Package 'SmoothWin'

May 11, 2026

Type Package

Title Soft Windowing for Linear and Non-Linear Models

Version 3.0.1

Date 2026-05-03

Description Fits symmetric soft windowing to linear and non-linear models by assigning exponential weights over time around specified modes; bandwidth and sharpness of the windows are chosen by a grid search and comparison diagnostics (Hamed Haselimashhadi et al (2019) <doi:10.1093/bioinformatics/btz744>).

Maintainer Hamed Haselimashhadi <hamedhaseli@gmail.com>

License LGPL (>= 2)

URL https://doi.org/10.1093/bioinformatics/btz744

Encoding UTF-8

Repository CRAN

Date/Publication 2026-05-11 19:10:30 UTC

Depends R (>= 3.5)

Imports nlme, Rfast

NeedsCompilation no

RoxygenNote 7.3.3

Author Hamed Haselimashhadi [aut, cre]

Contents

expWeight *This function computes the smooth windowing weights*

Description

The symmetric weight generating function (SWGF). This function computes the exponential weights/kernel (soft windowing weights) for different shapes (k) and bandwidth (l) and plots the weights.

Usage

```
expWeight(
    t                       ,
    k                       ,
    l                       ,
    m = 0                   ,
    direction = c(1, 1)     ,
    plot = FALSE            ,
    zeroCompensation = 0    ,
    cdf   = plogis          ,
    progress         = FALSE ,
    ...
)
```

Arguments

t	Vector of numeric time. A vector of positive continuous values for time
k	A single positive value for sharpness
l	A single non-negative value for bandwidth
m	Vector of indices. The index of the modes on 't' (modes are the peak of the windows)
direction	Vector of two numeric values. A vector of the form on (Left,right). The first element specifies the speed of expansion of the window(s) from the left and the second value for the right expansion. Setting to (0,1) and (1,0) lead to right and left expansions of the windows respectively. Default (1,1) that is the window(s) expand symmetrically from both sides.
plot	Logical flag. Setting to TRUE shows a plot of the weights
zeroCompensation	Single non-negative value. Setting to any non-negative value would replace all (weights =< zeroCompensation) with zeroCompensation. Default 0 (zero)
cdf	A cdf function preferably symmetric. The cdf function is used for the (window) weight generating function. The function must have two parameters precisely a location such as mean and a scale. Standard cdf functions such as pnorm, pcauchy and plogis (default) can be used. For an example of custom made function we define uniform function as below:

$punif0 = function(x, mean = 0.5, sd = sqrt(1/12))a = mean - sqrt(3) * sd; b = mean + sqrt(3$

progress	Logical flag. Setting to TRUE shows the progress of the function
...	Other parameters that can be passed to the 'plot()' function such as pch, colour etc.

Value

A numeric vector of length `length(t)` giving merged soft-window weights in $[0, 1]$ (after scaling within the observed range). Values are observation weights suitable for passing to weighted model fitting.

Author(s)

Hamed Haselimashhadi <hamedhaseli@gmail.com>

See Also

SmoothWin

Examples

```
oldpar <- par(mfrow = c(4, 1))
################################################
# Example 1 - no merging happends between windows
################################################
weight = expWeight(
  t = 1:100                                     ,
  k = 5                                         ,
  l = 10                                        ,
  m = c(25, 50, 75)                             ,
  plot = TRUE                                   ,
  ### Passed parameters to the plot function
  type = 'l'                                    ,
  lty = 2                                       ,
  lwd = 3                                       ,
  main = '1. If windows do not intersect, then wont merge! (l=10, k=5)'
)

################################################
# Example 2 - merging in windows
################################################
weight = expWeight(
  t = 1:100                                     ,
  k = 5                                         ,
  l = 15                                        ,
  m = c(25, 50, 75)                             ,
  plot = TRUE                                   ,
  ### Passed parameters to the plot function
  type = 'l'                                    ,
```

```
  lty = 2                                          ,
  lwd = 3                                          ,
  main = '2. If windows intersect, then merge! (l=15, k=5)'
)

################################################
# Example 3.1  - partial merging in windows
################################################
weight = expWeight(
  t = 1:100                                        ,
  k = 1                                            ,
  l = 12                                           ,
  m = c(25, 50, 75)                                ,
  plot = TRUE                                      ,
  ### Passed parameters to the plot function
  type = 'l'                                       ,
  lty = 2                                          ,
  lwd = 3                                          ,
  main = '3.1 If windows intersect with small k, then partially merge! (l=12, k=1)'
)

################################################
# Example 3.2  - partial merging in windows
################################################
  weight = expWeight(
  t = 1:100                                        ,
  k = .1                                            ,
  l = 12                                           ,
  m = c(25, 50, 75)                                ,
  plot = TRUE                                      ,
  ### Passed parameters to the plot function
  type = 'l'                                       ,
  lty = 2                                          ,
  lwd = 3                                          ,
  main = '3.2 If windows intersect with small k, then partially merge! (l=12, k=0.1)'
)

par(oldpar)
```

