

Package ‘ttrTests’

January 2, 2012

Type Package

Title Standard Backtests for Technical Trading Rules in Financial Data

Version 1.7

Date 2011-08-15

Author David St John

Maintainer David St John <dstjohn@math.uic.edu>

Depends fTrading, TTR

Description Five core functions evaluate the efficacy of a technical trading rule. - Conditional return statistics - Bootstrap resampling statistics - Reality Check for data snooping bias among parameter choices - Robustness, or Persistence, of parameter choices - Parameter Domain Correlation Test

License GPL (>= 3)

LazyLoad yes

Repository CRAN

Date/Publication 2011-08-16 07:53:20

R topics documented:

ttrTests-package	2
bootstrap	6
cReturns	7
dataSnoop	9
defaults	11
deleteNA	12
generateSample	13
indicator	14
macd4	16

nullModel	17
paramPersist	18
paramStats	20
position	23
returnStats	24
spData	25
subperiods	26

Index	29
--------------	-----------

ttrTests-package	<i>A series of tests for the efficacy of a technical trading rule (TTR)</i>
------------------	---

Description

Contains five major tests supported by other functions: Did the TTR strategy outperform a benchmark in the past data? Is the excess return significant, using bootstrapping to construct a confidence interval? Is the excess return explained by data snooping? Is the 'good' choice of parameters robust across sub-samples? Is this robustness significant, using bootstrapping to construct a confidence interval?

Details

Package:	ttrTests
Type:	Package
Version:	1.7
Date:	2011-08-15
License:	GPL
LazyLoad:	yes

First, data should be a univariate time series 'x'. Attempts have been made to accomodate 'High-Low-Close' data by coercing it into a univariate 'close' series, but stability of this behavior is not gauranteed.

All computations basically take place on the return series, $\text{diff}(\log(x))$ So it is assumed that the data x is such that this return series is stationary, a common assumption for financial data.

Second, a TTR should be any function whose input is the data 'x' and a list of parameters, and whose output is a ternary trading rule, i.e. 1 for 'long', -1 for 'short', and 0 for neither. Built in TTR functions have this behavior, and any user defined function with this behavior is accepted.

USER PROVIDED TTR FUNCTIONS MUST HAVE THE FOLLOWING FORM:

```
ttr <- function ( x , params , burn , short)
```

Where x is the data, params is a numeric vector of parameters, burn is an integer describing the 'burn' period, and short is logical. The function need not use these, but they must accept them.

PLEASE SEE THE EXAMPLES IF YOU ARE GETTING ERRORS!

Now, the major functions of the package:

1) `returnStats(x,ttr)` compares the performance of the TTR with some benchmark. A ttr that had good past performance might merit further study.

2) `nullModel(x,ttr)` constructs a confidence interval for this performance and gives a p-value for the excess return observed in (1). If the TTR fails this test (i.e. p-value too high), it means that the good performance was just as likely due to luck in how the data was constructed.

3) `dataSnoop(x,ttr)` constructs a p-value for the 'best' choice of parameters within a given domain (preferably large). If a TTR fails this test, it means that good performance was just as likely due to luck in picking a choice of parameters.

4) `subperiods(x,ttr)` asks whether or not good choices of parameters were robust across different time periods. This is a new type of 'out-of- sample' confirmation introduced by the author in 2010.

5) `paramPersist(x,ttr)` tests if the persistence measure from `subperiods()` is statistically significant. It constructs an asymptotically valid p-value for the observed persistence, assuming that the true correlation is zero. If this test fails (i.e. p-value too high), it means that regardless of how significant the performance of a TTR was, no choice of parameters performed consistently well over time.

Version \geq 1.1 allows the results of these tests to be written to a file in LaTeX code as a figure. It also includes two tests for data snooping, White's Reality Check and Hansen's test for Superior Predictive Ability, the later not included in the first version.

Version \geq 1.2 fixes some minor glitches in 1.1.

Version \geq 1.3 uses a different algorithm to compute the statistical significance of the observed correlation from test (4). Some new functions have been added to accomplish this. An option for the position/indicator functions was added to increase flexibility. Other minor changes here and there.

Version \geq 1.4 fixes some minor glitches in 1.3

Version \geq 1.5 fixes some major glitches in 1.4, now producing correct results for some tests that were behaving questionably, and with much more accurate p-values based on bootstrapping.

Version \geq 1.6 allows the user to pass the 'restrict' option to the `dataSnoop` test. Absence of this feature made it impossible for the user to use the 'restrict' option for a user defined ttr. Also a small glitch in cating the results of `dataSnoop` was fixed, and a relatively serious error in `paramStats` was corrected to allow it to run with rules using fewer than 4 parameters.

Version \geq 1.7 has new examples, a new data set, and minor changes to the documentation.

Note

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made directly or implied by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Please note that Dr. Halbert White owns U.S. patents 6,088,676 and 5,893,069, pertaining to the RC option for the function `dataSnoop()`. These routines are licensed to the author for use solely for the purposes of non-commercial academic research and study. License rights as granted in the GPL are therefore limited to uses that are legal under national and international patent law.

