Maximum Entropy Bootstrap Algorithm Enhancements

Hrishikesh D. Vinod *

June 25, 2013

Abstract

While moving block bootstrap (MBB) has been used for mildly dependent (m-dependent) time series, maximum entropy (ME) bootstrap (meboot) is perhaps the only tool for inference involving perfectly dependent, nonstationary time series, possibly subject to jumps, regime changes and gaps. This brief note describes the logic and provides the R code for two potential enhancements to the meboot algorithm in Vinod and López-de-Lacalle (2009), available as the 'meboot' package of the R software. The first 'rescaling enhancement' adjusts the of meboot resampled elements so that the population variance of the ME density equals that of the original data. Our second 'symmetrizing enhancement' forces the ME density to be symmetric. One simulation involving inference for regression standard errors suggests that the symmetrizing enhancement of the meboot continues to outperform the MBB.

Keywords: maximum entropy, block bootstrap, variance, symmetry, R-software.

1 Introduction

Since the invention of iid bootstrap by Efron around 1979, bootstrap has become a vital computer intensive tool for statistical inference (not estimation). It is particularly suited for complicated problems where traditional

*Professor of Economics, Fordham University, Bronx, New York, USA 10458. E-mail: vinod@fordham.edu.

(asymptotic) confidence intervals tend to be too wide, difficult to construct and unreliable in the absence of knowledge about the form of underlying distributions.

Vinod (2006, ch.9) explains that traditional inference for time series based on Wiener-Kolmogorov-Khintchine (WKK) theory using higher mathematics was developed in the 1930's, long before we had powerful computers. WKK construct a population of time series called ensemble Ω, heavily relying on the stationarity assumption. The meboot algorithm offers a new computer intensive construction of Ω for applications where the time series is short, non-stationary, perhaps with regime changes, gaps and jump discontinuities. While the moving block bootstrap (MBB) has been used for mildly dependent (m-dependent) series, maximum entropy bootstrap (meboot) is the only tool for highly dependent nonstationary time series.

The next section introduces the meboot package algorithm of the free R software in Vinod and López-de-Lacalle (2009) using a toy example included here for completeness. Section 2 may be skipped by readers familiar with the meboot algorithm. Sections 3 and 4 describe the rescaling and symmetrizing enhancements along with complete R code for implementing them.

2 Review of the Maximum Entropy Bootstrap

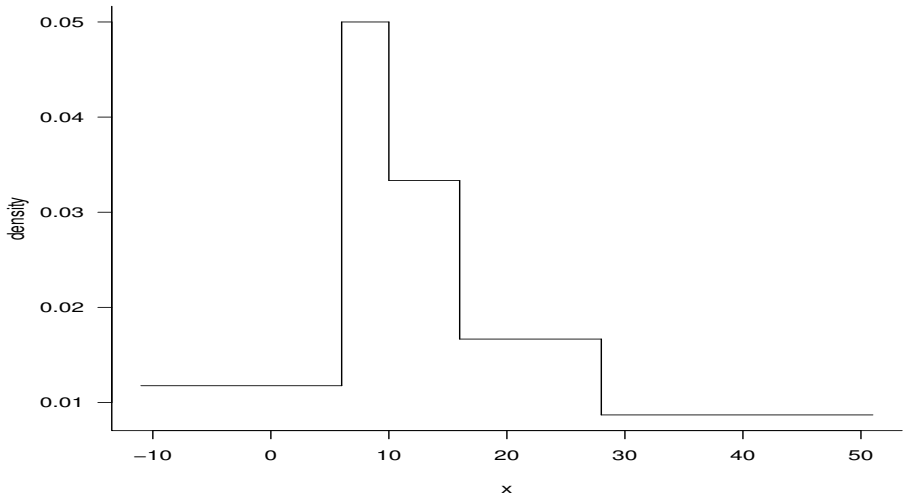
Maximum entropy is a powerful tool for avoiding unnecessary distributional assumptions. Let $f(x)$ denote the density of x_t. The entropy H is defined as:

$$H = E(-\log f(x)). \tag{1}$$

The maximum entropy (ME) density is maximally noncommittal about unavailable information regarding its functional form. When the limits of the support of $f(x)$ are known, it is well known that the uniform functional form is the ME density. In the context of the ME bootstrap we have a simple set of T uniformly distributed mixture of finite pieces joined together into what we call the ME density. Our ME density is further subject to mass and mean-preserving constraints as described in Vinod (2006). A brief description of the constraints follows.

