Package ‘kcpRS’

January 20, 2023

Type Package

Title Kernel Change Point Detection on the Running Statistics

Version 1.1.0

Maintainer Kristof Meers <kristof.meers+cran@kuleuven.be>

Description The running statistics of interest is first extracted using a time window which is slid across the time series, and in each window, the running statistics value is computed. KCP (Kernel Change Point) detection proposed by Arlot et al. (2012) <arXiv:1202.3878> is then implemented to flag the change points on the running statistics (Cabrieto et al., 2018, <doi:10.1016/j.ins.2018.03.010>). Change points are located by minimizing a variance criterion based on the pairwise similarities between running statistics which are computed via the Gaussian kernel. KCP can locate change points for a given k number of change points. To determine the optimal k, the KCP permutation test is first carried out by comparing the variance of the running statistics extracted from the original data to that of permuted data. If this test is significant, then there is sufficient evidence for at least one change point in the data. Model selection is then used to determine the optimal k>0.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Imports Rcpp (>= 1.0.0)

Depends RColorBrewer, stats, utils, graphics, roll, foreach, doParallel

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

LinkingTo Rcpp

RoxygenNote 7.2.3

NeedsCompilation yes

Author Jedelyn Cabrieto [aut],
Kristof Meers [aut, cre],
Evelien Schat [ctb],
Janne Adolf [ctb],

Peter Kuppens [ctb],
Francis Tuerlinckx [ctb],
Eva Ceulemans [ctb]

Repository CRAN

Date/Publication 2023-01-19 23:30:14 UTC

R topics documented:

kcpRS-package 2
CO2Inhalation 4
getScatterMatrix 4
kcpa 5
kcpRS 6
kcpRS_workflow 8
MentalLoad 10
permTest 11
runAR 12
runCorr 13
runMean 13
runVar 14

Index **15**

`kcpRS-package` *KCP on the running statistics*

Description

Flagging change points on a user-specified running statistics through KCP (Kernel Change Point) detection. A KCP permutation test is first implemented to confirm whether there is at least one change point (k>0) in the running statistics. If this permutation test is significant, a model selection procedure is implemented to choose the most optimal number of change points.

Details

Package:	**kcpRS**
Type:	Package
Version:	1.0.0
Date:	15-03-2019
License:	GPL (>=2)

This package contains the function `kcpRS` that can accept a user-defined function, `RS_fun`, which should derive the running statistics of interest. For examples, see `runMean`, `runVar`, `runAR` and `runCorr`. `kcpRS` performs a full change point analysis on the running statistics starting from locating the optimal change points given k, significance testing if k>0, and finally, determining the most optimal k. This function calls the function kcpa to find the most optimal change points given k and then the permTest function to carry out the permutation test. The model selection step is embedded in the kcpRS function.

This package also contains the function kcpRS_workflow which carries out a stepwise change point analysis to flag changes in 4 basic time series statistics: mean, variance, autocorrelation (lag 1) and correlations.

Two illustrative data sets are included: MentalLoad and CO2Inhalation

Author(s)

Jedelyn Cabrieto (<jed.cabrieto@kuleuven.be>) and Kristof Meers

For the core KCP analysis, the authors built upon the codes from the Supplementary Material available in doi:10.1080/01621459.2013.849605 by Matteson and James (2012).

References

Arlot, S., Celisse, A., & Harchaoui, Z. (2019). A kernel multiple change-point algorithm via model selection. *Journal of Machine Learning Research*, 20(162), 1-56.

Cabrieto, J., Tuerlinckx, F., Kuppens, P., Grassmann, M., & Ceulemans, E. (2017). Detecting correlation changes in multivariate time series: A comparison of four non-parametric change point detection methods. *Behavior Research Methods*, 49, 988-1005. doi:10.3758/s13428-016-0754-9

Cabrieto, J., Tuerlinckx, F., Kuppens, P., Hunyadi, B., & Ceulemans, E. (2018). Testing for the presence of correlation changes in a multivariate time series: A permutation based approach. *Scientific Reports*, 8, 769, 1-20. doi:10.1038/s41598-017-19067-2

Cabrieto, J., Tuerlinckx, F., Kuppens, P., Wilhelm, F., Liedlgruber, M., & Ceulemans, E. (2018). Capturing correlation changes by applying kernel change point detection on the running correlations. *Information Sciences*, 447, 117-139. doi:10.1016/j.ins.2018.03.010

Cabrieto, J., Adolf, J., Tuerlinckx, F., Kuppens, P., & Ceulemans, E. (2018). Detecting long-lived autodependency changes in a multivariate system via change point detection and regime switching models. *Scientific Reports*, 8, 15637, 1-15. doi:10.1038/s41598-018-33819-8

See Also

kcpRS

kcpRS_workflow

MentalLoad

CO2Inhalation

`CO2Inhalation` *CO2 Inhalation Data*

Description

Nine physiological measures during a CO2-inhalation experiment.

