Package: SynchWave (via r-universe)

October 28, 2026

Version 1.1.2

Date 2022-05-02

Title Synchrosqueezed Wavelet Transform

Author Matlab original by Eugene Brevdo; R port by Dongik Jang, Hee-Seok Oh and Donghoh Kim

Maintainer Donghoh Kim <donghoh.kim@gmail.com>

Depends R (>= 2.13), fields (>= 6.7.6)

Description The synchrosqueezed wavelet transform is implemented. The package is a translation of MATLAB Synchrosqueezing Toolbox, version 1.1 originally developed by Eugene Brevdo (2012). The C code for curve_ext was authored by Jianfeng Lu, and translated to Fortran by Dongik Jang. Synchrosqueezing is based on the papers: [1] Daubechies, I., Lu, J. and Wu, H. T. (2011) Synchrosqueezed wavelet transforms: An empirical mode decomposition-like tool. Applied and Computational Harmonic Analysis, 30. 243-261. [2] Thakur, G., Brevdo, E., Fukar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. Signal Processing, 93, 1079-1094.

License LGPL (>= 2)

NeedsCompilation yes

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

Date/Publication 2022-05-07 10:20:02 UTC

RemoteUrl https://github.com/cran/SynchWave

RemoteRef HEAD

RemoteSha 9ac80b2c3e60b417d0ee5567a66e8e64faa1a979

Contents

curve_ext 2
curve_ext_multi 3

curve_ext_recon 5
cwt_fw 6
cwt_iw 8
est_riskshrink_thresh 9
fftshift 10
ifftshift 11
synsq_cwt_fw 12
synsq_cwt_iw 14
synsq_filter_pass 16
wfiltfn 18
wfilth 19

Index **21**

`curve_ext` *Extract a Maximum Energy / Minimum Curvature Curve*

Description

This function extracts a maximum energy / minimum curvature curve from Synchrosqueezed Representation. This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
curve_ext(Tx, fs, lambda=0)
```

Arguments

`Tx`	synchrosqueezed output of x (columns associated with time t)
`fs`	frequencies associated with rows of `Tx`
`lambda`	lambda should be greater than or equal to 0. Default: lambda=0

Details

This function extracts a maximum energy, minimum curvature, curve from Synchrosqueezed Representation. Note, energy is given as: `abs(Tx)^2`.

This implements the solution to Eq. (8) of [1].

Original Author: Jianfeng Lu

Value

`C`	the curve locations (indices)
`E`	the (logarithmic) energy of the curve

References

[1] Thakur, G., Brevdo, E., Fuckar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, **93**, 1079–1094.

See Also

`synsq_cwt_fw`, `curve_ext_multi`, `curve_ext_recon`.

`curve_ext_multi` *Extract a Maximum Energy / Minimum Curvature Curves*

Description

This function consecutively extracts maximum energy / minimum curvature curves from a synchrosqueezing representation. This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
curve_ext_multi(Tx, fs, nc, lambda = 1e3, clwin = 4)
```

Arguments

`Tx`	same as input to `curve_ext`. synchrosqueezed output of `x` (columns associated with time `t`)
`fs`	same as input to `curve_ext`. frequencies associated with rows of `Tx`
`nc`	Number of curves to extract
`lambda`	same as input to `curve_ext`. lambda default = 1e3.
`clwin`	frequency clearing window; after curve extraction, a window of frequencies `(Cs[,i]-clwin) : (Cs[,i]+clwin)` is removed from the original representation. (default = 2)

Details

This function consecutively extracts maximum energy / minimum curvature curves from a synchrosqueezing representation. As curves are extracted, their associated energies are zeroed out in the synsq representation. Curve extraction is then again performed on the remaining representaton.

For more details, see help `curve_ext` and Sec. IIID of [1].

Value

`Cs`	`[N x nc]` matrix of curve indices (`N=ncol(Tx)`)
`Es`	`[nc x 1]` vector of associated (logarithmic) energies

References

[1] Thakur, G., Brevdo, E., Fuckar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, **93**, 1079–1094.

See Also

synsq_cwt_fw, curve_ext, curve_ext_recon.

