Package 'stepR'

October 17, 2016

Title Fitting Step-Functions

Version 1.0-6

Depends R (>= 2.15.3), stats, graphics

Description Allows to fit step-
functions to univariate serial data where neither the number of jumps nor their positions is known.

License GPL-3

Classification/MSC 62G08, 92C40, 92D20

LazyData yes

NeedsCompilation yes

Author Timo Aspelmeier [cre, aut],
Thomas Hotz [aut],
Hannes Sieling [aut],
Pein Florian [ctb]

Maintainer Timo Aspelmeier <timo.aspelmeier@mathematik.uni-goettingen.de>

Repository CRAN

Date/Publication 2016-10-17 19:00:07

R topics documented:

`stepR-package` *Fit step-functions*

Description

Allows to fit step-functions to univariate serial data where the number of jumps is used as a penalty.

Details

Package:	stepR
Version:	1.0
Date:	2013-09-17
Authors@R:	c(person("Thomas", "Hotz", role = c("aut", "cre"), email = "thomas.hotz@tu-ilmenau.de"), person("Hannes", "
Depends:	R (>= 2.15.0), stats
License:	GPL-3
LazyData:	yes
Built:	R 2.15.3; i686-pc-linux-gnu; 2013-09-17 07:05:08 UTC; unix

Index:

jsmurf	Reconstruct filtered piecewise constant functions with noise
neighbours	Neighbouring integers
sdrobnorm	Robust standard deviation estimate
smuceR	Piecewise constant regression with SMUCE
stepR-package	Fit step-functions
stepblock	Step function
stepbound	Jump estimation under restrictions
stepcand	Forward selection of candidate jumps
stepfit	Fitted step function
steppath	Solution path of step-functions
stepsel	Automatic selection of number of jumps
transit	TRANSIT algorithm for detecting jumps

Author(s)

Thomas Hotz and Hannes Sieling

Maintainer: Thomas Hotz <thomas.hotz@tu-ilmenau.de>

References

Boysen, L., Kempe, A., Liebscher, V., Munk, A., Wittich, O. (2009). Consistencies and rates of convergence of jump-penalized least squares estimators. *The Annals of Statistics* 37(1), 157-183.

Davies, P.L., Kovac, A. (2001). Local extremes, runs, strings and multiresolution. *The Annals of Statistics* 29, 1-65.

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. With discussion and rejoinder by the authors. *Journal of the Royal Statistical Society, Series B* 76(3), 495-580.

Friedrich, F., Kempe, A., Liebscher, V., Winkler, G. (2008). Complexity penalized M-estimation: fast computation. *Journal of Computational and Graphical Statistics* 17(1), 201-224.

Futschik, A., Hotz, T., Munk, A. Sieling, H. (2014). Multiresolution DNA partitioning: statistical evidence for segments. *Bioinformatics*, 30(16), 2255-2262.

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 Transactions on NanoBioscience* 12(4), 376-386.

VanDongen, A.M.J. (1996). A New Algorithm for Idealizing Single Ion Channel Data Containing Multiple Unknown Conductance Levels. *Biophysical Journal* 70(3), 1303-1315.

See Also

smuceR, jsmurf, stepbound, steppath, stepsel, transit, compareBlocks, family

Examples

```
# estimating step-functions with Gaussian white noise added
# simulate a Gaussian hidden Markov model of length 1000 with 2 states
# with identical transition rates 0.01, and signal-to-noise ratio 2
sim <- contMC(1e3, 0:1, matrix(c(0, 0.01, 0.01, 0), 2), param=1/2)
plot(sim$data, cex = 0.1)
```

```
lines(sim$cont, col="red")
# maximum-likelihood estimation under multiresolution constraints
fit.MRC <- smuceR(sim$data$y, sim$data$x)
lines(fit.MRC, col="blue")
# choose number of jumps using BIC
path <- steppath(sim$data$y, sim$data$x, max.blocks=1e2)
fit.BIC <- path[[stepsel.BIC(path)]]
lines(fit.BIC, col="green3", lty = 2)

# estimate after filtering
# simulate filtered ion channel recording with two states
set.seed(9)
# sampling rate 10 kHz
sampling <- 1e4
# tenfold oversampling
over <- 10
# 1 kHz 4-pole Bessel-filter, adjusted for oversampling
cutoff <- 1e3
df.over <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling / over))
# two states, leaving state 1 at 10 Hz, state 2 at 20 Hz
rates <- rbind(c(0, 10), c(20, 0))
# simulate 0.5 s, level 0 corresponds to state 1, level 1 to state 2
# noise level is 0.3 after filtering
Sim <- contMC(0.5 * sampling, 0:1, rates, sampling=sampling, family="gaussKern",
  param = list(df=df.over, over=over, sd=0.3))
plot(Sim$data, pch = ".")
lines(Sim$discr, col = "red")
# fit under multiresolution constraints using filter corresponding to sample rate
df <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling))
Fit.MRC <- jsmurf(Sim$data$y, Sim$data$x, param=df, r=1e2)
lines(Fit.MRC, col = "blue")
# fit using TRANSIT
Fit.trans <- transit(Sim$data$y, Sim$data$x)
lines(Fit.trans, col = "green3", lty=2)
```

`BesselPolynomial` *Bessel Polynomials*

Description

Recursively compute coefficients of Bessel Polynomials.

Usage

```
BesselPolynomial(n, reverse = FALSE)
```

Arguments

`n`	order
`reverse`	whether to return the coefficients of a reverse Bessel Polynomial

Value

Returns the polynom's coefficients ordered increasing with the exponent, i.e. starting with the intercept, as for polyroot.

See Also

bessel, polyroot

Examples

```
# 15 x^3 + 15 x^2 + 6 x + 1
BesselPolynomial(3)
```

---

bounds *Bounds based on MRC*

Description

Computes two-sided bounds for a collection of intervals based on a multiresolution criterion (MRC).

Usage

```
bounds(y, type = "MRC", ...)
bounds.MRC(y, q, alpha = 0.05, r = ceiling(50 / min(alpha, 1 - alpha)),
  lengths = if(family == "gaussKern")
    2^(floor(log2(length(y))):ceiling(log2(length(param$kern)))) else
    2^(floor(log2(length(y))):0), penalty = c("none", "len", "var", "sqrt"),
  name = if(family == "gaussKern") ".MRC.ktable" else ".MRC.table", pos = .GlobalEnv,
  family = c("gauss", "gaussvar", "poisson", "binomial","gaussKern"), param = NULL,
  subset, max.iter = 1e2, eps = 1e-3)
## S3 method for class 'bounds'
x[subset]
```

Arguments

y	a numeric vector containing the serial data
type	so far only bounds of type "MRC" are implemented
...	further arguments to be passed on to bounds.MRC
q	quantile of the MRC; if specified, alpha and r will be ignored
alpha	level of significance
r	number of simulations to use to obtain quantile of MRC for specified alpha
lengths	vector of interval lengths to use, dyadic intervals by default
penalty	penalty term in the multiresolution statistic: "none" for no penalty, "len" for penalizing the length of an interval, "var" for penalizing the variance over an interval, and "sqrt" for penalizing the square root of the MRC

family, param	specifies distribution of data, see family
subset	a subset of indices of y for which bounds should be aggregated
name,pos	under which name and where precomputed results are stored, or retrieved, see assign
max.iter	maximal iterations in Newton's method to compute non-Gaussian MRC bounds
eps	tolerance in Newton's method
x	an object of class bounds

Value

Returns an object of class bounds, i.e. a list whose entry bounds contains two-sided bounds (lower and upper) of the considered intervals (with left index li and right index ri) in a data.frame, along with a vector start specifying in which row of entry bounds intervals with corresponding li start (if any; specified as a C-style index), and a logical feasible telling whether a feasible solution exists for these bounds (always TRUE for MRC bounds which are not restricted to a subset).

