Package 'svpChange'

November 17, 2026

Type Package

Title Smallest Valid Partitioning for Change-Point Detection

Version 0.2.1

Author Vincent Runge [aut, cre, cph],
Alexandre Combeau [aut],
Gaetano Romano [aut],
Anica Kostic [ctb]

Maintainer Vincent Runge <vincent.runge@univ-evry.fr>

Description Provides methods for detecting multiple change-points and segmenting univariate time series using Smallest Valid Partitioning (SVP). SVP searches for a partition with the smallest number of segments whose segments satisfy a user-defined or built-in validity test. Among partitions with the same number of segments, it minimizes a within-segment sum-of-squared-errors criterion.

License GPL-3

Encoding UTF-8

URL https://github.com/vrunge/svpChange

BugReports https://github.com/vrunge/svpChange/issues

Imports Rcpp (>= 1.0.10), stats

Suggests stepR, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-11-17 18:50:02 UTC

Contents

AR1_rho . . . 2
AR1_single_change . . . 3
OP . . . 4
PELT . . . 5
smuce_cost . . . 6
SN . . . 7
SVP . . . 8
svp0 . . . 15
svpChange . . . 16
SVP_both_dyadic . . . 18
svp_smuce . . . 18
svp_smuce_cpp . . . 19
svp_smuce_dyadic_cpp . . . 20
ts_generator . . . 21
valid_AR1 . . . 23
valid_FOCUS . . . 24
valid_FOCUS_last . . . 25
valid_OP . . . 26
valid_QUANTILE . . . 27
valid_RANGE . . . 27
valid_RANGE_SLACK . . . 28
valid_SCALE . . . 29
valid_SSE . . . 30

Index **31**

`AR1_rho` *Robust AR(1) Autocorrelation Estimate*

Description

Computes the robust estimator from equation (3) of Chakar et al. (2017), using the squared ratio of the medians of absolute lag-two and lag-one differences, minus one.

Usage

```
AR1_rho(data)
```

Arguments

`data`	A numeric vector with at least three observations.

Value

A stationary AR(1) coefficient in the interval `[-0.999, 0.999]`.

References

Chakar, S., Lebarbier, E., Levy-Leduc, C. and Robin, S. (2017). A robust approach for estimating change-points in the mean of an AR(1) process. *Bernoulli*, 23(2), 1408–1447.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 60, parameters = 0, sd_noise = 1,
  rho = 0.6, type = "gaussAR1"
)
AR1_rho(data)
```

AR1_single_change *Single Mean Change Test for an AR(1) Process*

Description

Computes the conditional fixed-rho likelihood-ratio scan for a change in the marginal mean of an AR(1) process, conditional on the first observation. The residual crossing the candidate change uses both segment means. With known innovation variance, the statistic is the full RSS reduction divided by sigma2.

Usage

```
AR1_single_change(
  data,
  gamma,
  rho = NA_real_,
  sigma2 = 1,
  profile_sigma = FALSE
)
```

Arguments

data	A numeric AR(1) series with at least three observations.
gamma	The FOCuS validity threshold.
rho	An optional known AR(1) coefficient. If missing, it is estimated robustly.
sigma2	A positive innovation variance used for the known-variance likelihood-ratio statistic. It is ignored when profile_sigma = TRUE.
profile_sigma	If TRUE, profile the innovation variance and use the log RSS-ratio statistic; in this case, sigma2 is ignored.

Value

A list with the coefficient and variance used, test statistic, null and alternative residual sums of squares, estimated changepoint, and a logical validity indicator.

References

Chakar, S., Lebarbier, E., Levy-Leduc, C. and Robin, S. (2017). A robust approach for estimating change-points in the mean of an AR(1) process. *Bernoulli*, 23(2), 1408–1447.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(30, 60), parameters = c(0, 2), sd_noise = 1,
  rho = 0.6, type = "gaussAR1"
)
AR1_single_change(data, gamma = 20, rho = 0.6, sigma2 = 1)
```

OP *Optimal Partitioning Algorithm*

Description

Finds the least-squares segmentation minimizing the sum of within-segment squared errors plus `penalty` times the number of estimated change points.

Usage

```
OP(data, penalty)
```

Arguments

`data`	A numeric vector representing the data to segment.
`penalty`	Numeric penalty applied to each estimated change point.

Details

Optimal Partitioning Algorithm

A candidate boundary `s` and endpoint `t` represent the R segment `data[(s + 1):t]`. Setting the initial cost to `-penalty` makes the total penalty equal to `penalty * (K - 1)` for a partition with `K` segments.

Value

A list with the following components:

changepoints Increasing, one-based, inclusive segment endpoints, including `length(data)`.

lastIndexSet Always `NULL`; OP does not prune candidates.

nb Always `NULL`; OP does not prune candidates.

costQ Numeric vector of length `length(data)`. Element `t` is the minimum penalized cost for `data[1:t]`.

See Also

PELT() for the pruned version of the same objective, SN() for fixed numbers of segments, and SVP() for validity-constrained partitioning.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(40, 80, 120), parameters = c(0, 2, -1),
  sd_noise = 1, type = "gauss"
)
penalty <- 2 * log(length(data))
OPres <- OP(data, penalty)
OPres$changepoints
```

PELT *Optimal Partitioning using PELT*

Description

Finds the same penalized least-squares segmentation as OP(), while using the PELT rule to prune candidate boundaries.

Usage

```
PELT(data, penalty)
```

Arguments

data	A numeric vector representing the data to segment.
penalty	Numeric penalty applied to each estimated change point.

Details

Optimal Partitioning algorithm using PELT

A candidate boundary s and endpoint t represent the R segment data[(s + 1):t]. Setting the initial cost to -penalty makes the total penalty equal to penalty * (K - 1) for a partition with K segments. The pruning changes the candidate set, not the optimized objective.

Value

A list with the following components:

changepoints Increasing, one-based, inclusive segment endpoints, including length(data).

lastIndexSet Zero-based candidate boundaries remaining after the final iteration, returned in decreasing order and including length(data).

nb Numeric vector in time order. Element `t` is the number of candidates examined at endpoint `t`, before pruning.

costQ Numeric vector of length `length(data)`. Element `t` is the minimum penalized cost for `data[1:t]`.

See Also

`OP()` for the unpruned version of the same objective, `SN()` for fixed numbers of segments, and `SVP()` for validity-constrained partitioning.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(40, 80, 120), parameters = c(0, 2, -1),
  sd_noise = 1, type = "gauss"
)
penalty <- 2 * log(length(data))
resPELT <- PELT(data, penalty)
```

---

smuce_cost *Constrained Gaussian cost for a SMUCE-valid segment*

Description

Constrained Gaussian cost for a SMUCE-valid segment

Usage

```
smuce_cost(y, gamma, sigma2 = 1, n = length(y))
```

Arguments

`y`	Numeric observations in the candidate segment.
`gamma`	SMUCE threshold.
`sigma2`	Known Gaussian variance.
`n`	Total series length.

