Package: fCWTr (via r-universe)

November 13, 2026

Type Package

Title Fast Continuous Wavelet Transform

Version 0.4.1.9000

Maintainer Lukas Schneiderbauer <lukas.schneiderbauer@gmail.com>

Description Enables the usage of the fast continuous wavelet transform, originally implemented in the 'C++' library 'fCWT' by Lukas Arts. See Arts, P.A. and Van den Broek, E.L. (2022) <doi:10.1038/s43588-021-00183-z> for details.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Depends R (>= 4.2)

Suggests ggplot2 (>= 3.3.0), viridis (>= 0.6.0), rlang (>= 1.0.0), tibble (>= 3.0.0), testthat (>= 3.0.0), knitr, rmarkdown

LinkingTo cpp11

SystemRequirements C++17, fftw3 library with single-precision support, i.e. libfftw3f (deb: libfftw3-dev, rpm: fftw-devel and fftw-libs-single, brew: fftw, macOS recipe: fftw-s), pkg-config (optional), OpenMP-enabled single-precision fftw3, i.e. libfftw3f_omp (optional), OpenMP (optional)

Config/testthat/edition 3

URL https://lschneiderbauer.github.io/fCWTr/, https://github.com/lschneiderbauer/fCWTr

Roxygen list(markdown = TRUE)

BugReports https://github.com/lschneiderbauer/fCWTr/issues

Imports units (>= 0.8)

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libfftw3-dev pkg-config libudunits2-dev

Repository https://lschneiderbauer.r-universe.dev

Date/Publication 2026-11-13 14:20:03 UTC

RemoteUrl https://github.com/lschneiderbauer/fcwtr

RemoteRef HEAD

RemoteSha f5e3e7182005dc077a1e092f3e2d5413a172b939

Contents

[.fcwtr_scalogram . . . 2
as.data.frame.fcwtr_scalogram . . . 3
as.matrix.fcwtr_scalogram . . . 4
as_tibble.fcwtr_scalogram . . . 5
autoplot.fcwtr_scalogram . . . 6
avx_enabled . . . 7
du . . . 8
fcwt . . . 9
fcwt_batch . . . 11
openmp_enabled . . . 14
plot.fcwtr_scalogram . . . 15
print.fcwtr_scalogram . . . 16
rbind.fcwtr_scalogram . . . 17
sigma_freq_res_rel . . . 18
sigma_from_freq_res . . . 18
sigma_from_freq_res_rel . . . 19
sigma_from_time_res . . . 20
sigma_res . . . 21
ts_piano_sample . . . 22
ts_sin_440 . . . 22
ts_sin_sin . . . 23
ts_sin_superpos . . . 23
u . . . 24

Index **25**

`[.fcwtr_scalogram` *Extract parts of a scalogram*

Description

Selects a time range and/or a frequency range of a scalogram. The time offset and the frequency range of the result are adjusted accordingly.

Usage

```
## S3 method for class 'fcwtr_scalogram'
x[i, j]
```

Arguments

x	An "fcwtr_scalogram" object resulting from `fcwt()`.
i, j	Indices selecting time slices (rows, `i`) and frequencies (columns, `j`; ordered from the highest to the lowest frequency). Positive, negative, or logical indices are allowed, as long as they select a contiguous, ascending range (e.g. `10:20` or `-(1:5)`), since the result must again have evenly spaced time and frequency axes. A missing index selects everything. Use `as.matrix()` for arbitrary subsetting.

Value

Another "fcwtr_scalogram" object that contains only part of the data.

Examples

```
fcwt(
  sin((1:5000) * 2 * pi * 440 / 44100),
  x_sample_freq = u(44100, "Hz"),
  y_sample_freq = u(44100, "Hz"),
  n_freqs = 10
)[1:1000, 2:7]
```

---

as.data.frame.fcwtr_scalogram

Convert the scalogram matrix to a data frame

Description

Internally, the scalogram resulting from `fcwt()` is represented by a numeric `matrix()`. This method converts this matrix into a `data.frame()` in long format, with one row per matrix entry. Note that this conversion has a significant run time cost.

Usage

```
## S3 method for class 'fcwtr_scalogram'
as.data.frame(x, ...)
```

Arguments

x	An "fcwtr_scalogram" object resulting from `fcwt()`.
...	Additional arguments are ignored.