See Also

stepbound, family

Examples

```
# simulate signal of 100 data points
Y <- rpois(100, 1:100 / 10)
# compute bounds for intervals of dyadic lengths
b <- bounds(Y, penalty="len", family="poisson", q=4)
# compute bounds for all intervals
b <- bounds(Y, penalty="len", family="poisson", q=4, lengths=1:100)
```

---

compareBlocks *Compare fit blockwise with ground truth*

Description

Blockwise comparison of a fitted step function with a known ground truth using different criteria.

Usage

```
compareBlocks(truth, estimate, dist = 5e3)
```

Arguments

truth	an object of class stepblock giving the ground truth, or a list of such objects
estimate	corresponding estimated object(s) of class stepblock
dist	a single numeric specifying the distance for at which jumps will be considered as having matched in the qualitative criterion

Value

A `data.frame`, containing just one row if two single `stepblock` were given, with columns

`true.num, est.num`
: the true / estimated number of blocks

`true.pos, false.pos, false.neg, sens.rate, prec.rate`
: the number of true / false positive, false negatives, as well as the corresponding sensitivity and precision rates, where an estimated block is considered a true positive if it there is a corresponding block in the ground truth with both endpoints within `dist` of each other

`fpsle`
: false positive sensitive localization error: for each estimated block's midpoint find into which true block it falls, and sum distances of the respective borders

`fnsle`
: false negative sensitive localization error: for each true block's mid-point find into which estimated block it falls, and sum distances of the respective borders

`total.le`
: total localization error: sum of `fpsle` and `fnsle`

Note

No differences between true and fitted parameter *values* are taking into account, only the precision of the detected blocks is considered; also, differing from the criteria in Elhaik et al.~(2010), no blocks are merged in the ground truth if its parameter values are close, as this may punish sensitive estimators.

Beware that these criteria compare *blockwise*, i.e. they do *not* compare the precision of single jumps but for each block both endpoints have to match well at the same time.

References

Elhaik, E., Graur, D., Josić, K. (2010). Comparative testing of DNA segmentation algorithms using benchmark simulations. *Molecular Biology and Evolution* 27(5), 1015-24.

Futschik, A., Hotz, T., Munk, A. Sieling, H. (2014). Multiresolution DNA partitioning: statistical evidence for segments. *Bioinformatics*, 30(16), 2255-2262.

See Also

`stepblock`, `stepfit`, `contMC`

Examples

```
# simulate two Gaussian hidden Markov models of length 1000 with 2 states each
# with identical transition rates being 0.01 and 0.05, resp, signal-to-noise ratio is 5
sim <- lapply(c(0.01, 0.05), function(rate)
  contMC(1e3, 0:1, matrix(c(0, rate, rate, 0), 2), param=1/5))
plot(sim[[1]]$data)
lines(sim[[1]]$cont, col="red")
# use smuceR to estimate fit
fit <- lapply(sim, function(s) smuceR(s$data$y, s$data$x))
lines(fit[[1]], col="blue")
# compare fit with (discretised) ground truth
compareBlocks(lapply(sim, function(s) s$discr), fit)
```

contMC *Continuous time Markov chain*

Description

Simulate a continuous time Markov chain.

Usage

```
contMC(n, values, rates, start = 1, sampling = 1, family = c("gauss", "gaussKern"),
  param = NULL)
```

Arguments

n	number of data points to simulate
values	a numeric vector specifying signal amplitudes for different states
rates	a square matrix matching the dimension of values each with rates[i,j] specifying the transition rate from state i to state j; the diagonal entries are ignored
start	the state in which the Markov chain is started
sampling	the sampling rate
family	whether Gaussian white ("gauss") or coloured ("gaussKern"), i.e. filtered, noise should be added; cf. family
param	for family="gauss", a single non-negative numeric specifying the standard deviation of the noise; for family="gaussKern", param must be a list with entry df giving the dfilter object used for filtering, an integer entry over which specifies the oversampling factor of the filter, i.e. param$df has to be created for a sampling rate of sampling times over, and an additional non-negative numeric entry sd specifying the noise's standard deviation *after* filtering; cf. family

Value

A list with components

cont	an object of class stepblock containing the simulated true values in continuous time, with an additional column state specifying the corresponding state
discr	an object of class stepblock containing the simulated true values reduced to discrete time, i.e. containing only the observable blocks
data	a data.frame with columns x and y containing the times and values of the simulated observations, respectively

Note

This follows the description for simulating ion channels given by VanDongen (1996).

References

VanDongen, A.M.J. (1996). A New Algorithm for Idealizing Single Ion Channel Data Containing Multiple Unknown Conductance Levels. *Biophysical Journal* 70(3), 1303-1315.

See Also

stepblock, jsmurf, stepbound, steppath, family, dfilter

Examples

```
# Simulate filtered ion channel recording with two states
set.seed(9)
# sampling rate 10 kHz
sampling <- 1e4
# tenfold oversampling
over <- 10
# 1 kHz 4-pole Bessel-filter, adjusted for oversampling
cutoff <- 1e3
df <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling / over))
# two states, leaving state 1 at 1 Hz, state 2 at 10 Hz
rates <- rbind(c(0, 1e0), c(1e1, 0))
# simulate 5 s, level 0 corresponds to state 1, level 1 to state 2
# noise level is 0.1 after filtering
sim <- contMC(5 * sampling, 0:1, rates, sampling=sampling, family="gaussKern",
  param = list(df=df, over=over, sd=0.1))
sim$cont
plot(sim$data, pch = ".")
lines(sim$discr, col = "red")
# noise level after filtering, estimated from first block
sd(sim$data$y[1:sim$discr$rightIndex[1]])
# show autocovariance in first block
acf(ts(sim$data$y[1:sim$discr$rightIndex[1]], freq=sampling), type = "cov")
# power spectrum in first block
s <- spec.pgram(ts(sim$data$y[1:sim$discr$rightIndex[1]], freq=sampling), spans=c(200,90))
# cutoff frequency is where power spectrum is halved
abline(v=cutoff, h=s$spec[1] / 2, lty = 2)
```

---

dfilter *Digital filters*

Description

Create digital filters.