Future contributors, as defined in the GPL, assume full responsibility for their modifications and implementations.

If there is any doubt, please remove code that implements the reality check and enjoy the rest of the package. There should be very minimal loss of functionality, if any, since a second data snooping test is included.

Author(s)

David St John

Maintainer: David St John <dstjohn@math.uic.edu>

References

- [1] Sidney S. Alexander. Price movements in speculative markets: Trends or random walks. *Industrial Management Review*, 2(2):7-26, 1961.
- [2] David R. Aronson. *Evidence Based Technical Analysis*. John Wiley and Sons, 2007.
- [3] William Brock, Josef Lakonishok, and Blake LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, 47(5):1731-1764, 1992.
- [4] David P. Brown and Robert H. Jennings. On technical analysis. *The Review of Financial Studies*, 2(4):527-551, 1989.
- [5] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1-26, 1979.
- [6] Eugene F. Fama and Marshall E. Blume. Filter rules and stock market trading. *The Journal of Business*, 39(1):226-241, 1966.
- [7] John Hull. *Futures, Options, and Other Derivatives*. Prentice Hall, 2006.
- [8] Kenneth A. Kavajecz and Elizabeth R. Odders-White. Technical analysis and liquidity provision. *The Review of Financial Studies*, 17(4):1043- 1071, 2004.
- [9] Richard M. Levich and Lee R. Thomas. The significance of technical trading rule profits in the foreign exchange markets: A bootstrap approach. 1991.
- [10] Christopher Neely, Paul Weller, and Rob Dittmar. Is technical analysis in the foreign exchange market profitable? a genetic programming approach. *The Journal of Financial and Quantitative Analysis*, 32(4):405- 426, 1997.
- [11] Gary Norden. *Technical Analysis and the Active Trader*. McGraw - Hill, 2006.
- [12] Min Qi and Yangru Wu. Technical trading rule profitability, data snooping, and reality check: Evidence from the foreign exchange market. *Journal of Money, Credit, and Banking*, 38(8):2135-2158, 2006.
- [13] Mark J. Ready. Profits from technical trading rules. *Financial Management*, 31(3):43-61, 2002.
- [14] Halbert White. A Reality Check for Data Snooping. *Econometrica*, 68, 2000, pp1097-1126.
- [15] Ryan Sullivan, Allan Timmermann, and Halbert White. Data snooping, technical trading rule performance, and the bootstrap. *The Journal of Finance*, 54(5):1647-1691, 1999.
- [16] Jack L. Treynor and Robert Ferguson. In defense of technical analysis. *The Journal of Finance*, 40(3):757-773, 1985.
- [17] David St John. *Technical Analysis based on Moving Average Convergence and Divergence*. Phd thesis, University of Illinois, Chicago. 2010. <http://math.uic.edu/~dstjohn/thesis.pdf>

Examples

```

data(spData)

## First and foremost your ttr is a function.
## Call your ttr anything you like and use any defaults you like
## (they're ultimately irrelevant) but you must use
## 'x', 'params', 'burn', and 'short' exactly!

myTTR <- function(x, params=c(3,7,5), burn=0, short=TRUE)
{

crystalBall <- rexp(length(x),rate=1/params[3])

## note that there are no 'fixed' values used
## the output must depend explicitly on the input

position <- ifelse(crystalBall<=params[1],-1,0) +
ifelse(crystalBall>=params[2],1,0)

## the output will be a vector with values {1,0,-1}
## interpreted as long, neutral, short, respectively
## it should have the same length as the data

return(position)
}

## I know it's quirky, but when using your ttr, you have to include
## values for params (or start, nSteps, stepSize where needed),
## burn, and short for all of the following calls. No defaults
## will be used, and you will get an error if you exclude them

stat <- returnStats(spData, ttr=myTTR, params=c(3,7,5),burn=0,short=TRUE)

## Witness the amazing predictive power of my crystalBall!
## Go ahead and run returnStats over and over!
## We see a positive excess return every time!
## Man I am such a good fund manager

null <- nullModel(spData,ttr=myTTR,params=c(3,7,5),burn=0,short=TRUE,bSamples=5)
spa <- dataSnoop(spData,ttr=myTTR,start=c(3,7,5),nSteps=c(3,3,3),
stepSize=c(1,1,1),burn=0,short=TRUE,restrict=TRUE,bSamples=3)
pp <- paramPersist(spData,ttr=myTTR,start=c(3,7,5),nSteps=c(3,3,3),
stepSize=c(1,1,1),burn=0,short=TRUE,restrict=TRUE,bSamples=3)

## Sorry, for those who missed the joke, myTTR decides to be long, short, or
## neutral completely at random according to a distribution specified by
## the input parameters. Since the benchmark returns were negative, and
## the 'strategy' is short or neutral some of the time, it outperforms
## as expected, but with no real predictive power

#####