`lseq` *Logarithmically spaced sequence*

Description

Generates `length.out` positive values whose logarithms are equally spaced from `log(from)/adj` to `log(to)` (equivalently, values follow a geometric progression on that log interval). Used inside `SmoothWin` to build default grids for bandwidth (`l`) and sharpness (`k`).

Usage

```
lseq(from = 1, to = 5, length.out = 6, adj = 1)
```

Arguments

`from`	Positive numeric; start of the sequence on the original scale when `adj = 1` (first generated value equals `from`). Must be strictly positive so that `log(from)` is defined.
`to`	Positive numeric; end of the sequence on the original scale when `adj = 1` (last generated value equals `to`). Must be strictly positive.
`length.out`	Non-negative integer; desired length of the result. Passed to `seq` as `length.out`.
`adj`	Positive numeric; divisor applied to `log(from)` in the lower endpoint of the underlying arithmetic sequence in log space: `seq(log(from)/adj, log(to), length.out = length.out)`. The default `1` gives a geometric grid from `from` to `to`. Values other than `1` change the effective lower endpoint before exponentiation (so the first value is `exp(log(from)/adj)` rather than `from` when `from != 1`).

Details

The implementation is `exp(seq(log(from)/adj, log(to), length.out = length.out))`. With `adj = 1`, values are equally spaced on a logarithmic scale between `from` and `to`, which is natural for multiplicative search grids.

Value

A numeric vector of length `length.out` with strictly positive entries (assuming `from`, `to`, and `adj` are positive and `length.out` is at least 1).

Author(s)

Hamed Haselimashhadi <hamedhaseli@gmail.com>

See Also

`SmoothWin` for use in default `l` and `k` grids.

Examples

```
## Geometric sequence from 1 to 5 (compare to linear seq)
lseq(from = 1, to = 5, length.out = 6)

## Shorter grid similar to default sharpness values in SmoothWin
lseq(from = 0.5, to = 10, length.out = 8)
```

`plot.SmoothWin` *Plot function for the SmoothWin object*

Description

This function plots a SmoothWin object

Usage

```
  ## S3 method for class 'SmoothWin'
plot(x,
                          ylab   = 'Response'              ,
                          xlab   = 'Time (continuous)'    ,
                          sub    = NULL                   ,
                          col    = NULL                   ,
                          digits = 2                      ,
                          ...
                          )
```

Arguments

`x`	SmoothWin object
`ylab`	Label on the y axis. Default 'Response'
`xlab`	Label on the x axis. Default 'Time (continuous)'
`sub`	See the 'sub' parameter in 'plot()' function. If left NULL then some information about the final window will be shown. Default NULL
`col`	Colour parameter for the points. Set to NULL to use the default colouring (spectrum colouring). Default NULL
`digits`	The number of visible digits for l, k and SWS. Default 2
`...`	Optional parameters that can be passed to the 'plot'/'qqPlot' function. See 'car' package for the qqPlot function

Value

Called mainly for its side effect of drawing a scatterplot of the response against time, overlaying the scaled window weights and vertical lines at the modes.

If a final model is available, invisibly returns a list with components `weight` (numeric vector of full window weights along the time axis), `k` and `l` (the sharpness and bandwidth selection results from the fit, as stored in `x$final.k` and `x$final.l`), and `object` (the original `SmoothWin` object).

If windowing failed and no model is stored, prints a message and does not plot; the return value is `NULL` in that case.