Usage

```
dfilter(type = c("bessel", "gauss", "custom"), param = list(pole = 4, cutoff = 1 / 10),
  len = ceiling(3/param$cutoff))
## S3 method for class 'dfilter'
print(x, ...)
```

Arguments

type	allows to choose Bessel, Gauss or custom filters
param	for a "bessel" filter a list with entries pole and cutoff giving the filter's number of poles (order) and cut-off frequency, resp.; for a "gauss" filter the filter's bandwidth (standard deviation) as a single numeric; for a custom filter either a numeric vector specifying the filter's kernel or a list with items kern and step of the same length giving the filter's kernel and step-response, resp.
len	filter length (unnecessary for "custom" filters
x	the object
...	for generic methods only

Value

Returns a list of class dfilter that contains elements kern and step, the (digitised) filter kernel and step-response, resp., as well as an element param containing the argument param, for a "bessel" filter alongside the corresponding analogue kernel, step response, power spectrum, and autocorrelation function depending on time or frequency as elements kernfun, stepfun, spectrum, and acfun, resp.

See Also

filter, convolve, BesselPolynomial, Normal, family

Examples

```
# 6-pole Bessel filter with cut-off frequency 1 / 100, with length 100 (too short!)
dfilter("bessel", list(pole = 6, cutoff = 1 / 100), 100)
# custom filter: running mean of length 3
dfilter("custom", rep(1, 3))
dfilter("custom", rep(1, 3))$kern # normalised!
dfilter("custom", rep(1, 3))$step
# Gaussian filter with bandwidth 3 and length 11 (from -5 to 5)
dfilter("gauss", 3, 11)
```

---

family *Family of distributions*

Description

Families of distributions supported by package stepR.

Details

Package stepR supports several families of distributions (mainly exponential) to model the data, some of which require additional (fixed) parameters. In particular, the following families are available:

"gauss" normal distribution with unknown mean but known, fixed standard deviation given as a single numeric (will be estimated using sdrobnorm if omitted); cf. dnorm.

"gaussvar" normal distribution with unknown variance but known, fixed mean assumed to be zero; cf. dnorm.

"poisson" Poisson distribution with unknown intensity (no additional parameter); cf. dpois.

"binomial" binomial distribution with unknown success probability but known, fixed size given as a single integer; cf. dbinom.

"gaussKern" normal distribution with unknown mean and unknown, fixed standard deviation (being estimated using sdrobnorm), after filtering with a fixed filter which needs to be given as the additional parameter (a dfilter object); cf. dfilter.

The family is selected via the family argument, providing the corresponding string, while the param argument contains the parameters if any.

Note

Beware that not all families can be chosen for all functions.

See Also

Distributions, dnorm, dpois, dbinom, dfilter, sdrobnorm

Examples

```
# illustrating different families fitted to the same binomial data set
size <- 200
n <- 200
# truth
p <- 10^seq(-3, -0.1, length = n)
# data
y <- rbinom(n, size, p)
plot(y)
lines(size * p, col = "red")
# fit 4 jumps, binomial family
jumps <- 4
bfit <- steppath(y, family = "binomial", param = size, max.blocks = jumps)
lines(bfit[[jumps]], col = "orange")
# Gaussian approximation with estimated variance
gfit <- steppath(y, family = "gauss", max.blocks = jumps)
lines(gfit[[jumps]], col = "green3", lty = 2)
# Poisson approximation
pfit <- steppath(y, family = "poisson", max.blocks = jumps)
lines(pfit[[jumps]], col = "blue", lty = 2)
legend("topleft", legend = c("binomial", "gauss", "poisson"), lwd = 2,
  col = c("orange", "green3", "blue"))
```

jsmurf *Reconstruct filtered piecewise constant functions with noise*

Description

Reconstructs a piecewise constant function to which white noise was added and the sum filtered afterwards.

Usage

```
jsmurf(y, x = 1:length(y), x0 = 2 * x[1] - x[2], q, alpha = 0.05, r = 4e3,
  lengths = 2^(floor(log2(length(y))):floor(log2(max(length(param$kern) + 1,
    1 / param$param$cutoff)))), param, rm.out = FALSE,
    jumpint = confband, confband = FALSE)
```

Arguments

y	a numeric vector containing the serial data
x	a numeric vector of the same length as y containing the corresponding sample points
x0	a single numeric giving the last unobserved sample point directly before sampling started
q	threshold value, by default chosen automatically
alpha	significance level; if set to a value in (0,1), q is chosen as the corresponding quantile of the asymptotic (if r is not given) null distribution (and any value specified for q is silently ignored)
r	numer of simulations; if specified along alpha, q is chosen as the corresponding quantile of the simulated null distribution
lengths	length of intervals considered; by default up to a sample size of 1000 all lengths, otherwise only dyadic lengths
param	a dfilter object specifiying the filter
rm.out	a logical specifying whether outliers should be removed prior to the analysis
jumpint	logical (FALSE by default), indicates if confidence sets for jumps should be computed
confband	logical, indicates if a confidence band for the piecewise-continuous function should be computed

Value

An object object of class stepfit that contains the fit; if jumpint = TRUE function jumpint allows to extract the 1- alpha confidence interval for the jumps, if confband = TRUE function confband allows to extract the 1- alpha confidence band.

References

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 Transactions on NanoBioscience* 12(4), 376-386.

See Also

stepbound, bounds, family, MRC.asymptotic, sdrobnorm, stepfit

Examples

```
# simulate filtered ion channel recording with two states
set.seed(9)
# sampling rate 10 kHz
sampling <- 1e4
# tenfold oversampling
over <- 10
# 1 kHz 4-pole Bessel-filter, adjusted for oversampling
cutoff <- 1e3
df.over <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling / over))
# two states, leaving state 1 at 10 Hz, state 2 at 20 Hz
rates <- rbind(c(0, 10), c(20, 0))
# simulate 0.5 s, level 0 corresponds to state 1, level 1 to state 2
# noise level is 0.3 after filtering
sim <- contMC(0.5 * sampling, 0:1, rates, sampling=sampling, family="gaussKern",
  param = list(df=df.over, over=over, sd=0.3))
plot(sim$data, pch = ".")
lines(sim$discr, col = "red")
# fit using filter corresponding to sample rate
df <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling))
fit <- jsmurf(sim$data$y, sim$data$x, param=df, r=1e2)
lines(fit, col = "blue")
# fitted values take filter into account
lines(sim$data$x, fitted(fit), col = "green3", lty = 2)
```

---

jumpint *Confidence intervals for jumps and confidence bands for step functions*

Description

Extract and plot confidence intervals and bands from fits using stepbound.

Usage

```
jumpint(sb, ...)
## S3 method for class 'stepfit'
jumpint(sb, ...)
## S3 method for class 'jumpint'
points(x, pch.left = NA, pch.right = NA, y.left = NA, y.right = NA, xpd = NA, ...)
```

```
confband(sb, ...)
## S3 method for class 'stepfit'
confband(sb, ...)
## S3 method for class 'confband'
lines(x, dataspace = TRUE, ...)
```

Arguments

sb	the result of a fit by stepbound
x	the object
pch.left, pch.right	the plotting character to use for the left/right end of the interval with defaults "(" and "]" (see parameter pch of par)
y.left, y.right	at which height to plot the interval boundaries with default par()$usr[3]
xpd	see par
dataspace	logical determining whether the expected value should be plotted instead of the fitted parameter value, useful e.g. for family = "binomial", where it will plot the fitted success probability times the number of trials per observation
...	arguments to be passed to generic methods

Value

For jumpint an object of class jumpint, i.e. a data.frame whose columns rightEndLeftBound and rightEndRightBound specify the left and right end of the confidence interval for the block's right end, resp., given the number of blocks was estimated correctly, and similarly columns rightIndexLeftBound and rightIndexRightBound specify the left and right indices of the confidence interval, resp. Function points plots these intervals on the lower horizontal axis (by default).