Usage

```
data(CO2Inhalation)
```

Format

Dataframe with 239 rows and 10 columns. The first column indicates the experimental phase and the last nine columns correspond to the nine physiological measures tracked during the experiment: Breathing volume variables (ViVol, VeVol, Vent, PiaAB), breathing duration variables (Ti,Te,Tt), heart rate (HR) and RR interval (RR) or cardiac beat interval.

References

De Roover, K., Timmerman, M. E., Van Diest, I., Onghena, P., & Ceulemans, E. (2014). Switching principal component analysis for modeling means and covariance changes over time. *Psychological Methods*, 19, 113-132. doi:10.1037/a0034525

Examples

```
data(CO2Inhalation)
```

`getScatterMatrix` *Get the matrix of optimized scatters used in locating the change points.*

Description

Get the matrix of optimized scatters used in locating the change points.

Usage

```
getScatterMatrix(II_, X_, H_)
```

Arguments

`II_`	A D x N matrix where D is the maximum no. of segments (Kmax+1) and N is the no. of windows
`X_`	An N x r dataframe where N is the no. of windows and r the no. of running statistics monitored
`H_`	A D x N matrix where D is the maximum no. of segments (Kmax+1) and N is the no. of windows

Value

`II`	A matrix of optimized scatters
`H`	A matrix of candidate changes point locations
`medianK`	Median of the pairwise Euclidean distances

`kcpa` *KCP (Kernel Change Point) Detection*

Description

Finds the most optimal change point(s) in the running statistic time series `RunStat` by looking at their kernel-based pairwise similarities.

Usage

```
kcpa(RunStat, Kmax = 10, wsize = 25)
```

Arguments

`RunStat`	Dataframe of running statistics with rows corresponding to the windows and the columns corresponding to the variable(s) on which these running statistics were computed.
`Kmax`	Maximum number of change points
`wsize`	Window size

Value

`kcpSoln`	A matrix comprised of the minimized variance criterion *Rmin* and the optimal change point location(s) for each k from 1 to `Kmax`

References

Arlot, S., Celisse, A., & Harchaoui, Z. (2019). A kernel multiple change-point algorithm via model selection. *Journal of Machine Learning Research*, 20(162), 1-56.

Cabrieto, J., Tuerlinckx, F., Kuppens, P., Grassmann, M., & Ceulemans, E. (2017). Detecting correlation changes in multivariate time series: A comparison of four non-parametric change point detection methods. *Behavior Research Methods*, 49, 988-1005. doi:10.3758/s13428-016-0754-9

kcpRS — *KCP on the running statistics*

Description

Given a user-specified function RS_fun to compute the running statistics (see runMean, runVar, runAR and runCorr), a KCP permutation test (see permTest) is first implemented to test whether there is at least one significant change point, then through model selection most optimal number of change points is chosen.

Usage

```
kcpRS(
  data,
  RS_fun,
  RS_name,
  wsize = 25,
  nperm = 1000,
  Kmax = 10,
  alpha = 0.05,
  varTest = FALSE,
  ncpu = 1
)

## S3 method for class 'kcpRS'
plot(x, ...)

## S3 method for class 'kcpRS'
print(x, kcp_details = TRUE, ...)

## S3 method for class 'kcpRS'
summary(object, ...)
```

Arguments

data	data N x v dataframe where N is the number of time points and v the number of variables
RS_fun	Running statistics function: Should require wsize and wstep as input and return a dataframe of running statistics as output. The rows of this dataframe should correspond to the windows and the columns should correspond to the variable(s) on which the running statistics were computed.
RS_name	Name of the monitored running statistic.
wsize	Window size
nperm	Number of permutations used in the permutation test
Kmax	Maximum number of change points desired

`alpha` Significance level of the permutation test

`varTest` If set to FALSE, only the variance DROP test is implemented, and if set to TRUE, both the variance test and the variance DROP tests are implemented.

