Package 'mvMonitoring'

November 21, 2023

Type Package

Title Multi-State Adaptive Dynamic Principal Component Analysis for Multivariate Process Monitoring

Version 0.2.4

Date 2023-11-21

Description Use multi-state splitting to apply Adaptive-Dynamic PCA (ADPCA) to data generated from a continuous-time multivariate industrial or natural process. Employ PCA-based dimension reduction to extract linear combinations of relevant features, reducing computational burdens. For a description of ADPCA, see <doi:10.1007/s00477-016-1246-2>, the 2016 paper from Kazor et al. The multi-state application of ADPCA is from a manuscript under current revision entitled ``Multi-State Multivariate Statistical Process Control" by Odom, Newhart, Cath, and Hering, and is expected to appear in Q1 of 2018.

License GPL-2

Depends R (>= 2.10)

Imports dplyr, lazyeval, plyr, rlang, utils, xts, zoo, robustbase, graphics

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/gabrielodom/mvMonitoring

Config/testthat/edition 3

NeedsCompilation no

Author Melissa Innerst [aut],
Gabriel Odom [aut, cre],
Ben Barnard [aut],
Karen Kazor [aut],
Amanda Hering [aut]

Maintainer Gabriel Odom <gabriel.odom@fiu.edu>

Repository CRAN

Date/Publication 2023-11-21 17:30:02 UTC

R topics documented:

`dataStateSwitch` *Alternate Observations in a Data Frame over States*

Description

Split single-state process observations, apply multiple state projections, and combine these observations into a single data frame, arranged by process time or index.

Usage

```
dataStateSwitch(
  df,
  angles2 = list(yaw = 0, pitch = 90, roll = 30),
  scales2 = c(1, 0.5, 2),
  angles3 = list(yaw = 90, pitch = 0, roll = -30),
  scales3 = c(0.25, 0.1, 0.75)
)
```

Arguments

df	A data frame returned by processNOCdata() or faultSwitch().
angles2	Change the principal angles for State 2.
scales2	Change the principal scales for State 2.
angles3	Change the principal angles for State 3.
scales3	Change the principal scales for State 3.

Details

This function splits a process data frame by state, and rotates and scales the observations from states 2 and 3 by the scales and angles specified in the function arguments. After state-specific rotation and scaling, this function combines the observations back together and orders them by process time index. This function takes in data frame returned by processNOCdata() or faultSwitch(). This function calls rotateScale3D() and is called internally by mspProcessData().

Value

A data frame containing the time index, state, and feature values after state-specific rotation and scaling; this data frame also contains the other columns of df that aren't the feature values. This data frame has

dateTime - a POSIX column of the time stamps for each observation

state - column of state membership (1, 2, or 3)

x - the process values for the first feature, corresponding to t + random error

y - the process values for the second feature, corresponding to t ^ 2 - 3 * t + random error

z - the process values for the third feature, corresponding to -t ^ 3 + 3 * t ^ 2 + random error

t - the non-stationary and autocorrelated latent feature

err1 - a Gaussian white noise vector

err2 - a Gaussian white noise vector

err3 - a Gaussian white noise vector

See Also

Calls: processNOCdata, faultSwitch, rotateScale3D. Called by: mspProcessData

Examples

```
nrml <- processNOCdata()
dataStateSwitch(nrml)
```

`fault1A_xts` *Process Data under a System Shift Fault*

Description

Three-feature, three-state simulated process data including observations under normal operating conditions and observations after a positive shift for each feature in the system.

Usage

```
fault1A_xts
```

Format

An xts data matrix with 10080 rows and four columns, corresponding to one week worth of data recorded at a 1-minute interval. The columns under normal conditions are defined in the help file for normal_switch_xts. The fault is a system shock to each of the three features by 2. The fault starts at row 8500, and the four columns under the fault state are defined here:

state : the state indicator for the multivariate system, with three levels

x : x(t) = t + 2 + error

y : y(t) = t ^ 2 - 3t + 2 + error

z : z(t) = - t ^ 3 + 3t ^ 2 + 2 + error

where t is a 10080-entry vector of autocorrelated and non-stationary hidden process realizations. The states alternate each hour and are defined as follows:

State1 – As presented

State2 – Rotated by (yaw = 0, pitch = 90, roll = 30) and scaled by (1 * x, 0.5 * y , 2 * z).

State3 – Rotated by (yaw = 90, pitch = 0, roll = -30) and scaled by (0.25 * x, 0.1 * y , 0.75 * z).

See the vignette for more details.

Source

Simulated in R.

`fault2A_xts` *Process Data under a System Drift Fault*

Description

Three-feature, three-state simulated process data including observations under normal operating conditions and observations after a positive drift in values for each feature in the system.

Usage

```
fault2A_xts
```

Format

An xts data matrix with 10080 rows and four columns, corresponding to one week worth of data recorded at a 1-minute interval. The columns under normal conditions are defined in the help file for normal_switch_xts. The fault is a drift on each feature by s / 10 ^ 3, where s is the observation index. The fault starts at row 8500, and the four columns under the fault state are defined here:

state : the state indicator for the multivariate system, with three levels

x : x(t) = t + drift + error

y : y(t) = t ^ 2 - 3t + drift + error

z : z(t) = - t ^ 3 + 3t ^ 2 + drift + error

where t is a 10080-entry vector of autocorrelated and non-stationary hidden process realizations. The states alternate each hour and are defined as follows:

State1 – As presented

State2 – Rotated by (yaw = 0, pitch = 90, roll = 30) and scaled by (1 * x, 0.5 * y , 2 * z).

State3 – Rotated by (yaw = 90, pitch = 0, roll = -30) and scaled by (0.25 * x, 0.1 * y , 0.75 * z).

See the vignette for more details.

Source

Simulated in R.

`fault3A_xts` *Process Data under a System Signal Amplification*

Description

Three-feature, three-state simulated process data including observations under normal operating conditions and observations after an amplification of the underlying process for each feature in the system.

Usage

```
fault3A_xts
```

Format

An xts data matrix with 10080 rows and four columns, corresponding to one week worth of data recorded at a 1-minute interval. The columns under normal conditions are defined in the help file for normal_switch_xts. The fault is a signal amplification in the underlying determining t vector. The fault starts at row 8500, and the four columns under the fault state are defined here:

state : the state indicator for the multivariate system, with three levels

x : x(t_*) = t_* + error

y : y(t_*) = (t_*) ^ 2 - 3t + error

z : z(t_*) = - (t_*) ^ 3 + 3(t_*) ^ 2 + error

where t_* = 3 * t * (10080 - s) / (2 * 10080), where s is the observation index, and t is a 10080-entry vector of autocorrelated and non-stationary hidden process realizations. The states alternate each hour and are defined as follows:

State1 – As presented

State2 – Rotated by (yaw = 0, pitch = 90, roll = 30) and scaled by (1 * x, 0.5 * y , 2 * z).

State3 – Rotated by (yaw = 90, pitch = 0, roll = -30) and scaled by (0.25 * x, 0.1 * y , 0.75 * z).