For confband an object of class confband, i.e. a data.frame with columns lower and upper specifying a confidence band computed at every point x; this is a simultaneous confidence band assuming the true number of jumps has been determined. Function lines plots the confidence band.

Note

Observe that jumps may occur immediately before or after an observed x; this lack of knowledge is reflected in the visual impressions by the lower and upper envelopes jumping vertically early, so that possible jumps between xs remain within the band, and by the confidence intervals starting immediately after the last x for which there cannot be a jump, cf. the note in the help for stepblock.

See Also

stepbound, points, lines

Examples

```
# simulate Bernoulli data with four blocks
y <- rbinom(200, 1, rep(c(0.1, 0.7, 0.3, 0.9), each=50))
# fit step function
sb <- stepbound(y, family="binomial", param=1, confband=TRUE)
plot(y, pch="|")
lines(sb)
# confidence intervals for jumps
jumpint(sb)
points(jumpint(sb), col="blue")
# confidence band
confband(sb)
lines(confband(sb), lty=2, col="blue")
```

MRC *Compute Multiresolution Criterion*

Description

Computes multiresolution coefficients, the corresponding criterion, simulates these for Gaussian white or coloured noise, based on which p-values and quantiles are obtained.

Usage

```
MRC(x, lengths = 2^(floor(log2(length(x))):0), norm = sqrt(lengths),
  penalty = c("none", "log", "sqrt"))
MRCoeff(x, lengths = 2^(floor(log2(length(x))):0), norm = sqrt(lengths), signed = FALSE)
MRC.simul(n, r, lengths = 2^(floor(log2(n)):0), penalty = c("none", "log", "sqrt"))
MRC.pvalue(q, n, r, lengths = 2^(floor(log2(n)):0), penalty = c("none", "log", "sqrt"),
  name = ".MRC.table", pos = .GlobalEnv, inherits = TRUE)
MRC.FFT(epsFFT, testFFT, K = matrix(TRUE, nrow(testFFT), ncol(testFFT)), lengths,
  penalty = c("none", "log", "sqrt"))
MRC.quant(p, n, r, lengths = 2^(floor(log2(n)):0), penalty = c("none", "log", "sqrt"),
  name = ".MRC.table", pos = .GlobalEnv, inherits = TRUE, ...)
kMRC.simul(n, r, kern, lengths = 2^(floor(log2(n)):ceiling(log2(length(kern)))))
kMRC.pvalue(q, n, r, kern, lengths = 2^(floor(log2(n)):ceiling(log2(length(kern)))),
  name = ".MRC.ktable", pos = .GlobalEnv, inherits = TRUE)
kMRC.quant(p, n, r, kern, lengths = 2^(floor(log2(n)):ceiling(log2(length(kern)))),
  name = ".MRC.ktable", pos = .GlobalEnv, inherits = TRUE, ...)
```

Arguments

x	a vector of numerical observations
lengths	vector of interval lengths to use, dyadic intervals by default
signed	whether signed coefficients should be returned
q	quantile

n	length of data set
r	number of simulations to use
name,pos,inherits	under which name and where precomputed results are stored, or retrieved, see assign
K	a logical matrix indicating the set of valid intervals
epsFFT	a vector containg the FFT of the data set
testFFT	a matrix containing the FFTs of the intervals
kern	a filter kernel
penalty	penalty term in the multiresolution statistic: "none" for no penalty, "log" for penalizing the log-length of an interval, and "sqrt" for penalizing the square root of the MRC; or a function taking two arguments, the first being the multiresolution coefficients, the second the interval lenghts
norm	how the partial sums should be normalised, by default sqrt(lengths), so they are normalised to equal variance across all interval lengths
p	p-value
...	further arguments passed to function quantile

Value

for MRC	a vector giving the maximum as well as the indices of the corresponding interval's start and length
for MRCoeff	a matrix giving the multiresolution coefficients for all test intervals
for MRC.pvalue / MRC.quant / MRC.simul	the corresponding p-value / quantile / vector of simulated values under the assumption of standard Gaussian white noise
for kMRC.pvalue / kMRC.simul / kMRC.simul	the corresponding p-value / quantile / vector of simulated values under the assumption of filtered Gaussian white noise

References

Davies, P.L., Kovac, A. (2001). Local extremes, runs, strings and multiresolution. *The Annals of Statistics* 29, 1-65.

Dümbgen, L., Spokoiny, V. (2001). Multiscale testing of qualitative hypotheses. *The Annals of Statistics* 29, 124-152.

Siegmund, D.O., Venkatraman, E.S. (1995). Using the generalized likelihood ratio statistic for sequential detection of a change-point. *The Annals of Statistics* 23, 255-271.

Siegmund, D.O., Yakir, B. (2000). Tail probabilities for the null distribution of scanning statistics. *Bernoulli* 6, 191-213.

See Also

smuceR, jsmurf, stepbound, stepsel, quantile

Examples

```
# simulate signal of 100 data points
set.seed(100)
f <- rep(c(0, 2, 0), c(60, 10, 30))
# add gaussian noise
x <- f + rnorm(100)
# compute multiresolution criterion
m <- MRC(x)
# compute Monte-Carlo p-value based on 100 simulations
MRC.pvalue(m["max"], length(x), 100)
# compute multiresolution coefficients
M <- MRCoeff(x)
# plot multiresolution coefficients, colours show p-values below 5% in 1% steps
op <- par(mar = c(5, 4, 2, 4) + 0.1)
image(1:length(x), seq(min(x), max(x), length = ncol(M)), apply(M[,ncol(M):1], 1:2,
  MRC.pvalue, n = length(x), r = 100), breaks = (0:5) / 100,
  col = rgb(1, seq(0, 1, length = 5), 0, 0.75),
  xlab = "location / left end of interval", ylab ="measurement",
  main = "Multiresolution Coefficients",
  sub = paste("MRC p-value =", signif(MRC.pvalue(m["max"], length(x), 100), 3)))
axis(4, min(x) + diff(range(x)) * ( pretty(1:ncol(M) - 1) ) / dim(M)[2],
  2^pretty(1:ncol(M) - 1))
mtext("interval lengths", 4, 3)
# plot signal and its mean
points(x)
lines(f, lty = 2)
abline(h = mean(x))
par(op)
```

`MRC.1000` *Values of the MRC statistic for 1,000 observations (all intervals)*

Description

Simulated values of the MRC statistic with `penalty="sqrt"` based on all interval lengths computed from Gaussian white noise sequences of length 1,000.