Details

The cost is the Gaussian residual sum of squares divided by `sigma2`, minimized over the SMUCE-admissible constant means. It is therefore the constrained Gaussian negative log-likelihood cost up to an additive and multiplicative constant. `Inf` is returned when the segment has no admissible constant mean.

Value

Numeric constrained Gaussian residual cost.

References

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. *Journal of the Royal Statistical Society: Series B*, 76(3), 495–580. doi:10.1111/rssb.12047.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 20, parameters = 0, sd_noise = 1, type = "gauss"
)
smuce_cost(data, gamma = 2, sigma2 = 1)
```

SN *Segment Neighborhood*

Description

Finds the least-squares segmentation for every fixed number of segments from 1 through `Kmax`, using dynamic programming.

Usage

```
SN(data, Kmax)
```

Arguments

`data`	A numeric vector representing the data to segment.
`Kmax`	Maximum number of segments. It should be an integer between 1 and `length(data)`.

Details

Segment Neighborhood

A candidate boundary `s` and endpoint `t` represent the R segment `data[(s + 1):t]`. The segment cost is its sum of squared errors around its sample mean.

Value

A list with the following components:

changepoints List of length `Kmax`. Element `k` contains the `k` increasing, one-based, inclusive segment endpoints of the optimal `k`-segment partition, including `length(data)`.

lastIndexSet Always `NULL`; SN does not prune candidates.

nb Always `NULL`; SN does not prune candidates.

costQ Numeric matrix with `length(data) + 1` rows and `Kmax` columns. Entry `[t + 1, k]` is the minimum cost for `data[1:t]` with exactly `k` segments; infeasible entries are `Inf`.

See Also

OP() and PELT() for penalized segmentation, and SVP() for validity-constrained partitioning.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(20, 40), parameters = c(0, 4),
  sd_noise = 0.5, type = "gauss"
)
fit <- SN(data, Kmax = 2)
fit$changepoints[[2]]
fit$costQ[nrow(fit$costQ), ]
```

SVP *Smallest Valid Partitioning with Incremental Validity Tests*

Description

Segments a univariate series into the smallest number of segments that pass a selected validity test. A candidate segment is evaluated only at its current endpoint; its incremental state receives intervening observations, but earlier endpoints are not tested again. With `subtests = "none"`, an invalid endpoint does not remove the candidate and it can recover later. With `subtests = "right"` or `subtests = "both"`, it is removed immediately. With `subtests = "both"`, an invalid boundary also removes that boundary and every older boundary under the inclusive left-pruning rule. `subtests = "both_msl"`, `subtests = "both_ms_refined"`, and `subtests = "both_focus_lil"` use this pruning rule with multiscale Gaussian statistics. Among partitions with the same number of segments, it chooses the one with the smallest within-segment cost, selected with `cost`. Validity statistics are maintained incrementally in C++.

Usage

```
SVP(
  data,
  gamma,
  test = "gaussian_mean",
  subtests = "both",
  sigma2 = 1,
  rho = NA_real_,
  profile_sigma = FALSE,
  quantile = 0.01,
  cost = "auto",
  r_min = 1L,
  r_min_frac = 0,
  abc_A = NA_real_,
  abc_B = NA_real_,
  abc_C = NA_real_
)
```

Arguments

data	Numeric vector containing the univariate series. Missing and non-finite values are not supported.
gamma	Finite scalar validity threshold, positive except when subtests is "verzelen", "both_msl", "both_ms_refined", "both_focus_lil", "both_rmin", "both_calibrated", "both_correction", "both_empirical", "both_lengthonly", "both_mosum_sqrt", or "both_mosum_adaptive". A segment is valid when its statistic is less than or equal to gamma. Larger values generally accept longer or less homogeneous segments.
test	Character scalar selecting a validity test; see Details. Defaults to "gaussian_mean".
subtests	Character scalar selecting the validity-pruning rules: "both" (default), "right", "none", "verzelen", "both_msl", "both_ms_refined", "both_focus_lil", "both_rmin", "both_calibrated", "both_correction", "both_empirical", "both_lengthonly", "both_mosum_sqrt", "both_mosum", or "both_mosum_adaptive". All options after "none" require test = "gaussian_mean" and select multiscale, minimum-arm-restricted, length-adaptive, or moving-sum (MOSUM) Gaussian tests; see Details. "both_empirical" requires data of length 1000.
sigma2	Positive finite innovation variance for fixed-variance AR1 tests ("AR1" with profile_sigma = FALSE, and "AR1Focus"), or known observation noise variance for the multiscale Gaussian subtests options (every option after "none") (default 1). For AR(1), it is the variance of the new shock, not the marginal variance of the observed series. It is ignored by "AR1Profile" and by "AR1" when profile_sigma = TRUE.
rho	AR(1) coefficient for the AR1 tests. It must be finite and strictly between -1 and 1. If it is NA, it is estimated robustly from the full series using AR1_rho().
profile_sigma	If TRUE, estimate the innovation variance separately under the no-change and one-change AR(1) models when test = "AR1". This removes the need to know the innovation scale and uses the resulting residual sums of squares in the likelihood-ratio statistic. Using test = "AR1Profile" has the same effect. The argument is ignored by other tests.
quantile	Quantile level used by "quantile" and "quantileExact"; ignored by other tests.
cost	Character scalar selecting the within-segment cost used to rank partitions with the same number of segments: "auto" (default), "gaussian", or "ar1". See Details. "ar1" uses rho, estimating it from the full series with AR1_rho() when it is NA.
r_min	Positive integer absolute minimum arm length for subtests = "both_rmin". Ignored by every other subtests value.
r_min_frac	Finite scalar in [0, 0.5), the minimum arm length for subtests = "both_rmin" as a fraction of the interval's current length, combined with r_min by taking the larger of the two floors. Zero (default) disables the relative floor. Ignored by every other subtests value.
abc_A, abc_B, abc_C	Coefficients of the length-dependent penalty abc_A * log(m) + abc_B * log(log(m)) + abc_C subtracted from the Gaussian statistic of a candidate segment of length m

when `subtests = "both_calibrated"`. `NA` (default) uses the built-in calibrated values 1.682969, 6.208626 and 4.939248. Ignored by every other `subtests` value.

