The variable `N` in the formula above is `n/2`.

Value

Welch window, returned as a vector.

Author(s)

Muthiah Annamalai, <muthiah.annamalai@uta.edu>,
Mike Gross, <mike@appl-tech.com>,
Peter V. Lanspeary, <pvl@mecheng.adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
w <- welchwin(64)
plot (w, type = "l", xlab = "Samples", ylab =" Amplitude")

ws = welchwin(64,'symmetric')
wp = welchwin(63,'periodic')
plot (ws, type = "l", xlab = "Samples", ylab =" Amplitude")
lines(wp, col="red")
```

---

wkeep *Keep part of vector or matrix*

Description

Extract elements from a vector or matrix.

Usage

```
wkeep(x, l, opt = "centered")
```

Arguments

x	input data, specified as a numeric vector or matrix.
l	either a positive integer value, specifying the length to extract from the input *vector* x, or a vector of length 2, indicating the submatrix to extract from the *matrix* x. See the examples.
opt	One of: **character string** matched against c("centered", "left", "right"), indicating the location of the *vector* x to extract **positive integer** starting index of the input *vector* x **two-element vector** starting row and columns from the *matrix* x See the examples. Default: "centered".

Value

extracted vector or matrix

Author(s)

Sylvain Pelissier, <sylvain.pelissier@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
## create a vector
x <- 1:10
## Extract a vector of length 6 from the central part of x.
y <- wkeep(x, 6, 'c')

## Extract two vectors of length 6, one from the left part of x, and the
## other from the right part of x.
y <- wkeep(x, 6, 'l')
y <- wkeep(x, 6, 'r')

## Create a 5-by-5 matrix.
x <- matrix(round(runif(25, 0, 25)), 5, 5)

## Extract a 3-by-2 matrix from the center of x
y <- wkeep(x, c(3, 2))

## Extract from x the 2-by-4 submatrix starting at x[3, 1].
y <- wkeep(x, c(2, 4), c(3, 1))
```

xcorr *Cross-correlation*

Description

Estimate the cross-correlation between two sequences or the autocorrelation of a single sequence

Usage

```
xcorr(
  x,
  y = NULL,
  maxlag = if (is.matrix(x)) nrow(x) - 1 else max(length(x), length(y)) - 1,
  scale = c("none", "biased", "unbiased", "coeff")
)
```

Arguments

x	Input, numeric or complex vector or matrix. Must not be missing.
y	Input, numeric or complex vector data. If x is a matrix (not a vector), y must be omitted. y may be omitted if x is a vector; in this case xcorr estimates the autocorrelation of x.
maxlag	Integer scalar. Maximum correlation lag. If omitted, the default value is N-1, where N is the greater of the lengths of x and y or, if x is a matrix, the number of rows in x.
scale	Character string. Specifies the type of scaling applied to the correlation vector (or matrix). matched to one of: **"none"** return the unscaled correlation, R **"biased"** return the biased average, R / N **"unbiased"** return the unbiased average, R(k) / (N - |k|) **"coeff"** return the correlation coefficient, R / (rms(x) . rms(y)), where k is the lag, and N is the length of x If omitted, the default value is "none". If y is supplied but does not have the same length as x, scale must be "none".

Details

Estimate the cross correlation R_xy(k) of vector arguments x and y or, if y is omitted, estimate autocorrelation R_xx(k) of vector x, for a range of lags k specified by the argument maxlag. If x is a matrix, each column of x is correlated with itself and every other column.

The cross-correlation estimate between vectors x and y (of length N) for lag k is given by

$$R_{xy}(k) = \sum_{i=1}^{N} x_{i+k} Conj(y_i)$$

where data not provided (for example `x[-1], y[N+1]`) is zero. Note the definition of cross-correlation given above. To compute a cross-correlation consistent with the field of statistics, see xcov.

The cross-correlation estimate is calculated by a "spectral" method in which the FFT of the first vector is multiplied element-by-element with the FFT of second vector. The computational effort depends on the length N of the vectors and is independent of the number of lags requested. If you only need a few lags, the "direct sum" method may be faster.