Usage

```
MRC.1000
```

Format

A `numeric` vector containing 10,000 sorted values.

Examples

```
# threshold value for 95% confidence
quantile(stepR::MRC.1000, .95)
```

MRC.asymptotic *"Asymptotic" values of the MRC statistic (all intervals)*

Description

Simulated values of the MRC statistic with `penalty="sqrt"` based on all interval lengths computed from Gaussian white noise sequences of ("almost infinite") length 5,000.

Usage

```
MRC.asymptotic
```

Format

A `numeric` vector containing 10,000 sorted values.

Examples

```
# "asymptotic" threshold value for 95% confidence
quantile(stepR::MRC.asymptotic, .95)
```

MRC.asymptotic.dyadic *"Asymptotic" values of the MRC statistic (dyadic intervals)*

Description

Simulated values of the MRC statistic with `penalty="sqrt"` based on dyadic interval lengths computed from Gaussian white noise sequences of ("almost infinite") length 100,000.

Usage

```
MRC.asymptotic.dyadic
```

Format

A `numeric` vector containing 10,000 sorted values.

Examples

```
# "asymptotic" threshold value for 95% confidence
quantile(stepR::MRC.asymptotic.dyadic, .95)
```

neighbours *Neighbouring integers*

Description

Find integers within some radius of the given ones.

Usage

```
neighbours(k, x = 1:max(k), r = 0)
```

Arguments

k	integers within whose neighbourhood to look
x	allowed integers
r	radius within which to look

Value

Returns those integers in x which are at most r from some integer in k, i.e. the intersection of x with the union of the balls of radius r centred at the values of k. The return values are unique and sorted.

See Also

is.element, match, findInterval, stepcand

Examples

```
neighbours(c(10, 0, 5), r = 1)
neighbours(c(10, 0, 5), 0:15, r = 1)
```

sdrobnorm *Robust standard deviation estimate*

Description

Robust estimation of the standard deviation of Gaussian data.

Usage

```
sdrobnorm(x, p = c(0.25, 0.75), lag = 1)
```

Arguments

x	a vector of numerical observations
p	vector of two distinct probabilities
lag	the lag of the difference used, see diff

Details

Compares the difference between the estimated sample quantile corresponding to p after taking (lagged) differences) with the corresponding theoretical quantiles of Gaussian white noise to determine the standard deviation under a Gaussian assumption; if the data contain (few) jumps, this will (on average) be a slight overestimate of the true standard deviation.

This estimator has been inspired by equation (1.7) in *Davies & Kovac (2001)*.

Value

Returns the estimate of the sample's standard deviation.

References

Davies, P.L., Kovac, A. (2001). Local extremes, runs, strings and multiresolution. *The Annals of Statistics* 29, 1-65.

See Also

sd, diff, family

Examples

```
# simulate data sample
y <- rnorm(100, c(rep(1, 50), rep(10, 50)), 2)
# estimate standard deviation
sdrobnorm(y)
```

---

smuceR *Piecewise constant regression with SMUCE*

Description

Computes the SMUCE estimator for one-dimensional data.

Usage

```
smuceR(y, x = 1:length(y), x0 = 2 * x[1] - x[2], q = thresh.smuceR(length(y)), alpha, r,
  lengths, family = c("gauss", "gaussvar", "poisson", "binomial"), param,
  jumpint = confband, confband = FALSE)
thresh.smuceR(v)
```

Arguments

y	a numeric vector containing the serial data
x	a numeric vector of the same length as y containing the corresponding sample points
x0	a single numeric giving the last unobserved sample point directly before sampling started
q	threshold value, by default chosen automatically according to Frick et al.~(2013)
alpha	significance level; if set to a value in (0,1), q is chosen as the corresponding quantile of the asymptotic (if r is not given) null distribution (and any value specified for q is silently ignored)
r	numer of simulations; if specified along alpha, q is chosen as the corresponding quantile of the simulated null distribution
lengths	length of intervals considered; by default up to a sample size of 1000 all lengths, otherwise only dyadic lengths
family, param	specifies distribution of data, see family
jumpint	logical (FALSE by default), indicates if confidence sets for change-points should be computed
confband	logical, indicates if a confidence band for the piecewise-continuous function should be computed
v	number of data points

Value

For smuceR, an object of class stepfit that contains the fit; if jumpint = TRUE function jumpint allows to extract the 1- alpha confidence interval for the jumps, if confband = TRUE function confband allows to extract the 1- alpha confidence band.

For thresh.smuceR, a precomputed threshhold value, see reference.

References

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. With discussion and rejoinder by the authors. *Journal of the Royal Statistical Society, Series B* 76(3), 495-580.

Futschik, A., Hotz, T., Munk, A. Sieling, H. (2014). Multiresolution DNA partitioning: statistical evidence for segments. *Bioinformatics*, 30(16), 2255-2262.

See Also

stepbound, bounds, family, MRC.asymptotic, sdrobnorm, stepfit

Examples

```
# simulate poisson data with two levels
y <- rpois(100, c(rep(1, 50), rep(4, 50)))
# compute fit, q is chosen automatically
fit <- smuceR(y, family="poisson", confband = TRUE)
```

```
# plot result
plot(y)
lines(fit)
# plot confidence intervals for jumps on axis
points(jumpint(fit), col="blue")
# confidence band
lines(confband(fit), lty=2, col="blue")

# simulate binomial data with two levels
y <- rbinom(200,3,rep(c(0.1,0.7),c(110,90)))
# compute fit, q is the 0.9-quantile of the (asymptotic) null distribution
fit <- smuceR(y, alpha=0.1, family="binomial", param=3, confband = TRUE)
# plot result
plot(y)
lines(fit)
# plot confidence intervals for jumps on axis
points(jumpint(fit), col="blue")
# confidence band
lines(confband(fit), lty=2, col="blue")
```

`stepblock` *Step function*

Description

Constructs an object containing a step function sampled over finitely many values.

Usage

```
stepblock(value, leftEnd = c(1, rightEnd[-length(rightEnd)] + 1), rightEnd, x0 = 0)
## S3 method for class 'stepblock'
x[i, j, drop = if(missing(i)) TRUE else if(missing(j)) FALSE else length(j) == 1, ...]
## S3 method for class 'stepblock'
print(x, ...)
## S3 method for class 'stepblock'
plot(x, type = "c", xlab = "x", ylab = "y", main = "Step function", sub = NULL, ...)
## S3 method for class 'stepblock'
lines(x, type = "c", ...)
```

Arguments

`value`	a numeric vector containing the fitted values for each block; its length gives the number of blocks
`leftEnd`	a numeric vector of the same length as `value` containing the left end of each block
`rightEnd`	a numeric vector of the same length as `value` containing the right end of each block

x0	a single numeric giving the last unobserved sample point directly before sampling started, i.e. before leftEnd[1]
x	the object
i, j, drop	see [.data.frame
type	"c" to plot jumps in the middle between the end of the previous block (or x0) and the beginning of the following block; "e" to jump at the end of the previous block; "b" to jump at the beginning of the following block; capital letters also plot points
xlab, ylab, main, sub	see plot.default
...	for generic methods only

Value

For stepblock an object of class stepblock, i.e. a data.frame with columns value, leftEnd and rightEnd and attribute x0.