```

```
## Here's an example of another user defined ttr that will work:

twoSMA<-function(x,params=c(20,30),burn = 0, short = FALSE)
{
  mac2<-SMA(x,params[2])-SMA(x,params[1])
  mac2[is.na(mac2)]<-0
  sig<-ifelse(mac2>0,1,0)
  return(sig)
}

## This one is based on some actual technical analysis...
## not that that guarantees it will do much better than my crystalBall
```

bootstrap

Generates a Bootstrap Sample from Raw Data

Description

Given a data set, this function returns a randomly generated data set of the same size using the bootstrap procedure on the raw data. For now, standard (i.i.d.) bootstrapping and Stationary (block) bootstrapping are supported. If random data is desired from other model distributions, for example ARIMA or GARCH, a user defined function can be input. See function generateSample() for an implementation where the original data is transformed to returns data, resampled, and then exponentiated and aggregated back to a bootstrapped price series.

Usage

```
bootstrap(x, model = "bootstrap", userParams = 4)
```

Arguments

x	The data set (a univariate series)
model	Currently built in choices are "bootstrap" and "stationaryBootstrap". Also accepts a user defined function whose output is a series of the same length as the input data.
userParams	Will be passed to the function 'model', in the case that 'model' is a user defined function. Hence, a user defined function should take two parameters, the data and a list of other needed inputs. If "stationaryBootstrap" is used, userParams is the average block length from a geometric distribution, i.e. (1/lambda).

Details

By design the bootstrapping procedure produces samples with the same statistical properties as the original data. If a user defined function is used that generates samples with mis-matching statistical properties, these samples will not likely be useful.

Value

sample - a univariate series the same length as the input series

Note

A USER DEFINED MODEL MUST HAVE THE FOLLOWING FORM: function (x , userParams) Where x is the data, userParams is a numeric vector.

Author(s)

David St John

References

B. Efron. Bootstrap methods: Another look at the jackknife. The Annals of Statistics, 7(1):1-26, 1979.

Politis, Dimitris, and Joseph Romano, 1994, The stationary bootstrap, Journal of the American Statistical Association 89, 1303-1313.

Examples

```
foo <- runif(100)
mean(foo)
var(foo)
plot(foo)

sample <- bootstrap(foo)
mean(sample)
var(sample)
plot(sample)
```

cReturns

Return series conditioned on a TTR

Description

For a series $x(t)$ and a given TTR, computes the return series $r(t) = \ln(x(t+1) / x(t))$ and the conditional return series, $r(t)*s(t)$, where $s(t)$ is the position indicated by the TTR i.e. $s(t) = 1$ for long, -1 for short, 0 for neutral

Usage

```
cReturns(x, ttr = "macd4", params = 0, burn = 0, short = FALSE,
condition = NULL, TC = 0.001)
```

Arguments

x	The data set
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series $s(t)$. See 'defaults' for a list of built-in TTRs.
params	Used to compute the TTR. Will be passed to a user defined function. Hence a user defined function should have at least 2 inputs, the data set and a vector or list of parameters
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
TC	Trading cost, as a percentage. Used to compute an adjusted average return.

Value

cReturns	The conditional returns series
aReturns	The mean one-period return, adjusted for trading costs
lReturns	The conditional returns only during periods when long
sReturns	The conditional returns only during periods when short
nReturns	The conditional returns only during periods when neutral

Note

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

William Brock, Josef Lakonishok, and Blake LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, 47(5):1731-1764, 1992.

Examples

```
## Is the mean conditional return higher than the mean unconditional return?

data(spData)
mean(diff(log(spData)))

cr <- cReturns(spData)
mean(cr[[1]])
```

dataSnoop

Two Tests for Data Snooping: RC and SPA

Description

Tests for data snooping bias by doing bootstrap resampling, then finding the best parameterization in the bootstrapped samples and performing one of two popular tests - White's Reality Check or Hansen's test for Superior Predictive Ability. Can write a summary of results to a file as a LaTeX figure.

Please note that Dr. Halbert White owns U.S. patents 6,088,676 and 5,893,069, pertaining to the RC option for the function dataSnoop(). These routines are licensed to the author for use solely for the purposes of non-commercial academic research and study. License rights as granted in the GPL are therefore limited to uses that are legal under national and international patent law. Future contributors, as defined in the GPL, assume full responsibility for their modifications and implementations.

If there is any doubt, please remove code that implements the reality check and enjoy the rest of the package. There should be very minimal loss of functionality, if any, since a second data snooping test is included.