Value

A list containing the following variables:

R array of correlation estimates

lags vector of correlation lags `[-maxlag:maxlag]`

The array of correlation estimates has one of the following forms:

1. Cross-correlation estimate if X and Y are vectors.
2. Autocorrelation estimate if is a vector and Y is omitted.
3. If `x` is a matrix, `R` is a matrix containing the cross-correlation estimate of each column with every other column. Lag varies with the first index so that `R` has `2 * maxlag + 1` rows and P^2 columns where `P` is the number of columns in `x`.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Asbjorn Sabo, <asbjorn.sabo@broadpark.no>,
Peter Lanspeary. <peter.lanspeary@adelaide.edu.au>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

xcov.

Examples

```
## Create a vector x and a vector y that is equal to x shifted by 5
## elements to the right. Compute and plot the estimated cross-correlation
## of x and y. The largest spike occurs at the lag value when the elements
## of x and y match exactly (-5).
n <- 0:15
x <- 0.84^n
y <- pracma::circshift(x, 5)
rl <- xcorr(x, y)
plot(rl$lag, rl$R, type="h")

## Compute and plot the estimated autocorrelation of a vector x.
## The largest spike occurs at zero lag, when x matches itself exactly.
n <- 0:15
x <- 0.84^n
rl <- xcorr(x)
```

```
plot(rl$lag, rl$R, type="h")

## Compute and plot the normalized cross-correlation of vectors
## x and y with unity peak, and specify a maximum lag of 10.
n <- 0:15
x <- 0.84^n
y <- pracma::circshift(x, 5)
rl <- xcorr(x, y, 10, 'coeff')
plot(rl$lag, rl$R, type="h")
```

---

xcorr2 *2-D cross-correlation*

Description

Compute the 2D cross-correlation of matrices a and b.

Usage

```
xcorr2(a, b = a, scale = c("none", "biased", "unbiased", "coeff"))
```

Arguments

a	Input matrix, coerced to numeric. Must not be missing.
b	Input matrix, coerced to numeric. Default: a.
scale	Character string. Specifies the type of scaling applied to the correlation matrix. matched to one of:

"none" no scaling

"biased" Scales the raw cross-correlation by the maximum number of elements of a and b involved in the generation of any element of the output matrix.

"unbiased" Scales the raw correlation by dividing each element in the cross-correlation matrix by the number of products a and b used to generate that element.

"coeff" Scales the normalized cross-correlation on the range of [0 1] so that a value of 1 corresponds to a correlation coefficient of 1.

Details

If b is not specified, computes autocorrelation of a, i.e., same as `xcorr2 (a, a)`.

Value

2-D cross-correlation or autocorrelation matrix, returned as a matrix

Author(s)

Dave Cogdell, <cogdelld@asme.org>,
Paul Kienzle, <pkienzle@users.sf.net>,
Carne Draug, <carandraug+dev@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

conv2, xcorr.

Examples

```
m1 <- matrix(c(17, 24,  1,  8, 15,
               23,  5,  7, 14, 16,
                4,  6, 13, 20, 22,
               10, 12, 19, 21,  3,
               11, 18, 25,  2,  9), 5, 5, byrow = TRUE)
m2 <- matrix(c(8, 1, 6,
               3, 5, 7,
               4, 9, 2), 3, 3, byrow = TRUE)
R <- xcorr2(m1, m2)
```

---

xcov *Cross-covariance*

Description

Compute covariance at various lags (= correlation(x-mean(x), y-mean(y))).

Usage

```
xcov(
  x,
  y = NULL,
  maxlag = if (is.matrix(x)) nrow(x) - 1 else max(length(x), length(y)) - 1,
  scale = c("none", "biased", "unbiased", "coeff")
)
```

Arguments

x	Input, numeric or complex vector or matrix. Must not be missing.
y	Input, numeric or complex vector data. If x is a matrix (not a vector), y must be omitted. y may be omitted if x is a vector; in this case xcov estimates the autocovariance of x.
maxlag	Integer scalar. Maximum covariance lag. If omitted, the default value is N-1, where N is the greater of the lengths of x and y or, if x is a matrix, the number of rows in x.

