Package: lowpassFilter (via r-universe)

September 12, 2024

Title Lowpass Filtering

Version 1.0-2

Depends R (>= 3.0.0)

Imports Rcpp (>= 0.12.3), stats, methods

LinkingTo Rcpp

Suggests testthat

Description Creates lowpass filters which are commonly used in ion channel recordings. It supports generation of random numbers that are filtered, i.e. follow a model for ion channel recordings, see <doi:10.1109/TNB.2018.2845126>. Furthermore, time continuous convolutions of piecewise constant signals with the kernel of lowpass filters can be computed.

License GPL-3

Encoding UTF-8

NeedsCompilation yes

Author Pein Florian [aut, cre], Thomas Hotz [ctb], Inder Tecuapetla-Gómez [ctb]

Maintainer Pein Florian <f.pein@lancaster.ac.uk>

Date/Publication 2022-04-29 18:40:02 UTC

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

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

RemoteRef HEAD

RemoteSha 09d0a4b50616b808215031a4a757eb3819ad57ed

Contents

lowpassFilter-package 2
convolve 3
deconvolve 4
helpFunctionsFilter 5
lowpassFilter 7
randomGeneration 9

Index 13

`lowpassFilter-package` *Lowpass Filtering*

Description

Creates lowpass filters and offers further functionalities around them. Lowpass filters are commonly used in ion channel recordings.

Details

The main function of this package is `lowpassFilter` which creates lowpass filters, currently only Bessel filters are supported. `randomGeneration` and `randomGenerationMA` allow to generate random numbers that are filtered, i.e. follow a model for ion channel recordings, see *(Pein et al., 2018, 2020)*. `getConvolution`, `getConvolutionJump`, and `getConvolutionPeak` allow to compute the convolution of a signal with the kernel of a lowpass filter.

References

Pein, F., Bartsch, A., Steinem, C., and Munk, A. (2020) Heterogeneous idealization of ion channel recordings - Open channel noise. Submitted.

Pein, F., Tecuapetla-Gómez, I., Schütte, O., Steinem, C., Munk, A. (2018) Fully-automatic multiresolution idealization for filtered ion channel recordings: flickering event detection. IEEE Trans. Nanobioscience, 17(3):300-320.

Pein, F. (2017) Heterogeneous Multiscale Change-Point Inference and its Application to Ion Channel Recordings. PhD thesis, Georg-August-Universität Göttingen. http://hdl.handle.net/11858/00-1735-0000-002E-E34A-7.

Hotz, T., Schütte, O., Sieling, H., Polupanow, T., Diederichsen, U., Steinem, C., and Munk, A. (2013) Idealizing ion channel recordings by a jump segmentation multiresolution filter. IEEE Trans. Nanobioscience, 12(4):376-386.

See Also

`lowpassFilter`, `randomGeneration`, `randomGenerationMA`, `getConvolution`, `getConvolutionJump`, `getConvolutionPeak`

Examples

```
# creates a lowpass filter
filter <- lowpassFilter(type = "bessel", param = list(pole = 4, cutoff = 0.1), sr = 1e4)
time <- 1:4000 / filter$sr

# creates a piecewise constant signal with a single peak
stepfun <- getSignalPeak(time, cp1 = 0.2, cp2 = 0.2 + 3 / filter$sr,
                         value = 20, leftValue = 40, rightValue = 40)

# computes the convolution of the signal with the kernel of the lowpass filter
```

```
signal <- getConvolutionPeak(time, cp1 = 0.2, cp2 = 0.2 + 3 / filter$sr,
                             value = 20, leftValue = 40, rightValue = 40,
                             filter = filter)

# generates random numbers that are filtered
data <- randomGenerationMA(n = 4000, filter = filter, signal = signal, noise = 1.4)

# generated data
plot(time, data, pch = 16)

# zoom into the single peak
plot(time, data, pch = 16, xlim = c(0.199, 0.202), ylim = c(19, 45))
lines(time, stepfun, col = "blue", type = "s", lwd = 2)
lines(time, signal, col = "red", lwd = 2)

# use of data randomGeneration instead
data <- randomGeneration(n = 4000, filter = filter, signal = signal, noise = 1.4)

# similar result
plot(time, data, pch = 16, xlim = c(0.199, 0.202), ylim = c(19, 45))
lines(time, stepfun, col = "blue", type = "s", lwd = 2)
lines(time, signal, col = "red", lwd = 2)
```

convolve *Time discrete convolution*

Description

For developers only; computes a time discrete convolution.

