Package ‘meboot’

February 20, 2015

Version 1.4-5
Date 2014-09-28
Title Maximum Entropy Bootstrap for Time Series
Author Hrishikesh D. Vinod <vinod@fordham.edu> and Javier López-de-Lacalle
Maintainer Javier López-de-Lacalle <javlacalle@yahoo.es>
Encoding UTF-8
Depends R (>= 3.0.0), dynlm, nlme
Suggests boot, car, ConvergenceConcepts, geepack, lmtest, strucchange, plm, zoo
Description This package performs maximum entropy density based dependent data bootstrap. An algorithm is provided to create a population of time series (ensemble) without assuming stationarity. The reference paper (Vinod, H.D., 2004) explains how the algorithm satisfies the ergodic theorem and the central limit theorem.
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-09-28 16:35:47

R topics documented:

checkConv ......................................................... 2
elapsedtime ....................................................... 4
expand.sd .......................................................... 5
flexMeboot ......................................................... 6
force.clt .......................................................... 7
meboot ............................................................. 8
meboot.part ......................................................... 10
meboot.pdata.frame ............................................. 11
null.ci ............................................................ 12
checkConv

This function generates a 3D array giving \((X_n - X)\) in the notation of the ConvergenceConcepts package by Lafaye de Micheaux and Liquet for sample paths with dimensions \(= n999\) as first dimension, \(nover =\) range of \(n\) values as second dimension and number of items in \(key\) as the third dimension. It is intended to be used for checking convergence of \(meboot\) in the context of a specific real world time series regression problem.

Usage

checkConv \((y, bigx, trueb = 1, n999 = 999, nover = 5, \\
seed1 = 294, key = 0, trace = FALSE)\)

Arguments

- \(y\): vector of data containing the dependent variable.
- \(bigx\): vector of data for all regressor variables in a regression or \(ts\) object. \(bigx\) should not include column of ones for the intercept.
- \(trueb\): true values of regressor coefficients for simulation. If \(trueb = 0\) then use OLS coefficient values rounded to 2 digits as true values of beta for simulation purposes, to be close to but not exactly equal to OLS.
- \(n999\): number of replicates to generate in a simulation.
- \(nover\): number of values of \(n\) over which convergence calculated.
- \(seed1\): seed for the random number generator.
- \(key\): the subset of key regression coefficient whose convergence is studied if \(key = 0\) all coefficients are studied for convergence.
- \(trace\): logical. If \(TRUE\), tracing information on the process is printed.

Details

Use this only when lagged dependent variable is absent.

Warning: \(key = 0\) might use up too much memory for large regression problems.

The algorithm first creates data on the dependent variable for a simulation using known true values denoted by \(trueb\). It proceeds to create \(n999\) regression problems using the seven-step algorithm in
**checkConv**

meboot creating n999 time series for all variable in the simulated regression. It then creates sample paths over a range of n values for coefficients of interest denoted as key (usually a subset of original coefficients). For each key coefficient there are n999 paths as n increases. If meboot algorithm is converging to true values, the value of (Xn-X) based criteria for "convergence in probability" and "almost sure convergence" in the notation of the ConvergenceConcepts package should decline. The decline can be plotted and/or tested to check if it is statistically significant as sample size increases. This function permits the user of meboot working with a short time series to see if the meboot algorithm is working in his or her particular situation.

**Value**

A 3 dimensional array giving (Xn-X) for sample paths with dimensions = n999 as first dimension, nover = range of n values as second dimension and number of items in key as the third dimension ready for use in ConvergenceConcepts package.

**References**


**See Also**

meboot, criterion.