Note

For the purposes of this package step functions are taken to be left-continuous, i.e. the function jumps **after** the rightEnd of a block.

However, step functions are usually sampled at a discrete set of points so that the exact position of the jump is unknown, except that it has to occur before the next sampling point; this is expressed in the implementation by the specification of a leftEnd **within** the block so that every rightEnd and leftEnd is a sampling point (or the boundary of the observation window), there is no sampling point between one block's rightEnd and the following block's leftEnd, while the step function is constant at least on the closed interval with boundary leftEnd, rightEnd.

See Also

step, stepfit, family, [.data.frame, plot, lines

Examples

```
# step function consisting of 3 blocks: 1 on (0, 3]; 2 on (3, 6], 0 on (6, 8]
# sampled on the integers 1:10
f <- stepblock(value = c(1, 2, 0), rightEnd = c(3, 6, 8))
f
# show different plot types
plot(f, type = "C")
lines(f, type = "E", lty = 2, col = "red")
lines(f, type = "B", lty = 3, col = "blue")
legend("bottomleft", legend = c("C", "E", "B"), lty = 1:3, col = c("black", "red", "blue"))
```

stepbound *Jump estimation under restrictions*

Description

Computes piecewise constant maximum likelihood estimators with minimal number of jumps under given restrictions on subintervals.

Usage

```
stepbound(y, bounds, ...)
## Default S3 method:
stepbound(y, bounds, x = 1:length(y), x0 = 2 * x[1] - x[2],
  max.cand = NULL, family = c("gauss", "gaussvar", "poisson", "binomial", "gaussKern"),
  param = NULL, weights = rep(1, length(y)), refit = y,
  jumpint = confband, confband = FALSE, ...)
## S3 method for class 'stepcand'
stepbound(y, bounds, refit = TRUE, ...)
```

Arguments

y	a vector of numerical observations
bounds	bounds on the value allowed on intervals; typically computed with bounds
x	a numeric vector of the same length as y containing the corresponding sample points
x0	a single numeric giving the last unobserved sample point directly before sampling started
max.cand, weights	see stepcand
family, param	specifies distribution of data, see family
refit	logical, for family = "gaussKern"; determines whether a fit taken the filter kernel into account will be computed at the end
jumpint	logical (FALSE by default), indicates if confidence sets for jumps should be computed
confband	logical, indicates if a confidence band for the piecewise-continuous function should be computed
...	arguments to be passed to generic methods

Value

An object of class stepfit that contains the fit; if jumpint = TRUE function jumpint allows to extract the confidence interval for the jumps, if confband = TRUE function confband allows to extract the confidence band.

References

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. With discussion and rejoinder by the authors. *Journal of the Royal Statistical Society, Series B* 76(3), 495-580.

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 Transactions on NanoBioscience* 12(4), 376-386.

See Also

bounds, smuceR, jsmurf, stepsel, stepfit, jumpint, confband

Examples

```
# simulate poisson data with two levels
y <- rpois(100, c(rep(1, 50), rep(4, 50)))
# compute bounds
b <- bounds(y, penalty="len", family="poisson", q=4)
# fit step function to bounds
sb <- stepbound(y, b, family="poisson", confband=TRUE)
plot(y)
lines(sb)
# plot confidence intervals for jumps on axis
points(jumpint(sb), col="blue")
# confidence band
lines(confband(sb), lty=2, col="blue")
```

---

stepcand *Forward selection of candidate jumps*

Description

Find candidates for jumps in serial data by forward selection.

Usage

```
stepcand(y, x = 1:length(y), x0 = 2 * x[1] - x[2], max.cand = NULL,
  family = c("gauss", "gaussvar", "poisson", "binomial", "gaussKern"), param = NULL,
  weights = rep(1, length(y)), cand.radius = 0)
```

Arguments

y	a numeric vector containing the serial data
x	a numeric vector of the same length as y containing the corresponding sample points
x0	a single numeric giving the last unobserved sample point directly before sampling started

max.cand	single integer giving the maximal number of blocks to find; defaults to using all data (note: there will be one block more than the number of jumps
family	distribution of the errors, either "gauss", "poisson" or "binomial"; "gaussInhibit" is like "gauss" forbids jumps getting close together or to the ends in steppath.stepcand, "gaussInhibitBoth" already forbids this in stepcand (not recommended)
param	additional parameters specifying the distribution of the errors; the number of trials for family "binomial"; for gaussInhibit and gaussInhibitBoth a numeric of length 3 with components "start", "middle" and "end" preventing the first jump from getting closer to x0 than the "start" value, any two jumps from getting closer than the "middle" value, and the last jump from geting closer than the "end" value to the end, all distances measured by weights (cf. example below)
weights	a numeric vector of the same length as y containing non-negative weights
cand.radius	a non-negative integer: adds for each candidate found all indices that are at most cand.radius away

Value

An object of class stepcand extending class stepfit such that it can be used as an input to steppath.stepcand: additionally contains columns

cumSum	The cumulative sum of x up to rightEnd.
cumSumSq	The cumulative sum of squares of x up to rightEnd (for family = "gauss").
cumSumWe	The cumulative sum of weights up to rightEnd.
improve	The improvement this jump brought about when it was selected.

See Also

steppath, stepfit, family

Examples

```
# simulate 5 blocks (4 jumps) within a total of 100 data points
b <- c(sort(sample(1:99, 4)), 100)
f <- rep(rnorm(5, 0, 4), c(b[1], diff(b)))
rbind(b = b, f = unique(f), lambda = exp(unique(f) / 10) * 20)
# add gaussian noise
x <- f + rnorm(100)
# find 10 candidate jumps
stepcand(x, max.cand = 10)
# for poisson observations
y <- rpois(100, exp(f / 10) * 20)
# find 10 candidate jumps
stepcand(y, max.cand = 10, family = "poisson")
# for binomial observations
size <- 10
z <- rbinom(100, size, pnorm(f / 10))
# find 10 candidate jumps
stepcand(z, max.cand = 10, family = "binomial", param = size)
```

stepfit *Fitted step function*

Description

Constructs an object containing a step function fitted to some data.