Value

A `data.frame()` object representing the scalogram data with four columns:

time_index An integer index uniquely identifying time slices, starting at 0.

time The time of the time slice in physical units, relative to the beginning of the input signal (time slices that were cut off, e.g. by subsetting, are taken into account). The time unit (s, ms, us, or ns) is chosen depending on the magnitude of the values, and can be converted with `units::set_units()`.

freq The frequency, in the units chosen by the user for the `freq_begin` and `freq_end` arguments of `fcwt()`.

value The fCWT result for the particular time-frequency combination.

Examples

```
fcwt(
  sin((1:5000) * 2 * pi * 440 / 44100),
  x_sample_freq = u(44100, "Hz"),
  n_freqs = 10
) |>
  as.data.frame() |>
  head()
```

`as.matrix.fcwtr_scalogram`

Extract the data matrix from a scalogram

Description

Strips attributes and class from a scalogram object to retrieve a pure matrix.

Usage

```
## S3 method for class 'fcwtr_scalogram'
as.matrix(x, ...)
```

Arguments

`x`	An "fcwtr_scalogram" object resulting from `fcwt()`.
`...`	additional arguments to be passed to or from methods.

Value

A two dimensional numeric vector, inheriting the class "matrix".

Examples

```
options(max.print = 10)

fcwt(
  sin((1:5000) * 2 * pi * 440 / 44100),
  x_sample_freq = u(44100, "Hz"),
  n_freqs = 10
) |>
  as.matrix()
```

`as_tibble.fcwtr_scalogram` *Convert the scalogram matrix to a tibble*

Description

Internally, the scalogram resulting from `fcwt()` is represented by a numeric `matrix()`. This method converts this matrix into a `tibble::tibble()` in long format, with one row per matrix entry. Note that this conversion has a significant run time cost.

Usage

```
## S3 method for class 'fcwtr_scalogram'
as_tibble(x, ...)
```

Arguments

`x`	An "fcwtr_scalogram" object resulting from `fcwt()`.
`...`	Additional arguments are ignored.

Details

This method is conceptually identical to `as.data.frame()`, it simply returns a `tibble::tibble()` instead of a `data.frame()`.

Value

A `tibble::tibble()` object representing the scalogram data with four columns:

time_index An integer index uniquely identifying time slices, starting at 0.

time The time of the time slice in physical units, relative to the beginning of the input signal (time slices that were cut off, e.g. by subsetting, are taken into account). The time unit (s, ms, us, or ns) is chosen depending on the magnitude of the values, and can be converted with `units::set_units()`.

freq The frequency, in the units chosen by the user for the `freq_begin` and `freq_end` arguments of `fcwt()`.

value The fCWT result for the particular time-frequency combination.

Examples

```
library(tibble)

fcwt(
  sin((1:5000) * 2 * pi * 440 / 44100),
  x_sample_freq = u(44100, "Hz"),
  n_freqs = 10
) |>
  as_tibble()
```

`autoplot.fcwtr_scalogram` *Create a ggplot object resembling a scalogram*

Description

Create a ggplot object resembling a scalogram

Usage

```
## S3 method for class 'fcwtr_scalogram'
autoplot(
  object,
  n = 1000,
  time_unit = NULL,
  freq_unit = "Hz",
  transf = identity,
  ...
)
```

Arguments

`object`	A "fcwtr_scalogram" object resulting from `fcwt()`.
`n`	The plotting function reduces the time resolution by averaging to generate a reasonable graphics format. `n` is the number of time steps that are plotted. Defaults to `n = 1000`.
`time_unit`	A time unit that is used for the x-axis scale, e.g. `"s"` or `"ms"`. By default (`NULL`), a suitable unit (s, ms, us, or ns) is chosen depending on the length of the scalogram. See `units::valid_udunits()` and `units::valid_udunits_prefixes()` for valid expressions.
`freq_unit`	A frequency unit that is used for the y-axis scale. Defaults to "Hz" - "Hertz". See `units::valid_udunits()` and `units::valid_udunits_prefixes()` for valid expressions.
`transf`	A function, taking a vector and returning a vector, that is used to transform the scalogram values before plotting.
`...`	Additional arguments are ignored.

Value

A ggplot object.

Examples

```
library(ggplot2)

res <-
  fcwt(
    sin((1:4410) * 2 * pi * 440 / 44100),
    x_sample_freq = u(44100, "Hz"),
    freq_begin = u(50, "Hz"),
    freq_end = u(1000, "Hz"),
    n_freqs = 10,
    sigma = 5
  )

autoplot(res, time_unit = "ms")
```

`avx_enabled` *Check for AVX instruction set support*

Description

This function checks if fCWTr was compiled with AVX instruction set support. AVX instructions need to be supported by the users' CPU in order for this to work.