Usage

```
.convolve(val, kern)
```

Arguments

`val`	a numeric vector giving the values
`kern`	a numeric vector giving the time discrete kernel

Value

A numeric vector giving the convolution.

See Also

lowpassFilter

deconvolve *Deconvolution of a single jump / isolated peak*

Description

For developers only; computes the deconvolution of a single jump or an isolated peak assuming that the observations are lowpass filtered. More details are given in *(Pein et al., 2018)*.

Usage

```
.deconvolveJump(grid, observations, time, leftValue, rightValue,
                typeFilter, inputFilter, covariances)
.deconvolvePeak(gridLeft, gridRight, observations, time, leftValue, rightValue,
                typeFilter, inputFilter, covariances, tolerance)
```

Arguments

`grid`, `gridLeft`, `gridRight`
: numeric vectors giving the potential time points of the single jump, of the left and right jump points of the peak, respectively

`observations`
: a numeric vector giving the observed data

`time`
: a numeric vector of length `length(observations)` giving the time points at which the `observations` are observed

`leftValue`, `rightValue`
: single numerics giving the value (conductance level) before and after the jump / peak, respectively

`typeFilter`, `inputFilter`
: a description of the assumed lowpass filter, usually computed by `lowpassFilter`

`covariances`
: a numeric vector giving the (regularized) covariances of the observations

`tolerance`
: a single numeric giving a tolerance for the decision whether the left jump point is smaller than the right jump point

Value

For `.deconvolveJump` a single numeric giving the jump point. For `.deconvolvePeak` a list containing the entries `left`, `right` and `value` giving the left and right jump point and the value of the peak, respectively.

References

Pein, F., Tecuapetla-Gómez, I., Schütte, O., Steinem, C., Munk, A. (2018) Fully-automatic multiresolution idealization for filtered ion channel recordings: flickering event detection. IEEE Trans. Nanobioscience, 17(3):300-320.

See Also

`lowpassFilter`

`helpFunctionsFilter` *Convolved piecewise constant signals*

Description

Creates piecewise constant signals with a single jump / peak. Computes the convolution of piecewise constant signals with the kernel of a lowpass filter.

Usage

```
getConvolution(t, stepfun, filter, truncated = TRUE)
getSignalJump(t, cp, leftValue, rightValue)
getConvolutionJump(t, cp, leftValue, rightValue, filter, truncated = TRUE)
getSignalPeak(t, cp1, cp2, value, leftValue, rightValue)
getConvolutionPeak(t, cp1, cp2, value, leftValue, rightValue, filter, truncated = TRUE)
```

Arguments

`t`	a numeric vector giving the time points at which the signal / convolution should be computed
`stepfun`	specification of the piecewise constant signal, i.e. a `data.frame` with named arguments `leftEnd`, `rightEnd` and `value` giving the start and end points of the constant segments and the values on the segments, for instance an object of class `stepblock` as available by the package 'stepR'
`cp, cp1, cp2`	a single numeric giving the location of the single, first and second jump point, respectively
`value, leftValue, rightValue`	a single numeric giving the function value at, before and after the peak / jump, respectively
`filter`	an object of class `lowpassFilter` giving the analogue lowpass filter
`truncated`	a single logical (not NA) indicating whether the signal should be convolved with the truncated or the untruncated filter kernel

Value

a numeric of length `length(t)` giving the signal / convolution at time points `t`

References

Pein, F., Bartsch, A., Steinem, C., and Munk, A. (2020) Heterogeneous idealization of ion channel recordings - Open channel noise. Submitted.