`ncpu` number of cpu cores to use

`x` An object of the type produced by `kcpRS`

`...` Further plotting arguments.

`kcp_details` If *TRUE*, then the matrix of optimal change points solutions given *k* is displayed. If FALSE, then this output is suppressed.

`object` An object of the type produced by `kcpRS_workflow`

Value

`RS_name` Name indicated for the monitored running statistic

`RS` Dataframe of running statistics with rows corresponding to the time window and columns corresponding to the (combination of) variable(s) on which the running statistics were computed

`wsize` Selected window size

`varTest` Selected choice of implementation for varTest

`nperm` Selected number of permutations

`alpha` Selected significance level of the permutation test

`subTest_alpha` Significance level of each subtest. If `varTest=0`, `subTest_alpha` is equal to `alpha` since only the variance drop test is implemented. If `varTest=1`, `subTest_alpha=alpha/2` since two subtests are carried out and Bonferonni correction is applied.

`BestK` Optimal number of change points based on grid search

`changePoints` Change point location(s)

`p_var_test` P-value of the variance test

`p_varDrop_test` P-value of the variance drop test

`CPs_given_K` A matrix comprised of the minimized variance criterion *Rmin* and the optimal change point location(s) for each *k* from 1 to `Kmax`

`changePoints_scree_test`
Optimal number of change points based on scree test

`scree_test` A matrix comprised of the scree values for each *k* from 1 to `Kmax-1`

`medianK` Median Euclidean distance between all pairs of running statistics

References

Cabrieto, J., Tuerlinckx, F., Kuppens, P., Wilhelm, F., Liedlgruber, M., & Ceulemans, E. (2018). Capturing correlation changes by applying kernel change point detection on the running correlations. *Information Sciences*, 447, 117-139. doi:10.1016/j.ins.2018.03.010

Cabrieto, J., Adolf, J., Tuerlinckx, F., Kuppens, P., & Ceulemans, E. (2018). Detecting long-lived autodependency changes in a multivariate system via change point detection and regime switching models. *Scientific Reports*, 8, 15637, 1-15. doi:10.1038/s41598-018-33819-8

Cabrieto, J., Meers, K., Schat, E., Adolf, J. K., Kuppens, P., Tuerlinckx, F., & Ceulemans, E. (2022). kcpRS: An R package for performing kernel change point detection on the running statistics of multivariate time series. *Behavior Research Methods*, 54, 1092-1113. doi:10.3758/s13428-021-01603-8

Examples

```
phase1=cbind(rnorm(50,0,1),rnorm(50,0,1)) #phase1: Means=0
phase2=cbind(rnorm(50,1,1),rnorm(50,1,1)) #phase2: Means=1
X=rbind(phase1,phase2)
res=kcpRS(data=X,RS_fun=runMean,RS_name="Mean",wsize=25,
nperm=1000,Kmax=10,alpha=.05,varTest=FALSE,ncpu=1)

summary(res)
plot(res)
```

`kcpRS_workflow` *KCP on the Running Statistics Workflow*

Description

Any of the four basic running statistics (i.e., running means, running variances, running autocorrelations and running correlations) or a combination thereof can be scanned for change points.

Usage

```
kcpRS_workflow(
  data,
  RS_funs = c("runMean", "runVar", "runAR", "runCorr"),
  wsize = 25,
  nperm = 1000,
  Kmax = 10,
  alpha = 0.05,
  varTest = FALSE,
  bcorr = TRUE,
  ncpu = 1
)

## S3 method for class 'kcpRS_workflow'
plot(x, ...)

## S3 method for class 'kcpRS_workflow'
print(x, ...)

## S3 method for class 'kcpRS_workflow'
summary(object, ...)
```

Arguments

`data`	data N x v dataframe where N is the number of time points and v the number of variables
`RS_funs`	a vector of names of the functions that correspond to the running statistics to be monitored. Options available: "runMean"=running mean, "runVar"=running variance, "runAR"=running autocorrelation and "runCorr"=running correlation.
`wsize`	Window size
`nperm`	Number of permutations used in the permutation test
`Kmax`	Maximum number of change points desired
`alpha`	Significance level for the full KCP-RS workflow analysis if `bcorr=1`. Otherwise, this is the significance level for each running statistic.
`varTest`	If set to TRUE, only the variance DROP test is implemented, and if set to FALSE, both the variance test and the variance DROP tests are implemented.
`bcorr`	Set to TRUE if Bonferonni correction is desired for the workflow analysis and set to FALSE otherwise.
`ncpu`	number of cpu cores to use
`x`	An object of the type produced by `kcpRS_workflow`
`...`	Further plotting arguments
`object`	An object of the type produced by `kcpRS_workflow`