Usage

```
avx_enabled()
```

Details

By default, most compiler setups do not make use of AVX to increase portability of the binary. If you are an R user that has a CPU supporting AVX and want to make use of it, you might need to manually enable compiler flags to let R know about it, and install the package from source (so that it gets compiled on your machine).

Value

(`TRUE` | `FALSE`) Returns `TRUE` if AVX support is enabled, `FALSE` otherwise.

Examples

```
avx_enabled()
```

du *Drop physical unit*

Description

Drop units from a dimensionful quantity and retrieve a dimensionless number. The dimensionless number is defined by the ratio of `x` and one unit `u(1, unit)`.

Usage

```
du(x, unit = "1")
```

Arguments

`x`	A dimensionful numeric vector (S3 class "units"), e.g. generated with `u()` or subsequent calculations.
`unit`	A character indicating the unit. See `units::valid_udunits()` and `units::valid_udunits_prefixes(` for possible units.

Value

A dimensionless number defined by the ratio of `x` and one unit `u(1, unit)`.

See Also

`units::drop_units()`

Examples

```
x <- u(1, "km")

# retrieve a dimensionless number
# from x in units of "meter"
du(x, "m")

# drop the unit of a dimensionless "unit" quantity
du(x / x)
```

`fcwt` *Fast continuous wavelet transform*

Description

The core function of this package making use of the fCWT library. It processes an input signal in form of a real valued numeric vector interpreted as an evenly spaced time series and returns the absolute values of the continuous wavelet transform, i.e. a real valued positive matrix with a time and a frequency dimension.

Usage

```
fcwt(
  x,
  x_sample_freq,
  sigma = 2 * pi,
  y_sample_freq = 3/sigma_res(sigma, freq_end)$time,
  freq_begin = 2 * x_sample_freq/length(x),
  freq_end = x_sample_freq/2,
  n_freqs = 3 * ceiling(log(du(freq_end/freq_begin), base = 1 +
    sigma_freq_res_rel(sigma))),
  freq_scale = c("log", "linear"),
  rm_coi = TRUE,
  n_threads = 2L
)
```

Arguments

`x`	Real-valued time series. The time steps are assumed to be evenly spaced.
`x_sample_freq`	Sampling rate of input time series `x`. This number primarily establishes a connection to physical units which is used in other frequency definitions as well as the units of the output data. Expects either a value with frequency units, generated with `u()`, or a pure number, in which case it is interpreted in units of 'Hertz'. It does not need to be an integer.
`sigma`	Sets a dimensionless parameter Σ controlling the wavelet spread. Changing this parameter adjusts the time/frequency uncertainty balance, $\Delta t = 4\frac{\Sigma}{f}$, $\Delta f = 4\frac{f}{2\pi\Sigma}$. Larger (lower) value of sigma corresponds to a better (worse) frequency resolution and a worse (better) time resolution. For more information, see `vignette("sigma", package = "fCWTr")`. Defaults to 2π. Note that there is not really a natural choice for sigma, it depends on the use case. So the default choice can very well be quite a bad choice (it probably is for audio data). See `sigma_from_time_res()`, `sigma_from_freq_res()` and `sigma_from_freq_res_rel()` to determine sigma from resolution requirements.

`y_sample_freq` Sampling rate of output time series, in frequency units (see `u()`). The default value is set so that it aligns with the physical time resolution of the highest frequency modes. The maximum allowed sampling rate is the sampling rate of the input signal `x_sample_freq`. Since the output is generated by averaging over an integer number of input samples, the actual output sampling rate is `x_sample_freq / k`, where `k = floor(x_sample_freq / y_sample_freq)`. It is equal to or slightly higher than `y_sample_freq` and is stored in the result.

`freq_begin, freq_end` Optionally specifies the frequency range `[freq_begin, freq_end]` (both inclusive). If not specified the maximal meaningful frequency range, depending on the input signal, is taken: from `2 * x_sample_freq / length(x)` up to the Nyquist frequency `x_sample_freq / 2`. A frequency-valued number, generated with `u()`, or a pure number, that is interpreted in units of 'Hertz'.

`n_freqs` Number of frequency bins generated by the CWT. The frequencies are linearly or logarithmically distributed, depending on the `freq_scale` argument. Computation time increases when raising the number of frequency bins. The default number is chosen such that the frequency bandwidths are of the size of the physical frequency resolutions in case of a logarithmic scale. In some sense this value constitutes the "physical" limit. However, increasing `n_freqs` beyond this limit can still make sense: If we have a signal of a precise frequency, and the probing frequency band is not exactly aligned, the output signal will be dampened. In this case it can be useful to increase `n_freqs` and perform appropriate averaging afterwards: the output signal will show improved response.