Pein, F., Tecuapetla-Gómez, I., Schütte, O., Steinem, C., Munk, A. (2018) Fully-automatic multiresolution idealization for filtered ion channel recordings: flickering event detection. IEEE Trans. Nanobioscience, 17(3):300-320.

Pein, F. (2017) Heterogeneous Multiscale Change-Point Inference and its Application to Ion Channel Recordings. PhD thesis, Georg-August-Universität Göttingen. http://hdl.handle.net/11858/00-1735-0000-002E-E34A-7.

See Also

lowpassFilter

Examples

```
# creating and plotting a signal with a single jump at 0 from 0 to 1
time <- seq(-2, 13, 0.01)
signal <- getSignalJump(time, 0, 0, 1)
plot(time, signal, type = "l")

# setting up the filter
filter <- lowpassFilter(param = list(pole = 4, cutoff = 0.1))

# convolution with the truncated filter
convolution <- getConvolutionJump(time, 0, 0, 1, filter)
lines(time, convolution, col = "red")

# without truncating the filter, looks almost equal
convolution <- getConvolutionJump(time, 0, 0, 1, filter, truncated = FALSE)
lines(time, convolution, col = "blue")


# creating and plotting a signal with a single peak with jumps
# at 0 and at 3 from 0 to 1 to 0
time <- seq(-2, 16, 0.01)
signal <- getSignalPeak(time, 0, 3, 1, 0, 0)
plot(time, signal, type = "l")

# convolution with the truncated filter
convolution <- getConvolutionPeak(time, 0, 3, 1, 0, 0, filter)
lines(time, convolution, col = "red")

# without truncating the filter, looks almost equal
convolution <- getConvolutionPeak(time, 0, 3, 1, 0, 0, filter, truncated = FALSE)
lines(time, convolution, col = "blue")


# doing the same with getConvolution
# signal can also be an object of class stepblock instead,
# e.g. constructed by stepR::stepblock
signal <- data.frame(value = c(0, 1, 0), leftEnd = c(-2, 0, 3), rightEnd = c(0, 3, 16))

convolution <- getConvolution(time, signal, filter)
lines(time, convolution, col = "red")

convolution <- getConvolution(time, signal, filter, truncated = FALSE)
lines(time, convolution, col = "blue")


# more complicated signal
time <- seq(-2, 21, 0.01)
signal <- data.frame(value = c(0, 10, 0, 50, 0), leftEnd = c(-2, 0, 3, 6, 8),
```

```
                     rightEnd = c(0, 3, 6, 8, 21))

convolution <- getConvolution(time, signal, filter)
plot(time, convolution, col = "red", type = "l")

convolution <- getConvolution(time, signal, filter, truncated = FALSE)
lines(time, convolution, col = "blue")
```

`lowpassFilter` *Lowpass filtering*

Description

Creates a lowpass filter.

Usage

```
lowpassFilter(type = c("bessel"), param, sr = 1, len = NULL, shift = 0.5)
## S3 method for class 'lowpassFilter'
print(x, ...)
```

Arguments

`type`	a string specifying the type of the filter, currently only Bessel filters are supported
`param`	a `list` specifying the parameters of the filter depending on `type`. For `"bessel"` the entries `pole` and `cutoff` have to be specified and no other named entries are allowed. `pole` has to be a single integer giving the number of poles (order). `cutoff` has to be a single positive numeric not larger than `1` giving the normalized cutoff frequency, i.e. the cutoff frequency (in the temporal domain) of the filter divided by the sampling rate
`sr`	a single numeric giving the sampling rate
`len`	a single integer giving the filter length of the truncated and digitised filter, see *Value* for more details. By default (`NULL`) it is chosen such that the autocorrelation function is below `1e-3` at `len / sr` and at all lager lags `(len + i) / sr`, with `i` a positive integer
`shift`	a single numeric between `0` and `1` giving a shift for the digitised filter, i.e. kernel and step are obtained by evaluating the corresponding functions at `(0:len + shift) / sr`
`x`	the object
`...`	for generic methods only