Details

The workflow proceeds in two steps: First, the mean change points are flagged using KCP on the running means. If there are significant change points, the data is centered based on the yielded change points. Otherwise, the data remains untouched for the next analysis. Second, the remaining running statistics are computed using the centered data in the first step. The user can specify which running statistics to scan change points for (see `RS_funs` and examples). Bonferonni correction for tracking multiple running statistics can be implemented using the `bcorr` option.

Value

`kcpMean`	`kcpRS` solution for the running means. See output of `kcpRS` for further details.
`kcpVar`	`kcpRS` solution for the running variances. See output of `kcpRS` for further details.
`kcpAR`	`kcpRS` solution for the running autocorrelations. See output of `kcpRS` for further details.
`kcpCorr`	`kcpRS` solution for the running correlations. See output of `kcpRS` for further details.

References

Cabrieto, J., Adolf, J., Tuerlinckx, F., Kuppens, P., & Ceulemans, E. (2019). An objective, comprehensive and flexible statistical framework for detecting early warning signs of mental health problems. *Psychotherapy and Psychosomatics*, 88, 184-186. doi:10.1159/000494356

Examples

```
phase1=cbind(rnorm(50,0,1),rnorm(50,0,1)) #phase1: Means=0
phase2=cbind(rnorm(50,1,1),rnorm(50,1,1)) #phase2: Means=1
X=rbind(phase1,phase2)

#scan all statistics

res=kcpRS_workflow(data=X,RS_funs=c("runMean","runVar","runAR","runCorr"),
wsize=25,nperm=1000,Kmax=10,alpha=.05, varTest=FALSE, bcorr=TRUE, ncpu=1)
summary(res)
plot(res)


#scan the mean and the correlation only
res=kcpRS_workflow(data=X,RS_funs=c("runMean","runCorr"),wsize=25,nperm=1000,Kmax=10,
    alpha=.05, varTest=FALSE, bcorr=TRUE, ncpu=1)
summary(res)
plot(res)
```

`MentalLoad` *Mental Load Data*

Description

Three physiological measures during a mental load assessment experiment on aviation pilots

Usage

```
data(MentalLoad)
```

Format

Dataframe with 1393 rows and 4 columns. The first column indicates the experimental period, while the last three columns correspond to the three physiological measures monitored during the experiment: Heart rate (HR), respiration rate (RR) and petCO2.

References

Grassmann, M., Vlemincx, E., von Leupoldt, A., & Van den Bergh, O. (2016). The role of respiratory measures to assess mental load in pilot selection. *Ergonomics*, 59(6), 745-753. (PubMed)

Examples

```
data(MentalLoad)
```

permTest — *KCP Permutation Test*

Description

The KCP permutation test implements the variance test and the variance drop test to determine if there is at least one change point in the running statistics

Usage

```
permTest(
  data,
  RS_fun,
  wsize = 25,
  nperm = 1000,
  Kmax = 10,
  alpha = 0.05,
  varTest = FALSE
)
```

Arguments

`data`	data N x v dataframe where N is the number of time points and v the number of variables
`RS_fun`	Running statistics function: Should require the time series and `wsize` as input and return a dataframe of running statistics as output. This output dataframe should have rows that correspond to the time windows and columns that correspond to the variable(s) on which the running statistics were computed.
`wsize`	Window size
`nperm`	Number of permutations to be used in the permutation test
`Kmax`	Maximum number of change points desired
`alpha`	Significance level of the permutation test
`varTest`	If FALSE, only the variance DROP test is implemented, and if TRUE, both the variance and the variance DROP tests are implemented.

Value

`sig`	Significance of having at least one change point. 0 - Not significant, 1- Significant
`p_var_test`	P-value of the variance test.
`p_varDrop_test`	P-value of the variance drop test.
`perm_rmin`	A matrix of minimized variance criterion for the permuted data.
`perm_rmin_without_NA`	A matrix of minimized variance criterion for the permuted data without NA values.