Author(s)

Hamed Haselimashhadi <hamedhaseli@gmail.com>

See Also

SmoothWin

Examples

```
example(SmoothWin)
```

SmoothWin *Implementation of the soft windowing for linear models*

Description

Implementation of the (symmetric) soft windowing on a range of methods/models by imposing weights on the model.

- The function accepts a model fit, such as ‘lm‘, ‘lme‘, ‘glm‘ etc., as the input and fits a window to it.

- The parameters "k" and "l" control the shape and bandwidth of the windowing function respectively.

- There are several other parameters to cope with the different scenarios/models/window shapes.

- The default settings of the function is adapted to International Mouse Phenotyping Consortium (IMPC) statistical pipeline

Usage

```
SmoothWin(object                                          ,
          data                                            ,
          t                                               ,
          m                                               ,
          l = function(ignore.me.in.default) {
             r = lseq(
              from = 1                                    ,
              to = max(abs(t[m] - min(t, na.rm = TRUE))   ,
              abs(t[m] - max(t, na.rm = TRUE)), 1)        ,
              length.out = min(500, max(1, diff(range(
                t,na.rm = TRUE
              ))))
            )
            r = unique(round(r))
            return(r)
          }                                               ,
          k =             lseq(from = .5                  ,
                               to = 10                    ,
                               length.out = 50)           ,
          min.obs   = function(ignore.me.in.default) {
```

```
  lutm = length(unique(t[m]))
  r = ifelse(lutm > 1, 35, max(pi * sqrt(length(t)), 35))
  r = max(r * lutm, length(m), na.rm = TRUE)
  r = min(r       , length(t), na.rm = TRUE)
  return(r)
}                                                  ,
direction = c(1, 1)                                ,
weightFUN = function(x) {
  x
}                                                  ,
residFun = function(x) {
  resid(x)
}                                                  ,
predictFun = function(x) {
  predict(x)
}                                                  ,
weightORthreshold = 'weight'                       ,
cdf = plogis                                       ,
check = 2                                          ,
sensitivity   = c(1, 1, 1, 0)                      ,
pvalThreshold = c(0, 0, 0, 0)                      ,
threshold     = sqrt(.Machine$double.eps) * 10     ,
zeroCompensation = 0                               ,
messages      = TRUE                               ,
seed          = NULL                               ,
simple.output = FALSE                              ,
debug         = FALSE                              ,
...)
```

Arguments

object	The fitted model. The object must support 'update(weights =)'. See examples
data	data.frame. Input data that is used to fit the initial model
t	Vector of (numeric) time values.
m	Vector of integers (peaks). Mode indices on the time component. For example 10, 11, 12. Note that it is different from t[10], t[11], t[12]
l	Vector of numeric values for the bandwidth parameter, l. The default uses the maximum distance of the modes (t[m]) from the time boundaries, max(max(t)-t[m],t[m]-min(t)) split on 500 points on the logarithmic scale.
k	Vector of numeric values for the shape parameter, k. The default uses 50 splits of the values from 0.5 to 10 on the logarithmic scale.
min.obs	Single value. The minimum observations/sum weight scores (SWS) that must be in the total window(s). The default uses the following steps. 1. If there are more than one modes (peaks) in the data, then: 35*(the number of the unique modes)

2. If there is a single mode in the data, then: max(pi * sqrt(length(t)), 35)

* min.obs must not be less than the total number of observations in the mode time(s). For example, it can not be less than the number of mutant animals in the IMPC application.

** to function properly, min.obs should be less than the total number of observations

*** min.obs is applied on the total number of observations on all windows NOT each single window

**** if weightORthreshold='weight' then min.obs will be evaluated against SWS

`direction` Vector of two non-negative values. A non-negative vector of the form c(Left,right), for example c(1,1) [default] or c(0.5,0.5) or c(0,1). The first element specifies the speed of expansion of the window(s) from the left and the second value for the right expansion. Setting to c(0,1) and c(1,0) lead to right and left expansions of the window(s) respectively. Default c(1,1) that is the window(s) expand symmetrically from both sides.

`weightFUN` Weight function. By default, a vector of weights called "ModelWeight" is passed to this function. See the examples.