Details

Verzelen multiscale validity: With `test = "gaussian_mean"`, `subtests = "verzelen"`, a standalone scanner tests every pair of adjacent nonempty intervals contained in the candidate. Write N for the full data length and $C(a,b,c) = \sqrt{(b-a)(c-b)/(c-a)}(\bar{y}_{a+1:b} - \bar{y}_{b+1:c})$. The candidate $(s,t]$ is valid iff

$$V_s(t) = \max_{s \le a < b < c \le t} \left\{ C(a,b,c)^2/(4\sigma^2) - 2\log\left(\frac{N(c-a)}{(b-a)(c-b)}\right)\right\} \le \gamma.$$

Singletons have statistic minus infinity and are always valid. The global length N stays fixed for shorter segments. Each candidate maintains its current maximum using $V_s(t) = \max\{V_s(t-1), V_{s+1}(t), H(s,t)\}$, where $H(s,t)$ scans all splits with outer boundaries s,t. Containment justifies permanent pruning in both directions. FOCUS bounds skip scans that cannot improve the maximum; all splits are checked otherwise. Worst-case total time is cubic in N, with quadratic worst-case state storage.

This rule is inspired by the simultaneous CUSUM noise bound on slide 15 (PDF page 38) of Nicolas Verzelen's 2023 workshop presentation *Optimal multiple change-point detection and Localization*. It implements the bound with factor 4 and known noise variance `sigma2`. A finite negative `gamma` is allowed for empirical null calibration; such thresholds do not inherit the slide's positive-parameter probability bound. `gamma` is a direct threshold, not a multiplier of `log(N)`.

MS-l multiscale validity: With `test = "gaussian_mean"`, `subtests = "both_msl"`, consider every interval of length ℓ within a candidate segment. At split τ, let r_τ be the shorter side and Z_τ the Gaussian two-sample CUSUM. The interval passes if

$$\max_{\tau} \left\{ |Z_\tau|/\sqrt{\sigma^2} - \sqrt{2\log(e\ell/r_\tau)} \right\} \le \gamma.$$

The `"both"` pruning rule makes a candidate valid only if every contained interval passes. Singletons always pass. `gamma` is the direct threshold q, without a `log(n)` multiplier. A FOCUS bound skips exact split scans when the interval must pass; all potentially failing splits are checked exactly.

Refined MS-l multiscale validity: With `test = "gaussian_mean"`, `subtests = "both_ms_refined"`, use the same contained-interval scan and `"both"` pruning as MS-l, but replace its split penalty by

$$\sqrt{2\{1 + \log(\ell/r_\tau) + \tfrac{1}{2}\log(n/\ell)\}}.$$

Here n is the full series length, ℓ the interval length, and r_τ the shorter side of the split. The coefficient $1/2$ was selected in the separate Gaussian penalty study. The direct threshold γ is calibrated to the null distribution; for $n = 1000$, $\sigma^2 = 1$, and 1% nominal false detection, the study obtained $\gamma = 2.70947832002059$ from 5,000 null series and checked it on 2,000 independent null series. Other lengths or noise models require their own calibration.

FOCUS iterated-logarithm CUSUM: With `test = "gaussian_mean"`, `subtests = "both_focus_lil"`, consider a contained interval of length m and a split with lengths l and $r = m - l$. Define

$$h_m(l) = \max\{\min(l, m/l), \min(r, m/r)\}, \quad p_m(l) = 2\log\log(e h_m(l)).$$

The maximum of $h_m(l)$ over splits is

$$h_m^* = \max\{\lfloor\sqrt{m}\rfloor, m/\lceil\sqrt{m}\rceil\}.$$

The interval statistic is

$$\max_l Z_l^2/\sigma^2 - 8\log\log(eh_m^*),$$

where Z_l is the unstandardized two-sample CUSUM. FOCUS computes $\max_l Z_l^2$ exactly, since the penalty depends only on m. This statistic is no larger than the position-penalized statistic in Eq. (12) and Proposition 2 of Verzelen, Fromont, Lerasle and Reynaud-Bouret (2023), *Annals of Statistics* 51, 1586–1610, doi:10.1214/23AOS2297, taking their permitted $L = 2$. Consequently their single-interval level guarantee applies to this test. A candidate segment is valid when every contained interval has statistic no greater than `gamma`. The proposition does not directly cover the scan across all contained intervals, so calibrate `gamma` on the whole-series null maximum.

The available validity tests are:

- `"gaussian_mean"`: Gaussian FOCUS likelihood-ratio statistic evaluated at the current segment endpoint for a change in mean. Use for independent Gaussian data with constant variance. To require every prefix to be valid, use `svp0()` with `valid_FOCUS()`.
- `"gamma_rate"`: likelihood-ratio test for a rate change in positive Gamma data (unit shape in the current implementation).
- `"gaussian_variance"`: the Gamma-rate test applied to squared observations, for variance changes around a known zero mean.
- `"AR1"`: conditional fixed-`rho` Gaussian likelihood-ratio scan for a change in the marginal mean of an AR(1) series, conditional on the first observation. In the AR(1) model, an innovation is the new random shock after accounting for the previous observation and the AR(1) mean structure; `sigma2` is the variance of this shock, not the marginal variance of the observations.
- `"AR1Profile"`: the same conditional AR(1) scan, but profiling out the innovation variance. The variance is estimated separately under the no-change and change models from their residual sums of squares, which is useful when the innovation scale is unknown.
- `"AR1Focus"`: faster approximation applying Gaussian FOCUS to innovations `x[t] - rho * x[t - 1]`. Prefer `"AR1"` when exact boundary treatment matters.
- `"quantile"` and `"quantileExact"`: quantile-based robust tests, using `quantile` as their quantile level.
- `"varCost"`: robust variance cost test.
- `"WilcoxonCost"`: Wilcoxon rank-based cost test.
- `"MedianMoodCost"`: Median-Mood rank-based cost test.