Denote the order statistics of $x_t, t = 1, \ldots, T$ by $x_{(t)}$. Let m_{trm} denote the ten percent (say) trimmed mean of absolute distances between consecutive points $|x_t - x_{t-1}|$. Assume that the researcher knows the overall finite

Figure 1: Toy ME density for $x_t = (4, 12, 36, 20, 8)$



.

outer range of plausible limits $x_t \in [x_{\rm LO} = x_{(1)} - m_{trm}, x_{\rm UP} = x_{(T)} + m_{trm}]$, *supporting* the ME density on the finite interval along the horizontal axis.

Figure 1 reproduces the ME density for a toy example used in Vinod and López-de-Lacalle (2009).

Define $z_0 = x_{\rm LO}, z_T = x_{\rm UP}$, and

$$z_t = 0.5(x_{(t)} + x_{(t+1)}), \quad t = 1, \ldots, T-1, \tag{2}$$

as midpoints of T intervals based on extrapolated data beyond the observed range by the distance d_{trm} along both sides of the range.

Next, define half open intervals:

$$I_t = (z_{t-1}, z_t], \quad t = 1, \ldots, T, \tag{3}$$

of points around each observed $x_{(t)}$ from which we select the elements of our resample. We have exactly T intervals, each of which contains exactly one $x_{(t)}$ and our resample will contain one observation from each such interval with probability $1/T$.

The **mass-preserving** constraint says that, on an average, a fraction $1/T$ of the mass of the probability distribution must lie in each interval. The "meboot" algorithm achieves this by giving an equal chance to each half open

interval I_t defined in 3 of being included in the resample. The iid bootstrap selects uniform pseudo-random numbers between [0,1], and transforms them into a sequence of random integers in $[1, T]$ to define the shuffled selection from the set $\{x_t\}$. Thus, the iid bootstrap gives each observation an equal chance $(1/T)$ of being included in the resample. The advantages of the "meboot" are: (i) it uses the entire interval that contains the observation and not necessarily the observation itself, and (ii) it uses the pseudo-random numbers, not transformed integers.

The **mean-preserving** constraint (on order 1 moments of $f(x)$) is $\Sigma x_t = \Sigma x_{(t)} = \Sigma m_t$, where m_t denote the mean of $f(x)$ within the interval I_t. This property is satisfied by the iid bootstrap, since its $f(x)$ is a delta function with the mass $(1/T)$ concentrated at x_t in I_t. The mean-preserving requirement is a bit complicated for the "meboot," since it requires that the mean m_t in the interval I_t is equal to a weighted sum of the order statistic $x_{(t)}$ with weights from the set $\{0.25, 0.50, 0.75\}$ explained below. Theil and Laitinen (1980) chose the following interval means, m_t, to satisfy this constraint.

$$\begin{aligned}
f(x) \quad &= 1/(z_1 - z_0), \quad X \in I_{(1)}, \quad m_1 = 0.75x_{(1)} + 0.25x_{(2)}, \\
f(x) \quad &= 1/(z_k - z_{k-1}), \quad X \in (z_k - z_{k-1}], \\
&\text{with mean} \quad m_k = 0.25x_{(k-1)} + 0.50x_{(k)} + 0.25x_{(k+1)} \qquad (4) \\
&\text{for} \quad k = 2, \ldots, T-1, \\
f(x) \quad &= 1/(z_T - z_{T-1}), \quad X \in I_{(T)}, \quad m_T = 0.25x_{(T-1)} + 0.75x_{(T)}.
\end{aligned}$$

The intervals in equation (3) comprising the ME density as a mixture of uniform densities have means m_t defined in eq. (4) and the overall mean is a weighted sum:

$$E(X) = \sum_{t=1}^{T} m_t/T = \bar{x}, \qquad (5)$$

by construction of the mean preserving constraint. Vinod and López-de-Lacalle (2009) describe a seven-step algorithm to create a large number J of $x_{(t,j)}, j = 1, \ldots, J$, analogous time series satisfying both mass and mean preserving constraints. The algorithm retains the time-dependence of original x_t by additional steps whereby the $x_{(t,j)}$ and x_t have unit rank correlation coefficient. The implementation details with examples are available in the 'meboot' package of the free software R.