Usage

```
dataSnoop(x, ttr = "macd4", start = 0, nSteps = 0, stepSize = 0,
restrict=FALSE, burn = 0, short = FALSE, condition = NULL, silent = TRUE,
TC = 0.001, loud = TRUE, alpha = 0.025, crit = "sharpe" , begin = 1,
percent = 1, file = "", benchmark = "hold", bSamples = 100,
model = "stationaryBootstrap", userParams=4, test="SPA", latex="")
```

Arguments

x	A univariate series
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series s(t). See 'defaults' for a list of built-in TTRs.
start	Initial values for parameters
nSteps	How many parameter choices to use for each parameter

stepSize	The difference between successive choices of a parameter.
restrict	If restricted = TRUE, this will force the second parameter (and 4th, if applicable) to be strictly greater than the first (3rd, resp.) This is sensible if the pairs are moving average parameters.
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
silent	Logical. If TRUE, suppresses output from subroutines
TC	Percentage used to compute returns adjusted for trading costs.
loud	Logical. If FALSE, suppresses output from the main function(s)
alpha	Confidence level for 1-sided hypothesis testing
crit	The criterion used to evaluate the performance. Supported values are "sharpe" for the sharpe ratio (risk free rate assumed zero) which is consistent with Hansen's SPA, "return" which is the excess return, and "adjust" which is excess return adjusted for trading costs.
begin	The starting index of the data. The function assumes that the user wants a subset of the data, where the default subset is the entire data
percent	How much of the original data to use (default 100)
file	The full writable path string for a file to which output will be appended. Ideal for reviewing results.
benchmark	When computing 'excess' returns, all functions in this package subtract the conditional returns based on a given "trr" from the "benchmark" returns. Two different TTRs can be compared this way if desired.
bSamples	Number of bootstrapped samples to analyze
model	Currently built in choices are "bootstrap" and "stationaryBootstrap". Also accepts a user defined function whose output is a series of the same length as the input data.
userParams	Will be passed to the function 'model', in the case that 'model' is a user defined function. Hence, a user defined function should take two parameters, the data and a list of other needed inputs. If "stationaryBootstrap" is used, userParams is the average block length from a geometric distribution, i.e. $(1/\lambda)$.
test	Supports "RC" which is White's Reality Check, or "SPA", which is Hansen's test for Superior Predictive Ability
latex	Full path name for a writable file. The LaTeX code that generates a figure with a summary of the output will be appended to file.

Value

Returns the observed value for \hat{V} -N or T-SPA, the bootstrapped values of \hat{V} -N,b or T*-SPA, and P-value or values from the given test.

Note

See papers for details on the 'V' values an 'T' values

If p-value is significant, then the null hypothesis that good performance is due to data snooping is rejected. However, this does not preclude any other null hypothesis that might explain good results.

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

Ryan Sullivan, Allan Timmermann, and Halbert White. Data snooping, technical trading rule performance, and the bootstrap. *The Journal of Finance*, 54(5):1647-1691, 1999.

Peter R. Hansen. A Test for Superior Predictive Ability. *Journal of Business and Economic Statistics*, 2005.

Examples

```
data(spData)
rc <- dataSnoop(spData,bSamples=3,test="RC")
spa <- dataSnoop(spData,bSamples=3,test="SPA")
```

defaults

Default parameters for a given TTR

Description

Almost exclusively used by other functions in the package.

Usage

```
defaults(ttr)
```

Arguments

ttr A character string for built-in TTRs. Will accept a user-defined function for ttr, but in this case 0 is returned

Details

The following TTR choices from package 'TTR' are supported: - "aroon" - "cci" - "cmo" - "kst" - "macd" - "tdi" - "trix"

Also, the TTR "macd4" is a modification of "macd" by the author

TTR "none" returns a position of 0 always, i.e. no position

TTR "hold" returns a position of 1 always, i.e. always long

Value

Outputs several lists of parameters. Could be of different length depending on TTR. Used almost exclusively within other functions.

If the final output value is TRUE, this will force the second parameter (and 4th, if applicable) to be strictly greater than the first (3rd, resp.) This is sensible if the pairs are moving average parameters.

Author(s)

David St John

References

<http://cran.r-project.org/web/packages/TTR/>

Examples

```
## "macd4" has an extra parameter which is fixed by default
## to 1 in "macd", but can be varied using "macd4"

defaults("macd")
defaults("macd4")
```

deleteNA

Cleans data that contains NA entries

Description

Used in this package to create data for long, short, or neutral days only. Could have many other applications. Probably exists in other packages with more support.

Usage

```
deleteNA(x)
```

Arguments

x A univariate series or vector

Details

As long as x can be indexed, it should work fine.

Value

Outputs is a vector of length \leq length(x) with NA entries deleted

Author(s)

David St John

Examples

```
x <- c(1,NA,2,3,NA,4)
deleteNA(x)

t <- c(1,2,3)
x <- c(t[1],t[2],t[3],t[4],t[5])
deleteNA(x)
```

generateSample

Generates a Bootstrap Sample from Price Data

Description

Given a data set, this function returns a randomly generated data set of the same size using a modified bootstrap procedure. For now, standard (i.i.d.) bootstrapping and Stationary (block) bootstrapping are supported. If random data is desired from other model distributions, for example ARIMA or GARCH, a user defined function can be input.

Because it is assumed that we are using price data, this function will compute the returns series, bootstrap that series, and exponentiate and aggregate back to a price series.