See the vignette for more details.

Source

Simulated in R.

`faultDetect` *Process Fault Detection*

Description

Detect if a single multivariate observation is beyond normal operating conditions.

Usage

```
faultDetect(threshold_object, observation, ...)
```

Arguments

`threshold_object`	An object of classes "threshold" and "pca" returned by the internal threshold() function.
`observation`	A single row of an xts data matrix (a 1 x p matrix) to compare against the thresholds
`...`	Lazy dots for additional internal arguments

Details

This function takes in a threshold object returned by the threshold() function and a single observation as a matrix or xts row. Internally, the function multiplies the observation by the projection matrix returned within the threshold object, calculates the SPE and T2 process monitoring statistics for that observation, and compares these statistics against their corresponding threshold values to determine if the observation lies outside the expected boundaries. The function then returns a row vector of the SPE test statistic, a logical indicator marking if this statistic is beyond the threshold, the Hotelling's T2 statistic, and an indicator if this statistic is beyond the threshold. Observations with monitoring statistics beyond the calculated threshold are marked with a 1, while observations within normal operating conditions are marked with a 0. These threshold values are passed from the threshold() function through this function via a returned threshold object. This object will be used in higher function calls.

This internal function is called by faultFilter().

Value

A named 1 x 4 matrix with the following entries for the single row observation passed to this function:

SPE – the SPE statistic value

SPE_Flag – the SPE fault indicator, where 1 represents a flag and 0 marks that the observation is within the normal operating conditions

T2 – the T2 statistic value

T2_Flag – the T2 fault indicator, defined the same as SPE_Flag

See Also

Called by faultFilter and mspMonitor.

Examples

```
nrml <- mspProcessData(faults = "NOC")
scaledData <- scale(nrml[,-1])
pca_obj <- pca(scaledData)
thresh_obj <- threshold(pca_object = pca_obj)

faultDetect(threshold_object = thresh_obj,
            observation = scaledData[1,])
```

---

faultFilter *Process Fault Filtering*

Description

Flag and filter out observations beyond normal operating conditions, then return the observations within normal operating conditions.

Usage

```
faultFilter(trainData, testData, updateFreq, faultsToTriggerAlarm = 5, ...)
```

Arguments

trainData	An xts data matrix of initial training observations
testData	The data not included in the training data set
updateFreq	The number of observations from the test data matrix that must be returned to update the training data matrix and move it forward.
faultsToTriggerAlarm	Specifies how many sequential faults will cause an alarm to trigger. Defaults to 5.
...	Lazy dots for additional internal arguments

Details

This function is essentially a wrapper function to call and organize the output from these other internal functions: faultDetect(), threshold(), and pca(). It is applied over a rolling window, with observation width equal to updateFreq, of the larger full data matrix via the processMonitor() function, wherein the testing and training data sets move forward in time across the entire data matrix.

This internal function is called by processMonitor().

Value

A list of class "fault_ls" with the following:

faultObj – An xts flagging matrix with the same number of rows as "testData". This flag matrix has the following five columns:

SPE – The SPE statistic value for each observation in "testData". This statistic is defined as

$$SPE_i = (\mathbf{X}_i - \mathbf{Y}_i * \mathbf{P}^T) * (\mathbf{X}_i - \mathbf{Y}_i * \mathbf{P}^T)^T,$$

where $\mathbf{X}_i$ is the i^{th} observation vector, $\mathbf{Y}_i$ is the reduced-feature projection of the observation $\mathbf{X}_i$, and $\mathbf{P}$ is the projection matrix such that $\mathbf{X}_i\mathbf{P} = \mathbf{Y}_i$.

SPE_Flag – A vector of SPE indicators recording 0 if the test statistic is less than or equal to the critical value passed through from the threshold object.

T2 – The T2 statistic value for each observation in "testData". This statistic is defined as

$$T_i^2 = \mathbf{Y}_i * \mathbf{D}^{-1} * \mathbf{Y}_i^T,$$

where $\mathbf{Y}_i = \mathbf{X}_i\mathbf{P}$ is the reduced- feature projection of the observation $\mathbf{X}_i$, and $\mathbf{D}$ is the diagonal matrix of eigenvalues.

T2_Flag – A vector of T2 fault indicators, defined like SPE_Flag.

Alarm – A column indicating if there have been five flags in a row for either the SPE or T2 monitoring statistics or both. Alarm states are as follows: 0 = no alarm, 1 = Hotelling's T2 alarm, 2 = Squared Prediction Error alarm, and 3 = both alarms.

nonAlarmedTestObs – An xts matrix of the first updateFreq number of rows of the training data which were not alarmed.

trainSpecs – The threshold object returned by the internal threshold() function. See the threshold() function's help file for more details.

See Also

Calls: pca, threshold, faultDetect. Called by: processMonitor.

Examples

```
nrml <- mspProcessData(faults = "NOC")
# Select the data under state 1
data <- nrml[nrml[,1] == 1]

faultFilter(trainData = data[1:672, -1],
            testData = data[673:3360, -1],
            updateFreq = 336)
```

`faultSwitch` *Induce the Specified Fault on NOC Observations*

Description

Infect the input data frame with a specific fault, then return the infected data frame.

Usage

```
faultSwitch(
  df,
  fault,
  period = 7 * 24 * 60,
  faultStartIndex = round(0.8433 * period),
  shift = 2,
  postStateSplit = FALSE
)
```

Arguments

`df`	A data frame returned by the processNOCdata() function.
`fault`	A character string. Options are "NOC", "A1", "B1", "C1", "A2", "B2", "C2", "A3", "B3", or "C3". See "details" of mspProcessData() for more information.
`period`	The observation cycle length. Defaults to one week's worth of minute-level observations (10,080 observations).
`faultStartIndex`	An integer specifying the index at which the faults will start. Defaults to roughly 85 percent through the cycle.
`shift`	The fault parameter for faults "A1" and "B1" corresponding to the positive shock value added to features. Defaults to 2. See "details" of mspProcessData() for more information.
`postStateSplit`	Should we induce faults before or after state-splitting? Defaults to FALSE. Make this argument TRUE for faults 1C, 2C, 3C.

Details

The faults return data frames as follows:

A1 – A data frame with 10080 rows and five columns, corresponding by default to one week worth of data recorded at a 1-minute interval (as defined by the "period" argument of this function and the "increment" argument of the processNOCdata() function). The fault is a system shift to each of the three features by 2 (the "shift" argument). The fault starts at row 8500 (specified by the argument "faultStartIndex"), and the five columns under the fault state are defined here:

dateTime : a POSIXct column

state : the state indicator for the multivariate system, with three levels when the argument "multiState" is TRUE and one level otherwise

x : x(t) = t + shift + error

y : y(t) = t ^ 2 - 3t + shift + error

z : z(t) = -t ^ 3 + 3t ^ 2 + shift + error

where t is a 10080-entry vector of autocorrelated and non-stationary hidden process realizations generated within the processNOCdata() function.