`residFun` Residual computation function. The default is 'resid()'. However, the the user can define its own function. Note that the input of the function is the model object. The default is residFun = function(object){resid(object)}

`predictFun` Similar to residFun but instead defines the 'predict()' function. The default is predictFun = function(object){predict(object)}

`weightORthreshold`
select between 'weight' (default) or 'threshold'. If set to 'weight' then the sum of weights (Sum Weight Score (SWS)) would be used as the total number of (active) observations in the window, otherwise, total number of weights (count of weights) that are greater than a threshold (see 'threshold' below) (count weights>=threshold) would be used for the total number of samples in the window (see 'threshold').

`cdf` A cdf function preferably symmetric. The cdf function is used for the (window) weight generating function (WGF). The cdf function must have two parameters precisely a location such as mean and a scale. Standard cdf functions such as 'pnorm', 'pcauchy' and 'plogis' (default) can be used. For an example of custom made function we define uniform function as below:

$$punif0 = function(x, mean = .5, sd = sqrt(1/12))a = mean - sqrt(3) * sd; b = mean + sqrt(3)$$

`check` Single integer in {0,1,2}:

- check=1, the function selects the times (t) with more than one observations.

Further, the function only selects the values with weights greater than the 'threshold' (see threshold below). Mostly useful in fitting linear mixed model

- check=2 (default), the function only selects the values with weights greater than the 'threshold' parameter

- check=0, disables all checks

`sensitivity` (Default window selection criteria) Vector of four values (m, v, m*v, normality_test). For example (default) c(1,1,1,0) specifies the same weights for mean, variance, mean*variance interaction and zero weight for the test of normality (shapiro.test) in determining the optimal (final) window. We should stress that the window size is calculated by detecting the changes amongst the consecutive means (two sample t.test) and variances (two sample var.test) (as well as the normality of the first set) from each of predictFun() and residFun() combined together. For example, (m, v, m*v, normality_test) is calculated for predictFun() and the same for residFun(), then two means are combined under the 'sensitivity'[1]; and the same for variance, interactions and the normality.

`pvalThreshold` Vector of four values. It would be used as the significant level for the mean, variation and normality tests (for more details see 'sensitivity' above). If all zero (default) or (all) negative (<= 0) then the internal adaptive method (sensitivity - see above) would be used.

`threshold` Single positive value. The minimum value for weights before removing the corresponded samples, given check=1 or check=2 and also in 'weightORthreshold'. Default sqrt(.Machine$double.eps)*10 ~ 10^-7

`zeroCompensation`
Single non-negative value. Setting to any non-negative value would replace all (weights =< zeroCompensation) with 'zeroCompensation'. Useful for algorithms that have difficulties with zero. Default 0.

`messages` Logical value. Set to TRUE (default) to see the errors and warnings

`seed` seed. Default NULL

`simple.output` Logical flag. Setting to TRUE leads to not exporting the list of models for l and k. Useful for preventing memory overflow. Default FALSE

`debug` Logical flag. Setting to TRUE will show some plots for the parameter selection step. Useful for debuging. Default FALSE

`...` Other parameters that can be passed to the weightFUN()

Value

An object of class `"SmoothWin"` (a list) containing the fitted window and all objects needed to inspect or plot it.

`object` The original model object passed in.

`data` The `data` argument.

`final.k`, `final.l`
Lists describing the selected sharpness (`k`) and bandwidth (`l`), including at least `value` (chosen parameter) and `score` (diagnostic score, possibly `NA`).

finalModel	Result of the final weighted refit at the chosen (l, k): includes the updated model, merged weights, and summary output such as effective observation count in the window.
model.l, model.k	Lists of intermediate fits over the l and k grids (omitted if simple.output = TRUE).
min.obs	The minimum observation / sum-of-weights constraint used.
input	Captured call arguments (including resolved l, k, t, m, etc.) for reproducibility and plotting.

Author(s)