Examples

```
set.seed(7)
tt <- seq(0, 10, , 1024)
nv <- 32
f0 <- (1+0.6*cos(2*tt))*cos(4*pi*tt+1.2*tt^2)
sigma <- 0.5
f <- f0 + sigma*rnorm(length(tt))

# Continuous wavelet transform
opt <- list(type = "bump")
cwtfit <- cwt_fw(f, opt$type, nv, tt[2]-tt[1], opt)

# Hard thresholing
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0

# Denoised signal
opt$gamma <- thresh
fr <- cwt_iw(cwtfit$Wx, opt$type, opt)

# Synchrosqueezed wavelet transform using denoised signal
sstfit2 <- synsq_cwt_fw(tt, fr, nv, opt)

# Ridge extraction
lambda <- 1e+04
nw <- 16
imtfit <- curve_ext_multi(sstfit2$Tx, log2(sstfit2$fs), 1, lambda, nw)

# Reconstruction
curvefit <- curve_ext_recon(sstfit2$Tx, sstfit2$fs, imtfit$Cs, opt, nw)

par(mfrow=c(2,1))
image.plot(list(x=tt, y=sstfit2$fs, z=t(abs(sstfit2$Tx))), log="y",
    xlab="Time", ylab="Frequency", main="Time-Frequency Representation by SST",
    col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=c(0.5, 25))
lines(tt, sstfit2$fs[imtfit$Cs[,1]], col="red", lty=3, lwd=2)

plot(tt, f0, type="l")
lines(tt, curvefit, lty=2)
```

curve_ext_recon *Reconstruct Curves*

Description

This function reconstructs the curves given by `curve_ext` or `curve_ext_multi`.

This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
curve_ext_recon(Tx, fs, Cs, opt=NULL, clwin=4)
```

Arguments

`Tx`	same as input to `curve_ext_multi`. synchrosqueezed output of `x` (columns associated with time `t`)
`fs`	same as input to `curve_ext_multi`. frequencies associated with rows of `Tx`
`opt`	same as input to `synsq_cwt_fw` and `synsq_cwt_iw`
`Cs`	same as output to `curve_ext_multi`. `[N x nc]` matrix of curve indices (`N=ncol(Tx)`)
`clwin`	same as input to `curve_ext_multi`. frequency clearing window; after curve extraction, a window of frequencies `(Cs[,i]-clwin) : (Cs[,i]+clwin)` is removed from the original representation. (default = 2)

Details

This function reconstructs the curves given by `curve_ext` or `curve_ext_multi`.

Value

`xrs`	`[n x Nc]` matrix - the `Nc` reconstructed signals (each length `n`)

References

[1] Thakur, G., Brevdo, E., Fuckar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, **93**, 1079–1094.

See Also

`synsq_cwt_fw`, `curve_ext`, `curve_ext_multi`, `curve_ext_recon`.

Examples

```
set.seed(7)
tt <- seq(0, 10, , 1024)
nv <- 32
f0 <- (1+0.6*cos(2*tt))*cos(4*pi*tt+1.2*tt^2)
sigma <- 0.5
f <- f0 + sigma*rnorm(length(tt))

# Continuous wavelet transform
opt <- list(type = "bump")
cwtfit <- cwt_fw(f, opt$type, nv, tt[2]-tt[1], opt)

# Hard thresholing
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0

# Denoised signal
opt$gamma <- thresh
fr <- cwt_iw(cwtfit$Wx, opt$type, opt)

# Synchrosqueezed wavelet transform using denoised signal
sstfit2 <- synsq_cwt_fw(tt, fr, nv, opt)

# Ridge extraction
lambda <- 1e+04
nw <- 16
imtfit <- curve_ext_multi(sstfit2$Tx, log2(sstfit2$fs), 1, lambda, nw)

# Reconstruction
curvefit <- curve_ext_recon(sstfit2$Tx, sstfit2$fs, imtfit$Cs, opt, nw)

par(mfrow=c(2,1))
image.plot(list(x=tt, y=sstfit2$fs, z=t(abs(sstfit2$Tx))), log="y",
    xlab="Time", ylab="Frequency", main="Time-Frequency Representation by SST",
    col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=c(0.5, 25))
lines(tt, sstfit2$fs[imtfit$Cs[,1]], col="red", lty=3, lwd=2)

plot(tt, f0, type="l")
lines(tt, curvefit, lty=2)
```

---

cwt_fw *Forward Continuous Wavelet Transform*

Description

This function performs forward continuous wavelet transform, discretized, as described in Sec. 4.3.3 of [1] and Sec. IIIA of [2]. This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
cwt_fw(x, type, nv, dt, opt)
```

Arguments

x	input signal vector, length n (need not be dyadic length)
type	wavelet type, string (see wfiltfn)
nv	number of voices (suggest nv >= 32)
dt	sampling period (default, dt = 1)
opt	list of options. opt$padtype: type of padding, 'symmetric', 'replicate', or 'circular' (default = 'symmetric'). opt$rpadded: return padded Wx and dWx? (default = 0). opts, optmu, etc: wavelet options (see wfiltfn).