Value

An object of `class` `lowpassFilter`, i.e. a `list` that contains

`"type"`, `"param"`, `"sr"`, `"len"` the corresponding arguments

`"kernfun"` the kernel function of the filter, obtained as the Laplace transform of the corresponding transfer function

`"stepfun"` the step-response of the filter, i.e. the antiderivative of the filter kernel

`"acfun"` the autocorrelation function, i.e. the convolution of the filter kernel with itself

`"acAntiderivative"` the antiderivative of the autocorrelation function

`"truncatedKernfun"` the kernel function of the at `len / sr` truncated filter, i.e. `kernfun` truncated and rescaled such that the new kernel still integrates to `1`

`"truncatedStepfun"` the step-response of the at `len / sr` truncated filter, i.e. the antiderivative of the kernel of the truncated filter

`"truncatedAcfun"` the autocorrelation function of the at `len / sr` truncated filter, i.e. the convolution of the kernel of the truncated filter with itself

`"truncatedAcAntiderivative"` the antiderivative of the autocorrelation function of the at `len / sr` truncated filter

`"kern"` the digitised filter kernel normalised to one, i.e. `kernfun((0:len + shift) / sr) / sum(kernfun((0:len + shift) / sr))`

`"step"` the digitised step-response of the filter, i.e. `stepfun((0:len + shift) / sr)`

`"acf"` the discrete autocorrelation, i.e. `acfun(0:len / sr)`

`"jump"` the last index of the left half of the filter, i.e. `min(which(ret$step >= 0.5)) - 1L`, it indicates how much a jump is shifted in time by a convolution of the signal with the digitised kernel of the lowpassfilter; if all values are below `0.5`, `len` is returned with a `warning`

`"number"` for developers; an integer indicating the type of the filter

`"list"` for developers; a list containing precomputed quantities to recreate the filter in C++

References

Pein, F., Bartsch, A., Steinem, C., and Munk, A. (2020) Heterogeneous idealization of ion channel recordings - Open channel noise. Submitted.

Pein, F., Tecuapetla-Gómez, I., Schütte, O., Steinem, C., Munk, A. (2018) Fully-automatic multiresolution idealization for filtered ion channel recordings: flickering event detection. IEEE Trans. Nanobioscience, 17(3):300-320.

Pein, F. (2017) Heterogeneous Multiscale Change-Point Inference and its Application to Ion Channel Recordings. PhD thesis, Georg-August-Universität Göttingen. http://hdl.handle.net/11858/00-1735-0000-002E-E34A-7.

Hotz, T., Schütte, O., Sieling, H., Polupanow, T., Diederichsen, U., Steinem, C., and Munk, A. (2013) Idealizing ion channel recordings by a jump segmentation multiresolution filter. IEEE Trans. Nanobioscience, 12(4):376-386.

See Also

`filter`

Examples

```
filter <- lowpassFilter(type = "bessel", param = list(pole = 4L, cutoff = 1e3 / 1e4),
                        sr = 1e4)

# filter kernel, truncated version
plot(filter$kernfun, xlim = c(0, 20 / filter$sr))
t <- seq(0, 20 / filter$sr, 0.01 / filter$sr)
# truncated version looks very similar
lines(t, filter$truncatedKernfun(t), col = "red")

# filter$len (== 11) is chosen automatically
# this ensures that filter$acf < 1e-3 for this lag and at all larger lags
plot(filter$acfun, xlim = c(0, 20 / filter$sr), ylim = c(-0.003, 0.003))
abline(h = 0.001, lty = "22")
abline(h = -0.001, lty = "22")

abline(v = (filter$len - 1L) / filter$sr, col = "grey")
abline(v = filter$len / filter$sr, col = "red")

# filter with sr == 1
filter <- lowpassFilter(type = "bessel", param = list(pole = 4L, cutoff = 1e3 / 1e4))

# filter kernel and its truncated version
plot(filter$kernfun, xlim = c(0, 20 / filter$sr))
t <- seq(0, 20 / filter$sr, 0.01 / filter$sr)
# truncated version looks very similar
lines(t, filter$truncatedKernfun(t), col = "red")
# digitised filter
points((0:filter$len + 0.5) / filter$sr, filter$kern, col = "red", pch = 16)

# without a shift
filter <- lowpassFilter(type = "bessel", param = list(pole = 4L, cutoff = 1e3 / 1e4),
                        shift = 0)
# filter$kern starts with zero
points(0:filter$len / filter$sr, filter$kern, col = "blue", pch = 16)

# much shorter filter
filter <- lowpassFilter(type = "bessel", param = list(pole = 4L, cutoff = 1e3 / 1e4),
                        len = 4L)
points((0:filter$len + 0.5) / filter$sr, filter$kern, col = "darkgreen", pch = 16)
```