Hamed Haselimashhadi <hamedhaseli@gmail.com>

See Also

expWeight

Examples

```
################################################
################### Example in the manuscript
################################################
set.seed(1234)
oldpar <- par(mfrow = c(3, 1))
#############################
# Simulating data
#############################
n = 60
t = 1:n
sd = 1
m = n / 2
x = t
y = c(0 * x[t <= n / 3]                         ,
      x[t < 2 * n / 3 & t > n / 3] * 1          ,
      0 * x[t >= 2 * n / 3]) + rnorm(n, 0, sd)
# True weights
w = weights = expWeight(
  t = t                                         ,
  k = 5                                         ,
  l = n/6                                       ,
  m = m                                         ,
  plot = 0
)
#############################
# Fitting and ploting data and models
#############################
l = lm(y ~ x, weights = w)
plot(
  x                                              ,
  y                                              ,
```

```
  ylim = c(min(y), max(y) * 1.5)                ,
  col = t %in% seq(n / 3 + 1, 2 * n / 3 - 1) + 1,
  cex = 1.5                                     ,
  pch = 16                                      ,
  xlab = 'Time'                                 ,
  main = 'Simulated data'
)
abline(v = x[c(n / 3 + 1, 2 * n / 3 - 1)],
       lty = 2    ,
       lwd = 4    ,
       col = 'gray')
abline(l, col = 2 , lty = 2, lwd = 4)
abline(lm(y ~ x)  ,
       col = 3    ,
       lty = 3    ,
       lwd = 4)
plot(
  t,
  w ,
  type = 'b'         ,
  main = 'True weights',
  ylab = 'Weight'    ,
  xlab = 'Time'
)
##############################
# Fitting the Windowing model
##############################
r = SmoothWin(
  object = l                      ,
  data = data.frame(y = y, x = x),
  t = t                           ,
  m = m                           ,
  min.obs = 4                     ,
  debug = FALSE
)
##############################
# Plot fitted (windowed) model
##############################
plot(r, main = 'Estimated weights from WGF')


################################################
################### Other examples
################################################
# All examples import the Orthodont dataset from the nlme package
library(nlme)
# Sort the data on the time component (age)
Orthodont =  Orthodont[order(Orthodont$age), ]
##############################
# Modes
##############################
mode = which(Orthodont$age  %in%  c(12))
##############################
```

```
# Time component
############################
time = Orthodont$age
f  = formula(distance ~ Sex)


################################################
################## Examples ##################
################################################
### Example 1. Linear model
############################
# Method 1 (recommanded)
############################
lm = do.call('lm', list(formula = f, data = Orthodont))
rm(f)

############################
# Method 2 (can cause error if you pass the formula to the lm function)
# lm = lm(distance ~ Sex, data = Orthodont)
############################

lm.result = SmoothWin(
  object = lm,
  data = Orthodont,
  t = time,
  m = mode,
  check = 0,
  weightFUN = function(x) {
    x
  },
  debug = TRUE
)
plot(
  lm.result,
  col = Orthodont$Sex,
  pch = as.integer(Orthodont$Sex),
  main = 'Simple liner model'
)

############################
#### Example 2. Linear Model Using Generalized Least Squares
# Method 1 (recommanded)
############################
f   = formula(distance ~ Sex)
gls = do.call('gls', list(model = f, data = Orthodont))
rm(f)

############################
# Method 2 (can cause error if you pass the formula to the gls function)
# gls = gls(distance ~ Sex, data = Orthodont)
############################
gls.result = SmoothWin(
  object = gls,
```

```
  data = Orthodont,
  t = time,
  m = mode,
  check = 2,
  weightFUN = function(ignore.me) {
    varFixed(~ 1 / ModelWeight) #nlme package uses the inverse weights
  },
  debug = TRUE
)
plot(
  gls.result,
  col = Orthodont$Sex,
  pch = as.integer(Orthodont$Sex),
  main = 'Linear model using GLS'
)

################################################
#### Example 3. Linear mixed model
################################################
# Method 1 (recommanded)
############################
fixed  = formula(distance ~ Sex)
random = formula(~ 1 | Subject)
lme = do.call('lme', list(
  fixed = fixed,
  random = random,
  data = Orthodont
))
rm(fixed, random)

############################
# Method 2 (can cause error if you pass the formula to the lme function)
# lme = lme(fixed = distance ~ Sex, random=~1|Subject , data = Orthodont)
############################
lme.result = SmoothWin(
  object = lme,
  data = Orthodont,
  t = time,
  m = mode,
  # Remove zero weights as well as single observation dates
  check = 1,
  weightFUN = function(ignore.me) {
    varFixed(~ 1 / ModelWeight)
  },
  debug = TRUE
)
plot(
  lme.result,
  col = Orthodont$Sex,
  pch = as.integer(Orthodont$Sex),
  main = 'Linear mixed model'
)
```

```
par(oldpar)
```

Index