Details

This function performs forward continuous wavelet transform, discretized, as described in Sec. 4.3.3 of [1] and Sec. IIIA of [2]. This algorithm uses the FFT and samples the wavelet atoms in the fourier domain. Options such as padding of the original signal are allowed. Returns the vector of scales and, if requested, the analytic time-derivative of the wavelet transform (as described in Sec. IIIB of [2].

Value

Wx	[na x n] size matrix (rows = scales, cols = times) containing samples of the CWT of x.
asc	na length vector containing the associated scales.
dWx	[na x n] size matrix containing samples of the time-derivatives of the CWT of x.

References

[1] Mallat, S (2009) *A Wavelet Tour of Signal Processing*, New York: Academic Press.

[2] Thakur, G., Brevdo, E., Fuckar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, **93**, 1079–1094.

See Also

cwt_iw, wfiltfn, est_riskshrink_thresh.

Examples

```
tt <- seq(0, 10, , 1024)
f0 <- (1+0.6*cos(2*tt))*cos(4*pi*tt+1.2*tt^2)
sigma <- 0.5
f <- f0 + sigma*rnorm(length(tt))

# Continuous wavelet transform
```

```
nv <- 32
opt <- list(type = "bump")
cwtfit <- cwt_fw(f, opt$type, nv, tt[2]-tt[1], opt)
```

---

cwt_iw *Inverse Wavelet Transform*

Description

This function performs the inverse wavelet transform of signal Wx.

This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
cwt_iw(Wx, type, opt=NULL)
```

Arguments

Wx	wavelet transform of a signal, see cwt_fw.
type	wavelet used to take the wavelet transform, see cwt_fw and wfiltfn.
opt	options list used for forward wavelet transform.

Details

This function performs the inverse wavelet transform of signal Wx, and implements Eq. (4.67) of [1].

Value

x	the signal, as reconstructed from Wx.

References

[1] Mallat, S (2009) *A Wavelet Tour of Signal Processing*, New York: Academic Press.

See Also

cwt_fw, wfiltfn, est_riskshrink_thresh.

Examples

```
tt <- seq(0, 10, , 1024)
nv <- 32
f0 <- (1+0.6*cos(2*tt))*cos(4*pi*tt+1.2*tt^2)
sigma <- 0.5
f <- f0 + sigma*rnorm(length(tt))

# Continuous wavelet transform
opt <- list(type = "bump")
cwtfit <- cwt_fw(f, opt$type, nv, tt[2]-tt[1], opt)

# Hard thresholing
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0

# Reconstruction
opt$gamma <- thresh
cwtrec <- cwt_iw(cwtfit$Wx, opt$type, opt)

par(mfrow=c(1,1))
plot(tt, f, type="p", lty=2, xlab="time", ylab="f", col="red", cex=0.1)
lines(tt, f0, col="blue")
lines(tt, cwtrec)
```

---

est_riskshrink_thresh *Estimate the RiskShrink Hard Thresholding Level*

Description

This function estimates the RiskShrink hard thresholding level.

This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
est_riskshrink_thresh(Wx, nv)
```

Arguments

Wx	wavelet transform of a signal, see cwt_fw
nv	number of voices

Details

This function implements Defn. 1 of Sec. 2.4 in [1], using the suggested noise estimator from the discussion "Estimating the noise level" in that same section.

Value

the RiskShrink hard threshold estimate

References

[1] Donoho, D. L. and Johnstone, I. M. (1994) Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, **81**, 425–455.

See Also

cwt_fw, cwt_iw.

Examples

```
tt <- seq(0, 10, , 1024)
nv <- 32
f0 <- (1+0.6*cos(2*tt))*cos(4*pi*tt+1.2*tt^2)
sigma <- 0.5
f <- f0 + sigma*rnorm(length(tt))

# Continuous wavelet transform
opt <- list(type = "bump")
cwtfit <- cwt_fw(f, opt$type, nv, tt[2]-tt[1], opt)

# Hard thresholing
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0
```

---

fftshift *FFT Shift*

Description

This function exchanges the left halves of a vector with the right halves.