**Examples**

```r
## Not run:
library("ConvergenceConcepts")
library("strucchange")
data("PhillipsCurve")
## Not run:
uk <- window(PhillipsCurve, start = 1948)
attach(data.frame(uk))
y <- uk[,5]
bigx <- cbind(dp1,du,u1) # collect all regressors
reg1 <- lm(y ~ bigx)
coef(reg1)
nover <- 16 # choose range allowing n=25, 26, ..., 40

# the following step will take some time to run
sc <- checkconv(y, bigx, trueb=0, n999=999, nover=nover, seed1=294)

dim(sc) # n999 x nover x length(key)
j <- 3 # choose key coefficient no. 3 for lagged income
epsilon <- 0.1 # needed for p.as.plot command below
```
nb.sp <- 10 # needed for p.as.plot command below
mode <- "p" # needed for p.as.plot command below

# criterion function in ConvergenceConcepts package wants 999 rows
dat <- sC[, , ] # 999 sample paths for diff inflation n=25,26,..40
critp <- criterion(data = dat, epsilon = epsilon, mode = "p")$crit
critas <- criterion(data = dat, epsilon = epsilon, mode = "as")$crit

p.as.plot(dat, critp, critas, epsilon, nb.sp, mode = mode)
nstart <- length(y) - nover + 1
nn <- seq(nstart, length(y)) # choose the range of n the sample size

opar <- par(mfrow = c(2,1)) #plot 2 plots in one
plot(nn, critp, type="l",
    main="Convergence in probability: Diff(Inflation) Coefficient",
    xlab="Sample size", ylab="Criterion using 999 sample paths")
plot(nn, critas, type="l",
    main="Almost sure convergence: Diff(Inflation) Coefficient",
    xlab="Sample size", ylab="Criterion using 999 sample paths")
par(opar)

regp <- lm(critp ~ nn) # OLS of conv. in prob. criterion
sup <- summary(regp) # regressed on sample size
sup$coef
# slope coeff. should be negative in sign for convergence
# the t statistic on the slope coefficient should be large
regas <- lm(critas ~ nn) #OLS of almost sure conv criterion
suas <- summary(regas)#regressed on sample size
suas$coef
# slope coeff. should be negative in sign for convergence
# the t statistic on the slope coefficient should be large

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

---

**elapsedtime**

**Internal Function**

**Description**

Internal function.

**Usage**

`elapsedtime(ptm1, ptm2)`

**Arguments**

- `ptm1` *proc.time* object.
- `ptm2` *proc.time* object.
Value

List giving the elapsed time between ptm1 and ptm2.

Description

This function expands the standard deviation of the simulated data. Expansion is needed since some of the ratios of the actual standard deviation to that of the original data are lower than 1 due to attenuation.

Usage

\texttt{expand.sd (x, ensemble, fiv=5)}

Arguments

\begin{itemize}
  \item \textbf{x} \texttt{a vector of data or a time series object.}
  \item \textbf{ensemble} \texttt{a matrix or \texttt{mts} object containing resamples of the original data \texttt{x}.}
  \item \textbf{fiv} \texttt{reference value for the upper limit of a uniform distribution used in expansion. For example, if equal to 5 the standard deviation of each resample is expanded through a value from a uniform random distribution with lower limit equal to 1 and upper limit equal to 1+(5/100)=1.05.}
\end{itemize}

Value

Resamples (by columns) with expanded standard deviations.

Examples

\begin{verbatim}
set.seed(345)
out <- meboot(x=AirPassengers, reps=100, trim=0.10, reachbnd=FALSE, elaps=TRUE)
exp.ens <- expand.sd(x=AirPassengers, out$ensemble)
\end{verbatim}
Description

This function extends the maximum entropy bootstrap procedure implemented in `meboot` to allow for a flexible trend up, flat or down.

Usage

```r
flexMeboot(x, reps = 9, segment = 5, forc = FALSE, myseq = seq(-1, 1, by = 1))
```

Arguments

- `x` vector of data, `ts` object.
- `reps` number of replicates to generate.
- `segment` block size.
- `forc` logical. If `TRUE` the ensemble is forced to satisfy the central limit theorem. See `forceNclt`.
- `myseq` directions for trend within a block of data is chosen randomly with the user’s choice limited by the range of values given by `myseq`. For example, `myseq = seq(-1, 1, by = 0.5)` provides five options for direction changes. If the user specifies any single number instead of a sequence, (e.g., `myseq = 1`) then `flexMeboot` will not change the directions of trends at all, but will modify the original `meboot` function to resample separately within several non-overlapping blocks, before joining them into resampled time series. This may be desirable for long series and for some applications.

Details

`flexMeboot` uses non-overlapping blocks having only `m` observations. A trend `a + bt` is replaced by `a + Bt`, where `B = sample(myseq) * b`.