Usage

```
stepfit(cost, family, value, param = NULL, leftEnd, rightEnd, x0,
  leftIndex = leftEnd, rightIndex = rightEnd)
## S3 method for class 'stepfit'
x[i, j, drop = if(missing(i)) TRUE else
  if(missing(j)) FALSE else length(j) == 1, refit = FALSE]
## S3 method for class 'stepfit'
print(x, ...)
## S3 method for class 'stepfit'
plot(x, dataspace = TRUE, ...)
## S3 method for class 'stepfit'
lines(x, dataspace = TRUE, ...)
## S3 method for class 'stepfit'
fitted(object, ...)
## S3 method for class 'stepfit'
residuals(object, y, ...)
## S3 method for class 'stepfit'
logLik(object, df = NULL, nobs = object$rightIndex[nrow(object)], ...)
```

Arguments

`cost`	the value of the cost-functional used for the fit: RSS for family `gauss`, log-likelihood (up to a constant) for families `poisson` and `binomial`
`family`	distribution of the errors, either `"gauss"`, `"poisson"` or `"binomial"`
`value`	a numeric vector containing the fitted values for each block; its length gives the number of blocks
`param`	additional paramters specifying the distribution of the errors, the number of trials for family `"binomial"`
`leftEnd`	a numeric vector of the same length as `value` containing the left end of each block
`rightEnd`	a numeric vector of the same length as `value` containing the left end of each block
`x0`	a single numeric giving the last unobserved sample point directly before sampling started, i.e. before `leftEnd[0]`
`leftIndex`	a numeric vector of the same length as `value` containing the index of the sample points corresponding to the block's left end, cf. `stepcand`

rightIndex	a numeric vector of the same length as value containing the index of the sample points corresponding to the block's right end, cf. stepcand
x, object	the object
y	a numeric vector containing the data with which to compare the fit
df	the number of estimated parameters: by default the number of blocks for families poisson and binomial, one more (for the variance) for family gauss
nobs	the number of observations used for estimating
...	for generic methods only
i,j,drop	see "[.data.frame"
refit	logical; determines whether the function will be refitted after subselection, i.e. whether the selection should be interpreted as a fit with fewer jumps); in that case, for family = "gaussKern", refit needs to be set to the original data, i.e. y
dataspace	logical determining whether the expected value should be plotted instead of the fitted parameter value, useful e.g. for family = "binomial", where it will plot the fitted success probability times the number of trials per observation

Value

for stepfit	an object of class stepfit which extends stepblock, additionally containing attributes cost, family and param, as well as columns leftIndex and rightIndex
for [.stepfit	an object of class stepfit which contains the selected subset
for fitted.stepfit	a numeric vector of length rightIndex[length(rightIndex)] giving the fit at the original sample points
for residuals.stepfit	a numeric vector of length rightIndex[length(rightIndex)] giving the residuals at the original sample points
for logLik.stepfit	an object of class logLik giving the likelihood of the data given this fit, e.g. for use with AIC and stepsel; this will (incorrectly) treat family = "gaussKern" as if it were fitted with family = "gauss"
for plot.stepfit, plot.stepfit	the corresponding functions for stepblock are called

See Also

stepblock, stepbound, steppath, stepsel, family, "[.data.frame", fitted, residuals, logLik, AIC

Examples

```
# simulate 5 blocks (4 jumps) within a total of 100 data points
b <- c(sort(sample(1:99, 4)), 100)
p <- rep(runif(5), c(b[1], diff(b))) # success probabilities
```

```
# binomial observations, each with 10 trials
y <- rbinom(100, 10, p)
# find solution with 5 blocks
fit <- steppath(y, family = "binomial", param = 10)[[5]]
plot(y, ylim = c(0, 10))
lines(fit, col = "red")
# residual diagnostics for Gaussian data
yg <- rnorm(100, qnorm(p), 1)
fitg <- steppath(yg)[[5]]
plot(yg, ylim = c(0, 10))
lines(fitg, col = "red")
plot(resid(fitg, yg))
qqnorm(resid(fitg, yg))
```

---

steppath *Solution path of step-functions*

Description

Find optimal fits with step-functions having jumps at given candidate positions for all possible subset sizes.

Usage

```
steppath(y, ..., max.blocks)
## Default S3 method:
steppath(y, x = 1:length(y), x0 = 2 * x[1] - x[2], max.cand = NULL,
  family = c("gauss", "gaussvar", "poisson", "binomial", "gaussKern"), param = NULL,
   weights = rep(1, length(y)), cand.radius = 0, ..., max.blocks = max.cand)
## S3 method for class 'stepcand'
steppath(y, ..., max.blocks = sum(!is.na(y$number)))
## S3 method for class 'steppath'
x[[i]]
## S3 method for class 'steppath'
length(x)
## S3 method for class 'steppath'
print(x, ...)
## S3 method for class 'steppath'
logLik(object, df = NULL, nobs = object$cand$rightIndex[nrow(object$cand)], ...)
```

Arguments

	for steppath:
	either an object of class stepcand for steppath.stepcand or a numeric vector containing the serial data for steppath.default
x, x0, max.cand, family, param, weights, cand.radius	
y	for steppath.default which calls stepcand; see there

max.blocks	single integer giving the maximal number of blocks to find; defaults to number of candidates (note: there will be one block more than the number of jumps
...	for generic methods only for methods on a steppath object x or object:
object	the object
i	if this is an integer returns the fit with i blocks as an object of class stepcand, else the standard behaviour of a list
df	the number of estimated parameters: by default the number of blocks for families poisson and binomial, one more (for the variance) for family gauss
nobs	the number of observations used for estimating

Value

For steppath an object of class steppath, i.e. a list with components

path	A list of length length(object) where the ith element contains the best fit by a step-function having i-1 jumps (i.e. i blocks), given by the candidates indices
cost	A numeric vector of length length(object) giving the value of the cost functional corresponding to the solutions.
cand	An object of class stepcand giving the candidates among which the jumps were selected.

[[.steppath returns the fit with i blocks as an object of class stepfit; length.steppath the maximum number of blocks for which a fit has been computed. logLik.stepfit returns an object of class logLik giving the likelihood of the data given the fits corresponding to cost, e.g. for use with AIC.

References

Friedrich, F., Kempe, A., Liebscher, V., Winkler, G. (2008). Complexity penalized M-estimation: fast computation. *Journal of Computational and Graphical Statistics* 17(1), 201-224.

See Also

stepcand, stepfit, family, logLik, AIC

Examples

```
# simulate 5 blocks (4 jumps) within a total of 100 data points
b <- c(sort(sample(1:99, 4)), 100)
f <- rep(rnorm(5, 0, 4), c(b[1], diff(b)))
# add Gaussian noise
x <- f + rnorm(100)
# find 10 candidate jumps
cand <- stepcand(x, max.cand = 10)
cand
# compute solution path
path <- steppath(cand)
path
plot(x)
```

```
lines(path[[5]], col = "red")
# compare result having 5 blocks with truth
fit <- path[[5]]
fit
logLik(fit)
AIC(logLik(fit))
cbind(fit, trueRightEnd = b, trueLevel = unique(f))
# for poisson observations
y <- rpois(100, exp(f / 10) * 20)
# compute solution path, compare result having 5 blocks with truth
cbind(steppath(y, max.cand = 10, family = "poisson")[[5]],
  trueRightEnd = b, trueIntensity = exp(unique(f) / 10) * 20)
# for binomial observations
size <- 10
z <- rbinom(100, size, pnorm(f / 10))
# compute solution path, compare result having 5 blocks with truth
cbind(steppath(z, max.cand = 10, family = "binomial", param = size)[[5]],
  trueRightEnd = b, trueIntensity = pnorm(unique(f) / 10))
# an example where stepcand is not optimal but indices found are close to optimal ones
blocks <- c(rep(0, 9), 1, 3, rep(1, 9))
blocks
stepcand(blocks, max.cand = 3)[,c("rightEnd", "value", "number")]
# erroneously puts the "1" into the right block in the first step
steppath(blocks)[[3]][,c("rightEnd", "value")]
# putting the "1" in the middle block is optimal
steppath(blocks, max.cand = 3, cand.radius = 1)[[3]][,c("rightEnd", "value")]
# also looking in the 1-neighbourhood remedies the problem
```

`stepsel` *Automatic selection of number of jumps*

Description

Select the number of jumps.