`randomGeneration` *Random number generation*

Description

Generate random numbers that are filtered. Both, signal and noise, are convolved with the given lowpass filter, see details. Can be used to generate synthetic data resembling ion channel recordings, please see *(Pein et al., 2018, 2020)* for the exact models.

Usage

```
randomGeneration(n, filter, signal = 0, noise = 1, oversampling = 100L, seed = n,
                 startTime = 0, truncated = TRUE)
randomGenerationMA(n, filter, signal = 0, noise = 1, seed = n,
                   startTime = 0, truncated = TRUE)
```

Arguments

n	a single positive integer giving the number of observations that should be generated
filter	an object of class lowpassFilter giving the analogue lowpass filter
signal	either a numeric of length 1 or of length n giving the convolved signal, i.e. the mean of the random numbers, or an object that can be passed to getConvolution, i.e. an object of class stepblock, see *Examples*, giving the signal that will be convolved with the kernel of the lowpass filter filter
noise	for randomGenerationMA a single positive finite numeric giving the constant noise level, for randomGeneration either a numeric of length 1 or of length (n + filter$len - 1L) * oversampling or an object of class stepblock, see *Examples*, giving the noise of the random errors, see *Details*
oversampling	a single positive integer giving the factor by which the errors should be oversampled, see *Details*
seed	will be passed to set.seed to set a seed, set.seed will not be called if this argument is set to "no", i.e. a single value, interpreted as an integer, NULL or "no"
startTime	a single finite numeric giving the time at which sampling should start
truncated	a single logical (not NA) indicating whether the signal should be convolved with the truncated or the untruncated filter kernel

Details

As discussed in *(Pein et al., 2018)* and *(Pein et al., 2020)*, in ion channel recordings the recorded data points can be modelled as equidistant sampled at rate filter$sr from the convolution of a piecewise constant signal perturbed by Gaussian white noise scaled by the noise level with the kernel of an analogue lowpass filter. The noise level is either constant (homogeneous noise, see *(Pein et al., 2018)*) or itself varying (heterogeneous noise, see *(Pein et al., 2020)*). randomGeneration and randomGenerationMA generate synthetic data from such models. randomGeneration allows homogeneous and heterogeneous noise, while randomGenerationMA only allows homogeneous noise, i.e. noise has to be a single numeric giving the constant noise level. The resulting observations represent the conductance at time points startTime + 1:n / filter$sr.

The generated observations are the sum of a convolved signal evaluated at those time points plus centred Gaussian errors that are correlated (coloured noise), because of the filtering, and scaled by the noise level. The convolved signal evaluated at those time points can either by specified in signal directly or signal can specify a piecewise constant signal that will be convolved with the filter using getConvolution and evaluated at those time points. randomGenerationMA computes a moving average process with the desired autocorrelation to generate random errors. randomGeneration oversamples the error, i.e. generates errors at time points startTime + (seq(1 - filter$len + 1

/ oversampling, n, 1 / oversampling) - 1 / 2 / oversampling) / filter$sr, which will then be convolved with the filter. For this function noise can either give the noise levels at those oversampled time points or specify a piecewise constant function that will be automatically evaluated at those time points.

Value

a numeric vector of length n giving the generated random numbers

References

Pein, F., Bartsch, A., Steinem, C., and Munk, A. (2020) Heterogeneous idealization of ion channel recordings - Open channel noise. Submitted.