Per-candidate update complexity: Let m be the current length of one candidate segment and let a_m be the number of active functional-pruning pieces, with $a_m \leq m$. A state update incorporates one new observation. A complete endpoint check also computes the statistic needed to decide validity. These are per-candidate costs; total SVP time also depends on the number of candidates examined under the selected `subtests` mode.

Test	State update	Update and endpoint check
`"gaussian_mean"`	amortized $O(1)$, worst $O(m)$	expected $O(\log m)$, worst $O(r$

"gaussian_mean", "both_focus_lil"	amortized $O(1)$, worst $O(m)$	exact FOCUS maximum, expe
"gaussian_mean", "both_msl" or "both_ms_refined"	amortized $O(1)$, worst $O(m)$	FOCUS bound, exact $O(m)$ sc
"gamma_rate"	$O(a_m)$, worst $O(m)$	$O(a_m)$, worst $O(m)$
"gaussian_variance"	$O(a_m)$, worst $O(m)$	$O(a_m)$, worst $O(m)$
"AR1"	$O(1)$	$O(m)$
"AR1Profile"	$O(1)$	$O(m)$
"AR1Focus"	$O(a_m)$, worst $O(m)$	$O(a_m)$, worst $O(m)$
"quantile"	$O(1)$	$O(1)$
"quantileExact"	expected $O(\log m)$, worst $O(m)$	expected $O(\log m)$, worst $O(r$
"varCost"	$O(1)$	$O(1)$
"WilcoxonCost"	$O(m)$	$O(m)$
"MedianMoodCost"	expected $O(\log m)$, worst $O(m)$	$O(m)$

For the Gaussian FOCuS test, the standard FOCuS result gives $E[a_m] = O(\log m)$ under independent continuous noise with a constant or single-change mean. Its complete endpoint update is therefore expected $O(\log m)$ and worst-case $O(m)$. The recurrence itself is amortized $O(1)$; the logarithmic term comes from maximizing over the active pieces. The Gamma, Gaussian-variance, and AR1Focus implementations use the same active-piece strategy and cost $O(a_m)$, but this package does not claim the Gaussian theorem for those different data models without additional model-specific assumptions.

The genuinely constant-time complete endpoint updates are `"quantile"` and `"varCost"`. `"quantileExact"` has expected logarithmic cost. Functional-pruning tests can be fast when a_m remains small, but their worst-case endpoint cost is linear.

For an AR(1) series generated by `ts_generator(type = "gaussAR1")`, the model is `x[t] = mu[t] + e[t]`, with `e[t] = rho * e[t - 1] + eta[t]` and `eta[t] ~ N(0, sigma2)`. Thus, `rho` controls serial dependence and `sigma2` is the innovation variance, that is, the variance of the new shock after accounting for the previous observation. It is not the marginal variance of the observed series; for a stationary AR(1) noise process, the marginal variance is `sigma2 / (1 - rho^2)`. In `ts_generator()`, `sd_noise` is the innovation standard deviation, so use `sigma2 = sd_noise^2`.

`"AR1"` treats `rho` and `sigma2` as fixed and uses the exact conditional Gaussian likelihood scan given the first observation. `"AR1Profile"` uses the same conditional scan but estimates the innovation variance separately under the no-change and change models. This is useful when the innovation scale is unknown. `"AR1Focus"` is faster because it applies the Gaussian FOCUS calculation to the transformed innovations; it is an approximation and can produce different boundaries near a change point.

Within-segment cost: The validity test fixes the number of segments; `cost` decides between the partitions that attain it, so it controls where the boundaries are placed and not how many there are. Two cost series are available:

- `"gaussian"`: the sum of squared deviations of `data[(s + 1):t]` from the segment mean. This is the right choice for independent observations.
- `"ar1"`: the same quantity for the innovations `z[u] = data[u] - rho * data[u - 1]`. Inside a segment with constant mean `mu` these have constant mean `(1 - rho) * mu`, so their sum of squared deviations is the profiled conditional cost of the AR(1) mean model. The innovation `z[s + 1]` that straddles the boundary is skipped, because its mean is `mu_new - rho * mu_old` rather than `(1 - rho) * mu_new`; keeping it would charge every segment for the jump in front

of it and pull the boundary away from the change. Costs are only ever compared across partitions with the same number of segments K, and every such partition skips exactly K - 1 innovations, so the comparison remains fair.

"auto", the default, uses "ar1" for "AR1", "AR1Profile", and "AR1Focus", and "gaussian" for every other test. Under serial dependence the Gaussian cost is misspecified, so an AR(1) test combined with the Gaussian cost detects the right number of changes but places them less accurately. Selecting "ar1" with a non-AR(1) test is allowed and uses rho in the same way.

The subtests argument selects validity-based candidate pruning. "right" removes a candidate when its current endpoint is invalid; when the one-segment candidate is valid, later candidates are deferred until it fails. If the shortcut ends, deferred candidates are replayed through the missed endpoints so that the right-pruning history is preserved; after that, they are tested only at the current endpoint. "both" additionally removes that boundary and every older boundary; candidates are examined from newest to oldest and the scan stops at the first invalid boundary. "none" retains all candidates and groups them by their predecessor segment count, stopping after the first group containing a valid candidate. Invalid candidates are never used in the current optimum. Pruning is exact only when the selected validity statistic has the corresponding monotonicity properties.

The available nonparametric and robust cost tests are "quantile", "quantileExact", "varCost", "WilcoxonCost", and "MedianMoodCost". They are selected through the same test argument and use the same pruning options as the Gaussian tests. The nb output records the number of active candidates at the beginning of each endpoint iteration, before pruning.

Value

A list containing:

changepoints Inclusive end of every segment, including length(data).

lastIndexSet Candidate boundaries remaining at termination.

nb Number of active candidate boundaries at the beginning of each endpoint iteration, before pruning.

costQ Currently NULL.

R Dynamic-programming matrix. Row t contains the best cumulative cost on the scale selected by cost, the number of segments, and the previous boundary for data[1:t].

rho, sigma2 Returned by the three AR(1) tests only.