We conclude this introductory section showing the practical usage of the R function 'meboot' with the help of a toy example used in other papers. Let $x_t = (4, 12, 36, 20, 8)$, for $t = 1, \ldots, T$, respectively, having the mean $\bar{x} = 16$. The following R code creates J resamples $x_{(t,j)}$ in a $T \times J$ matrix representing the ensemble.

```
require(meboot)
set.seed(234)
x=c(4,12,36,20,8)
xtj=meboot(x,reps=4)$ensem
xtj
```

Now the order statistics are $x_{(t)} = (4, 8, 12, 20, 36)$. The trimmed mean of absolute consecutive deviations is $m_{trm} = 15$, leading to the following limits z_t of the $T = 5$ half-open intervals of eq. (3): $z_0 = x_{\mathrm{LO}} = (x_{(1)} - m_{trm}) = -11$, is the lower limit of the first interval. Now z_1 to $z_4 = (6, 10, 16, 28)$ are the intermediate limits. Finally, $z_T = x_{\mathrm{UP}} = (x_{(T)} + m_{trm}) = 51$ is the upper limit of the fifth interval.

The desired means of these intervals using (4) to ensure mean-preserving constraint are: m_t=(5, 8, 13, 22, 32), whose grand mean is 16 as desired. The output of the above code provides the small ensemble Ω below.

```
> xtj
          [,1]      [,2]      [,3]      [,4]
[1,] -8.248707  4.101817 -14.61687  2.808591
[2,] 25.785338 19.475746  13.99363 13.284345
[3,] 27.950884 39.296600  14.90885 38.305254
[4,] 27.613262 39.106913  14.90408 26.451677
[5,] -4.264504 15.104922  13.72431 10.313860
```

The above code shows that it is easy to implement 'meboot'. The toy example has $T = 5, J = 4$ for brevity. A real world example will have a larger T and usual $J \geq 999$ for a meaningful ensemble Ω.

3 Scale Adjustment to the meboot algorithm

Theil and Laitinen (1980) show that under certain conditions when the two tails are assumed to be exponentially distributed, the overall variance of the

ME density is smaller than σ_x^2 of the original data defined by:

$$\sigma_x^2 = \sum_{t=1}^{T} \frac{(x_t - \bar{x})^2}{T-1} \tag{6}$$

before any resampling to create the $x_{(t,j)}, j = 1, \ldots, J$, series. As with the usual bootstrap, the resamples are randomly chosen (iid) from the cumulative distribution function associated with the ME density.

It is well known that the usual iid bootstrap sometimes uses a scale up of observed regression residuals using a constant $\sqrt{[T/(T-p)]} > 1$ to make their variance close to the variance of true regression errors, Vinod (2008, p.395). In this subsection we investigate a new scale adjustment to the meboot algorithm to make the population variance of the ME density from which resamples $x_{(t,j)}$ are chosen to equal σ_x^2.

Let us begin by evaluating the conditional variance 'within' each half open interval $t = 1, \ldots T$:

$$\sigma_t^2 = var(X|X \in \{z_{t-1}, z_t\}) = \frac{(z_t - z_{t-1})^2}{12}, \tag{7}$$

from the properties of the uniform density.

Now the overall population variance of the ME density is obtained by the sum of T expressions weighted by $1/T$ as:

$$\sigma_{me}^2 = E(X - \bar{x})^2 = \frac{1}{T} \sum_{t=1}^{T} [(m_t - \bar{x})^2 + \sigma_t^2]. \tag{8}$$

upon adding the 'between' and 'within' sum of squares components of the usual analysis of variance.

In general, it is difficult to know exactly how much the theoretical unconditional variance var(X) of the ME density based on eq. (8) will differ from the variance of x_t in eq. (6) above. Fortunately, in our bootstrap context we can compute both quantities from the available data and use them to determine a suitable linear transformation of the ME bootstrap resamples $x_{(t,j)}$ described below. Consider the ratio of the desired population standard deviation in (6) to the population standard deviation from the ME density in (8):

$$\kappa_1 = \frac{\sigma_x}{\sigma_{me}}, \tag{9}$$

where we would need to expand σ^2_{me} only if $\kappa_1 > 1$ holds. If on the other hand, we have $\kappa_1 < 1$, we must shrink the ME-density-based resampled elements $x_{(t,j)}$ so that their variance equals the original variance of the time series σ^2_x. Hence, we can always use κ_1 to expand or shrink the variance of elements $x_{(t,j)}$ from the ME density to the correct scale as needed. Unfortunately simple scale change also changes the mean. Hence the following is needed to ensure that the 'mean preserving constraint' remains satisfied.