`freq_scale` One of `"log"` (default) or `"linear"`: should the `n_freqs` frequencies be distributed logarithmically or linearly between `freq_begin` and `freq_end`? The default scale is logarithmic since differences on a logarithmic scale are proportional to the frequency uncertainties. In this sense, the logarithmic frequency scale is actually the natural scale for the continuous wavelet transform.

`rm_coi` `TRUE` (default) or `FALSE`. Boundary effects can result in nonphysical artifacts. If `rm_coi = TRUE`, those are effectively removed by setting corresponding values to `NA`. We define the essential support of the (Gaussian) wavelet at frequency f to be four times its standard deviation Σ/f, i.e. $\Delta t = 4\frac{\Sigma}{f}$, and so a wavelet touches the boundary if the distance of the center of the wavelet to the boundary is less than $\Delta t/2$. Values that fall into that range (the "cone of influence") are removed if `rm_coi = TRUE`.

`n_threads` Number of threads used by the computation, if supported by your platform. Defaults to 2 threads (to accommodate CRAN requirements). If `openmp_enabled()` returns `FALSE`, this argument is ignored, and only a single thread is used.

Details

The wavelet used in this calculation is the so called Morlet wavelet, a sinusoidal wave modulated by a Gaussian whose spread is controlled by the argument `sigma`.

See the original paper Arts, L.P.A., van den Broek, E.L. The fast continuous wavelet transformation (fCWT) for real-time, high-quality, noise-resistant time–frequency analysis. Nat Comput Sci 2, 47–58 (2022). doi:10.1038/s4358802100183z

Value

The scalogram, a numeric real-valued matrix with a time (rows) and a frequency (columns) dimension, curated with some additional properties. Its dimensions are `c(ceiling(length(x) / k), n_freqs)`, where `k` is the number of input samples averaged per output sample (see `y_sample_freq`; `k = 1` if `y_sample_freq = x_sample_freq`). The columns are ordered from the highest (`freq_end`) to the lowest (`freq_begin`) frequency.

This matrix is wrapped into a S3-class `fcwtr_scalogram` so that plotting and coercion functions can be used conveniently. Use `as.matrix()` to strip the curated information. Or use `as.data.frame()` to convert to another data format.

See Also

`fcwt_batch()`

Examples

```
ts_sin_440 <- sin((1:5000) * 2 * pi * 440 / 44100)

fcwt(
  ts_sin_440,
  x_sample_freq = u(44.1, "kHz"),
  sigma = 5,
  freq_begin = u(50, "Hz"),
  freq_end = u(1000, "Hz"),
  n_freqs = 10
)
```

`fcwt_batch` *Fast continuous wavelet transform - Batch processing*

Description

Performs a fast continuous wavelet transform on long sequences by sequentially processing chunks (batches) of the input signal. This is only useful for very long signals that cannot be processed by `fcwt()` in one go. It should not be used on short signals since boundary artefacts are automatically discarded (and those potentially dominate for short signals).

Usage

```
fcwt_batch(
  x,
  x_sample_freq,
  sigma = 2 * pi,
  y_sample_freq = 3/sigma_res(sigma, freq_end)$time,
  freq_begin = 2 * x_sample_freq/length(x),
  freq_end = x_sample_freq/2,
  n_freqs = 3 * ceiling(log(du(freq_end/freq_begin), base = 1 +
```

```
    sigma_freq_res_rel(sigma))),
  freq_scale = c("log", "linear"),
 max_batch_size = default_max_batch_size(x_sample_freq, y_sample_freq, n_freqs),
  n_threads = 2L,
  progress_bar = FALSE
)
```