`scale` Character string. Specifies the type of scaling applied to the covariation vector (or matrix). matched to one of:

"none" return the unscaled covariance, C

"biased" return the biased average, C/N

"unbiased" return the unbiased average, C(k)/(N-|k|)

"coeff" return C/(covariance at lag 0), where `k` is the lag, and `N` is the length of `x`

If omitted, the default value is `"none"`. If `y` is supplied but does not have the same length as `x`, scale must be `"none"`.

Value

A list containing the following variables:

C array of covariance estimates

lags vector of covariance lags `[-maxlag:maxlag]`

The array of covariance estimates has one of the following forms:

1. Cross-covariance estimate if X and Y are vectors.
2. Autocovariance estimate if is a vector and Y is omitted.
3. If `x` is a matrix, `C` is a matrix containing the cross-covariance estimates of each column with every other column. Lag varies with the first index so that `C` has `2 * maxlag + 1` rows and P^2 columns where `P` is the number of columns in `x`.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

See Also

xcorr.

Examples

```
N <- 128
fs <- 5
t <- seq(0, 1, length.out = N)
x <- sin(2 * pi * fs * t) + runif(N)
cl <- xcov(x, maxlag = 20, scale = 'coeff')
plot (cl$lags, cl$C, type = "h", xlab = "", ylab = "")
points (cl$lags, cl$C)
abline(h = 0)
```

zerocrossing *Zero Crossing*

Description

Estimate zero crossing points of waveform.

Usage

```
zerocrossing(x, y)
```

Arguments

x	the x-coordinates of points in the function.
y	the y-coordinates of points in the function.

Value

Zero-crossing points

Author(s)

Carlo de Falco, <carlo.defalco@gmail.com>.
Conversion to R by Geert van Boxtel, <G.J.M.vanBoxtel@gmail.com>.

Examples

```
x <- seq(0, 1, length.out = 100)
y <- runif(100) - 0.5
x0 <- zerocrossing(x, y)
plot(x, y, type ="l", xlab = "", ylab = "")
points(x0, rep(0, length(x0)), col = "red")
```

zp2sos *Zero-pole-gain to second-order section format*

Description

Convert digital filter zero-pole-gain data to second-order section form.

Usage

```
zp2sos(z, p, g = 1, order = c("down", "up"))
```

Arguments

z	complex vector of the zeros of the model (roots of B(z))
p	complex vector of the poles of the model (roots of A(z))
g	overall gain (B(Inf)). Default: 1
order	row order, specified as:

"up" order the sections so the first row contains the poles farthest from the unit circle.

"down" (Default) order the sections so the first row of sos contains the poles closest to the unit circle.

The ordering influences round-off noise and the probability of overflow.

Value

A list with the following list elements:

sos Second-order section representation, specified as an nrow-by-6 matrix, whose rows contain the numerator and denominator coefficients of the second-order sections: sos <- rbind(cbind(B1, A1), cbind(...), cbind(Bn, An)), where B1 <- c(b0, b1, b2), and A1 <- c(a0, a1, a2) for section 1, etc. The b0 entry must be nonzero for each section.

g Overall gain factor that effectively scales the output b vector (or any one of the input Bi vectors).

Author(s)

Julius O. Smith III, <jos@ccrma.stanford.edu>.
Conversion to R by Geert van Boxtel, <gjmvanboxtel@gmail.com>

See Also

as.Sos, filter, sosfilt

Examples

```
zpk <- tf2zp (c(1, 0, 0, 0, 0, 1), c(1, 0, 0, 0, 0, .9))
sosg <- zp2sos (zpk$z, zpk$p, zpk$g)
```

---

zp2tf *Zero-pole-gain to transfer function*

Description

Convert digital filter zero-pole-gain data to transfer function form

Usage

```
zp2tf(z, p, g = 1)
```

Arguments

z	complex vector of the zeros of the model
p	complex vector of the poles of the model
g	overall gain. Default: 1.