Usage

```
generateSample(x, model = "stationaryBootstrap", userParams = 4)
```

Arguments

x	The data set (a univariate series)
model	Currently built in choices are "bootstrap" and "stationaryBootstrap". Also accepts a user defined function whose output is a series of the same length as the input data.
userParams	Will be passed to the function 'model', in the case that 'model' is a user defined function. Hence, a user defined function should take two parameters, the data and a list of other needed inputs. If "stationaryBootstrap" is used, userParams is the average block length from a geometric distribution, i.e. $(1/\lambda)$.

Details

By design the bootstrapping procedure produces samples with the same statistical properties as the original data. If a user defined function is used that generates samples with mis-matching statistical properties, these samples will not likely be useful.

Value

sample - a univariate series the same length as the input series

Note

A USER DEFINED MODEL MUST HAVE THE FOLLOWING FORM: function (x , userParams) Where x is the data, userParams is a numeric vector.

Author(s)

David St John

References

B. Efron. Bootstrap methods: Another look at the jackknife. The Annals of Statistics, 7(1):1-26, 1979.

Politis, Dimitris, and Joseph Romano, 1994, The stationary bootstrap, Journal of the American Statistical Association 89, 1303-1313.

Examples

```
data(spData)
mean(diff(log(spData)))
var(diff(log(spData)))
plot(spData)

sample <- generateSample(spData)
mean(diff(log(sample)))
var(diff(log(sample)))
plot(sample)
```

indicator

The canonical position / indicator functions

Description

Computes the "position" function corresponding to a given univariate series and TTR. Position is 1 for long, -1 for short, and 0 for neutral. The indicator function is just the difference of the position from one period to the next, i.e. 1 for buy one unit, -2 for sell 2 units, etc.

Usage

```
indicator(x, ttr = "macd4", params = 0, burn = 0, short = FALSE,
condition = NULL)
```

Arguments

x	The data set
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series $s(t)$. See 'defaults' for a list of built-in TTRs.
params	Used to compute the TTR. Will be passed to a user defined function. Hence a user defined function should have at least 2 inputs, the data set and a vector or list of parameters
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.

Value

Output is a list containing the position and indicator (resp.)

Author(s)

David St John

References

William Brock, Josef Lakonishok, and Blake LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, 47(5):1731-1764, 1992.

Examples

```
## How many days does the TTR 'macd4' indicate a long position?
## How many trades are indicated?

data(spData)
position <- indicator(spData,short=FALSE)

length(spData)
sum(position[[1]])
sum(abs(position[[2]]))
```

macd4	<i>MACD oscillator</i>
-------	------------------------

Description

A new way of computing the MACD oscillator where a fourth parameter is allowed to vary. Typically this parameter is fixed at 1 arbitrarily.

Usage

```
macd4(x, params = c(12, 26, 1, 9))
```

Arguments

x	A univariate series
params	First parameter - the 'fast' average parameter Second parameter - the 'slow' average parameter Third parameter - the new parameter, indicating a 'fast' averaging of the MACD line instead of the typical choice of the MACD line itself Fourth parameter - the 'slow' averaging for the MACD signal line

Details

The standard interpretation of the MACD is a 'crossover' rule, i.e. when the MACD oscillator crosses from positive to negative, that's a sell signal, and vice versa.

Value

A univariate series computed as the difference between the MACD line and the MACD signal line. "macd4" replaces the MACD line with a fast average of the MACD line.

Author(s)

David St John

References

<http://cran.r-project.org/web/packages/TTR/> <http://en.wikipedia.org/wiki/MACD>

Examples

```
data(spData)
oscillator <- macd4(spData)
plot(oscillator)
```

nullModel

Hypothesis test for efficacy of TTR

Description

One of the four main functions in the package. Creates a confidence interval for the observed excess return via bootstrap resampling. Can write summary of output to a file as a latex figure.

Usage

```
nullModel(x, model = "stationaryBootstrap", userParams = 4,
bSamples = 100, ttr = "macd4", params = 0, burn = 0, short = FALSE,
condition = NULL, silent = TRUE, loud = TRUE, alpha = 0.025,
crit = "return", TC = 0.001, benchmark = "hold", latex = "")
```

Arguments

x	A univariate series
model	Passed to the function 'generateSample'
userParams	Passed to the function 'generateSample'
bSamples	How many bootstrapped samples to generate
ttr	Could be a character string for a built in TTR, or a user defined function. User defined functions must take a univariate series and a list/vector of inputs and must output a series with values 1,0,-1 only
params	Used to calculate the position based on the given TTR
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
silent	Logical. If TRUE, output from subroutines will be suppressed.
loud	Logical. If FALSE, output from the main function will be suppressed.
alpha	Confidence interval for 1-sided hypothesis test
crit	The criterion used to evaluate the performance. Supported values are "sharpe" for the sharpe ratio (risk free rate assumed zero) which is consistent with Hansen's SPA, "return" which is the excess return, and "adjust" which is excess return adjusted for trading costs.
TC	Percentage trading costs. Used to adjust return statistics.
benchmark	When computing 'excess' returns, all functions in this package subtract the conditional returns based on a given "ttr" from the "benchmark" returns. Two different TTRs can be compared this way if desired.
latex	Full path name for a writable file. The LaTeX code that generates a figure with a summary of the output will be appended to file.