Arguments

x	Real-valued time series. The time steps are assumed to be evenly spaced.
x_sample_freq	Sampling rate of input time series x. This number primarily establishes a connection to physical units which is used in other frequency definitions as well as the units of the output data. Expects either a value with frequency units, generated with u(), or a pure number, in which case it is interpreted in units of 'Hertz'. It does not need to be an integer.
sigma	Sets a dimensionless parameter Σ controlling the wavelet spread. Changing this parameter adjusts the time/frequency uncertainty balance, $\Delta t = 4\frac{\Sigma}{f}$, $\Delta f = 4\frac{f}{2\pi\Sigma}$. Larger (lower) value of sigma corresponds to a better (worse) frequency resolution and a worse (better) time resolution. For more information, see vignette("sigma", package = "fCWTr")). Defaults to 2π. Note that there is not really a natural choice for sigma, it depends on the use case. So the default choice can very well be quite a bad choice (it probably is for audio data). See sigma_from_time_res(), sigma_from_freq_res() and sigma_from_freq_res_rel() to determine sigma from resolution requirements.
y_sample_freq	Sampling rate of output time series, in frequency units (see u()). The default value is set so that it aligns with the physical time resolution of the highest frequency modes. The maximum allowed sampling rate is the sampling rate of the input signal x_sample_freq. Since the output is generated by averaging over an integer number of input samples, the actual output sampling rate is x_sample_freq / k, where k = floor(x_sample_freq / y_sample_freq). It is equal to or slightly higher than y_sample_freq and is stored in the result.
freq_begin, freq_end	Optionally specifies the frequency range [freq_begin, freq_end] (both inclusive). If not specified the maximal meaningful frequency range, depending on the input signal, is taken: from 2 * x_sample_freq / length(x) up to the Nyquist frequency x_sample_freq / 2. A frequency-valued number, generated with u(), or a pure number, that is interpreted in units of 'Hertz'.
n_freqs	Number of frequency bins generated by the CWT. The frequencies are linearly or logarithmically distributed, depending on the freq_scale argument. Computation time increases when raising the number of frequency bins. The default number is chosen such that the frequency bandwidths are of the size of the physical frequency resolutions in case of a logarithmic scale. In some sense this value constitutes the "physical" limit. However, increasing n_freqs beyond this limit can still make sense: If we have a signal of a precise frequency, and the probing frequency band is not exactly aligned, the output signal will be dampened. In

this case it can be useful to increase `n_freqs` and perform appropriate averaging afterwards: the output signal will show improved response.

`freq_scale` One of `"log"` (default) or `"linear"`: should the `n_freqs` frequencies be distributed logarithmically or linearly between `freq_begin` and `freq_end`? The default scale is logarithmic since differences on a logarithmic scale are proportional to the frequency uncertainties. In this sense, the logarithmic frequency scale is actually the natural scale for the continuous wavelet transform.

`max_batch_size` The maximal batch size (number of input samples) that is used for splitting up the input sequence. This limits the memory that is needed while processing a batch, which is roughly

$$50 + 16 \cdot \frac{n_{freqs}}{k}$$

bytes per input sample, where $k = x_{sample_freq}/y_{sample_freq}$ is the number of input samples that are averaged into one output sample: about 50 bytes are needed for the transform itself, the rest for the output of the batch (and a temporary copy of it). The default is the largest power of two for which this amounts to at most about 500 MB. (The Fourier transforms are fastest for powers of two, other sizes are zero-padded to the next power of two, which also increases the memory needed for the transform.)

Independent of the batch size, the outputs of all batches are kept and finally joined, which needs about twice the memory of the final result.

The actual batch size is the largest batch size that is not larger than `max_batch_size` and compatible with the requested `y_sample_freq`. Each batch loses about `4 * sigma * x_sample_freq / freq_begin` samples to boundary effects, so for low values of `freq_begin`, increase the batch size as far as your memory allows (see also the loss ratio reported with `progress_bar = TRUE`).

`n_threads` Number of threads used by the computation, if supported by your platform. Defaults to 2 threads (to accommodate CRAN requirements). If `openmp_enabled()` returns `FALSE`, this argument is ignored, and only a single thread is used.

`progress_bar` Monitoring progress can sometimes be useful when performing time consuming operations. Setting `progress_bar = TRUE` enables printing a progress bar to the console, printing the "loss ratio" and the number of batches. The loss ratio is a number between 0 and 1 and indicates how much of the batch computation has to be thrown away due to boundary artefacts. The higher the batch size the smaller the loss ratio will be. Defaults to `FALSE`.

Details

`fcwt()` computes the result frequency by frequency and averages over time on the fly (see its `y_sample_freq` argument), so the size of its output is usually not the limiting factor. However, it performs Fourier transforms over the *whole* input signal. This requires memory proportional to the signal length (several single-precision complex copies of the signal, zero-padded to the next power of two), and the signal length is limited to 2^{30} samples. For instance, a recording of 24 hours at a sampling rate of 44100 Hz has about $3.8 \cdot 10^9$ samples.

This function splits up the input sequence into overlapping batches, processes each batch separately with `fcwt()`, removes the boundary artefacts of each batch, and joins the outputs together.

Attention: In contrast to `fcwt()` boundary artefacts are automatically removed, so some information at the beginning and the end of the sequence is lost. (The amount depends on the lowest frequency `freq_begin`, since low-frequency wavelets are spread out the most in time.)