Its steps are as follows:

1. Choose block size `segment` denoted here as `m` (default equal to `m = 5`) and divide the original time series `x` of length `T` into `k = floor(T/m)` blocks or subsets. Note that when `T/m` is not an integer the `k`-th block will have a few more than `m` items. Hence let us denote the number of observations in each block as `m` which equals `m` for most blocks, except the `k`-th.
2. Regress each block having `m` observations as subsets of `x` on the set `τ = 1, 2, ..., m`, and store the intercept `b0`, the slope `b1` of `τ` and the residuals `r`.
3. Note that the positive (negative) sign of the slope `b1` in this regression determines the up (down) direction of the time series in that block. Hence the next step of the algorithm replaces `b1` by `B1 = b1 * w`, defined by a randomly chosen weight `win(-1, 0, 1)`. For example, when the random choice yields `w = -1`, the sign of `b1` is reversed. Our weighting independently injects some limited flexibility to the directions of values block segments of the original time series.
4. Reconstruct all time series blocks as: \( b_0 + b_1 \ast w \ast \tau + r \), by adding back the residual \( r \) of the regression on \( \tau \).

5. The next step applies the function `meboot` to each block of time series now having a modified trend and create a large number, \( J \), of resampled time series for each of the \( k \) blocks.

6. Sequentially join the \( J \) replicates of all \( k \) blocks or subsets together.

**Value**

A matrix containing by columns the bootstrapped replicated of the original data \( x \).

**References**


**See Also**

`meboot`.

**Examples**

```r
set.seed(235)
myseq <- seq(-1, 1, by = 0.5)
xx <- flexMeboot(x = AirPassengers, myseq = myseq, reps = 3)
matplot(cbind(AirPassengers, xx), type = "l")
```

---

**force.clt**

*Enforce Central Limit Theorem*

**Description**

Function to enforce the maximum entropy bootstrap resamples to satisfy the central limit theorem.

**Usage**

```r
force.clt(x, ensemble)
```

**Arguments**

- \( x \) : a vector of data or a time series object.
- \( \text{ensemble} \) : a matrix or \texttt{mts} object containing resamples of the original data \( x \).

**Value**

Revised matrix satisfying the central limit theorem.
Examples

```r
set.seed(345)
out <- meboot(x=AirPassengers, reps=100, trim=0.10, reachbnd=FALSE, elaps=TRUE)
cm1 <- colMeans(out$ensemble)
# Note that the column means are somewhat non-normal
qqnorm(cm1)

clt.ens <- force.clt(x=AirPassengers, ensemble=out$ensemble)
cm2 <- colMeans(clt.ens)
# Note that the columns are closer to being normal
qqnorm(cm2)
```

---

**meboot**

*Generate Maximum Entropy Bootstrapped Time Series Ensemble*

**Description**

Generates maximum entropy bootstrap replicates for dependent data. (See details.)

**Usage**

```r
meboot (x, reps=999, trim=list(trim=0.10, xmin=NULL, xmax=NULL), reachbnd=TRUE,
expand.sd=TRUE, force.clt=TRUE, scl.adjustment = FALSE, sym = FALSE,
elaps=FALSE, colsubj, coldata, coltimes, ...)
```

**Arguments**

- **x**: vector of data, ts object or pdata.frame object.
- **reps**: number of replicates to generate.
- **trim**: a list object containing the elements: trim, the trimming proportion; xmin, the lower limit for left tail and xmax, the upper limit for right tail.
- **reachbnd**: logical. If TRUE potentially reached bounds (xmin = smallest value - trimmed mean and xmax=largest value + trimmed mean) are given when the random draw happens to be equal to 0 and 1, respectively.
- **expand.sd**: logical. If TRUE the standard deviation in the ensemble in expanded. See expand.sd.
- **force.clt**: logical. If TRUE the ensemble is forced to satisfy the central limit theorem. See force.clt.
- **scl.adjustment**: logical. If TRUE scale adjustment is performed to ensure that the population variance of the transformed series equals the variance of the data.
- **sym**: logical. If TRUE an adjustment is performed to ensure that the ME density is symmetric.
- **elaps**: logical. If TRUE elapsed time during computations is displayed.
colsubj 
the column in x that contains the individual index. It is ignored if the input data x is not a pdata.frame object.

coldata 
the column in x that contains the data of the variable to create the ensemble. It is ignored if the input data x is not a pdata.frame object.

coltimes 
an optional argument indicating the column that contains the times at which the observations for each individual are observed. It is ignored if the input data x is not a pdata.frame object.