Consider the following linear transformation:

$$y_{(t,j)} = x_{(t,j)} + \kappa(x_{(t,j)} - \bar{x}), \tag{10}$$

having the following properties: (i) The population mean of transformed data $E(y_{(t,j)}) = \bar{x} + \kappa(\bar{x} - \bar{x}) = \bar{x}$. (ii) The transformation changes the variance of $x_{(t,j)}$ according to the formula:

$$var(y_{(t,j)}) = (1+\kappa)^2 var(x_{(t,j)}) = (1+\kappa)^2 \sigma^2_{me}. \tag{11}$$

In our context, we want to transform $x_{(t,j)}$ derived from the ME density [having population mean $\bar{x}$ by mean preserving constraint and population variance σ^2_{me}] to $y_{(t,j)}$ with the same population mean $\bar{x}$ and σ^2_x as population (not sample) variance. Next, we need to determine the κ appearing in the transformation (10). In terms of standard deviations we want the κ to satisfy

$$\sqrt{[var(y_{(t,j)})]} = (1+\kappa)\,\sigma_{me}, \tag{12}$$

where the left side is simply σ_x. Hence using the definition (9) verify that

$$\kappa = (\kappa_1) - 1. \tag{13}$$

Note that $y_{(t,j)}$ have a larger variance than $x_{(t,j)}$ as long as $\kappa > 0$. Now we are ready to transform the resampled meboot values from $x_{(t,j)}$ to $y_{(t,j)}$ by using (10) with the indicated choice of the scale factor κ from (13) in order to adjust for the correct population variance without changing the population mean. The steps described above are implemented in the R code given here as a function called 'findKapa.'

```
findKapa=function (x, trim = 0.1)
{    #find factor by which to multiply sd of each ensemble
n <- length(x)
    xx <- sort(x)
```

```
#    ordxx <- order(x)
    z <- rowMeans(embed(xx, 2))
    dv <- abs(diff(as.numeric(x)))
    dvtrim <- mean(dv, trim = trim)
    xmin <- xx[1] - dvtrim
    xmax <- xx[n] + dvtrim
aux <- colSums(t(embed(xx, 3)) * c(0.25, 0.5, 0.25))
desintxb <- c(0.75 * xx[1] + 0.25 * xx[2], aux,
0.25 * xx[n - 1] + 0.75 * xx[n])

zz=c(xmin,z,xmax)#extended list of z values
v=rep(NA,n) #storing within variances
for (i in 2: (n+1)){
v[i-1]=((zz[i]-zz[i-1])^2)/12
}
xb=mean(x)
s1=sum((desintxb-xb)^2)
uv=(s1+sum(v))/n
desired.sd=sd(x)
actualME.sd=sqrt(uv)
if (actualME.sd<=0) print("actualME.sd<=0 Error")
out=desired.sd/actualME.sd
return(out-1)
}
```

We shall see with the help of our toy example how to call the 'findKapa' function. Note that the κ does not depend on the seed used for random number generation for the toy example, since our five x_t values are fixed, not random.

The steps in finding the κ for the toy example may be instructive. The within interval variances using (7) for the toy example are: $\sigma_t^2 = (24.083333, 1.333333, 3, 12, 44.083333)$, respectively. The 'between interval' variance $\Sigma(m_t - \bar{x})^2 = 486$ is added to the sum of within variances (=84.5) to yield the ME density population variance using eq. (8) to be $\sigma_{me}^2 = 114.1$, after dividing by $T = 5$. We have $\kappa_1 = \sigma_x/\sigma_{me} = 1.184178$. The standard deviation of x is over 18% higher than the population standard deviation of the ME density. This suggests using $\kappa = (\kappa_1) - 1 = 0.184178$ for this example.

```
require(meboot)
set.seed(234)
x=c(4,12,36,20,8)
kap=findKapa(x);kap
xtj=meboot(x,reps=4, expand.sd=FALSE)$ensem
xbar=mean(x);xbar
ytj=xtj+kap*(xtj-xbar)
apply(ytj,2,sd) #report sd for y(t,j)
apply(ytj,2,mean)/apply(xtj,2,mean)
apply(ytj,2,sd)/apply(xtj,2,sd)
```