Value

The scalogram, a numeric real-valued matrix with dimensions roughly `dim ~ c(length(x) * y_sample_freq / x_sample_freq, n_freqs)`. The exact length of the output depends on boundary effect details. This matrix is wrapped into a S3-class "fcwtr_scalogram" so that plotting and coercion functions can be used conveniently.

See Also

`fcwt()`

Examples

```
fcwt_batch(
  ts_sin_sin,
  x_sample_freq = u(44.1, "kHz"),
  sigma = 10,
  y_sample_freq = u(100, "Hz"),
  freq_begin = u(100, "Hz"),
  freq_end = u(11000, "Hz"),
  n_freqs = 30
)
```

---

openmp_enabled *Check for OpenMP support*

Description

This function checks if fCWTr was compiled with OpenMP support. It is responsible for the multithreading capabilities of `fcwt()`. If OpenMP is not enabled the parameter `n_threads` of `fcwt()` is ignored.

Usage

```
openmp_enabled()
```

Details

OpenMP is used by fCWTr if the compiler supports it. In addition, the Fourier transforms are multithreaded only if the fftw library on your platform comes with OpenMP support (`libfftw3f_omp`); otherwise they run single-threaded.

On Windows, OpenMP support is disabled, since the fftw library provided by Rtools is compiled without OpenMP support.

When using pre-built package binaries, it depends on the (CRAN) build server whether or not these conditions are met. When installing the package from source, the installation script detects whether these conditions are met.

Value

(TRUE | FALSE) Returns TRUE if OpenMP support is enabled, FALSE otherwise.

Examples

```
openmp_enabled()
```

---

plot.fcwtr_scalogram *Scalogram plotting*

Description

Plots the scalogram resulting from fcwt(). Requires ggplot2.

Usage

```
## S3 method for class 'fcwtr_scalogram'
plot(x, n = 1000, time_unit = NULL, freq_unit = "Hz", transf = identity, ...)
```

Arguments

x	An "fcwtr_scalogram" object resulting from fcwt().
n	The plotting function reduces the time resolution by averaging to generate a reasonable graphics format. n is the number of time steps that are plotted. Defaults to n = 1000.
time_unit	A time unit that is used for the x-axis scale, e.g. "s" or "ms". By default (NULL), a suitable unit (s, ms, us, or ns) is chosen depending on the length of the scalogram. See units::valid_udunits() and units::valid_udunits_prefixes() for valid expressions.
freq_unit	A frequency unit that is used for the y-axis scale. Defaults to "Hz" - "Hertz". See units::valid_udunits() and units::valid_udunits_prefixes() for valid expressions.
transf	A function, taking a vector and returning a vector, that is used to transform the scalogram values before plotting.
...	Additional arguments are ignored.

Value

No return value, called for side effects.

Examples

```
ts_sin_440 <- sin((1:4410) * 2 * pi * 440 / 44100)

res <-
  fcwt(
    ts_sin_440,
    x_sample_freq = u(44100, "Hz"),
    freq_begin = u(50, "Hz"),
    freq_end = u(1000, "Hz"),
    n_freqs = 10,
    sigma = 5
  )

plot(res, time_unit = "ms")
```

---

print.fcwtr_scalogram *Print the scalogram*

Description

`print()` prints its argument and returns it invisibly (via `invisible(x)`).

Usage

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

Arguments

`x`	The "fcwtr_scalogram" object resulting from `fcwt()` to print.
`...`	further arguments passed to or from other methods.

Value

Returns the argument `x` invisibly.

`rbind.fcwtr_scalogram` *Combine scalograms in "time" direction*

Description

Given two or more scalograms with identical sampling frequencies, frequency scales and sigma, it can be useful to combine several into a single object creating a longer time series. The function errs if these conditions are not satisfied.

Usage

```
## S3 method for class 'fcwtr_scalogram'
rbind(...)
```

Arguments

`...`	One or more "fcwtr_scalogram" objects, generated by `fcwt()`.

Details

The scalograms are stitched together in chronological fashion (i.e. the first argument will be the initial piece, etc.). Time offset information is kept from the first piece.

Value

Returns a new time-wise combined "fcwtr_scalogram" object.

Examples

```
ts_sin_440 <- sin((1:5000) * 2 * pi * 440 / 44100)

res <-
  fcwt(
    ts_sin_440,
    x_sample_freq = u(44.1, "kHz"),
    freq_begin = u(50, "Hz"),
    freq_end = u(1000, "Hz"),
    n_freqs = 10,
    sigma = 5
  )

print(res)

# doubled scalogram
res_doubled <- rbind(res, res)

print(res_doubled)
```

sigma_freq_res_rel *Calculates Relative Frequency Resolution*

Description

For a given `sigma` this function calculates the expected relative frequency resolution respectively.