Usage

```
fftshift(x)
```

Arguments

x	a vector

Details

This function exchanges the left halves of a vector with the right halves. This function is adapted from Matlab.

Value

shifted vector

See Also

ifftshift.

Examples

```
x <- 1:4
fftshift(fftshift(x))
ifftshift(fftshift(x))

x <- 1:5
fftshift(fftshift(x))
ifftshift(fftshift(x))
```

---

ifftshift *Inverse FFT Shift*

Description

This function exchanges the left halves of a vector with the right halves.

Usage

```
ifftshift(x)
```

Arguments

x	a vector

Details

This function exchanges the left halves of a vector with the right halves. This function is adapted from Matlab.

Value

shifted vector

See Also

fftshift.

Examples

```
x <- 1:4
fftshift(fftshift(x))
ifftshift(fftshift(x))

x <- 1:5
fftshift(fftshift(x))
ifftshift(fftshift(x))
```

`synsq_cwt_fw` *Synchrosqueezing Transform*

Description

This function calculates the synchrosqueezing transform.

This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
synsq_cwt_fw(tt, x, nv=16, opt=NULL)
```

Arguments

`tt`	vector of times samples are taken (e.g. `seq(0, 1,length=2)`)
`x`	vector of signal samples (e.g. `x = cos(20*pi*tt)`)
`nv`	number of voices (default: 16, recommended: 32 or 64)
`opt`	list of options. `opt.type`: type of wavelet (see `wfiltfn`), `opt$s`, `opt$mu`, etc (wavelet parameters: see `opt` from `wfiltfn`), `opt.disp`: display debug information?, `opt.gamma`: wavelet hard thresholding value (see `cwt_freq_direct`, default: sqrt(machine epsilon)) `opt.dtype`: direct or numerical differentiation (default: 1, uses direct). if `dtype=0`, uses differentiation algorithm instead (see `cwt_freq`), which is faster and uses less memory, but may be less accurate.

Details

This function calculates the synchrosqueezing transform of vector `x`, with samples taken at times given in vector `tt`. Uses `nv` voices. `opt` is a struct of options for the way synchrosqueezing is done, the type of wavelet used, and the wavelet parameters. This implements the algorithm described in Sec. III of [1].

Note that `Wx` itself is not thresholded by `opt$gamma`. The instantaneoues frequency `w` is calculated using `Wx` by `cwt_freq_direct` and `cwt_freq`. After the calculation, instantaneoues frequency `w` is treated as `NA` where `abs(Wx) < opt$gamma`.

Value

Tx	synchrosqueezed output of x (columns associated with time tt)
fs	frequencies associated with rows of Tx
Wx	wavelet transform of x (see cwt_fw)
asc	scales associated with rows of Wx
w	instantaneous frequency estimates for each element of Wx (see cwt_freq_direct)

References

[1] Thakur, G., Brevdo, E., Fuckar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, **93**, 1079–1094.

See Also

cwt_fw, est_riskshrink_thresh, wfiltfn.

Examples

```
tt <- seq(0, 10, , 1024)
nv <- 32
f0 <- (1+0.6*cos(2*tt))*cos(4*pi*tt+1.2*tt^2)
sigma <- 0.5
f <- f0 + sigma*rnorm(length(tt))

# Continuous wavelet transform
opt <- list(type = "bump")
cwtfit <- cwt_fw(f, opt$type, nv, tt[2]-tt[1], opt)

# Hard thresholing
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0

# Reconstruction
opt$gamma <- thresh
#[1] 0.0593984
#opt$gamma <- 10^-5
cwtrec <- cwt_iw(cwtfit$Wx, opt$type, opt)

par(mfrow=c(1,1))
plot(tt, f, type="p", lty=2, xlab="time", ylab="f", col="red", cex=0.1)
lines(tt, f0, col="blue")
lines(tt, cwtrec)

# Synchrosqueezed wavelet transform
sstfit <- synsq_cwt_fw(tt, f, nv, opt)

#par(mfrow=c(2,2))
#plot(tt, f, type="p", lty=2, xlab="time", ylab="f", col="red", cex=0.1)
#lines(tt, f0, col="blue")
```