See Also

svp0, AR1_rho, AR1_single_change

Examples

```
# Gaussian mean: FOCuS test for independent Gaussian observations.
set.seed(1)
gaussian_data <- ts_generator(
  chpts = c(20, 40, 60), parameters = c(0, 2, -1),
  sd_noise = 1, type = "gauss"
)
SVP(gaussian_data, gamma = 4 * log(length(gaussian_data)),
```

```
    test = "gaussian_mean")$changepoints

# Gamma rate: positive exponential observations with changing rates.
gamma_data <- ts_generator(
  chpts = c(20, 40, 60), parameters = c(1, 4, 2),
  type = "exp"
)
SVP(gamma_data, gamma = 2 * log(length(gamma_data)),
    test = "gamma_rate")$changepoints

# Gaussian variance: parameters are segment standard deviations.
variance_data <- ts_generator(
  chpts = c(20, 40, 60), parameters = c(0.5, 1.5, 0.75),
  type = "variance"
)
SVP(variance_data, gamma = 2 * log(length(variance_data)),
    test = "gaussian_variance")$changepoints

# Quantile and exact quantile tests: robust tests for changes in spread.
quantile_data <- ts_generator(
  chpts = c(20, 40, 60), parameters = c(0.5, 1.5, 0.75),
  type = "variance"
)
SVP(quantile_data, gamma = 2, quantile = 0.1,
    test = "quantile")$changepoints
SVP(quantile_data, gamma = 2, quantile = 0.1,
    test = "quantileExact")$changepoints

# Robust variance, Wilcoxon, and Median-Mood tests.
robust_data <- ts_generator(
  chpts = c(20, 40, 60), parameters = c(0, 2, -1),
  sd_noise = 1, type = "gauss"
)
SVP(robust_data, gamma = 2, test = "varCost")$changepoints
SVP(robust_data, gamma = 2 * log(length(robust_data)),
    test = "WilcoxonCost")$changepoints
SVP(robust_data, gamma = 2 * log(length(robust_data)),
    test = "MedianMoodCost")$changepoints

# Exact AR(1), with the known innovation variance.
ar1_data <- ts_generator(
  chpts = c(20, 40, 60), parameters = c(0, 2, -1),
  sd_noise = 0.8, rho = 0.7, type = "gaussAR1"
)
SVP(ar1_data, gamma = 4 * log(length(ar1_data)), test = "AR1",
    rho = 0.7, sigma2 = 0.8^2)$changepoints

# Exact AR(1) with the innovation variance profiled out.
SVP(ar1_data, gamma = 2 * log(length(ar1_data)),
    test = "AR1Profile", rho = 0.7, sigma2 = 1)$changepoints

# Faster approximate AR(1) FOCUS test on the transformed innovations.
SVP(ar1_data, gamma = 2 * log(length(ar1_data)),
```

```
    test = "AR1Focus", rho = 0.7, sigma2 = 0.8^2)$changepoints

# The same test ranking partitions with the Gaussian cost instead, which keeps
# the number of segments but can move the boundaries.
SVP(ar1_data, gamma = 2 * log(length(ar1_data)),
    test = "AR1Focus", rho = 0.7, sigma2 = 0.8^2,
    cost = "gaussian")$changepoints
```

`svp0` *Smallest Valid Partitioning with a User-Defined Validity Test*

Description

This function uses dynamic programming to find a partition of a univariate signal whose segments all pass a user-defined validity test. It first minimizes the number of segments and, among partitions with the same number of segments, minimizes the within-segment sum of squares.

Usage

```
svp0(data, gamma, test, subtests = "both", PELT_pruning = FALSE)
```

Arguments

`data`	Numeric vector containing the univariate signal to segment.
`gamma`	Numeric threshold passed to `test`.
`test`	Function of the form `function(segment, gamma)` returning one non-missing logical value: `TRUE` if the segment is valid. The function is not called for singleton segments; they are always valid.
`subtests`	Character scalar controlling validity-based pruning. The choices are `"both"` (default), `"right"`, and `"none"`. `"right"` removes a candidate start when its current segment is invalid; `"both"` additionally removes that start and all older starts; and `"none"` applies neither rule. Invalid candidates are never used for the current optimum.
`PELT_pruning`	Logical; whether to apply the additional cost-based candidate pruning rule.

Details

Smallest Valid Partitioning with a User-Defined Validity Test

A candidate boundary `s` at endpoint `t` represents the R segment `data[(s + 1):t]`; `s` is zero-based and `t` is one-based. The quadratic cost is the residual sum of squares around the segment mean. Singleton segments are always valid, regardless of the result of `test`.

Value

A list with the following components:

changepoints Numeric vector of increasing segment-ending positions, including `length(data)`. Internal values are estimated change-point positions.

lastIndexSet Numeric vector of candidate boundaries remaining at termination. These are zero-based boundaries and are returned in decreasing order.

nb Numeric vector in time order. Element `t` is the number of candidate entries examined at end-point `t`, before pruning.

costQ Always `NULL`; cumulative costs are stored in the first column of `R`.

R Numeric matrix with `length(data)` rows and three columns. Row `t` describes the optimum ending at observation `t`:

- **Q** cumulative cost
- **K** number of segments in Q
- **s** zero-based previous boundary; the final segment is `data[(s + 1):t]`

Examples

```
range_test <- function(segment, gamma) {
  diff(range(segment)) <= gamma
}
set.seed(1)
data <- ts_generator(
  chpts = c(30, 60), parameters = c(0, 4),
  sd_noise = 0.25, type = "gauss"
)
fit <- svp0(data, gamma = 2, test = range_test,
           subtests = "both")
fit$changepoints
```

---

svpChange *svpChange: Smallest Valid Partitioning for Change-Point Detection*

Description

Provides methods for detecting multiple change-points and segmenting univariate time series using Smallest Valid Partitioning (SVP).

SVP searches for a partition with the smallest number of contiguous segments whose observations satisfy a user-defined or built-in validity test. Among partitions with the same number of segments, it minimizes a within-segment sum-of-squared-errors criterion.

Details

The package provides two main SVP interfaces:

`SVP` A compiled implementation with built-in validity tests, including Gaussian mean, MS-l multi-scale Gaussian mean, Gamma rate, variance, quantile, rank-based, and AR(1) tests.

`svp0` A flexible implementation accepting an arbitrary R function of the form `function(segment, gamma)` as its validity test.

The standalone `SVP_both_dyadic()` entrypoint applies length-dependent dyadic thresholds to the Gaussian-mean test while using both pruning.

