Package 'DriftBurstHypothesis'

October 12, 2022

Type Package

Title Calculates the Test-Statistic for the Drift Burst Hypothesis

Version 0.4.0.1

Date 2020-07-13

Author Emil Sjoerup

Maintainer Emil Sjoerup <emilsjoerup@live.dk>

Description Calculates the T-Statistic for the drift burst hypothesis from the working paper Christensen, Oomen and Reno (2018) <DOI:10.2139/ssrn.2842535>. The authors' MATLAB code is available upon request, see: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2842535>.

License GPL-3

BugReports https://github.com/emilsjoerup/DriftBurstHypothesis/issues

URL https://github.com/emilsjoerup/DriftBurstHypothesis

NeedsCompilation yes

Depends R (>= 3.5.0)

Imports Rcpp (>= 0.12.18), xts, zoo,

LinkingTo Rcpp, RcppArmadillo

Suggests testthat

Repository CRAN

Date/Publication 2020-07-13 20:40:02 UTC

R topics documented:

`DriftBurstHypothesis-package` *Calculates the Test-Statistic for the Drift Burst Hypothesis*

Description

Calculates the T-Statistic for the drift burst hypothesis from the working paper Christensen, Oomen and Reno (2018) <DOI:10.2139/ssrn.2842535>. The authors' MATLAB code is available upon request, see: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2842535>.

Details

The DESCRIPTION file:

Package:	DriftBurstHypothesis
Type:	Package
Title:	Calculates the Test-Statistic for the Drift Burst Hypothesis
Version:	0.4.0.1
Date:	2020-07-13
Author:	Emil Sjoerup
Maintainer:	Emil Sjoerup <emilsjoerup@live.dk>
Description:	Calculates the T-Statistic for the drift burst hypothesis from the working paper Christensen, Oomen and
License:	GPL-3
BugReports:	https://github.com/emilsjoerup/DriftBurstHypothesis/issues
URL:	https://github.com/emilsjoerup/DriftBurstHypothesis
NeedsCompilation:	Yes
Depends:	R (>= 3.5.0)
Imports:	Rcpp (>= 0.12.18), xts, zoo,
LinkingTo:	Rcpp, RcppArmadillo
Suggests:	testthat

Index of help topics:

```
DriftBurstHypothesis-package
                        Calculates the Test-Statistic for the Drift
                        Burst Hypothesis
driftBursts             Drift Bursts
print.DBH               Methods for the DBH class
```

Author(s)

Emil Sjoerup

Maintainer: Emil Sjoerup <emilsjoerup@live.dk>

References

Christensen, Oomen and Reno (2018) <DOI:10.2139/ssrn.2842535>.

`driftBursts` *Drift Bursts*

Description

Calculates the Test-Statistic for the Drift Burst Hypothesis.

Usage

```
driftBursts(timestamps = NULL, logPrices, testTimes = seq(34260, 57600, 60),
                        preAverage = 5, ACLag = -1L,
                        meanBandwidth = 300L, varianceBandwidth = 900L,
                        sessionStart = 34200, sessionEnd = 57600,
                        parallelize = FALSE, nCores = NA, warnings = TRUE)
```

Arguments

`timestamps`	Either: A `numeric` of `timestamps` for the trades in seconds after midnight. Or: `NULL`, when the time argument is `NULL`, the `logprices` argument must be an `xts` object.
`logPrices`	A `numeric` or `xts` object containing the log-prices.
`testTimes`	A `numeric` containing the times at which to calculate the tests. The standard of `seq(34260, 57600, 60)` denotes calculating the test -statistic once per minute, i.e. 390 times for a typical 6.5 hour trading day from 9:31:00 to 16:00:00. See details.
`preAverage`	An `integer` denoting the length of pre-averaging estimates of the log-prices.
`ACLag`	A positive `integer` greater than 1 denoting how many lags are to be used for the HAC estimator of the variance - the standard of `-1` denotes using an automatic lag selection algorithm for each iteration.
`meanBandwidth`	An `integer` denoting the bandwidth for the left-sided exponential kernel for the mean.
`varianceBandwidth`	An `integer` denoting the bandwidth for the left-sided exponential kernel for the variance.
`sessionStart`	A `double` to denote the start of the trading session in seconds after midnight.
`sessionEnd`	A `double` to denote the end of the trading session in seconds after midnight.
`parallelize`	A `Boolean` to determine whether to parallelize the underlying C++ code. (Using OpenMP)
`nCores`	An `integer` denoting the number of cores to use for calculating the code when paralellized. If this argument is not provided, sequential evaluation will be used even though `bParallelize` is TRUE
`warnings`	A `logical` denoting whether warnings should be shown.