Usage

```
sigma_freq_res_rel(sigma)
```

Arguments

`sigma`	The dimensionless positive parameter in `fcwt()` controlling the wavelet spread.

Value

A number `p` representing the relative frequency resolution. I.e. at a given frequency `f`, the frequency resolution is `f*p`.

See Also

Other sigma: `sigma_from_freq_res()`, `sigma_from_freq_res_rel()`, `sigma_from_time_res()`, `sigma_res()`

Examples

```
sigma_freq_res_rel(1)
```

sigma_from_freq_res *Determine Sigma from a frequency resolution requirement*

Description

Setting the correct value for `sigma` in `fcwt()` for a particular use case is important. Sigma controls the wavelet spread used by the FCWT and so determines the time-/frequency resolution (depending on a given reference frequency).

Usage

```
sigma_from_freq_res(res, freq)
```

Arguments

res	The required frequency resolution in frequency units (see u()).
freq	The reference frequency where the required resolution must hold. Must be a quantity with frequency units (see u()).

Details

If you set a high frequency resolution, the time resolution of the output signal will suffer, and vice versa. Determining the correct balance for a particular use case can be crucial. This function determines the value of sigma for a particular frequency resolution requirement.

Value

A dimensionless number controlling the FCWT wavelet spread, corresponding to the sigma argument in fcwt().

See Also

Other sigma: sigma_freq_res_rel(), sigma_from_freq_res_rel(), sigma_from_time_res(), sigma_res()

Examples

```
ref_freq <- u(440, "Hz")
sigma <- sigma_from_freq_res(u(7, "Hz"), ref_freq)

sigma_res(sigma, ref_freq)
```

---

sigma_from_freq_res_rel

Determine Sigma from a relative frequency resolution requirement

---

Description

Setting the correct value for sigma in fcwt() for a particular use case is important. Sigma controls the wavelet spread used by the FCWT and so determines the time-/frequency resolution (depending on a given reference frequency).

Usage

```
sigma_from_freq_res_rel(res)
```

Arguments

res	The required relative frequency resolution, a number between 0 and 1.

Details

If you set a high frequency resolution, the time resolution of the output signal will suffer, and vice versa. Determining the correct balance for a particular use case can be crucial. This function determines the value of sigma for a particular relative frequency resolution requirement. Note that the relative frequency resolution does not depend on the frequency itself, in contrast to the absolute frequency resolution.

Value

A dimensionless number controlling the FCWT wavelet spread, corresponding to the `sigma` argument in `fcwt()`.

See Also

Other sigma: `sigma_freq_res_rel()`, `sigma_from_freq_res()`, `sigma_from_time_res()`, `sigma_res()`

Examples

```
ref_freq <- u(440, "Hz")
sigma <- sigma_from_freq_res_rel(0.2)

sigma_res(sigma, ref_freq)
```

---

sigma_from_time_res *Determine Sigma from a time resolution requirement*

Description

Setting the correct value for `sigma` in `fcwt()` for a particular use case is important. Sigma controls the wavelet spread used by the FCWT and so determines the time-/frequency resolution (depending on a given reference frequency).

Usage

```
sigma_from_time_res(res, freq)
```

Arguments

`res`	The required time resolution in time units (see `u()`).
`freq`	The reference frequency where the required resolution must hold. Must be a quantity with frequency units (see `u()`).

Details

If you set a high frequency resolution, the time resolution of the output signal will suffer, and vice versa. Determining the correct balance for a particular use case can be crucial. This function determines the value of sigma for a particular time resolution requirement.

Value

A dimensionless number controlling the FCWT wavelet spread, corresponding to the `sigma` argument in `fcwt()`.

See Also

Other sigma: `sigma_freq_res_rel()`, `sigma_from_freq_res()`, `sigma_from_freq_res_rel()`, `sigma_res()`

Examples

```
ref_freq <- u(440, "Hz")
sigma <- sigma_from_time_res(u(10, "ms"), ref_freq)

sigma_res(sigma, ref_freq)
```

`sigma_res` *Calculates Time/Frequency Resolution*

Description

For a given `sigma` and reference frequency `freq` this function calculates the expected time and frequency resolution respectively.

Usage

```
sigma_res(sigma, freq)
```

Arguments

`sigma`	The dimensionless positive parameter in `fcwt()` controlling the wavelet spread.
`freq`	The reference frequency where the frequency and time resolution are calculated. Must be a quantity with frequency units (see `u()`).