Pein, F., Tecuapetla-Gómez, I., Schütte, O., Steinem, C., Munk, A. (2018) Fully-automatic multiresolution idealization for filtered ion channel recordings: flickering event detection. IEEE Trans. Nanobioscience, 17(3):300-320.

Pein, F. (2017) Heterogeneous Multiscale Change-Point Inference and its Application to Ion Channel Recordings. PhD thesis, Georg-August-Universität Göttingen. http://hdl.handle.net/11858/00-1735-0000-002E-E34A-7.

See Also

lowpassFilter, getConvolution

Examples

```
filter <- lowpassFilter(type = "bessel", param = list(pole = 4, cutoff = 0.1), sr = 1e4)
time <- 1:4000 / filter$sr
stepfun <- getSignalPeak(time, cp1 = 0.2, cp2 = 0.2 + 3 / filter$sr,
                         value = 20, leftValue = 40, rightValue = 40)
signal <- getConvolutionPeak(time, cp1 = 0.2, cp2 = 0.2 + 3 / filter$sr,
                             value = 20, leftValue = 40, rightValue = 40, filter = filter)
data <- randomGenerationMA(n = 4000, filter = filter, signal = signal, noise = 1.4)

# generated data
plot(time, data, pch = 16)

# zoom into the single peak
plot(time, data, pch = 16, xlim = c(0.199, 0.202), ylim = c(19, 45))
lines(time, stepfun, col = "blue", type = "s", lwd = 2)
lines(time, signal, col = "red", lwd = 2)

# use of randomGeneration instead
data <- randomGeneration(n = 4000, filter = filter, signal = signal, noise = 1.4)

# similar result
plot(time, data, pch = 16, xlim = c(0.199, 0.202), ylim = c(19, 45))
lines(time, stepfun, col = "blue", type = "s", lwd = 2)
lines(time, signal, col = "red", lwd = 2)

## heterogeneous noise
```

```
# manual creation of an object of class 'stepblock'
# instead the function stepblock in the package stepR can be used
noise <- data.frame(leftEnd = c(0, 0.2, 0.2 + 3 / filter$sr),
                    rightEnd = c(0.2, 0.2 + 3 / filter$sr, 0.4),
                    value = c(1, 30, 1))
attr(noise, "x0") <- 0
class(noise) <- c("stepblock", class(noise))

data <- randomGeneration(n = 4000, filter = filter, signal = signal, noise = noise)

plot(time, data, pch = 16, xlim = c(0.199, 0.202), ylim = c(19, 45))
lines(time, stepfun, col = "blue", type = "s", lwd = 2)
lines(time, signal, col = "red", lwd = 2)
```

Index

∗ **nonparametric**
helpFunctionsFilter, 5
lowpassFilter-package, 2
randomGeneration, 9
∗ **package**
lowpassFilter-package, 2
∗ **ts**
lowpassFilter, 7
lowpassFilter-package, 2
.convolve *(*convolve*)*, 3
.deconvolveJump *(*deconvolve*)*, 4
.deconvolvePeak *(*deconvolve*)*, 4

class, *8*
convolve, 3

data.frame, *5*
deconvolve, 4
deconvolveJump *(*deconvolve*)*, 4
deconvolvePeak *(*deconvolve*)*, 4

filter, *8*

getConvolution, *2*, *10, 11*
getConvolution *(*helpFunctionsFilter*)*, 5
getConvolutionJump, *2*
getConvolutionJump *(*helpFunctionsFilter*)*, 5
getConvolutionPeak, *2*
getConvolutionPeak *(*helpFunctionsFilter*)*, 5
getSignalJump *(*helpFunctionsFilter*)*, 5
getSignalPeak *(*helpFunctionsFilter*)*, 5

helpFunctionsFilter, 5

integer, *10*

list, *7, 8*
lowpassFilter, *2–6*, 7, *10, 11*
lowpassFilter-package, 2

print.lowpassFilter *(*lowpassFilter*)*, 7

randomGeneration, *2*, 9
randomGenerationMA, *2*
randomGenerationMA *(*randomGeneration*)*, 9

set.seed, *10*