Value

CR	A vector of conditional returns of length 'bSamples'
AR	CR, adjusted for trading costs
SR	Sharp ratio for these returns using $r_f = 0$
Z	Z-score for observed excess return, using mean and standard deviation of CR for a confidence interval
P	P-value associated with observed Z-score

Note

A significant P-value is enough to reject the null hypothesis that the TTR had results due solely to randomness in the data. However, there are several other null hypotheses to explain good results, chiefly the data snooping hypothesis, addressed using the function 'realityCheck'.

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

William Brock, Josef Lakonishok, and Blake LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, 47(5):1731-1764, 1992.

Examples

```
data(spData)
null <- nullModel(spData,bSamples=5)
```

 paramPersist

Robustness, or Persistence, of Parameter Choices for TTR

Description

Divides the given data set into 2 subperiods and computes the correlation coefficient using a call to the 'subperiods' function. A p-value for the null hypothesis that there is no correlation in the underlying distribution is given using the bootstrapping procedure. This is the Parameter Domain Correlation (PDC) test, as defined in Chapter 3 of the author's PhD thesis (see below reference).

Usage

```
paramPersist(x, ttr = "macd4", start = 0, nSteps = 0, stepSize = 0,
restrict = FALSE, bSamples = 25, model = "stationaryBootstrap",
userParams = 4, burn = 0, short = FALSE, condition = NULL,
silent = TRUE, TC = 0.001, loud = TRUE, plot = TRUE,
alpha = 0.05, periods = 2, file = "", latex = "")
```

Arguments

x	A univariate series
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series s(t). See 'defaults' for a list of built-in TTRs.
start	Initial values for parameters
nSteps	How many parameter choices to use for each parameter
stepSize	The difference between successive choices of a parameter.
restrict	If restricted = TRUE, this will force the second parameter (and 4th, if applicable) to be strictly greater than the first (3rd, resp.) This is sensible if the pairs are moving average parameters.
bSamples	How many bootstrapped samples to generate
model	Passed to the function 'generateSample'
userParams	Passed to the function 'generateSample'
burn	When computing the position function s(t), values for t < burn will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function s(t) will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
silent	Logical. If TRUE, suppresses output from subroutines
TC	Percentage used to compute returns adjusted for trading costs.
loud	Logical. If FALSE, suppresses output from the main function(s)
plot	Logical. If FALSE, suppresses plot of regression data
alpha	Confidence level for 2-sided hypothesis testing
periods	How many periods to split up the original data. If default, the number of periods is decided based on the length of the data.
file	The full writable path string for a file to which output will be appended. Ideal for reviewing results.
latex	Full path name for a writable file. The LaTeX code that generates a figure with a summary of the output will be appended to file.

Details

See the help file for 'paramStats' for important information about using "start,nSteps,stepSize" to define a domain of parameters

Value

Output is a list containing the observed correlation coefficient, the observed correlation coefficients in bootstrapped samples, and the p-value for hypothesis testing.

Note

This procedure is very computationally intensive and requires a lot of resources for large data sets and large numbers of bootstrapped samples.

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

David St John. Technical Analysis based on Moving Average Convergence and Divergence. Phd thesis, University of Illinois, Chicago. 2010. <http://math.uic.edu/~dstjohn/thesis.pdf>

Examples

```
data(spData)
pp <- paramPersist(spData,bSamples=3)
```

paramStats

Analyzes a domain of parameter choices to pick the best one

Description

This function calls the 'returnStats' function with every choice of parameters in a given domain and records the result. This allows 'good' choices for parameters to be identified.

Usage

```
paramStats(x, ttr = "macd4", start = 0, nSteps = 0, stepSize = 0,
restrict = FALSE, burn = 0, short = FALSE, condition = NULL,
silent = TRUE, TC = 0.001, loud = TRUE, plot = TRUE, alpha = 0.025,
begin = 1, percent = 1, file = "", benchmark = "hold")
```

Arguments

x	A univariate series
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series $s(t)$. See 'defaults' for a list of built-in TTRs.
start	Initial values for parameters
nSteps	How many parameter choices to use for each parameter
stepSize	The difference between successive choices of a parameter.
restrict	If restricted = TRUE, this will force the second parameter (and 4th, if applicable) to be strictly greater than the first (3rd, resp.) This is sensible if the pairs are moving average parameters.
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
silent	Logical. If TRUE, suppresses output from subroutines
TC	Percentage used to compute returns adjusted for trading costs.
loud	Logical. If FALSE, suppresses output from the main function(s).
plot	Logical. If FALSE, suppresses plot of results by parameter choice.
alpha	Confidence level for 2-sided hypothesis testing
begin	The starting index of the data. The function assumes that the user wants a subset of the data, where the default subset is the entire data
percent	How much of the original data to use (default 100)
file	The full writable path string for a file to which output will be appended. Ideal for reviewing results.
benchmark	When computing 'excess' returns, all functions in this package subtract the conditional returns based on a given "ttr" from the "benchmark" returns. Two different TTRs can be compared this way if desired.