Usage

```
stepsel(path, y, type = c("MRC", "AIC", "BIC"), ...)
stepsel.MRC(path, y, q, alpha = 0.05, r = ceiling(50 / min(alpha, 1 - alpha)),
  lengths = if(attr(path$cand, "family") == "gaussKern")
   2^(floor(log2(length(y))):ceiling(log2(length(attr(path$cand, "param")$kern)))) else
    2^(floor(log2(length(y))):0),
  penalty = c("none", "log", "sqrt"), name = if(attr(path$cand, "family") == "gaussKern")
    ".MRC.ktable" else ".MRC.table",
  pos = .GlobalEnv)
stepsel.AIC(path, ...)
stepsel.BIC(path, ...)
```

Arguments

path	an object of class steppath
y	for type=MRC only: a numeric vector containing the serial data
type	how to select, dispatches specific method
...	further argument passed to specific method
q, alpha, r, lengths, penalty, name, pos	see bounds

Value

A single integer giving the number of blocks selected, with attribute crit containing the values of the criterion (MRC / AIC / BIC) for each fit in the path.

Note

To obtain the threshold described in Boysen et al.~(2009, Theorem~5), set q=(1+delta) * sdrobnorm(y) * sqrt(2*lengt for some positive dela and penalty="none".

References

Boysen, L., Kempe, A., Liebscher, V., Munk, A., Wittich, O. (2009). Consistencies and rates of convergence of jump-penalized least squares estimators. *The Annals of Statistics* 37(1), 157-183.

Yao, Y.-C. (1988). Estimating the number of change-points via Schwarz' criterion. *Statistics & Probability Letters* 6, 181-189.

See Also

steppath, stepfit, family, stepbound

Examples

```
# simulate 5 blocks (4 jumps) within a total of 100 data points
b <- c(sort(sample(1:99, 4)), 100)
f <- rep(rnorm(5, 0, 4), c(b[1], diff(b)))
rbind(b = b, f = unique(f))
# add gaussian noise
y <- f + rnorm(100)
# find 10 candidate jumps
path <- steppath(y, max.cand = 10)
# select number of jumps by simulated MRC with sqrt-penalty
# thresholded with positive delta, and by BIC
sel.MRC <- stepsel(path, y, "MRC", alpha = 0.05, r = 1e2, penalty = "sqrt")
sel.MRC
delta <- .1
sel.delta <- stepsel(path, y, "MRC",
  q = (1 + delta) * sdrobnorm(y) * sqrt(2 * length(y)), penalty = "none")
sel.delta
sel.BIC <- stepsel(path, type="BIC")
sel.BIC
```

```
# compare results with truth
fit.MRC <- path[[sel.MRC]]
as.data.frame(fit.MRC)
as.data.frame(path[[sel.delta]])
as.data.frame(path[[sel.BIC]])
```

`transit` *TRANSIT algorithm for detecting jumps*

Description

Reimplementation of VanDongen's algorithm for detecting jumps in ion channel recordings.

Usage

```
transit(y, x = 1:length(y), x0 = 2 * x[1] - x[2], sigma.amp = NA, sigma.slope = NA,
  amp.thresh = 3, slope.thresh = 2, rel.amp.n = 3, rel.amp.thresh = 4,
  family = c("gauss", "gaussKern"), param = NULL, refit = FALSE)
```

Arguments

`y`	a numeric vector containing the serial data
`sigma.amp`	amplitude (i.e. raw data within block) standard deviation; estimated using `sdrobnorm` if omitted
`sigma.slope`	slope (i.e. central difference within block) standard deviation; estimated using `sdrobnorm` if omitted
`amp.thresh`	amplitude threshold
`slope.thresh`	slope threshold
`rel.amp.n`	relative amplitude threshold will be used for blocks with no more datapoints than this
`rel.amp.thresh`	relative amplitude threshold
`x`	a numeric vector of the same length as `y` containing the corresponding sample points
`x0`	a single numeric giving the last unobserved sample point directly before sampling started
`family, param`	specifies distribution of data, see family
`refit`	should the values for `family = "gaussKern"` be obtained by fitting in the end (otherwise they are meaningless)

Value

Returns an object of class `stepfit` which encodes the jumps and corresponding mean values.

Note

Only central, no forward differences have been used in this implementation. Moreover, the standard deviations will be estimated by sdrobnorm if omitted (respecting the filter's effect if applicable).

References

VanDongen, A.M.J. (1996). A New Algorithm for Idealizing Single Ion Channel Data Containing Multiple Unknown Conductance Levels. *Biophysical Journal* 70(3), 1303-1315.

See Also

stepfit, sdrobnorm, jsmurf, stepbound, steppath

Examples

```
# estimating step-functions with Gaussian white noise added
# simulate a Gaussian hidden Markov model of length 1000 with 2 states
# with identical transition rates 0.01, and signal-to-noise ratio 2
sim <- contMC(1e3, 0:1, matrix(c(0, 0.01, 0.01, 0), 2), param=1/2)
plot(sim$data, cex = 0.1)
lines(sim$cont, col="red")
# maximum-likelihood estimation under multiresolution constraints
fit.MRC <- smuceR(sim$data$y, sim$data$x)
lines(fit.MRC, col="blue")
# choose number of jumps using BIC
path <- steppath(sim$data$y, sim$data$x, max.blocks=1e2)
fit.BIC <- path[[stepsel.BIC(path)]]
lines(fit.BIC, col="green3", lty = 2)

# estimate after filtering
# simulate filtered ion channel recording with two states
set.seed(9)
# sampling rate 10 kHz
sampling <- 1e4
# tenfold oversampling
over <- 10
# 1 kHz 4-pole Bessel-filter, adjusted for oversampling
cutoff <- 1e3
df.over <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling / over))
# two states, leaving state 1 at 10 Hz, state 2 at 20 Hz
rates <- rbind(c(0, 10), c(20, 0))
# simulate 0.5 s, level 0 corresponds to state 1, level 1 to state 2
# noise level is 0.3 after filtering
Sim <- contMC(0.5 * sampling, 0:1, rates, sampling=sampling, family="gaussKern",
  param = list(df=df.over, over=over, sd=0.3))
plot(Sim$data, pch = ".")
lines(Sim$discr, col = "red")
# fit under multiresolution constraints using filter corresponding to sample rate
df <- dfilter("bessel", list(pole=4, cutoff=cutoff / sampling))
Fit.MRC <- jsmurf(Sim$data$y, Sim$data$x, param=df, r=1e2)
lines(Fit.MRC, col = "blue")
# fit using TRANSIT
```

```
Fit.trans <- transit(Sim$data$y, Sim$data$x)
lines(Fit.trans, col = "green3", lty=2)
```

Index