B1 – A matrix as defined in A1, but with x, y, and z feature columns defined as follows:

x : x(t) = t + shift + error

y : y(t) = t ^ 2 - 3t + error

z : z(t) = -t ^ 3 + 3t ^ 2 + error

C1 – A matrix as defined in A1, but with x, y, and z feature columns defined as follows:

x : x(t) = t + shift / 4 + error

y : y(t) = t ^ 2 - 3t + error

z : z(t) = -t ^ 3 + 3t ^ 2 + shift / 4 + error

This shift is applied only in State 3.

A2 – The fault is a drift on each feature by (s - faultStartIndex / 10 ^ 3, where s is the observation index. The fault starts at "faultStartIndex", and the x, y, and z feature columns are defined as follows:

x : x(t) = t + drift + error

y : y(t) = t ^ 2 - 3t + drift + error

z : z(t) = -t ^ 3 + 3t ^ 2 + drift + error

B2 – The fault is a drift a drift on the "y" and "z" feature by (s - faultStartIndex / 10 ^ 3, where s is the observation index. The fault starts at "faultStartIndex", and the x, y, and z feature columns are defined as follows:

x : x(t) = t + error

y : y(t) = t ^ 2 - 3t + drift + error

z : z(t) = -t ^ 3 + 3t ^ 2 + drift + error

C2 – The fault is a negative drift on the "y" feature by 1.5 * (s - faultStartIndex) / (period - faultStartIndex). Thus,

x : x(t) = t + error

y : y(t) = t ^ 2 - 3t - drift + error

z : z(t) = -t ^ 3 + 3t ^ 2 + error

This drift is applied only in State 2.

A3 – The fault is a signal amplification in the determining latent t vector. The fault starts at "faultStartIndex", and the x, y, and z features under the fault state are defined here:

x : x(t_*) = t_* + error

y : y(t_*) = (t_*) ^ 2 - 3t_* + error

z : z(t_*) = -(t_*) ^ 3 + 3(t_*) ^ 2 + error

where t_* = 5 x t x (period - s) / (period - faultStartIndex) and s is the observation index.

B3 – The fault is a signal amplification in the determining latent t vector for the "z" feature only. The fault starts at "faultStartIndex", and the x, y, and z features under the fault state are defined here:

x : x(t) = t + error

y : y(t) = (t) ^ 2 - 3t + error

z : z(t_*) = -(t_*) ^ 3 + 3(t_*) ^ 2 + error

where t_* = 3 x t x (period - s) / (2 x period) and s is the observation index.

C3 – This fault is a change in the error structure of feature "y". We let errorNew = 2 * error - 0.25, so that

x : x(t) = t + error

y : y(t) = t ^ 2 - 3t + errorNew

z : z(t) = -t ^ 3 + 3t ^ 2 + error

This new error structure is applied only in State 2.

Value

A data frame with the same structure as df, but with faults induced across all observations. The mspProcessData() function then subsets the observations necessary to corrupt the normal data frame, and binds them together by row. This function is called by mspProcessData(). See ?mspProcessData for more details.

See Also

Called by: `mspProcessData`.

Examples

```
nrml <- processNOCdata()
faultSwitch(nrml, fault = "NOC")
```

`mspContributionPlot` *Contribution Plots*

Description

This function plots the contribution value for each variable of a newly monitored observation and compares them to the contribution values of the training data.

Usage

```
mspContributionPlot(
  trainData,
  trainLabel,
  newData,
  newLabel,
  var.amnt,
  trainObs
)
```

Arguments

trainData	an xts data matrix containing the training observations
trainLabel	Class labels for the training data as a logical (two states only) or finite numeric (two or more states) vector or matrix column (not from a data frame) with length equal to the number of rows in "data." For data with only one state, this will be a vector of 1s.
newData	an xts data matrix containing the new observation
newLabel	the class label for the new observation
var.amnt	the energy proportion to preserve in the projection, which dictates the number of principal components to keep
trainObs	the number of observations upon which to train the algorithm. This will be split based on class information by a priori class membership proportions.

Value

A contribution plot and a list with the following items:

TrainCV – A list vectors containing the contribution values corresponding to each observation in the set of training observations.

NewCV – The vector of contribution values associated with the new observation

Examples

```
## Not run:
# Create some data
dataA1 <- mspProcessData(faults = "B1")
traindataA1 <- dataA1[1:8567,]

# Train on the data that should be in control
trainResults <- mspTrain(traindataA1[,-1], traindataA1[,1], trainObs = 4320)

# Lag an out of control observation
testdataA1 <- dataA1[8567:8568,-1]
testdataA1 <- lag.xts(testdataA1,0:1)
testdataA1 <- testdataA1[-1,]
testdataA1 <- cbind(dataA1[8568,1],testdataA1)

tD <- traindataA1[,-1]
tL <- traindataA1[,1]
nD <- testdataA1[,-1]
nL <- testdataA1[,1]
tO <- 4320
vA <- 0.95

mspContributionPlot(tD, tL, nD, nL, vA, tO)
## End(Not run)
```

`mspMonitor` *Real-Time Process Monitoring Function*

Description

Monitor and flag (if necessary) incoming multivariate process observations.

Usage

```
mspMonitor(observations, labelVector, trainingSummary, ...)
```

Arguments

`observations`	an n x p xts matrix. For real-time monitoring via a script within a batch file, n = 1, so this must be a 1 x p matrix. If lags were included at the training step, then these observations will also have lagged features.
`labelVector`	an n x 1 integer vector of class memberships
`trainingSummary`	the TrainingSpecs list returned by the mspTrain() function. This list contains—for each class—the SPE and T2 thresholds, as well the projection matrix.
`...`	Lazy dots for additional internal arguments

Details

This function is designed to be run at specific time intervals (e.g.every 10 seconds, 30 seconds, 1 minute, 5 minutes, 10 minutes) through a scheduled operating script which calls this function and mspWarning(). We expect this script to be set up in Windows "Task Scheduler" or Macintosh OX "launchd" application suites. This function takes in the specific observations to monitor and their class memberships (if any) and returns an xts matrix of these observation columns concatenated with their monitoring statistic values, flag statuses, and an empty alarm column. Users should then append these rows onto a previously existing matrix of daily observations. The mspWarning() function will then take in the daily observation xts matrix with updated rows returned by this function and check the monitoring statistic flag indicators to see if an alarm status has been reached. For further details, see the mspWarning() function.

This function calls the faultDetect() function, and requires the training information returned by the mspTrain function. This function will return the xts matrix necessary for the mspWarning() function.

Value

An n x (p + 5) xts matrix, where the last five columns are:

SPE – the SPE statistic value for each observation in "observations"

SPE_Flag – a vector of SPE indicators recording 0 if the test statistic is less than or equal to the critical value passed through from the threshold object

T2 – the T2 statistic value for each observation in "observations"

T2_Flag – a vector of T2 fault indicators, defined like SPE_Flag

Alarm – a column indicating if there have been five flags in a row for either the SPE or T2 monitoring statistics or both. Alarm states are as follows: 0 = no alarm, 1 = Hotelling's T2 alarm, 2 = Squared Prediction Error alarm, and 3 = both alarms.