Details

If the testTimes vector contains instructions to test before the first trade, or more than 15 minutes after the last trade, these entries will be deleted, as not doing so may cause crashes.

The test statistic is unstable before max(meanBandwidth , varianceBandwidth) seconds has passed.

If timestamps is provided and logPrices is an xts object, the indices of logPrices will be used regardless.

Note that using an xts logPrices argument is slower than using a numeric due to the creation of the timestamps from the index of the input. When using xts objects, be careful to use the correct time zones. For example, if I as a dane use the "America/New_York" time zone for my xts objects, I have to add 14400 to my testing times. Same correction will have to be made to the startTime and endTime arguments in the plotting methods.

The lags from the Newey-West algorithm is increased by 2 * (preAveage-1) due to the pre-averaging we know atleast this many lags should be corrected for. The maximum of 20 lags is also increased by this same factor for the same reason.

The methods getMu(DBH) and getSigma(DBH) have ad-hoc adjustments made to the output that are based on simulations. These adjustments give a slight down-ward bias in the value of the drift and sigma if the preAverage argument is VERY high, i.e. 10 or 20 for a noise-less price series of e.g. 23400 trades. If you find any better adjustments, please contact me.

To get the unadjusted series, use DBH$mu or DBH$sigma for the drift, and volatility respectively.

Value

An object of class DBH and list containing the series of the drift burst hypothesis test-statistic as well as the estimated spot drift and variance series. The list also contains some information such as the variance and mean bandwidths along with the pre-averaging setting and the amount of observations. Additionally, the list will contain information on whether testing happened for all testTimes entries.

warning

When using getMu(DBH), beware that the series may be padded with zeros (a warning will be shown when running the driftBursts in this case). This will for example happen when using testTimes for an entire trading day but the data is from a half trading day. Therefore, methods for calculating the mean and variance of the series that take this into account are provided and documented here.

Author(s)

Emil Sjoerup

References

Christensen, Oomen and Reno (2018) <DOI:10.2139/ssrn.2842535>.

See Also

The DBH methods for the DBH class are documented here.