Details

Note that this resolution depends on the reference frequency, since the FCWT scales the wavelet spread depending on the frequency in a natural way. (in contrast to a simple fixed-windowed Fourier Transform).

Value

A named list with two elements: "time" and "freq", in time and frequency units. They obey the property $time \times freq = \frac{8}{\pi}$.

See Also

Other sigma: sigma_freq_res_rel(), sigma_from_freq_res(), sigma_from_freq_res_rel(), sigma_from_time_res()

Examples

```
sigma_res(1, u(440, "Hz"))
```

---

ts_piano_sample *A simple piano audio sample*

Description

A piano playing the note A4 (440 Hz). Its sample frequency is 11025 Hz.

Usage

```
ts_piano_sample
```

Format

A numeric vector of length 438461 containing a time series signal.

---

ts_sin_440 *Pure sine wave*

Description

Assuming a sample rate of 44100 Hz, the sine wave's frequency is 440 Hz, and the signal is 1 second long.

Usage

```
ts_sin_440
```

Format

A numeric vector of length 44100 containing a time series signal.

ts_sin_sin *A sinusoidal wave with varying frequency*

Description

The frequency itself is changing in a sinusoidal fashion. Assuming a sample rate of 44100 Hz, the signal is 1 second long and its frequency varies between about 50 Hz and 10 kHz.

Usage

```
ts_sin_sin
```

Format

A numeric vector of length 44100 containing a time series signal.

ts_sin_superpos *Superimposed sine waves of different frequencies and different amplitudes*

Description

Assuming a sample rate of 44100 Hz, the signal consists of two consecutive parts of 3000 samples each: superimposed sine waves of 440 Hz, 880 Hz and 100 Hz in the first part, and of 500 Hz, 1200 Hz and 50 Hz in the second part.

Usage

```
ts_sin_superpos
```

Format

A numeric vector of length 6000 containing a time series signal.

u *Augment physical unit*

Description

A wrapper around `units::set_units()` mostly to have a short name to create a dimensionful quantity.

Usage

```
u(x, unit = "1")
```

Arguments

x	A numeric vector to be augmented with a unit.
unit	A character indicating the unit. See `units::valid_udunits()` and `units::valid_udunits_prefixes(` for possible units.

See Also

`units::set_units()`

Examples

```
u(10, "km") # 10 kilometer
u(10, "mm") # 10 mm

# comparison taking into account the unit
u(100, "m") > u(1, "km")
```

Index

∗ **datasets**
ts_piano_sample, 22
ts_sin_440, 22
ts_sin_sin, 23
ts_sin_superpos, 23
∗ **sigma**
sigma_freq_res_rel, 18
sigma_from_freq_res, 18
sigma_from_freq_res_rel, 19
sigma_from_time_res, 20
sigma_res, 21
∗ **sys_support**
avx_enabled, 7
openmp_enabled, 14
[.fcwtr_scalogram, 2

as.data.frame(), *5*, *11*
as.data.frame.fcwtr_scalogram, 3
as.matrix(), *3*, *11*
as.matrix.fcwtr_scalogram, 4
as_tibble.fcwtr_scalogram, 5
autoplot.fcwtr_scalogram, 6
avx_enabled, 7

data.frame(), *3–5*
du, 8

fcwt, 9
fcwt(), *3–6*, *11*, *13–16*, *18–21*
fcwt_batch, 11
fcwt_batch(), *11*

matrix(), *3*, *5*

openmp_enabled, 14
openmp_enabled(), *10*, *13*

plot.fcwtr_scalogram, 15
print.fcwtr_scalogram, 16

rbind.fcwtr_scalogram, 17

sigma_freq_res_rel, 18
sigma_freq_res_rel(), *19–22*
sigma_from_freq_res, 18
sigma_from_freq_res(), *9*, *12*, *18*, *20–22*
sigma_from_freq_res_rel, 19
sigma_from_freq_res_rel(), *9*, *12*, *18*, *19*, *21*, *22*
sigma_from_time_res, 20
sigma_from_time_res(), *9*, *12*, *18–20*, *22*
sigma_res, 21
sigma_res(), *18–21*

tibble::tibble(), *5*
ts_piano_sample, 22
ts_sin_440, 22
ts_sin_sin, 23
ts_sin_superpos, 23

u, 24
u(), *8–10*, *12*, *19–21*
units::drop_units(), *8*
units::set_units(), *24*
units::valid_udunits(), *8*, *24*
units::valid_udunits_prefixes(), *8*, *24*