See Also

Calls: `faultDetect`. Pipe flow: `mspTrain` into `mspMonitor` into `mspWarning`.

Examples

```
## Not run: # cut down on R CMD check time

  nrml <- mspProcessData(faults = "NOC")
  n <- nrow(nrml)

  # Calculate the training summary, but save five observations for monitoring.
  trainResults_ls <- mspTrain(data = nrml[1:(n - 5), -1],
                              labelVector = nrml[1:(n - 5), 1],
                              trainObs = 4320)

  # While training, we included 1 lag (the default), so we will also lag the
  #   observations we will test.
  testObs <- nrml[(n - 6):n, -1]
  testObs <- xts:::lag.xts(testObs, 0:1)
  testObs <- testObs[-1,]
  testObs <- cbind(nrml[(n - 5):n, 1], testObs)

  mspMonitor(observations = testObs[, -1],
             labelVector = testObs[, 1],
             trainingSummary = trainResults_ls$TrainingSpecs)


## End(Not run)
```

`mspProcessData` *Simulate Normal or Fault Observations from a Single-State or Multi-State Process*

Description

Generate single- or multi-state observations under normal operating conditions or under fault conditions.

Usage

```
mspProcessData(
  faults,
  period = 7 * 24 * 60,
  faultStartIndex = round(0.8433 * period),
  startTime = "2015-05-16 10:00:00 CST",
  multiState = TRUE,
  angles2 = list(yaw = 0, pitch = 90, roll = 30),
  scales2 = c(1, 0.5, 2),
  angles3 = list(yaw = 90, pitch = 0, roll = -30),
  scales3 = c(0.25, 0.1, 0.75),
  adpcaTest = FALSE,
  msadpcaTest = FALSE,
  ...
)
```

Arguments

faults	A character vector of faults chosen. Options are "NOC", "A1", "B1", "C1", "A2", "B2", "C2", "A3", "B3", "C3", or "All". See details for more information.
period	The observation cycle length. Defaults to one week's worth of minute-level observations (10,080 observations).
faultStartIndex	An integer specifying the index at which the faults will start. Defaults to roughly 85 percent through the cycle.
startTime	a POSIXct object specifying the day and time for the starting observation.
multiState	Should the observations be generated from a multi-state process? Defaults to TRUE.
angles2	Change the principal angles for State 2. Defaults to yaw = 0, pitch = 90, and roll = 30.
scales2	Change the principal scales for State 2. Defaults to 1, 0.5, and 2.
angles3	Change the principal angles for State 3. Defaults to yaw = 90, pitch = 0, and roll = -30.
scales3	Change the principal scales for State 3. Defaults to 0.25, 0.1, and 0.75.
adpcaTest	If "multiState" is TRUE, incorrectly label all the states the same. This should only be used to test AD-PCA performance under a true multi-state model. Defaults to FALSE.
msadpcaTest	If "multiState" is FALSE, incorrectly label all the states at random. This should only be used to test MSAD-PCA performance under a true single-state model. Defaults to FALSE.
...	Lazy dots for internal arguments

Details

For details on how the faults are induced, see the "details" of the faultSwitch() function. This function also includes AD-PCA versus MSAD-PCA treatment arm testing. There are four possibilities to test:

1. The true process has one state, and we correctly assume the true process has one state. In this case, AD-PCA and MSAD-PCA are exactly the same. Draw observations from this state by setting the "multiState" argument to FALSE. The "state" label will correctly mark each observation as from the same state.
2. The true process has one state, but we incorrectly assume the true process has multiple states. In this case, AD-PCA should outperform MSAD-PCA in false alarm rates and waiting time to the first alarm. Draw observations from this state by setting the "multiState" argument to FALSE and the "msadpcaTest" argument to TRUE. The "state" label will be contain randomly generated state values (1, 2, and 3 are all equally likely) for each observation.
3. The true process has multiple states, but we incorrectly assume the true process has one single states. In this case, MSAD-PCA should outperform AD-PCA in false alarm rates and waiting time to the first alarm. Draw observations from this state by setting the "multiState" argument to TRUE and the "adpcaTest" argument to TRUE. The "state" label will be identical for each observation.
4. The true process has multiple states, and we correctly assume the true process has multiple states. In this case, MSAD-PCA should outperform AD-PCA in false alarm rates and waiting time to the first alarm. Draw observations from this state by setting the "multiState" argument to TRUE. The "state" label will correctly mark each observation as from the same state.

Value

A list of data frames named with the names of the given faults with the following information:

dateTime – A POSIXct column of times starting at the user- defined ‘startTime‘ argument, length given by the ‘period‘ argument, and spacing given by the ‘increment‘ argument. For example, if the starting value is "2016-01-10", period is 10080, and the incrementation is in minutes, then this sequence will be one week’s worth of observations recorded every minute from midnight on the tenth of January.

state – An integer column of all 1’s (when the ‘multiState‘ argument is FALSE), or a column of the state values (1, 2 or 3).

altState – If either adpcaTest or msadpcaTest are TRUE, this column will contain incorrect state information used for testing the different treatment arms against their respective controls.

x – A double column of generated values for the first feature.

y – A double column of generated values for the second feature.

z – A double column of generated values for the third feature.

If the user only specifies one fault, then this function will return the single xts matrix, instead of a list of one matrix. For details on how these features are defined, see the "details" of the processNOCdata() function.

See Also

Calls: `processNOCdata`, `faultSwitch`, `dataStateSwitch`. Simulation pipe flow: `mspProcessData` into `mspTrain` into `mspMonitor` into `mspWarning`.

Examples

```
## Not run: # cut down on R CMD check time

   mspProcessData(faults = "All")


## End(Not run)
```

`mspSPEPlot` *Squared Prediction Error Contribution Plots*

Description

Plots a variation of the squared prediction error (SPE) statistic to visualize the contribution of each variable to a fault.