Examples

```
#Simulate a price series containing both a flash crash and a flash-rally
flashCrashSim = function(iT, dSigma, dPhi, dMu){
  vSim = numeric(iT)
  vEps = rnorm(iT , sd =dSigma)
  vEpsy = rnorm(iT)
  vEps[30001:32000] = rnorm(2000 ,sd =seq(from = dSigma ,
                                          to = 2*dSigma , length.out = 2000))
  vEps[32001:34000] = rnorm(2000 ,sd =seq(from = 2*dSigma ,
                                          to = dSigma , length.out = 2000))
  vEpsy[30001:32000] = -rnorm(2000 ,mean =seq(from = 0,
                                              to = 0.3 , length.out = 2000))
  vEpsy[32001:34000] = -rnorm(2000 ,mean =seq(from = 0.3,
                                              to = 0 , length.out = 2000))


  vEps[60001:63000] = rnorm(3000,sd = seq(from = dSigma ,
                                          to = 2*dSigma , length.out = 3000))
  vEps[63001:66000] = rnorm(3000,  sd = seq(from = 2*dSigma ,
                                            to =  dSigma, length.out = 3000))

  vEpsy[60001:63000] = rnorm(3000 ,mean =seq(from = 0,
                                             to = 0.2 , length.out = 3000))
  vEpsy[63001:66000] = rnorm(3000 ,mean =seq(from = 0.2,
                                             to = 0 , length.out = 3000))
  vSim[1] = dMu + dPhi *rnorm(1 , mean = dMu , sd = dSigma /sqrt(1-dPhi^2))
  for (i in 2:iT) {
    vSim[i] = dMu + dPhi * (vSim[(i-1)] - dMu) + vEps[i]
  }
  vY = exp(vSim/2) * vEpsy
  return(vY)
}
#Set parameter values of the simulation
iT = 66500; dSigma = 0.3; dPhi = 0.98; dMu = -10;
#set seed for reproducibility
set.seed(123)
#Simulate the series
y = 500+cumsum(flashCrashSim(iT, dSigma, dPhi, dMu))

#insert an outlier to illustrate robustness.
y[50000] = 500

#Here, the observations are equidistant, but the code can handle unevenly spaced observations.
timestamps = seq(34200 , 57600 , length.out = iT)
testTimes = seq(34260, 57600, 60)
logPrices = log(y)

library("DriftBurstHypothesis")

#calculating drift burst test statistic

DBH = driftBursts(timestamps,  logPrices,
```

```
                        testTimes, preAverage = 5, ACLag = -1L,
                        meanBandwidth = 300L, varianceBandwidth = 900L,
                        parallelize = FALSE)


#plot test statistic
plot(DBH)
#plot both test statistic and price
plot(DBH, price = y, timestamps = timestamps)
#Plot the mu series
plot(DBH, which = "Mu")
#plot the sigma series
plot(DBH, which = "Sigma")
#plot both
plot(DBH, which = c("Mu", "Sigma"))


#Means of the tstat, sigma, and mu series.
mean(getDB(DBH))
mean(getSigma(DBH))
mean(getMu(DBH))



## Not run:
################## same example with xts object:
library("xts")
#Set parameter values of the simulation
iT = 66500; dSigma = 0.3; dPhi = 0.98; dMu = -10;
#set seed for reproducibility
set.seed(123)
#Simulate the series
y = 500+cumsum(flashCrashSim(iT, dSigma, dPhi, dMu))

#insert an outlier to illustrate robustness.
y[50000] = 500

#Here, the observations are equidistant, but the code can handle unevenly spaced observations.
timestamps = seq(34200 , 57600 , length.out = iT)
startTime = strptime("1970-01-01 00:00:00.0000", "%Y-%m-%d %H:%M:%OS", tz = "GMT")
testTimes = seq(34260, 57600, 60)


price = xts(vY, startTime + timestamps)


DBHxts = driftBursts(timestamps = NULL,  log(price),
                   testTimes, preAverage = 5, ACLag = -1L,
                   meanBandwidth = 300L, varianceBandwidth = 900L,
                   parallelize = FALSE)

#check for equality
all.equal(as.numeric(getDB(DBH)), as.numeric(getDB(DBHxts)))
```

```
## End(Not run)
```

`methods for DBH objects` *Methods for the DBH class*

Description

These are the methods for the class DBH which are currently implemented.

Usage

```
## S3 method for class 'DBH'
print(x, ...)
## S3 method for class 'DBH'
plot(x, ...)
## S3 method for class 'DBH'
getDB(x)
## S3 method for class 'DBH'
getCriticalValues(x, alpha)
## S3 method for class 'DBH'
getMu(x, annualize = FALSE, nDays = 252)
## S3 method for class 'DBH'
getSigma(x, annualize = FALSE, nDays = 252)
## S3 method for class 'DBH'
getMean(x, which = 'all')
## S3 method for class 'DBH'
getVar(x, which = 'all', annualize = FALSE, nDays = 252)
```

Arguments

`x`	DBH object
`...`	Additional arguments for the plotting and printing routines. See details
`alpha`	`double` confidence level for the critical values to be extracted. Default = 0.95 designating 95%
`annualize`	`logical` determining whether or not to annualize the series.
`nDays`	`numeric` determining how many days to use for annualization. The standard of 252 reflects the average 252 trading days in a year.
`which`	`character` determining which series to retrieve the mean and variance of in `getMean` and `getVar`. This argument can also be used in the `print` and `plot` methods, see details for use in these methods.

Details

For the `print` method, the `...` argument can be used to pass the following arguments:

- `criticalValue`: `numeric` determining the critical value to use when determining whether a drift burst is present in the data. If this critical value is omitted, a method to determine the critical value based on a interpolations of simulated data-set of critical values for different confidence levels and auto-correlations of the test statistic. See appendix B in the article.
- `alpha`: `numeric` of length 1 which determines the confidence level of critical values extracted using the method above.