... 
possible argument fiv to be passed to expand.sd.

Details

Seven-steps algorithm:

1. Sort the original data in increasing order and store the ordering index vector.
2. Compute intermediate points on the sorted series.
3. Compute lower limit for left tail (xmin) and upper limit for right tail (xmax). This is done by computing the trim (e.g. 10)
4. Compute the mean of the maximum entropy density within each interval in such a way that the mean preserving constraint is satisfied. (Denoted as mt in the reference paper.) The first and last interval means have distinct formulas. See Theil and Laitinen (1980) for details.
5. Generate random numbers from the [0,1] uniform interval and compute sample quantiles at those points.
6. Apply to the sample quantiles the correct order to keep the dependence relationships of the observed data.
7. Repeat the previous steps several times (e.g. 999).

The scale and symmetry adjustments are described in Vinod (2013) referenced below.

In some applications, the ensembles must be ensured to be non-negative. Setting trim$xmin = 0 ensures positive values of the ensembles. It also requires force.clt = FALSE and expand.sd = FALSE. These arguments are set to FALSE if trim$xmin = 0 is defined and a warning is returned to inform that the value of those arguments were overwritten. Note: The choice of xmin and xmax cannot be arbitrary and should be cognizant of range(x) in data. Otherwise, if there are observations outside those bounds, the limits set by these arguments may not be met. If the user is concerned only with the trimming proportion, then it can be passed as argument simply trim = 0.1 and the default values for xmin and xmax will be used.

Value

x 
original data provided as input.

ensemble 
maximum entropy bootstrap replicates.

xx 
sorted order stats (xx[1] is minimum value).

z 
class intervals limits.

dv 
deviations of consecutive data values.

dvtrim 
trimmed mean of dv.

xmin 
data minimum for ensemble=xx[1]-dvtrim.
References


Examples

```r
## Ensemble for the AirPassenger time series data
set.seed(345)
out <- meboot(x=AirPassengers, reps=100, trim=0.10, elaps=TRUE)

## Ensemble for T=5 toy time series used in Vinod (2004)
set.seed(345)
out <- meboot(x=c(4, 12, 36, 20, 8), reps=999, trim=0.25, elaps=TRUE)
mean(out$ens) # ensemble mean should be close to sample mean 16
```

Description

Internal function.

Usage

`meboot.part (x, n, z, xmin, xmax, desintxb, reachbnd)`

Arguments

- `x` vector of data, ts object or `pdata.frame` object.
- `n` length of `x`.
- `z` class intervals limits.
- `xmin` lower limit in the left tail.
- `xmax` upper limit in the left tail.
- `desintxb` desired interval means.
reachbnd logical. If TRUE potentially reached bounds (xmin = smallest value - trimmed mean and xmax=largest value + trimmed mean) are given when the random draw happens to be equal to 0 and 1, respectively.

Value
A vector of resampled data.

See Also
meboot.

Description
This function applies the maximum entropy bootstraped in a panel of time series data.

Usage
meboot.pdata.frame (x, reps=999, trim=0.10, reachbnd=TRUE, expand.sd=TRUE, force.clt=TRUE, scl.adjustment = FALSE, sym = FALSE, elaps=FALSE, colsubj, coldata, coltimes, ...)

Arguments
x a pdata.frame object containing by columns: the individual index, an optional time index and a panel of time series data.
reps number of replicates to generate for each subject in the panel.
trim the trimming proportion.
reachbnd logical. If TRUE potentially reached bounds (xmin = smallest value - trimmed mean and xmax=largest value + trimmed mean) are given when the random draw happens to be equal to 0 and 1, respectively.
expand.sd logical. If TRUE the standard deviation in the ensemble in expanded. See expand.sd.
force.clt logical. If TRUE the ensemble is forced to satisfy the central limit theorem. See force.clt.
scl.adjustment logical. If TRUE scale adjustment is performed to ensure that the population variance of the transformed series equals the variance of the data.
sym logical. If TRUE an adjustment is performed to ensure that the ME density is symmetric.
elaps logical. If TRUE elapsed time during computations is displayed.
colsubj the column in x that contains the individual index.
coldata  the column in x that contains the data of the variable to create the ensemble.
coltimes an optional argument indicating the column that contains the times at which the
observations for each individual are observed.
... possible argument fiv to be passed to expand.sd.