Usage

```
mspSPEPlot(
  trainData,
  trainLabel,
  trainSPE,
  newData,
  newLabel,
  newSPE,
  trainObs,
  var.amnt
)
```

Arguments

`trainData`	an xts data matrix containing the training observations
`trainLabel`	Class labels for the training data as a logical (two states only) or finite numeric (two or more states) vector or matrix column (not from a data frame) with length equal to the number of rows in "data." For data with only one state, this will be a vector of 1s.
`trainSPE`	the SPE values corresponding to the newLabel state calculated by mspTrain using the full training data with all variables included
`newData`	an xts data matrix containing the new observation
`newLabel`	the class label for the new observation
`newSPE`	the SPE value returned by mspMonitor using the full new observation with all variables included
`trainObs`	the number of observations upon which to train the algorithm. This will be split based on class information by a priori class membership proportions.
`var.amnt`	the energy proportion to preserve in the projection, which dictates the number of principal components to keep

Examples

```
## Not run:
# Create some data
dataA1 <- mspProcessData(faults = "B1")
traindataA1 <- dataA1[1:8567,]

# Train on the data that should be in control
trainResults <- mspTrain(traindataA1[,-1], traindataA1[,1], trainObs = 4320)


# Lag an out of control observation
testdataA1 <- dataA1[8567:8568,-1]
testdataA1 <- lag.xts(testdataA1,0:1)
testdataA1 <- testdataA1[-1,]
testdataA1 <- cbind(dataA1[8568,1],testdataA1)

# Monitor this observation
monitorResults <- mspMonitor(observations = testdataA1[,-1],
                             labelVector = testdataA1[,1],
                             trainingSummary = trainResults$TrainingSpecs)


tD <- traindataA1[,-1]
tL <- traindataA1[,1]
nD <- testdataA1[,-1]
nL <- testdataA1[,1]
tO <- trainObs
vA <- 0.95
nSPE <- monitorResults$SPE
tSPE <- trainResults$TrainingSpecs[[nL]]$SPE

mspSPEPlot(tD,tL,tSPE,nD,nL,nSPE,tO,vA)

## End(Not run)
```

`mspSubset` *Multi-State Subsetting*

Description

This function separates the data into k subsets, one for each of the k states, containing the subset of the original variables that are of interest for a given state.

Usage

```
mspSubset(
  data,
  labelVector = rep(1, nrow(data)),
 subsetMatrix = matrix(TRUE, nrow = length(unique(labelVector)), ncol = ncol(data))
)
```

Arguments

data	An xts data matrix
labelVector	Class labels as a logical (two states only) or finite numeric (two or more states) vector or matrix column (not from a data frame) with length equal to the number of rows in "data." For data with only one state, this will be a vector of 1s.
subsetMatrix	A matrix of logicals with number of rows equal to the number of states and number of columns equal to the number of columns in data. The i,j entry in the matrix indicates whether or not to monitor the jth variable in the ith state.

Details

This function is designed to be used in conjunction with mspTrain and to allow the user to monitor a different subset of the variables during each state.

Value

A list with the following components:

Class1Data --	an xts data matrix containing the subset of the state 1 data.
Class2Data --	an xts data matrix containing the subset of the state 2 data.
Class3Data --	an xts data matrix containing the subset of the state 3 data.

Examples

```
nrml <- mspProcessData(faults = "NOC")

sub1 <- c(TRUE,TRUE,FALSE)
sub2 <- c(TRUE,FALSE,TRUE)
sub3 <- c(TRUE,FALSE,FALSE)
submatrix <- t(matrix(c(sub1,sub2,sub3),nrow=3,ncol=3))

subsets <- mspSubset(data = nrml[,-1],
labelVector = nrml[,1],
subsetMatrix = submatrix)
```

---

mspT2Plot *T-Squared Contribution Plots*

Description

Plots a variation of the Hotelling's T-squared statistic to visualize the contribution of each variable to a fault.

Usage

```
mspT2Plot(
  trainData,
  trainLabel,
  trainT2,
  newData,
  newLabel,
  newT2,
  trainObs,
  var.amnt
)
```

Arguments

trainData	an xts data matrix containing the training observations
trainLabel	Class labels for the training data as a logical (two states only) or finite numeric (two or more states) vector or matrix column (not from a data frame) with length equal to the number of rows in “data." For data with only one state, this will be a vector of 1s.
trainT2	the Hotelling’s T-squared values corresponding to the newLabel state calculated by mspTrain using the full training data with all variables included
newData	an xts data matrix containing the new observation
newLabel	the class label for the new observation
newT2	the Hotelling’s T-squared value returned by mspMonitor using the full new observation with all variables included
trainObs	the number of observations upon which to train the algorithm. This will be split based on class information by a priori class membership proportions.
var.amnt	the energy proportion to preserve in the projection, which dictates the number of principal components to keep

Examples

```
## Not run:
# Create some data
dataA1 <- mspProcessData(faults = "B1")
traindataA1 <- dataA1[1:8567,]

# Train on the data that should be in control
trainResults <- mspTrain(traindataA1[,-1], traindataA1[,1], trainObs = 4320)


# Lag an out of control observation
testdataA1 <- dataA1[8567:8568,-1]
testdataA1 <- lag.xts(testdataA1,0:1)
testdataA1 <- testdataA1[-1,]
testdataA1 <- cbind(dataA1[8568,1],testdataA1)

# Monitor this observation
```

```
monitorResults <- mspMonitor(observations = testdataA1[,-1],
                             labelVector = testdataA1[,1],
                             trainingSummary = trainResults$TrainingSpecs)


tD <- traindataA1[,-1]
tL <- traindataA1[,1]
nD <- testdataA1[,-1]
nL <- testdataA1[,1]
tO <- 4320
vA <- 0.95
nT2 <- monitorResults$T2
tT2 <- trainResults$TrainingSpecs[[nL]]$T2

mspT2Plot(tD,tL,tT2,nD,nL,nT2,tO,vA)

## End(Not run)
```

`mspTrain` *Multi-State Adaptive-Dynamic Process Training*

Description

This function performs Multi-State Adaptive-Dynamic PCA on a data set with time-stamped observations.

Usage

```
mspTrain(
  data,
  labelVector,
  trainObs,
  updateFreq = ceiling(0.5 * trainObs),
  Dynamic = TRUE,
  lagsIncluded = c(0, 1),
  faultsToTriggerAlarm = 5,
  ...
)
```

Arguments

`data`	An xts data matrix
`labelVector`	Class labels as a logical (two states only) or finite numeric (two or more states) vector or matrix column (not from data frame) with length equal to the number of rows in "data". For data with only one state, this will be a vector of 1s.
`trainObs`	The number of observations upon which to train the algorithm. This will be split based on class information by a priori class membership proportion.

`updateFreq` The algorithm update frequency. Defaults to half as many observations as the training frequency.

`Dynamic` Specify if the PCA algorithm should include lagged variables. Defaults to TRUE.

`lagsIncluded` A vector of lags to include. If Dynamic = TRUE, specify which lags to include. Defaults to c(0, 1), signifying that the Dynamic process observations will include current observations and observations from one time step previous. See "Details" for more information.

`faultsToTriggerAlarm`
The number of sequential faults needed to trigger an alarm. Defaults to 5.

`...` Lazy dots for additional internal arguments

Details

This function is designed to identify and sort out sequences of observations which fall outside normal operating conditions. We assume that the process data are time-dependent in both seasonal and non-stationary effects (which necessitate the Adaptive and Dynamic components, respectively). We further assume that this data is drawn from a multivariate process under multiple mutually exclusive states, implying that the linear dimension reduction projection matrices may be different for each state. Therefore, in summary, this function lags the features to account for correlation between sequential observations, splits the data by classes, and re-estimates projection matrices on a rolling window to account for seasonality. Further, this function uses non-parametric density estimation to calculate the 1 - alpha quantiles of the SPE and Hotelling's T2 statistics from a set of training observations, then flags any observation in the testing data set with process monitoring statistics beyond these calculated critical values. Because of natural variability inherent in all real data, we do not remove observations simply because they are have been flagged as outside normal operating conditions. This function records an alarm only for observations having five flags in a row, as set by the default argument value of "faultsToTriggerAlarm". These alarm-positive observations are then removed from the data set and held in a separate xts matrix for inspection.