Details

This function will only allow a list of parameters with length at most 4. If a TTR requires more than 4 parameters, it is not supported here (yet).

If a TTR uses 3 or 4 parameters, it may be 'restricted'. In this case it is assumed that the first 2 parameters are related, and forces the second parameter to be strictly greater than the first. The 3rd and 4th parameters are treated similarly. Built in TTRs 'macd' and 'macd4' are restricted. Users may wish to 'restrict' user defined TTR, if appropriate.

Example: (4 parameters) start = c(2,4,3,6) nSteps = c(3,5,1,2) stepSize = c(4,5,2,3)

The values of the first parameter would be (2,6,10) The values of the second parameter would be (4,9,14,19,24) PLUS THE FIRST!!! The values of the third parameter would be (3) The values of the fourth parameter would be (6,9) PLUS THE THIRD!!!

So there would be 30 parameterizations in this domain. They would be: (2,6,3,6) , (2,11,3,6) , ... (notice the second parameter is NOT 4,9,...) (6,10,3,6) , (6,15,3,6) , ... (it is forced to be strictly greater) (10,14,3,6) , (10,19,3,6) , ... (by adding the first parameter)

If restrict = FALSE, no such adjustment will be made to the choice of domain for parameters.

Value

Output is a list of vectors. When read as a matrix, each row contains: The conditional mean return The z-score of this amongst all parameter choices The adjusted return (after trading costs) The Sharpe ratio (The best choice of parameters, as a vector. Not part of the overall matrix) The parameters

For a unique choice of parameters

Note

Of all parameter choices, the 'best' is used for the function 'dataSnoop'

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

Ryan Sullivan, Allan Timmermann, and Halbert White. Data snooping, technical trading rule performance, and the bootstrap. The Journal of Finance, 54(5):1647-1691, 1999.

Examples

```

data(spData)
ps <- paramStats(spData)

max(ps[[1]])
max(ps[[2]])

```

position	<i>Position Function</i>
----------	--------------------------

Description

Obsoleted by 'indicator' function, but called from there so must be included in the package.

Usage

```

position(x, ttr = "macd4", params = 0, burn = 0, short = FALSE,
condition = NULL)

```

Arguments

x	A univariate series
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series $s(t)$. See 'defaults' for a list of built-in TTRs.
params	Used to compute the TTR. Will be passed to a user defined function. Hence a user defined function should have at least 2 inputs, the data set and a vector or list of parameters
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See notes for more details.

Value

Value returned to function 'indicator' is returned as 'pos'

Note

The user-supplied 'Condition' series should be zero during any time period when there should not be any position, and one otherwise. If none is given, a string of all ones is default. Such a restriction could be no positions held during certain times or any other external condition.

Author(s)

David St John

returnStats

*Conditional statistics for given data and TTR***Description**

The first of four main backtests for a technical analysis strategy. Gives returns conditioned on the TTR above and beyond some benchmark.

Usage

```
returnStats(x, ttr = "macd4", params = 0, burn = 0, short = FALSE,
condition = NULL, silent = FALSE, TC = 0.001, benchmark = "hold", latex="")
```

Arguments

x	A univariate series
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series s(t). See 'defaults' for a list of built-in TTRs.
params	A list of parameters to use for the TTR
burn	When computing the position function s(t), values for t < burn will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function s(t) will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
silent	Logical. If TRUE, supresses output from subroutines
TC	Percentage used to compute returns adjusted for trading costs.
benchmark	When computing 'excess' returns, all functions in this package subtract the conditional returns based on a given "ttr" from the "benchmark" returns. Two different TTRs can be compared this way if desired.
latex	Full path name for a writable file. The LaTeX code that generates a figure with a summary of the output will be appended to file.

Details

"excess return" means the conditional return minus the benchmark return

Value

uResults and cResults are each of length 10 and include: mean, variance, sharp ratio, skew, kurtosis, and first 5 autocorrelation coefficients

The third item of output is the excess return adjusted for trading costs.

Note

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

William Brock, Josef Lakonishok, and Blake LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, 47(5):1731-1764, 1992.

Examples

```
data(spData)
stat <- returnStats(spData)
```

spData	<i>Daily Close of SP Index</i>
--------	--------------------------------

Description

This data set gives three years worth of daily data from 1/1/2006 through 12/31/2008.

Usage

```
spData
```

Format

A vector with 755 observations

Source

www.yahoo.com accessed 8/15/2011

subperiods *Computes the Correlation between Conditional Returns in Two Non-Overlapping Sub-Periods*

Description

This function computes the conditional returns for each parameterization of a given TTR in a given domain for each of subperiod in a given data set. The correlation is then computed between the results in each pair of sub-periods. This is a measure of the persistence, or robustness, of the parameter choices for a given TTR.