Validity-test helpers include: `valid_FOCUS`, `valid_AR1`, `valid_SSE`, `valid_RANGE`, `valid_RANGE_SLACK`, `valid_QUANTILE`, `valid_SCALE`, and `valid_OP`.

The package also provides specialized SMUCE routines for fixed-variance Gaussian observations. `svp_smuce` is an R implementation, while `svp_smuce_cpp` provides a compiled implementation. Both minimize the number of SMUCE-valid segments and then minimize the constrained Gaussian residual cost.

`OP`, `PELT`, and `SN` are available for comparisons with other dynamic-programming algorithms. `AR1_rho` and `AR1_single_change` provide AR(1) diagnostics, and `ts_generator` generates simulation data.

The computational implementations are written in C++ and exposed to R through Rcpp. The R implementations of the SMUCE routines are retained as transparent reference implementations.

Author(s)

Maintainer: Vincent Runge <vincent.runge@univ-evry.fr> [copyright holder]

Authors:

- Vincent Runge <vincent.runge@univ-evry.fr> [copyright holder]
- Alexandre Combeau <alexandre.combeau.77@gmail.com>
- Gaetano Romano <g.romano@lancaster.ac.uk>

Other contributors:

- Anica Kostic <anica.kostic@gmail.com> [contributor]

References

Romano, G., Eckley, I. A., Fearnhead, P., and Rigaill, G. (2023). Fast Online Changepoint Detection via Functional Pruning CUSUM Statistics. *Journal of Machine Learning Research*, 24(81), 1–36. https://www.jmlr.org/papers/v24/21-1230.html

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. *Journal of the Royal Statistical Society: Series B*, 76(3), 495–580. doi:10.1111/rssb.12047

Chakar, S., Lebarbier, E., Levy-Leduc, C., and Robin, S. (2017). A robust approach for estimating change-points in the mean of an AR(1) process. *Bernoulli*, 23(2), 1408–1447.

See Also

`SVP`, `svp0`, `valid_FOCUS`, `svp_smuce`

SVP_both_dyadic *Smallest Valid Partitioning with dyadic Gaussian thresholds and both pruning*

Description

The threshold used for a candidate segment depends on its length. For a series of length `n`, band `j` covers lengths in `(n / 2^(j + 1), n / 2^j]`, with `j = 0` denoting the longest segments. Single-observation segments are always valid. Candidate removal follows the same both-direction pruning rules as `SVP()`.

Usage

```
SVP_both_dyadic(data, gamma)
```

Arguments

`data`	Finite numeric observations.
`gamma`	Finite positive numeric vector with one threshold per dyadic band, ordered from longest to shortest segment lengths.

Value

The same partition and dynamic-programming diagnostics returned by `SVP()`.

svp_smuce *SVP with SMUCE validity and constrained Gaussian cost*

Description

SVP with SMUCE validity and constrained Gaussian cost

Usage

```
svp_smuce(y, gamma, sigma2 = 1)
```

Arguments

`y`	Numeric observations.
`gamma`	SMUCE threshold.
`sigma2`	Known Gaussian variance.

Details

For independent Gaussian observations with known variance, this function computes the SMUCE dynamic program: it first minimizes the number of constant segments satisfying the multiscale constraint, then minimizes the constrained Gaussian residual cost among those partitions. The function returns one optimal partition as its segment endpoints, not fitted levels or confidence intervals. Exact changepoint locations are not unique when multiple optimal partitions have the same objective; ties are resolved deterministically by the dynamic program.

Value

Integer segment-end indices. Internal segment endpoints are the estimated changepoints; the final endpoint is `length(y)`.

References

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. *Journal of the Royal Statistical Society: Series B*, 76(3), 495–580. doi:10.1111/rssb.12047.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(20, 40), parameters = c(0, 2),
  sd_noise = 1, type = "gauss"
)
svp_smuce(data, gamma = 1.5, sigma2 = 1)
```

`svp_smuce_cpp` *C++ SVP with SMUCE validity and constrained Gaussian cost*

Description

C++ SVP with SMUCE validity and constrained Gaussian cost

Usage

```
svp_smuce_cpp(y, q, sigma2 = 1)
```

Arguments

`y`	Numeric observations.
`q`	SMUCE threshold.
`sigma2`	Known Gaussian variance.

Details

For independent Gaussian observations with known variance, this function returns the segment endpoints of one SMUCE-optimal partition. Internal endpoints are estimated changepoints.

Value

Integer segment-end indices.

References

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. *Journal of the Royal Statistical Society: Series B*, 76(3), 495–580. doi:10.1111/rssb.12047.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(20, 40), parameters = c(0, 2),
  sd_noise = 1, type = "gauss"
)
svp_smuce_cpp(data, q = 1.5, sigma2 = 1)
```

---

svp_smuce_dyadic_cpp *C++ SVP with dyadic-length SMUCE validity and constrained Gaussian cost*

Description

C++ SVP with dyadic-length SMUCE validity and constrained Gaussian cost

Usage

```
svp_smuce_dyadic_cpp(y, q, sigma2 = 1)
```

Arguments

y	Numeric observations.
q	SMUCE threshold.
sigma2	Known Gaussian variance.

Details

Identical to svp_smuce_cpp except that the multiscale constraint is only enforced on sub-windows whose length is a power of two (the "dyaLen" interval system of the **stepR** package), instead of on every sub-window (the "all" interval system svp_smuce_cpp uses). Every starting position is still tested at each dyadic length, so no positional resolution is lost; only the set of tested lengths shrinks from n to $O(\log n)$ per position, from $n(n+1)/2$ to $O(n \log n)$ intervals overall. Because fewer simultaneous tests need to be union-bounded, the calibrated threshold for a given null false-detection rate is smaller than svp_smuce_cpp's, which can translate into higher detection power at the same false-positive rate.

Value

Integer segment-end indices.

References

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale Change-Point Inference. *Journal of the Royal Statistical Society: Series B*, 76(3), 495–580. doi:10.1111/rssb.12047.

Pein, F. and Munk, A. (2026). R package stepR: Multiscale change-point inference. Package vignette, Section 6 ("Interval systems").