Details

The observations in x should be arranged by individuals. The observations for each individual must be sorted by time.
The argument colsubj can be either a numeric or a character index indicating the individual or the time series to which each observation is related.
Only one variable can be replicated at a time, coldata must be of length one.
If the times at which observations are observed is provided specifying the column with the times through the argument coltimes, these times are used only to label the rows of the data.frame returned as output.

Value

A data.frame object of dimension: number of rows of x times number of replicates indicated in reps. The replicates for the panel of data are arranged by columns. Each replicate in each column is sorted with the same order stablished in the input x.

See Also

meboot.

Examples

## Ensemble for a panel of series of stock prices
data("ullwan")
out <- meboot(ullwan, reps=99, colsubj=2, coldata=4)

null.ci  Get Confidence Interval Around Specified NullZero Total

Description

Function to get two sided confidence interval around zero as the true value. Confidence interval is adjusted so that it covers the true zero (1-‘level’)\*100 times. Symmetry is not assumed.

Usage

null.ci (x, level=0.95, null.value=0, type=8, ...)

nullNci Get Confidence Interval Around Specified NullZero Total

## Get Confidence Interval Around Specified NullZero Total

## Function to get two sided confidence interval around zero as the true value. Confidence interval is adjusted so that it covers the true zero (1-‘level’)\*100 times. Symmetry is not assumed.

Usage

null.ci (x, level=0.95, null.value=0, type=8, ...)

nullNci Get Confidence Interval Around Specified NullZero Total

## Get Confidence Interval Around Specified NullZero Total

## Function to get two sided confidence interval around zero as the true value. Confidence interval is adjusted so that it covers the true zero (1-‘level’)\*100 times. Symmetry is not assumed.

Usage

null.ci (x, level=0.95, null.value=0, type=8, ...)

nullNci Get Confidence Interval Around Specified NullZero Total

## Get Confidence Interval Around Specified NullZero Total

## Function to get two sided confidence interval around zero as the true value. Confidence interval is adjusted so that it covers the true zero (1-‘level’)\*100 times. Symmetry is not assumed.

Usage

null.ci (x, level=0.95, null.value=0, type=8, ...)
Arguments

- `x` a vector of data.
- `level` confidence level.
- `null.value` a specified value of the null, e.g., 0.
- `type` type of quantile, a number between 1 and 9. See `quantile`.
- `...` further arguments passed to or from other methods.

Value

Lower limit and upper limit of the confidence interval.

Examples

```r
x <- runif(25, 0, 1)
null.ci(x)
```

Description

Compute OLS coefficients in a regression model for the consumption variable. See details.

Usage

```r
olshallNb(y, x)
```

Arguments

- `y` dependent variable (consumption).
- `x` regressor variable (disposable income).

Details

The regression model is: \( c(t) = b_1 + b_2c(t) + b_3y(t-1) + u(t) \), where 'c' is consumption and 'y' is disposable income.

This function is intended to speed up the ME bootstrap procedure for inference. Instead of using the `lm` or `dynlm` interfaces the function calls directly to the Fortran procedure 'dqrls'.

Value

Coefficient estimates by OLS.
Examples

data("USconsum")
USconsum <- log(USconsum)

# lm interface
lmcf1 <- lm(USconsum[-1,1] ~ USconsum[-51,1] + USconsum[-51,2])
coefficients(lmcf1)

# dynlm interface
library("dynlm")
lmcf2 <- dynlm(consum ~ L(consum, 1) + L(dispinc, 1), data=USconsum)
coefficients(lmcf2)

# olshALL.b
olsHALL.b(y=USconsum[,1], x=USconsum[,2])

ullwan

Data about Some of the S\&P 500 Stock Prices

Description

This data set collects information about seven S\&P 500 stocks whose market capitalization exceeds \$27 billion. The seven companies are labelled as ABT, AEG, ATI, ALD, ALL, AOL and AXP. For each company data from May 1993 to November 1997 are available (469 observations).