Concerning the lagsIncluded variable: the argument lagsIncluded = c(0,1) will column concatenate the current data with the same data from one discrete time step back. This will necessarily remove the first row of the data matrix, as we will have NA values under the lagged features. The argument lagsIncluded = 0:2 will column concatenate the current observations with the observations from one step previous and the observations from two steps previous, which will necessarily require the removal of the first two rows of the data matrix. To include only certain lags with the current data, specify lagsIncluded = c(0, lag_1, lag_2, ..., lag_K). This induce NA values in the first max(lag_k) rows, for k = 1, ..., K, and these rows will be removed from consideration. From the lag.xts() function helpfile: "The primary motivation for having methods specific to xts was to make use of faster C-level code within xts. Additionally, it was decided that lag's default behavior should match the common time-series interpretation of that operator — specifically that a value at time 't' should be the value at time 't-1' for a positive lag. This is different than lag.zoo() as well as lag.ts()."

Of note when considering performance: the example has 10080 rows on three features alternating between three states, and trains on 20 percent of the observations, while updating every 1008 (10 percent) observation. On a 2016 Macbook Pro with 16Gb of RAM, this example function call takes 15 second to run. Increasing the update frequency will decrease computation time, but may increase false alarm rates or decrease flagging accuracy. We recommend that you set the update frequency based on the natural and physical designs of your system. For example, if your system

has a multi-state process which switches across one of four states every two hours, then test the update frequency at an eight or 12 hour level — enough observations to measure two to three full cycles of the switching process. For observations recorded every five minutes, try updateFreq = (60 / 5) * 8 = 96 or (60 / 5) * 12 = 144.

This user-facing function calls the processMonitor() function, and returns the training arguments necessary to call the mspMonitor() and mspWarning() functions.

For more details, see Kazor et al (2016):

doi:10.1007/s0047701612462

Value

A list with the following components:

FaultChecks – an xts flagging matrix with the same number of rows as "data". This flag matrix has the following five columns:

- **SPE –** the SPE statistic value for each observation in "data"
- **SPE_Flag –** a vector of SPE indicators recording 0 if the test statistic is less than or equal to the critical value passed through from the threshold object
- **T2 –** the T2 statistic value for each observation in "data"
- **T2_Flag –** a vector of T2 fault indicators, defined like SPE_Flag
- **Alarm –** a column indicating if there have been five flags in a row for either the SPE or T2 monitoring statistics or both. Alarm states are as follows: 0 = no alarm, 1 = Hotelling's T2 alarm, 2 = Squared Prediction Error alarm, and 3 = both alarms.

Non_Alarmed_Obs – an xts data matrix of all the non-alarmed observations

Alarms – an xts data matrix of the features and specific alarms for Alarmed observations with the alarm codes are listed above

TrainingSpecs – a list of k lists, one for each class, with each list containing the specific threshold object returned by the internal threshold() function for that class. See the threshold() function's help file for more details.

See Also

Calls: `processMonitor`. Pipe flow: `mspTrain` into `mspMonitor` into `mspWarning`.

Examples

```
## Not run: # cut down on R CMD check time

  nrml <- mspProcessData(faults = "NOC")

  mspTrain(data = nrml[, -1],
         labelVector = nrml[, 1],
         trainObs = 4320)


## End(Not run)
```

mspWarning *Process Alarms*

Description

Trigger an alarm, if necessary, for incoming multivariate process observations.

Usage

```
mspWarning(mspMonitor_object, faultsToTriggerAlarm = 5)
```

Arguments

mspMonitor_object	An xts matrix returned by the mspMonitor() function
faultsToTriggerAlarm	Specifies how many sequential faults will cause an alarm to trigger. Defaults to 5.

Details

This function and the mspMonitor() function are designed to be ran via a scheduled task through Windows "Task Scheduler" or Macintosh OX "launchd" application suites. The file flow is as follows: at each time interval, run the mspMonitor() function on the matrix of daily observations to add a flag status to the most recent incoming observation in the matrix, and return this new xts matrix. Then, pass this updated daily observation matrix to the mspWarning() function, which will check if the process has recorded five or more sequential monitoring statistic flags in a row. Of note, because these functions are expected to be repeatedly called in real time, this function will only check for an alarm within the last row of the xts matrix. To check multiple rows for an alarm state, please use the mspTrain() function, which was designed to check multiple past observations.

This function requires an xts matrix returned by the mspMonitor() function.

Value

An xts matrix of the same dimensions as mspMonitor_object, with a recorded negative or positive and type-specific alarm status. Alarm codes are: 0 = no alarm, 1 = Hotelling's T2 alarm, 2 = Squared Prediction Error alarm, and 3 = both alarms.

See Also

Pipe flow: mspTrain into mspMonitor into mspWarning.

Examples

```
## Not run: # cut down on R CMD check time

  nrml <- mspProcessData(faults = "NOC")
  n <- nrow(nrml)

  # Calculate the training summary, but save five observations for monitoring.
  trainResults_ls <- mspTrain(data = nrml[1:(n - 5), -1],
                              labelVector = nrml[1:(n - 5), 1],
                              trainObs = 4320)

  # While training, we included 1 lag (the default), so we will also lag the
  #   observations we will test.
  testObs <- nrml[(n - 6):n, -1]
  testObs <- xts:::lag.xts(testObs, 0:1)
  testObs <- testObs[-1,]
  testObs <- cbind(nrml[(n - 5):n, 1], testObs)

  # Run the monitoring function.
  dataAndFlags <- mspMonitor(observations = testObs[, -1],
                             labelVector = testObs[, 1],
                             trainingSummary = trainResults_ls$TrainingSpecs)

  # Alarm check the last row of the matrix returned by the mspMonitor
  #   function
  mspWarning(dataAndFlags)


## End(Not run)
```

`mvMonitoring` *A Package for Multivariate Statistical Process Monitoring*

Description

The `mvMonitoring` package has four main functions for external use, all of which begin with the string "msp" (for "multivariate statistical process") followed by the function use. Functions without this "msp" key are primarily internal functions. They are available to see and use, but will largely be unnecessary to call in common workflows.

`mvMonitoring` external functions

`mspProcessData` - A function for synthetic process data generation. Use this data to test new process monitoring methods.

`mspTrain` - A function to take in observations for training under normal conditions, and to return the training summary from these observations.

`mspMonitor` - A function to take in real-time process observations and detect system anomalies based on the training summary returned by `mspTrain`.

`mspWarning` - A function to take in observations returned by `mspMonitor` and check for alarms by measuring sequential anomalies. This function will also be equipped to send SMS notifications to process technicians in future versions.