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(20, 40), parameters = c(0, 2),
  sd_noise = 1, type = "gauss"
)
svp_smuce_dyadic_cpp(data, q = 1.5, sigma2 = 1)
```

`ts_generator` *ts_generator*

Description

Generating univariate time series for multiple change-point detection based on uni-parametric models of the exponential family. The `"gaussAR1"` model generates pure AR(1) noise around the piecewise-constant signal, matching the DeCAFS model with `sdEta = 0`.

Usage

```
ts_generator(
  chpts = 100,
  parameters = 0.5,
  sd_noise = 1,
  rho = 0,
  nb_trials = 10,
  nb_success = 10,
  type = "gauss",
  df = 2,
  scale = 1
)
```

Arguments

`chpts`	a vector of increasing change-point indices (the last value is data length)
`parameters`	vector of successive segment parameters (as many parameters as values in `chpts` vector); for `"exp"` these are rates and for `"variance"` these are standard deviations

sd_noise	(types "gauss" and "gaussAR1") standard deviation of each Gaussian innovation
rho	(type "gaussAR1") AR(1) coefficient for the Gaussian noise
nb_trials	(type "binom") number of trials
nb_success	(type "negbin") number of successes
type	the model: "gauss", "gaussAR1", "student", "exp", "poisson", "geom", "bern", "binom", "negbin", "variance"
df	(type "student") positive degrees of freedom for the Student-t noise
scale	(type "student") non-negative scale of the Student-t noise

Details

For type = "gaussAR1", the generated series is y[t] = parameters[t] + e[t], where e[t] = rho * e[t - 1] + eta[t] and eta[t] ~ N(0, sd_noise^2). The first residual is drawn from the stationary distribution. Consequently, the marginal noise standard deviation is sd_noise / sqrt(1 - rho^2), not sd_noise. To obtain marginal standard deviation target_sd, use sd_noise = target_sd * sqrt(1 - rho^2). The "student" model uses parameters + scale * T, where T has a Student-t distribution with df degrees of freedom; for df <= 2, its variance is not finite.

Value

a univariate time series following the chosen model type and parameters

Examples

```
set.seed(1)
# Independent Gaussian noise with a few mean changes.
ts_generator(
  chpts = c(50, 100, 150, 200),
  parameters = c(0, 2, -1, 1),
  sd_noise = 1,
  type = "gauss"
)

# Pure AR(1) Gaussian noise around the same type of signal.
ts_generator(
  chpts = c(50, 100, 150, 200),
  parameters = c(0, 2, -1, 1),
  sd_noise = 1,
  rho = 0.8,
  type = "gaussAR1"
)

# Heavy-tailed Student-t noise.
ts_generator(
  chpts = c(50, 100),
  parameters = c(0, 2),
  df = 2,
  scale = 1,
```

```
  type = "student"
)

# Other supported distributions.
ts_generator(chpts = c(50, 100), parameters = c(2, 7), type = "exp")
ts_generator(chpts = c(50, 100), parameters = c(3, 5), type = "poisson")
ts_generator(chpts = c(50, 100), parameters = c(0.6, 0.3), type = "geom")
ts_generator(chpts = c(50, 100), parameters = c(0.7, 0.2), type = "bern")
ts_generator(
  chpts = c(50, 100), parameters = c(0.7, 0.3), nb_trials = 5,
  type = "binom"
)
ts_generator(
  chpts = c(50, 100), parameters = c(0.4, 0.7), nb_success = 10,
  type = "negbin"
)
ts_generator(
  chpts = c(50, 100, 180), parameters = c(3, 1, 6), type = "variance"
)
```

`valid_AR1` *AR(1) Mean-Change Validity Test*

Description

Tests whether a segment is valid under a conditional Gaussian AR(1) single-mean-change statistic implemented in R.

Usage

```
valid_AR1(
  y,
  gamma,
  rho = NA_real_,
  sigma2 = 1,
  profile_sigma = FALSE
)
```

Arguments

`y`	A numeric AR(1) segment. At least four observations are needed for a non-trivial single-change scan.
`gamma`	A threshold for the AR(1) likelihood-ratio statistic.
`rho`	An optional known AR(1) coefficient. If `NA`, it is estimated robustly from `y`.
`sigma2`	A positive innovation variance used by the known-variance statistic. In the AR(1) model, this is the variance of the new random shock after accounting for the previous observation and the AR(1) mean structure; it is not the marginal variance of the observed series. It is ignored when `profile_sigma = TRUE`.

profile_sigma If TRUE, estimate the innovation variance separately under the no-change and one-change models from their residual sums of squares. This is useful when the innovation scale is unknown.

Details

The function conditions on the first observation, transforms the remaining observations into AR(1) innovations, and scans change locations that leave at least two observations on each side. The known-variance statistic is the full RSS reduction divided by sigma2, using the same convention as AR1_single_change. If rho is NA, the robust estimator requires at least three observations. The function can be passed to svp0 directly when rho is fixed, or through a closure when other optional arguments are needed. The caller should enforce any minimum segment length required by the chosen SVP procedure. When profile_sigma = FALSE, the statistic uses the supplied sigma2; when profile_sigma = TRUE, the innovation variance is profiled out and estimated under each competing model.

Value

TRUE if the AR(1) statistic is less than or equal to gamma; otherwise, FALSE.

See Also

AR1_single_change, svp0

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 60, parameters = 0, sd_noise = 1,
  rho = 0.6, type = "gaussAR1"
)
valid_AR1(data, gamma = 20, rho = 0.6, sigma2 = 1)
```

---

valid_FOCUS *FOCuS Validity Test for Every Prefix*

Description

Tests whether the FOCuS statistic remains at most gamma for every prefix of the segment. It is the prefix-wise validity rule available for use with svp0().

Usage

```
valid_FOCUS(y, gamma)
```

Arguments

y	A numeric vector representing a segment of the signal.
gamma	A numeric threshold for the FOCuS statistic.

Details

FOCuS Validity Test for Every Prefix

`valid_FOCUS()` updates the FOCuS statistic after each observation and returns `FALSE` as soon as a prefix statistic exceeds `gamma`. Use it with `svp0()` when every prefix must be valid. In contrast, `SVP()` with `test = "gaussian_mean"` checks only the statistic at the current segment endpoint, and `valid_FOCUS_last()` checks only the final statistic of a supplied segment.

Value

`TRUE` if the statistic is less than or equal to `gamma` for every prefix of `y`; otherwise, `FALSE`.