The abridged output from the code is:

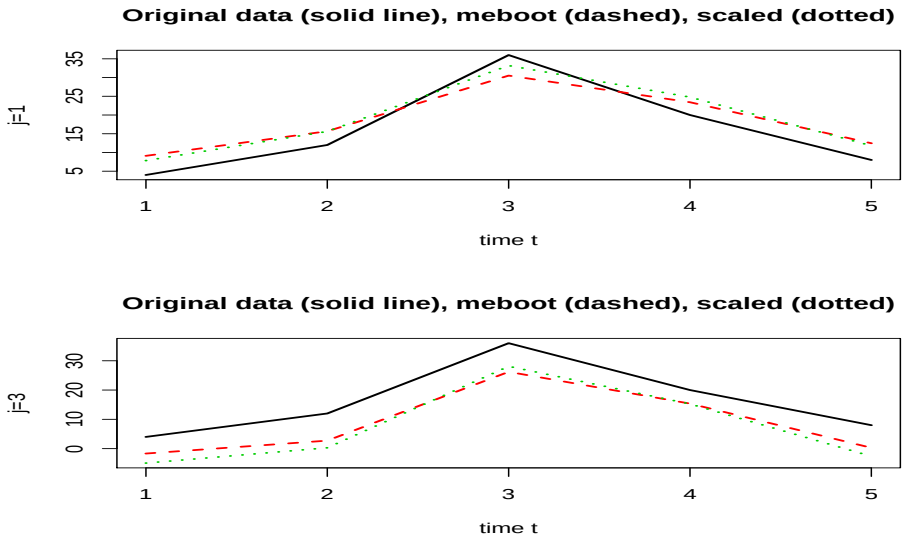
```
> kap=findKapa(x);kap
[1] 0.1841785
> apply(ytj,2,sd) #report sd for y(t,j)
[1] 21.73192 18.30730 13.73703 16.70024
> apply(ytj,2,sd)/apply(xtj,2,sd)
[1] 1.184178 1.184178 1.184178 1.184178
```

Although some output is suppressed for brevity, note that $\kappa = 0.184718$ holds. Also, verify that our transformation changes only the population variance. The sample standard deviation of $y_{(t,j)}$ for any particular j-th resample (column of ytj) need not equal $\sigma_x = 12.64911$. The last reported line of output above shows that the standard deviations of transformed data get multiplied by the common factor 1.184718. The following code creates a graphical representations of two resamples $j = 1, 3$ showing the effect of rescaling by κ.

```
par(mfrow=c(2,1))
my3=cbind(x, xtj[,1], ytj[,1])
matplot(my3, typ="l",xlab="time t", ylab="j=1",lwd=2)
title("Original data (solid line), meboot (dashed), scaled (dotted)")
my3=cbind(x, xtj[,3], ytj[,3])
matplot(my3, typ="l",xlab="time t", ylab="j=3",lwd=2)
title("Original data (solid line), meboot (dashed), scaled (dotted)")
```

Figure 2 plots the x_t of the toy example (solid line) along with two realizations $j = 1, j = 3$ respectively in the upper and lower panels. The dashed

Figure 2: Resamples from toy example meboot and its scaled by κ version



.

line refers to the usual meboot resamples (without any enhancement) and dotted line refers to its scaled (by κ version in both panels.

What is the motivation behind this scale adjustment? Given a time series x_t the unadjusted meboot constructs a large number J of similar time series $x_{(t,j)}$ to form an ensemble of time series to represent the population of time series using the ME density. Our scale adjustment from $x_{(t,j)}$ to $y_{(t,j)}$ makes sure that the population variance of the transformed series equals σ_x^2. This seems to be intuitively desirable, at least in some cases.

The transformation further induces a similar scale adjustment to the sampling distribution of a statistic under study. Since many of the sample statistics have asymptotically Normal distributions (based on central limit theorem type arguments), it may be desirable to have symmetric sampling distributions. This motivation leads to the next section.