`normal_switch_xts` *Process Data under Normal Conditions*

Description

Three-feature, three-state simulated process data under normal operating conditions as example data for different included functions.

Usage

```
normal_switch_xts
```

Format

An xts data matrix with 10080 rows and four columns, corresponding to one week worth of data recorded at a 1-minute interval, and four columns as defined here:

state – the state indicator for the multivariate system, with three levels

x : x(t) = t + error

y : y(t) = t ^ 2 - 3t + error

z : z(t) = - t ^ 3 + 3t ^ 2 + error

where t is a 10080-entry vector of autocorrelated and non-stationary hidden process realizations. The states alternate each hour and are defined as follows:

State1 – As presented

State2 – Rotated by (yaw = 0, pitch = 90, roll = 30) and scaled by (1 * x, 0.5 * y , 2 * z).

State3 – Rotated by (yaw = 90, pitch = 0, roll = -30) and scaled by (0.25 * x, 0.1 * y , 0.75 * z).

See the vignette for more details.

Source

Simulated in R.

oneDay_clean *Real Process Data for Testing*

Description

Data from the SM-MBR Bioreactor system over 12 hours. This data will be used for testing the mvMonitoring package.

Usage

```
oneDay_clean
```

Format

An xts matrix of 75 rows and 35 features recorded over 2017-01-27 at 00:10 to 2017-01-27 at 12:30.

Source

Kathryn Newhart

pca *PCA for Data Scatter Matrix*

Description

Calculate the principal component analysis for a data matrix, and also find the squared prediction error (SPE) and Hotelling's T2 test statistic values for each observation in this data matrix.

Usage

```
pca(data, var.amnt = 0.9, ...)
```

Arguments

data	A centred-and-scaled data matrix or xts matrix
var.amnt	The energy proportion to preserve in the projection, which dictates the number of principal components to keep. Defaults to 0.90.
...	Lazy dots for additional internal arguments

Details

This function takes in a training data matrix, without the label column, and the energy preservation proportion, which defaults to 95 percent per Kazor et al (2016). This proportion is the sum of the q largest eigenvalues divided by the sum of all p eigenvalues, where q is the number of columns of the p x q projection matrix P. This function then returns the projection matrix P, a diagonal matrix of the reciprocal eigenvalues (LambdaInv), a vector of the SPE test statistic values corresponding to the rows of the data matrix, and a T2 test statistic vector similar to the SPE vector.

This internal function is called by faultFilter().

Value

A list of class "pca" with the following:

projectionMatrix – the q eigenvectors corresponding to the q largest eigenvalues as a p x q projection matrix

LambdaInv – the diagonal matrix of inverse eigenvalues

SPE – the vector of SPE test statistic values for each of the n observations contained in "data"

T2 – the vector of Hotelling's T2 test statistic for each of the same n observations

See Also

Called by: faultFilter.

Examples

```
nrml <- mspProcessData(faults = "NOC")
scaledData <- scale(nrml[,-1])
pca(scaledData)
```

---

processMonitor *Adaptive Process Training*

Description

Apply Adaptive-Dynamic PCA to state-specific data matrices.

Usage

```
processMonitor(
  data,
  trainObs,
  updateFreq = ceiling(0.5 * trainObs),
  faultsToTriggerAlarm = 5,
  ...
)
```

Arguments

data	An xts data matrix
trainObs	The number of training observations to be used
updateFreq	The number of non-flagged observations to collect before the function updates. Defaults to half as many observations as the number of training observations.
faultsToTriggerAlarm	The number of sequential faults needed to trigger an alarm. Defaults to 5.
...	Lazy dots for additional internal arguments

Details

This function is the class-specific implementation of the Adaptive- Dynamic PCA described in the details of the mspTrain() function. See the mspTrain() function's help file for further details.

This internal function is called by mspTrain(). This function calls the faultFilter() function.

Value

A list with the following components:

FaultChecks – a class-specific xts flagging matrix with the same number of rows as "data". This flag matrix has the following five columns:

- **SPE –** the SPE statistic value for each observation in "data"
- **SPE_Flag –** a vector of SPE indicators recording 0 if the test statistic is less than or equal to the critical value passed through from the threshold object
- **T2 –** the T2 statistic value for each observation in "data"
- **T2_Flag –** a vector of T2 fault indicators, defined like SPE_Flag
- **Alarm –** a column indicating if there have been five flags in a row for either the SPE or T2 monitoring statistics or both. Alarm states are as follows: 0 = no alarm, 1 = Hotelling's T2 alarm, 2 = Squared Prediction Error alarm, and 3 = both alarms.

Non_Alarmed_Obs – a class-specific xts data matrix of all the non-alarmed observations (observations with alarm state equal to 0)

Alarms – a class-specific xts data matrix of the features and specific alarms of Alarmed observations, where the alarm codes are listed above

trainSpecs – a threshold object returned by the internal threshold() function. See the threshold() function's help file for more details.

See Also

Calls: `faultFilter`. Called by: `mspTrain`.

Examples

```
nrml <- mspProcessData(faults = "NOC")
data <- nrml[nrml[,1] == 1]

processMonitor(data = data[,-1], trainObs = 672)
```

`processNOCdata` *Simulate NOC Observations from a Single-State or Multi-State Process*

Description

This function generates data under normal operating conditions from a single-state or multi-state process model.

Usage

```
processNOCdata(
  startTime = "2015-05-16 10:00:00 CST",
  period = 7 * 24 * 60,
  stateDuration = 60,
  increment = "min",
  multiState = TRUE,
  autocorellation = 0.75,
  tLower = 0.01,
  tUpper = 2,
  errVar = 0.01
)
```

Arguments

startTime	a POSIXct object specifying the day and time for the starting observation.
period	The observation cycle length. Defaults to one week's worth of minute-level observations (10,080 observations).
stateDuration	The number of observations generated during a stay in each state. Defaults to 60.
increment	The time-sequence base increment. See "Details" of the seq.POSIXt() function options. Defaults to "min" for minutes.
multiState	Should the observations be generated from a multi-state process? Defaults to TRUE.
autocorellation	The autocorrelation parameter. Must be less than 1 in absolute value, or the process generated will be nonstationary. Defaults to 0.75 in accordance to Kazor et al (2016).
tLower	Lower bound of the latent t variable. Defaults to 0.01.
tUpper	Upper bound of the latent t variable. Defaults to 2.
errVar	Error variance of the normal white noise process on the feature variables.

Details

This function randomly generates a non-stationary (sinusoidal) and autocorrelated latent variable t with lower and upper bounds given by the arguments "tLower" and "tUpper", respectively, with autocorrelation governed by the "autocorrelation" argument. Necessarily, this coefficient must be less than 1 in absolute value, otherwise the latent variable will be unbounded. Next, this function draws a realization of this random variable t and calculates three functions of it, then jitters these functions with a normal white noise variable (with variance set by "errVar"). These three functions are:

x : x(t) = t + error

y : y(t) = t ^ 2 - 3t + error

z : z(t) = -t ^ 3 + 3t ^ 2 + error

This function is called by the mspProcessData() function. See ?mspProcessData for more details.