Usage

```
subperiods(x, ttr = "macd4", start = 0, nSteps = 0, stepSize = 0,
restrict = FALSE, burn = 0, short = FALSE, condition = NULL,
silent = TRUE, TC = 0.001, loud = TRUE, plot = TRUE, alpha = 0.05,
periods = 0, file = "", latex = "", benchmark = "hold")
```

Arguments

x	A univariate series
ttr	The TTR to be used. Can be a character string for built-in TTRs, or a user defined function whose output is a position series $s(t)$. See 'defaults' for a list of built-in TTRs.
start	Initial values for parameters
nSteps	How many parameter choices to use for each parameter
stepSize	The difference between successive choices of a parameter.
restrict	If restricted = TRUE, this will force the second parameter (and 4th, if applicable) to be strictly greater than the first (3rd, resp.) This is sensible if the pairs are moving average parameters.
burn	When computing the position function $s(t)$, values for $t < \text{burn}$ will be forced to 0, i.e. no position held during the 'burn' period
short	Logical. If false the position function $s(t)$ will be forced to 0 when it would otherwise be -1, i.e. no short selling
condition	An extra opportunity to restrict the TTR so that position is forced to 0 under some condition. Must be a binary string of the same length as the data 'x'. See 'position' for more details.
silent	Logical. If TRUE, suppresses output from subroutines
TC	Percentage used to compute returns adjusted for trading costs.
loud	Logical. If FALSE, suppresses output from the main function(s).
plot	Logical. If FALSE, suppresses plot of results by parameter choice.
alpha	Confidence level for 2-sided hypothesis testing
periods	How many subperiods into which the function will divide the data.

file	The full writable path string for a file to which output will be appended. Ideal for reviewing results.
latex	Full path name for a writable file. The LaTeX code that generates a figure with a summary of the output will be appended to file.
benchmark	When computing 'excess' returns, all functions in this package subtract the conditional returns based on a given "ttr" from the "benchmark" returns. Two different TTRs can be compared this way if desired.

Details

This function will only allow a list of parameters with length at most 4. If a TTR requires more than 4 parameters, it is not supported here (yet).

If a TTR uses 3 or 4 parameters, it may be 'restricted'. In this case it is assumed that the first 2 parameters are related, and forces the second parameter to be strictly greater than the first. The 3rd and 4th parameters are treated similarly. Built in TTRs 'macd' and 'macd4' are restricted. Users may wish to 'restrict' user defined TTR, if appropriate.

Example: (4 parameters) start = c(2,4,3,6) nSteps = c(3,5,1,2) stepSize = c(4,5,2,3)

The values of the first parameter would be (2,6,10) The values of the second parameter would be (4,9,14,19,24) PLUS THE FIRST!!! The values of the third parameter would be (3) The values of the fourth parameter would be (6,9) PLUS THE THIRD!!!

So there would be 30 parameterizations in this domain. They would be: (2,6,3,6) , (2,11,3,6) , ... (notice the second parameter is NOT 4,9,...) (6,10,3,6) , (6,15,3,6) , ... (it is forced to be strictly greater) (10,14,3,6) , (10,19,3,6) , ... (by adding the first parameter)

If restrict = FALSE, no such adjustment will be made to the choice of domain for parameters.

Value

Output is a list containing: The observed covariance The observed correlation coefficient The raw ordered pair data used to measure these

Note

EXTREMELY IMPORTANT NOTE: The functions in this package evaluate past performance only. No warranty is made that the results of these tests should, or even can, be used to inform business decisions or make predictions of future events.

The author does not make any claim that any results will predict future performance. No such prediction is made, directly or implied, by the outputs of these function, and any attempt to use these function for such prediction is done solely at the risk of the end user.

Author(s)

David St John

References

Ryan Sullivan, Allan Timmermann, and Halbert White. Data snooping, technical trading rule performance, and the bootstrap. *The Journal of Finance*, 54(5):1647-1691, 1999.

Examples

```
data(spData)
sp <- subperiods(spData)
```

```
sp[[1]]
sp[[2]]
```

Index

- *Topic **datasets**
 - spData, [25](#)
 - *Topic **design**
 - dataSnoop, [9](#)
 - nullModel, [17](#)
 - paramPersist, [18](#)
 - paramStats, [20](#)
 - subperiods, [26](#)
 - *Topic **distribution**
 - bootstrap, [6](#)
 - generateSample, [13](#)
 - *Topic **package**
 - ttrTests-package, [2](#)
 - *Topic **ts**
 - indicator, [14](#)
 - macd4, [16](#)
 - position, [23](#)
 - *Topic **univar**
 - cReturns, [7](#)
 - returnStats, [24](#)
- [bootstrap, 6](#)
- [cReturns, 7](#)
- [dataSnoop, 9](#)
[defaults, 11](#)
[deleteNA, 12](#)
- [generateSample, 13](#)
- [indicator, 14](#)
- [macd4, 16](#)
- [nullModel, 17](#)
- [paramPersist, 18](#)
[paramStats, 20](#)
[position, 23](#)
- [returnStats, 24](#)
- [spData, 25](#)
[subperiods, 26](#)
- [ttrTests \(ttrTests-package\), 2](#)
[ttrTests-package, 2](#)