See Also

`valid_FOCUS_last()`

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 40, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_FOCUS(data, gamma = 4 * log(length(data)))
```

---

valid_FOCUS_last *FOCuS Validity Test for the Final Segment*

Description

Tests whether the FOCuS statistic for the complete segment is at most `gamma`, without checking intermediate prefixes.

Usage

```
valid_FOCUS_last(y, gamma)
```

Arguments

`y`	A numeric vector representing a segment of the signal.
`gamma`	A numeric threshold for the FOCuS statistic.

Details

FOCuS Validity Test for the Final Segment

`valid_FOCUS_last()` processes the complete segment and checks only the final statistic. An earlier prefix may have exceeded `gamma` without making the final segment invalid. Use `valid_FOCUS()` to require every prefix to remain valid.

Value

TRUE if the final statistic is less than or equal to gamma; otherwise, FALSE.

See Also

valid_FOCUS()

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 40, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_FOCUS_last(data, gamma = 4 * log(length(data)))
```

---

valid_OP *Optimal Partitioning Cost Test*

Description

Tests whether the total cost of the segment is smaller than the best two-part penalized segmentation cost.

Usage

```
valid_OP(y, gamma)
```

Arguments

y	A numeric vector representing a segment of the signal.
gamma	A numeric threshold for penalizing the introduction of a new segment.

Details

Optimal Partitioning Cost Test

Value

TRUE if the segment cannot be split into two parts with lower (penalized) cost.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = c(20, 40), parameters = c(0, 2),
  sd_noise = 1, type = "gauss"
)
valid_OP(data, gamma = 2 * log(length(data)))
```

valid_QUANTILE *Validity Test Based on Interquantile Range*

Description

Checks whether the quantile range between `probs[1]` and `probs[2]` is below a threshold.

Usage

```
valid_QUANTILE(y, gamma, probs = c(0.05, 0.95))
```

Arguments

`y`	A numeric vector representing a segment of the signal.
`gamma`	A numeric threshold for the interquantile range.
`probs`	A numeric vector of length two giving the lower and upper quantile probabilities. Defaults to `c(0.05, 0.95)` and must be in increasing order.

Details

Validity Test Based on Interquantile Range

The test computes the difference between the quantiles at `probs[2]` and `probs[1]`.

Value

TRUE if the interquantile range is less than or equal to gamma.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 30, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_QUANTILE(data, gamma = 4)
```

valid_RANGE *Validity Test Based on Range*

Description

Checks whether the range (max - min) of the segment is below a threshold.

Usage

```
valid_RANGE(y, gamma)
```

Arguments

y	A numeric vector representing a segment of the signal.
gamma	A numeric threshold for the maximum allowed range.

Details

Validity Test Based on Range

Value

TRUE if the range is less than or equal to gamma.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 30, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_RANGE(data, gamma = 5)
```

---

valid_RANGE_SLACK *Validity Test Based on Trimmed Range*

Description

Applies a slack range test using trimmed minimum and maximum (ignores `trim` observations at each end).

Usage

```
valid_RANGE_SLACK(y, gamma, trim = 3)
```

Arguments

y	A numeric vector representing a segment with at least `2 * trim + 1` observations.
gamma	A numeric threshold for the trimmed range.
trim	Number of observations to remove from each end. Defaults to 3 and must be a non-negative integer.

Details

Validity Test Based on Trimmed Range

The test discards `trim` smallest and `trim` largest observations before computing the range. At least `2 * trim + 1` observations are therefore required, leaving one observation after trimming. The minimum segment length is enforced by the caller.

Value

TRUE if the trimmed range is less than or equal to gamma.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 30, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_RANGE_SLACK(data, gamma = 4, trim = 3)
```

---

valid_SCALE *Validity Test Based on Robust Scale*

Description

Checks whether the Gaussian-normalized median absolute deviation of a segment is below a threshold.

Usage

```
valid_SCALE(y, gamma)
```

Arguments

y	A numeric vector representing a segment of the signal.
gamma	A numeric threshold for the robust scale of the segment.

Details

The scale is computed with `stats::mad()` using the segment median as centre and the constant `1.4826`. This constant makes the MAD estimate the standard deviation for Gaussian data. Because the test is based on the complete segment, its result is not necessarily preserved when observations are appended; use `subtests = "none"` and `PELT_pruning = FALSE` in `svp0` unless the required pruning properties have been established for the intended application.

Value

`TRUE` if the normalized median absolute deviation is less than or equal to `gamma`; otherwise, `FALSE`.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 30, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_SCALE(data, gamma = 2)
```

valid_SSE *Validity Test Based on Sum of Squared Errors*

Description

Checks if the sum of squared deviations from the segment mean is lower than or equal to a threshold.

Usage

```
valid_SSE(y, gamma)
```

Arguments

y	A numeric vector representing a segment of the signal.
gamma	A numeric threshold for the maximum allowed sum of squared errors.

Details

Validity Test Based on Sum of Squared Errors

Value

TRUE if the sum of squared errors is less than or equal to gamma.

Examples

```
set.seed(1)
data <- ts_generator(
  chpts = 30, parameters = 0, sd_noise = 1, type = "gauss"
)
valid_SSE(data, gamma = 40)
```

Index

∗ **package**
svpChange, 16

AR1_rho, 2, *13*, *17*
AR1_single_change, 3, *13*, *17*, *24*

OP, 4, *17*
OP(), *5, 6*, *8*

PELT, 5, *17*
PELT(), *5*, *8*

smuce_cost, 6
SN, 7, *17*
SN(), *5, 6*
stats::mad(), *29*
SVP, 8, *17*
SVP(), *5, 6*, *8*, *18*, *25*
svp0, *13*, 15, *17*, *24*, *29*
svp0(), *24, 25*
SVP_both_dyadic, 18
SVP_both_dyadic(), *17*
svp_smuce, *17*, 18
svp_smuce_cpp, *17*, 19
svp_smuce_dyadic_cpp, 20
svpChange, 16
svpChange-package *(*svpChange*)*, 16

ts_generator, *17*, 21

valid_AR1, *17*, 23
valid_FOCUS, *17*, 24
valid_FOCUS(), *25, 26*
valid_FOCUS_last, 25
valid_FOCUS_last(), *25*
valid_OP, *17*, 26
valid_QUANTILE, *17*, 27
valid_RANGE, *17*, 27
valid_RANGE_SLACK, *17*, 28
valid_SCALE, *17*, 29
valid_SSE, *17*, 30