Value

A list of class "Arma" with the following list elements:

b moving average (MA) polynomial coefficients

a autoregressive (AR) polynomial coefficients

Author(s)

Geert van Boxtel, <gjmvanboxtel@gmail.com>

See Also

as.Arma, filter

Examples

```
g <- 1
z <- c(0, 0)
p <- pracma::roots(c(1, 0.01, 1))
ba <- zp2tf(z, p, g)
```

Zpg *Zero pole gain model*

Description

Create an zero pole gain model of an ARMA filter, or convert other forms to a Zpg model.

Usage

```
Zpg(z, p, g)

as.Zpg(x, ...)

## S3 method for class 'Arma'
as.Zpg(x, ...)

## S3 method for class 'Ma'
as.Zpg(x, ...)
```

```
## S3 method for class 'Sos'
as.Zpg(x, ...)

## S3 method for class 'Zpg'
as.Zpg(x, ...)
```

Arguments

z	complex vector of the zeros of the model.
p	complex vector of the poles of the model.
g	overall gain of the model.
x	model to be converted.
...	additional arguments (ignored).

Details

as.Zpg converts from other forms, including Arma and Ma.

Value

A list of class Zpg with the following list elements:

z complex vector of the zeros of the model

p complex vector of the poles of the model

g gain of the model

Author(s)

Tom Short, <tshort@eprisolutions.com>,
adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>.

See Also

See also Arma

Examples

```
## design notch filter at pi/4 radians = 0.5/4 = 0.125 * fs
w = pi/4
# 2 poles, 2 zeros
# zeroes at r = 1
r <- 1
z1 <- r * exp(1i * w)
z2 <- r * exp(1i * -w)
# poles at r = 0.9
r = 0.9
p1 <- r * exp(1i * w)
p2 <- r * exp(1i * -w)
```

```
zpg <- Zpg(c(z1, z2), c(p1, p2), 1)
zplane(zpg)
freqz(zpg)

## Sharper edges: increase distance between zeros and poles
r = 0.8
p1 <- r * exp(1i * w)
p2 <- r * exp(1i * -w)
zpg <- Zpg(c(z1, z2), c(p1, p2), 1)
zplane(zpg)
freqz(zpg)
```

`zplane` *Zero-pole plot*

Description

Plot the poles and zeros of a filter or model on the complex Z-plane

Usage

```
zplane(filt, ...)

## S3 method for class 'Arma'
zplane(filt, ...)

## S3 method for class 'Ma'
zplane(filt, ...)

## S3 method for class 'Sos'
zplane(filt, ...)

## S3 method for class 'Zpg'
zplane(filt, ...)

## Default S3 method:
zplane(filt, a, ...)
```

Arguments

`filt`	for the default case, the moving-average coefficients of an ARMA model or filter. Generically, `filt` specifies an arbitrary model or filter operation.
`...`	additional arguments are passed through to plot.
`a`	the autoregressive (recursive) coefficients of an ARMA filter.

Details

Poles are marked with an `x`, and zeros are marked with an `o`.

Value

No value is returned.

Note

When results of `zplane` are printed, `plot` will be called. As with lattice plots, automatic printing does not work inside loops and function calls, so explicit calls to print or plot are needed there.

Author(s)

Paul Kienzle, <pkienzle@users.sf.net>,
Stefan van der Walt <stefan@sun.ac.za>,
Mike Miller.
Conversion to R by Tom Short,
adapted by Geert van Boxtel, <gjmvanboxtel@gmail.com>

References

https://en.wikipedia.org/wiki/Pole-zero_plot

See Also

`freqz`

Examples

```
## elliptic low-pass filter
elp <- ellip(4, 0.5, 20, 0.4)
zplane(elp)
```

Index