```
#image.plot(list(x=tt, y=sstfit$asc, z=t(abs(sstfit$Wx))), log="y",
#    xlab="Time", ylab="Scale", main="Time-Scale Representation by CWT",
#   col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=rev(range(sstfit$asc)))
#image.plot(list(x=tt, y=sstfit$fs, z=t(abs(sstfit$Tx))), log="y",
#    xlab="Time", ylab="Frequency", main="Time-Frequency Representation by SST",
#    col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=c(0.5, 25))
#image.plot(list(x=tt, y=sstfit$asc, z=t(sstfit$w)), log="y",
#    xlab="Time", ylab="Scale", main="Instantaneous Frequency",
#   col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=rev(range(sstfit$asc)))
```

`synsq_cwt_iw` *Invese Synchrosqueezing Transform*

Description

This function performs inverse synchrosqueezing transform.

This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
synsq_cwt_iw(Tx, fs, opt=NULL)
```

Arguments

`Tx`	synchrosqueezed output of `x` (columns associated with time `t`)
`fs`	frequencies associated with rows of `Tx`
`opt`	list of options. See `synsq_cwt_fw`. `opt.type`: type of wavelet used in `synsq_cwt_fw`, other wavelet options (`opt.mu`, `opt.s`) should also match those used in `synsq_cwt_fw`

Details

This function performs inverse synchrosqueezing transform of `Tx` with associated frequencies in `fs`. This implements Eq. 5 of [1].

Value

`x`	reconstructed signal

References

[1] Thakur, G., Brevdo, E., Fuckar, N. S. and Wu, H-T. (2013) The Synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications. *Signal Processing*, **93**, 1079–1094.

See Also

`synsq_cwt_fw`, `synsq_filter_pass`.

Examples

```
set.seed(7)
n <- 2048
tu <- seq(0,10,, n)
dt <- tu[2]-tu[1]

feq1 <- function(t) (1+0.2*cos(t))*cos(2*pi*(2*t+0.3*cos(t)))
feq2 <- function(t) (1+0.3*cos(2*t))*exp(-t/15)*cos(2*pi*(2.4*t+0.5*t^(1.2)+0.3*sin(t)))
feq3 <- function(t) cos(2*pi*(5.3*t-0.2*t^(1.3)))
feq <- function(t) feq1(t) + feq2(t) + feq3(t)
s2 <- 2.4
noise <- sqrt(s2)*rnorm(length(tu))

fu0 <- feq(tu);
fu <- fu0 + noise;
fus <- cbind(feq1(tu), feq2(tu), feq3(tu))

# Continuous wavelet transform
nv <- 32
opt <- list(type = "bump")

cwtfit <- cwt_fw(fu, opt$type, nv, dt, opt)
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)

# Hard thresholding and Reconstruction
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0
fur <- cwt_iw(cwtfit$Wx, opt$type, opt)

# Synchrosqueezed wavelet transform using denoised signal
sstfit <- synsq_cwt_fw(tu, fur, nv, opt)

#par(mfrow=c(1,2))
#image.plot(list(x=tu, y=sstfit$fs, z=t(abs(sstfit$Tx))), log="y",
#    xlab="Time", ylab="Frequency", main="Time-Frequency Representation by SST",
#    col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=c(1, 8))

# Extracting the second component by filtering of synchrosqueezed wavelet transform
fm <- fM <- (2.4+0.5*1.2*tu^0.2+0.3*cos(tu))

#lines(tu, 0.88*fm, col="red", lty=3, lwd=2)
#lines(tu, 1.22*fM, col="red", lty=3, lwd=2)

tmp <- synsq_filter_pass(sstfit$Tx, sstfit$fs, 0.88*fm, 1.12*fM);
fursst <- synsq_cwt_iw(tmp$Txf, w, opt);

#plot(tu, fursst, type="l", main="SST", xlab="time", ylab="f", col="red",
#    xlim=c(1.5,8.5), ylim=c(-1,1))
#lines(tu, feq2(tu), col="blue")

# Example:
# tmp <- synsq_cwt_fw(t, x, 32)  # Synchrosqueezing
# Txf <- synsq_filter_pass(tmp$Tx, tmp$fs, -Inf, 1) # Pass band filter
```

```
# xf <- synsq_cwt_iw(Txf, fs)  # Filtered signal reconstruction
```