Value

An data frame with the following information:

dateTime – A POSIXct column of times starting at the user-defined ‘startTime‘ argument, length given by the ‘period‘ argument, and spacing given by the ‘increment‘ argument. For example, if the starting value is "2016-01-10", period is 10080, and the incrementation is in minutes, then this sequence will be one week’s worth of observations recorded every minute from midnight on the tenth of January.

state – An integer column of all 1’s (when the ‘multiState‘ argument is FALSE), or a column of the state values (1, 2 or 3).

x – A double column of generated values for the first feature.

y – A double column of generated values for the second feature.

z – A double column of generated values for the third feature.

See Also

Called by: `mspProcessData`.

Examples

```
processNOCdata()
```

`quantile.density` *Extract Quantiles from ’density’ Objects*

Description

Quantiles for objects of class `density`

Usage

```
## S3 method for class 'density'
quantile(x, probs = seq(0.25, 0.75, 0.25), names = TRUE, normalize = TRUE, ...)
```

Arguments

`x`	a object of class `density` or a list of densities
`probs`	numeric vector of probabilities with values in [0,1]. Note that elements very close to the boundaries return `Inf` or `-Inf`
`names`	logical; if `TRUE`, the result has a names attribute, resp. a rownames and colnames attributes. Set to `FALSE` for speedup with many probabilities
`normalize`	logical; if `TRUE` then the values in `x$y` are multiplied with a factor such that their integral is equal to one.
`...`	further arguments passed to or from other methods (currently unused)

Details

This function is a near-exact copy of the `quantile.density` function from package BMS (https://CRAN.R-project.org/package=BMS). In spring of 2022, CRAN informed us that the BMS has been orphaned, so we copied the code (and corresponding documentation) we needed from it. See doi:10.18637/jss.v068.i04 for their paper.

The function `quantile.density()` applies generically to the built-in class `density` (as least for versions where there is no such method in the pre-configured packages). Note that this function relies on trapezoidal integration in order to compute the cumulative densities necessary for the calculation of quantiles.

Value

If x is of class `density` (or a list with exactly one element), a vector with quantiles. If x is a list of densities, then the output is a matrix of quantiles, with each matrix row corresponding to the respective density.

Author(s)

Stefan Zeugner, <stefan.zeugner@ec.europa.eu>

Martin Feldkircher, <martin.feldkircher@da-vienna.ac.at>

Examples

```
rNorm_dens <- density(rnorm(100000))
quantile(rNorm_dens)
```

---

rotate3D *Three-Dimensional Rotation Matrix*

Description

Render a 3-Dimensional projection matrix given positive or negative degree changes in yaw, pitch, and / or roll.

Usage

```
rotate3D(yaw, pitch, roll)
```

Arguments

`yaw`	z-axis change in degrees; look left (+) or right (-). Consider this a rotation on the x-y plane.
`pitch`	y-axis change in degrees; look up (-) or down (+). Consider this a rotation on the x-z plane.
`roll`	x-axis change in degrees; this change appears as if you touch head to shoulders: right roll (+) and left roll (-).

Details

When plotting with the package scatterplot3d, the default perspective is such that the pitch action appears as a roll while the roll action appears as a pitch.

This function is used only in data generation of the package vignette. This function is called by rotateScale3D().

Value

A 3 x 3 projection matrix corresponding to the degree changes entered.

See Also

Called by: `rotateScale3D`.

Examples

```
rotate3D(yaw = -10, pitch = 0, roll = 15)
```

---

rotateScale3D *Three-Dimensional Rotation and Scaling Matrix*

Description

Render a 3-Dimensional projection matrix given positive or negative degree changes in yaw, pitch, and / or roll and increment or decrement feature scales.

Usage

```
rotateScale3D(rot_angles = c(0, 0, 0), scale_factors = c(1, 1, 1))
```

Arguments

`rot_angles`	a list or vector containing the rotation angles in the order following: yaw, pitch, roll. Defaults to <0,0,0>.
`scale_factors`	a list or vector containing the values by which to multiply each dimension. Defaults to <1,1,1>.

Details

See the help file of function rotate_3D() for a brief explanation of how these angles behave in scatterplot3d functionality (from package scatterplot3d).

This function is used only in data generation in the package vignette (version 1) and the dataStateSwitch() function within the mspProcessData() function. This function calls rotate3D().

Value

A 3 x 3 projection matrix corresponding to the degree and scale changes entered.

See Also

Calls: rotate3D. Called by dataStateSwitch.

Examples

```
rotateScale3D(rot_angles = list(yaw = -10, pitch = 0, roll = 15),
              scale_factors = c(0.2, 1, 5))
```

---

tenDay_clean *Real Process Data for Training*

Description

Data from the SM-MBR Bioreactor system over ten days. This data will be used for training the mvMonitoring package.

Usage

```
tenDay_clean
```

Format

An xts matrix of 1,299 rows and 35 features recorded over 2017-01-17 at 00:10 to 2017-01-27 at 00:00.

Source

Kathryn Newhart

---

threshold *Non-parametric Threshold Estimation*

Description

Calculate the non-parametric critical value threshold estimates for the SPE and T2 monitoring test statistics.

Usage

```
threshold(pca_object, alpha = 0.001, ...)
```

Arguments

`pca_object`	A list with class "pca" from the internal pca() function
`alpha`	The upper 1 - alpha quantile of the SPE and T2 densities from the training data passed to this function. Defaults to 0.001.
`...`	Lazy dots for additional internal arguments

Details

This function takes in a pca object returned by the pca() function and a threshold level defaulting to alpha = 0.1 percent of the observations. This critical quantile is set this low to reduce false alarms, as described in Kazor et al (2016). The function then returns a calculated SPE threshold corresponding to the 1 - alpha critical value, a similar T2 threshold, and the projection and Lambda Inverse (1 / eigenvalues) matrices passed through from the pca() function call.

This internal function is called by faultFilter().

Value

A list with classes "threshold" and "pca" containing:

SPE_threshold – the 1 - alpha quantile of the estimated SPE density

T2_threshold – the 1 - alpha quantile of the estimated Hotelling's T2 density

projectionMatrix – a projection matrix from the data feature space to the feature subspace which preserves some pre-specified proportion of the energy of the data scatter matrix. This pre-specified energy proportion is user supplied as the var.amnt argument in the pca() function. See the pca() function's help file for more details.

LambdaInv – a diagonal matrix of the reciprocal eigenvalues of the data scatter matrix

T2 – the vector of Hotelling's T2 test statistic values for each of the n observations in "data"

SPE – the vector of SPE test statistic values for each of the n observations in "data"

See Also

Called by: `faultFilter`. This function uses a port of the `quantile.density()` function from the now-orphaned BMS package.

Examples

```
nrml <- mspProcessData(faults = "NOC")
scaledData <- scale(nrml[,-1])
pca_obj <- pca(scaledData)
threshold(pca_object = pca_obj)
```

Index