References

Cabrieto, J., Tuerlinckx, F., Kuppens, P., Hunyadi, B., & Ceulemans, E. (2018). Testing for the presence of correlation changes in a multivariate time series: A permutation based approach. *Scientific Reports*, 8, 769, 1-20. doi:10.1038/s41598-017-19067-2

`runAR` *Running Autocorrelations*

Description

Extracts the running autocorrelations by sliding a window comprised of `wsize` time points, and in each window, the autocorrelation for each variable is computed. Each time the window is slid, the oldest time point is discarded and the latest time point is added.

Usage

```
runAR(data, wsize = 25)
```

Arguments

`data`	*N* x *v* dataframe where *N* is the no. of time points and *v* the no. of variables
`wsize`	Window size

Value

Running autocorrelations time series

Examples

```
phase1=cbind(rnorm(50,0,1),rnorm(50,0,1)) #phase1: AutoCorr=0
phase2=cbind(rnorm(50,0,1),rnorm(50,0,1))
phase2=filter(phase2,.50, method="recursive") #phase2: AutoCorr=.5
X=rbind(phase1,phase2)
RS=runAR(data=X,wsize=25)
ts.plot(RS, gpars=list(xlab="Window", ylab="Autocorrelation", col=1:2,lwd=2))
```

runCorr — *Running Correlations*

Description

Extracts the running correlations by sliding a window comprised of `wsize` time points, and in each window, the correlation of each pair of variables is computed. Each time the window is slid, the oldest time point is discarded and the latest time point is added.

Usage

```
runCorr(data, wsize = 25)
```

Arguments

`data`	*N* x *v* dataframe where *N* is the no. of time points and *v* the no. of variables
`wsize`	window size

Value

Running correlations time series

Examples

```
data(MentalLoad)
RS<-runCorr(data=MentalLoad,wsize=25)
ts.plot(RS, gpars=list(xlab="Window", ylab="Correlations", col=1:3,lwd=2))
```

runMean — *Running Means*

Description

Extracts the running means by sliding a window comprised of `wsize` time points, and in each window, the mean for each variable is computed. Each time the window is slid, the oldest time point is discarded and the latest time point is added.

Usage

```
runMean(data, wsize = 25)
```

Arguments

`data`	*N* x *v* dataframe where *N* is the no. of time points and *v* the no. of variables
`wsize`	Window size

Value

Running means time series

Examples

```
phase1=cbind(rnorm(50,0,1),rnorm(50,0,1)) #phase1: Means=0
phase2=cbind(rnorm(50,1,1),rnorm(50,1,1)) #phase2: Means=1
X=rbind(phase1,phase2)
RS=runMean(data=X,wsize=25)
ts.plot(RS, gpars=list(xlab="Window", ylab="Means", col=1:2,lwd=2))
```

---

runVar *Running Variances*

Description

Extracts the running variances by sliding a window comprised of `wsize` time points, and in each window, the variance for each variable is computed. Each time the window is slid, the oldest time point is discarded and the latest time point is added.

Usage

```
runVar(data, wsize = 25)
```

Arguments

`data`	N x v dataframe where N is the no. of time points and v the no. of variables
`wsize`	Window size

Value

Running variances time series

Examples

```
phase1=cbind(rnorm(50,0,1),rnorm(50,0,1)) #phase1: SD=1
phase2=cbind(rnorm(50,0,2),rnorm(50,0,2)) #phase2: SD=2
X=rbind(phase1,phase2)
RS=runVar(data=X,wsize=25)
ts.plot(RS, gpars=list(xlab="Window", ylab="Variances", col=1:2,lwd=2))
```

Index

∗ **datasets**
  CO2Inhalation, 4
  MentalLoad, 10

CO2Inhalation, *3*, 4

getScatterMatrix, 4

kcpa, *3*, 5
kcpRS, *2, 3*, 6, *9*
kcpRS-package, 2
kcpRS_workflow, *3*, 8

MentalLoad, *3*, 10

permTest, *3*, *6*, 11
plot.kcpRS *(*kcpRS*)*, 6
plot.kcpRS_workflow *(*kcpRS_workflow*)*, 8
print.kcpRS *(*kcpRS*)*, 6
print.kcpRS_workflow *(*kcpRS_workflow*)*, 8

runAR, *2*, *6*, 12
runCorr, *2*, *6*, 13
runMean, *2*, *6*, 13
runVar, *2*, *6*, 14

summary.kcpRS *(*kcpRS*)*, 6
summary.kcpRS_workflow *(*kcpRS_workflow*)*, 8