`synsq_filter_pass` *Filtering of the Synchrosqueezing Representation*

Description

This function filters the Synchrosqueezing representation `Tx`, having associated frequencies fs (see `synsq_cwt_fw`). This band-pass filter keeps frequencies in the range `[fm, fM]`. This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~

Usage

```
synsq_filter_pass(Tx, fs, fm, fM)
```

Arguments

`Tx`	synchrosqueezed output of `x` (columns associated with time `t`). output of `synsq_cwt_fw`
`fs`	frequencies associated with rows of `Tx`
`fm`	Minimum band pass values. scalars, or vectors with each value associated with a time index (`length(fm)=ncol(Tx)`)
`fM`	Maximum band pass values. scalars, or vectors with each value associated with a time index (`length(fM)=ncol(Tx)`)

Details

This function filters the Synchrosqueezing representation `Tx`, having associated frequencies `fs` (see `synsq_cwt_fw`). This band-pass filter keeps frequencies in the range `[fm, fM]`.

Value

`Txf`	Filtered version of `Tx` (same size), with zeros outside the pass band rows.
`fmi`	time-length vector of min-frequency row indices
`fMi`	time-length vector of max-frequency row indices

See Also

`synsq_cwt_fw`, `synsq_cwt_fw`.

Examples

```
set.seed(7)
n <- 2048
tu <- seq(0,10,, n)
dt <- tu[2]-tu[1]

feq1 <- function(t) (1+0.2*cos(t))*cos(2*pi*(2*t+0.3*cos(t)))
feq2 <- function(t) (1+0.3*cos(2*t))*exp(-t/15)*cos(2*pi*(2.4*t+0.5*t^(1.2)+0.3*sin(t)))
feq3 <- function(t) cos(2*pi*(5.3*t-0.2*t^(1.3)))
feq <- function(t) feq1(t) + feq2(t) + feq3(t)
s2 <- 2.4
noise <- sqrt(s2)*rnorm(length(tu))

fu0 <- feq(tu);
fu <- fu0 + noise;
fus <- cbind(feq1(tu), feq2(tu), feq3(tu))

# Continuous wavelet transform
nv <- 32
opt <- list(type = "bump")

cwtfit <- cwt_fw(fu, opt$type, nv, dt, opt)
thresh <- est_riskshrink_thresh(cwtfit$Wx, nv)

# Hard thresholding and Reconstruction
cwtfit$Wx[which(abs(cwtfit$Wx) < thresh)] <- 0.0
fur <- cwt_iw(cwtfit$Wx, opt$type, opt)

# Synchrosqueezed wavelet transform using denoised signal
sstfit <- synsq_cwt_fw(tu, fur, nv, opt)

#par(mfrow=c(2,2))
#image.plot(list(x=tu, y=sstfit$asc, z=t(abs(sstfit$Wx))), log="y",
#    xlab="Time", ylab="Scale", main="Time-Scale Representation by CWT",
#    col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=c(1, 0.0625))

# Extracting the second component by filtering of continuous wavelet transform
am <- 0.2 * rep(1, length(tu))
aM <- 0.3 * rep(1, length(tu))

#lines(tu, am, col="red", lty=3, lwd=2)
#lines(tu, aM, col="red", lty=3, lwd=2)

tmp <- synsq_filter_pass(sstfit$Wx, sstfit$asc, am, aM);
furcwt <- cwt_iw(tmp$Txf, opt$type, opt);

#image.plot(list(x=tu, y=sstfit$fs, z=t(abs(sstfit$Tx))), log="y",
#    xlab="Time", ylab="Frequency", main="Time-Frequency Representation by SST",
#    col=designer.colors(64, c("azure", "cyan", "blue", "darkblue")), ylim=c(1, 8))

# Extracting the second component by filtering of synchrosqueezed wavelet transform
fm <- fM <- (2.4+0.5*1.2*tu^0.2+0.3*cos(tu))
```

```
#lines(tu, 0.88*fm, col="red", lty=3, lwd=2)
#lines(tu, 1.22*fM, col="red", lty=3, lwd=2)

tmp <- synsq_filter_pass(sstfit$Tx, sstfit$fs, 0.88*fm, 1.12*fM);
fursst <- synsq_cwt_iw(tmp$Txf, w, opt);

#plot(tu, fursst, type="l", main="SST", xlab="time", ylab="f", col="red",
#    xlim=c(1.5,8.5), ylim=c(-1,1))
#lines(tu, feq2(tu), col="blue")

#plot(tu, furcwt, type="l", main="CWT", xlab="time", ylab="f", col="red",
#    xlim=c(1.5,8.5), ylim=c(-1,1))
#lines(tu, feq2(tu), col="blue")

# Remove all energy for normalized frequencies above 1.
# synsq_filter_pass(Tx, fs, -Inf, 1)
```