Usage

data (ullwan)

Format

The data are stored in an object of classes pdata.frame (a data.frame class with further attributes useful for panel data from the plm package) and data.frame.

Details

The following information is contained by columns:

- **Subj**: Company index.
- **Tim**: Times at which the data were observed (on a monthly basis).
- **MktVal**: Market capitalization.
- **Price**: Stock prices.
- **Pupdn**: Binary variable, takes the value 1 if there is a turning point (a switch from a bull to a bear market or vice versa) and 0 otherwise.
- **Tb3**: Interest on 3-month Treasury bills.
**Source**

Compustat database.

**References**


---

**USconsum**

*Consumption and Disposable Income Data (Annual 1948-1998)*

**Description**

Data set employed in Murray (2006, pp.46-47, 799-801) to discuss the Keynesian consumption function on the basis of the Friedman’s permanent income hypothesis and Robert Hall’s model.

**Usage**

```r
data (USconsum)
```

**Format**

A .rda file storing the data as an mts object.

**Details**

Annual data. Available time series: (Each corresponding label in the list object appears in quotes.)

- `consum`: consumption per capita in thousands of dollars (1948-1998). Log of this variable is the dependent variable and its lagged value is a regressor in Murray’s Table 18.1.
- `dispinc`: disposable income per capita in thousands of dollars (1948-1998). Lagged value of the Log of this variable is the second regressor (proxy for permanent income) in Murray’s Table 18.1.

**Source**


**References**

USfygt  

Long-term Treasury Bond Rates and Deficit Data Set (Annual 1948-200)

Description

Data set employed in Murray (2006, pp.795-797) to test the null hypothesis that per capita federal deficits explain long-term Treasury bond interest rates based on the Stock and Watson’s dynamic OLS model.

Usage

data (USfygt)

Format

A .rda file storing the data as an mts object.

Details

Annual data. Available time series: (Each corresponding label in the list object appears in quotes.)

- "dy": mean changes in real per capita income (1949-1998).
- "infl": inflation (1949-2000).
- "usdef": per capita real federal deficit (1948-2000).
- "reallir": real long term interest rates (not used in Murray’s Table 18.12).
- "realsir": real short term interest rates (not used in Murray’s Table 18.12).

Source

Data was made available by James Stock and Mark Watson to readers of their famous Econometrica paper, 1993, 61, pp 783-820, who in turn used standard sources for US macroeconomic data from government publications.

References

**zero.ci**  
*Get Confidence Interval Around Zero*

**Description**
Function to get two sided confidence interval around zero as the true value. Confidence interval is adjusted so that it covers the true zero (1-`confl`)*100 times. Symmetry is not assumed.

**Usage**
```
zero.ci(x, confl=0.05)
```

**Arguments**
- `x`: a vector of data.
- `confl`: confidence level.

**Value**
- `bnlo`: count of number of items below lower limit.
- `bnup`: count of number of items above upper limit.
- `lolim`: lower limit of the confidence interval.
- `uplim`: upper limit of the confidence interval.

**Examples**
```
x <- runif(25, 0, 1)
zero.ci(x)
```
Index

*Topic **datagen**
  - ullwan, 14
  - USconsum, 15
  - USfyygt, 16

*Topic **ts**
  - checkConv, 2
  - elapsedTime, 4
  - expand.sd, 5
  - flexMeboot, 6
  - force.clt, 7
  - meboot, 8
  - meboot.part, 10
  - meboot_pdata.frame, 11
  - null.ci, 12
  - zero.ci, 17

checkConv, 2

criterion, 3

elapsedtime, 4
expand.sd, 5, 8, 9, 11, 12

flexMeboot, 6
force.clt, 6, 7, 8, 11

meboot, 3, 6, 7, 8, 11, 12
meboot.part, 10
meboot_pdata.frame, 11

null.ci, 12

olsHALL.b, 13

proc.time, 4

quantile, 13

ullwan, 14
USconsum, 15
USfyygt, 16

zero.ci, 17