For the `plot` method, the `...` argument can be used to pass the following arguments:

- `which`: `character` Used for choosing which series to plot. Valid choices are: `"DriftBursts"`, `"DB"`, `"Sigma"`, `"Mu"`, and `c("Sigma","Mu")`, the order of the latter is irrelevant. The case of the input does not matter. Default = `"driftbursts"`
- `price`:The price series which, if provided, will be overlayed in a red dotted line and the level will be shown at the right y-axis. (Only used if `which` is `"DriftBursts"`). Default = `NULL`
- `time`:Timestamps for the trades in seconds after midnight, which will be used for the x-axis of the plot if the price is overlayed. Default = `NULL`
- `startTime`:Start of the trading day in seconds after midnight. Default = `34200`
- `endTime`:End of the trading day in seconds after midnight. Default = `57600`
- `leg.x`:X-position of the legend in the case `which` is `"DriftBursts"` AND the price is overlayed. Default = `"topleft"`. Usage is as in the base `R` engine.
- `leg.y`:Y-position of the legend in the case `which` is `"DriftBursts"` AND the price is overlayed. Default = `NULL`. Usage is as in the base `R` engine.
- `tz`: `character` denoting the time-zone. Default = `"GMT"`
- `annualize`: `logical` denoting whether to annualize in case which contains `"mu"`, `"sigma"`, or both. Default = `FALSE`
- `nDays`: `numeric` denoting how many to use for annualization if `annualize` is `TRUE`. Default = `252`

Value

For `print` and `plot`, nothing is returned.

For `getCriticalValues`, the critical value, and the normalized critical value is returned in a `list`.

For `getDB`, `getMu`, and `getSigma`, vectors with the same length as testTimes, containing the test statistic, drift, or volatility respecitvely, is returned

For the methods `getMean` and `getVar`:

When the `which` argument is `'all'`, getMean and getVar returns lists containing the mean or variance of the test statistic, drift, and volatility respectively.

When the `which` argument is `'db'` (or `'driftbursts'`), `'mu'`, or `'sigma'`, doubles containing the mean or variance is returned.

Note that the reason for having the getMean and getVar methods is the possibility of testing being mandated while data is not present. This primarily happens when loading in tick data for an entire

year in a loop, including for example Christmas Eve in the U.S. where the markets close early. In this case if the user reuses the same `testTimes`, a warning will be thrown in `driftBursts`, and no testing will be done, but the output series will be padded with zeros to keep the same size as `testTimes`.

Author(s)

Emil Sjoerup

Examples

```
library(DriftBurstHypothesis)
set.seed(1)
# Set mean and variance bandwidth parameters
meanBandwidth = 300L
varianceBandwidth = 900L

# Simulate noise-less price series with 23400 observations, denoting 1 trader per second
# and generate corresponding timestamps.
iT = 23399
r = rnorm(iT, mean = 0, sd = 1)/sqrt(iT)
p = c(0,cumsum(r))
timestamps = seq(34200, 57600, length.out = iT+1)

# Test every minute after the instability period is over.
testTimes  = seq(34260 + varianceBandwidth, 57600, 60L)

# Calculate drift burst test statistic
DBH = driftBursts(timestamps, p, testTimes, preAverage = 1, ACLag = -1,
                  meanBandwidth = meanBandwidth, varianceBandwidth = varianceBandwidth)



print(DBH)
# Plots the test statistic with prices overlaid.
plot(DBH, timestamps = timestamps, price = p)

# Plots the annualized volatility
plot(DBH, which = 'sigma', annualize = TRUE)

# Plots the annualized drift and volatility
plot(DBH, which = c('sigma', 'mu'), annualize = TRUE)

# Retrieve the critical values of the drift burst test statistic
getCriticalValues(DBH)

# Calculate the mean of the test statistic, drift, and volatility
getMean(DBH, which = 'all')

# Calculate the variance of the test statistic
getVar(DBH, which = 'db')
```

```
# Extracts the annualized drift
annualizedDrift = getMu(DBH, annualize = TRUE)

# Extracts the annualized volatility
annualizedVolatility = getSigma(DBH, annualize = TRUE)
```

Index