`wfiltfn` *Wavelet Transform Function of the Wavelet Filter*

Description

This function provides wavelet transform function of the wavelet filter in question, fourier domain. This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
wfiltfn(type, opt)
```

Arguments

`type`	wavelet type. 'gauss', 'mhat', 'cmhat', 'morlet', 'shannon', 'hshannon', 'hhat', 'hhhat', 'bump'
`opt`	list of options for wavelet type. For 'gauss', `s` and `mu` with default 1/6 and 2. For 'mhat', `s` with default 1. For 'cmhat', `s` and `mu` with default 1 and 1. For 'morlet', `mu` with default `2*pi`. For 'hhhat', `mu` with default 5. For 'bump', `s` and `mu` with default 1 and 5. The wavelet types 'mhat', 'shannon' and 'hhat' are not used for synchrosqueezed transform. The wavelet types 'shannon' and 'hshannon' are NOT recommended for analysis.

Details

This function provides wavelet transform function of the wavelet filter in question, fourier domain.

Value

wavelet transform function

See Also

wfilth, cwt_fw, cwt_iw.

Examples

```
psihfn <- wfiltfn("bump", list(mu=1, s=.5))
plot(psihfn(seq(-5, 5, by=0.01)), type="l", xlab="", ylab="")
```

---

wfilth *FFT of Wavelet Transform Function*

Description

This function outputs the FFT of the wavelet. This code is translated from MATLAB Synchrosqueezing Toolbox, version 1.1 developed by Eugene Brevdo (http://www.math.princeton.edu/~ebrevdo/).

Usage

```
wfilth(type, N, a, opt)
```

Arguments

type	wavelet type. See wfiltfn. 'gauss', 'mhat', 'cmhat', 'morlet', 'shannon', 'hshannon', 'hhat', 'hhhat', 'bump'
N	number of samples to calculate
a	wavelet scale parameter (default = 1)
opt	list of options for wavelet type. See wfiltfn. opt.dt: sampling period, default = 1.

Details

This function outputs the FFT of the wavelet of family 'type' with parameters in 'opt', of length N at scale a: (psi(-t/a)).

Note that the output is made so that the inverse fft of the result is zero-centered in time. This is important for convolving with the derivative(dpsih). To get the correct output, perform an ifftshift. That is,

psi = ifftshift(fft(psih, inverse=TRUE) / length(psih)),

xfilt = ifftshift(fft(fft(x) * psih, inverse=TRUE) / length(fft(x) * psih))

Value

psih	wavelet sampling in frequency domain (for use in fft)
dpsih	derivative of same wavelet, sampled in frequency domain (for fft)
xi	associated fourier domain frequencies of the samples.

See Also

wfiltfn, cwt_fw, cwt_iw.

Examples

```
tmp <- wfilth("morlet", 1024, 4)
plot(fftshift(tmp$xi/(2*pi)), fftshift(abs(tmp$psih)), type="l", col="blue", xlab="", ylab="")
```

Index

∗ **nonparametric**
curve_ext, 2
curve_ext_multi, 3
curve_ext_recon, 5
cwt_fw, 6
cwt_iw, 8
est_riskshrink_thresh, 9
fftshift, 10
ifftshift, 11
synsq_cwt_fw, 12
synsq_cwt_iw, 14
synsq_filter_pass, 16
wfiltfn, 18
wfilth, 19

curve_ext, 2, *4, 5*
curve_ext_multi, 3, *3*, *5*
curve_ext_recon, *3, 4*, 5, *5*
cwt_fw, 6, *8*, *10*, *13*, *19, 20*
cwt_iw, *7*, 8, *10*, *19, 20*

est_riskshrink_thresh, *7, 8*, 9, *13*

fftshift, 10, *11*

ifftshift, 11, *11*

synsq_cwt_fw, *3–5*, 12, *14*, *16*
synsq_cwt_iw, 14
synsq_filter_pass, *14*, 16

wfiltfn, *7, 8*, *13*, 18, *20*
wfilth, 19, *19*