4 Adjustment to make the ME density Symmetric

This subsection considers another potentially useful adjustment to the meboot algorithm. Note that the ME density is not generally symmetric. How-

ever statistical inference usually relies on the symmetric Normal density. The traditional confidence intervals use the symmetry when they cut off a fixed percent of realizations from both sides of a bootstrap approximation to the sampling distribution.

Theil (1980) first considered this problem for a version of the ME density having exponential tails. His derivation is based on the following property of order statistics $x_{(t)}$ of symmetric densities when the mean is $\bar{x}$: The sampling distribution of $x_{(t)} - \bar{x}$ is the same as that of $\bar{x} - x_{(T+1-t)}$. Let the symmetrized version of original data be denoted by y_t with order statistics $y_{(t)}$. Hence the symmetrized order statistics are defined by:

$$y_{(t)} = \bar{x} + 0.5[x_{(t)} - x_{(T+1-t)}] \tag{14}$$

Recall the 'meboot' algorithm relies heavily on the order statistics $x_{(t)}$. The modification to the algorithm would be replacing them by revised order statistics $y_{(t)}$ in every relevant step. This seemingly innocuous change has a noticeable effect on the resampled values $x_{(t,j)}$ as can be seen from the plot for a toy example where $x_t = (4, 12, 36, 20, 8)$, respectively.

The following code replaces the R function 'meboot' by a modified function called 'mebootSym' which provides an option to use the symmetric version by setting the value of the argument 'sym' to be 'TRUE'. The default value is FALSE.

```
mebootSym=function (x, reps = 999, trim = 0.1,
reachbnd = TRUE, expand.sd = TRUE,
  force.clt = TRUE, elaps = FALSE,
  sym=FALSE, colsubj, coldata, coltimes,
    ...)
{
    if ("pdata.frame" %in% class(x)) {
 res <- meboot.pdata.frame(x, reps, trim, reachbnd, expand.sd,
     force.clt, elaps, colsubj, coldata, coltimes, ...)
        return(res)
    }
    ptm1 <- proc.time()
    n <- length(x)
    xx <- sort(x)
if (sym){
    xxr=rev(xx) #reordered values
```

```
    xx.sym=mean(xx)+0.5*(xx-xxr)#symmetrized order stats
    xx=xx.sym #replace order stats by symmetrized ones
}
    ordxx <- order(x)
    z <- rowMeans(embed(xx, 2))
    dv <- abs(diff(as.numeric(x)))
    dvtrim <- mean(dv, trim = trim)
    xmin <- xx[1] - dvtrim
    xmax <- xx[n] + dvtrim
    aux <- colSums(t(embed(xx, 3)) * c(0.25, 0.5, 0.25))
    desintxb <- c(0.75 * xx[1] + 0.25 * xx[2], aux, 0.25 * xx[n -
        1] + 0.75 * xx[n])
    ensemble <- matrix(x, nrow = n, ncol = reps)
    ensemble <- apply(ensemble, 2, meboot.part,
    n, z, xmin, xmax,
        desintxb, reachbnd)
    qseq <- apply(ensemble, 2, sort)
    ensemble[ordxx, ] <- qseq
    if (expand.sd)
        ensemble <- expand.sd(x = x, ensemble = ensemble, ...)
    if (force.clt)
        ensemble <- force.clt(x = x, ensemble = ensemble)
    if (is.ts(x)) {
        ensemble <- ts(ensemble, frequency = frequency(x),
        start = start(x))
        dimnames(ensemble)[[2]] <- paste("Series", 1:reps)
    }
    ptm2 <- proc.time()
    elapsr <- elapsedtime(ptm1, ptm2)
    if (elaps)
    cat("\n  Elapsed time:", elapsr$elaps, paste(elapsr$units,
            ".", sep = ""), "\n")
    list(x = x, ensemble = ensemble, xx = xx, z = z, dv = dv,
        dvtrim = dvtrim, xmin = xmin, xmax = xmax,
        desintxb = desintxb,
        ordxx = ordxx, elaps = elapsr)
}
```

After entering the above code in R the following code calls the function mebootSym for the toy example mentioned earlier.

```
x=c(4,12,36,20,8)
require(meboot)
out1=meboot(x,reps=4)
out3=mebootSym(x,reps=4,sym=TRUE)
```

Now the code to produce a graph for the toy example is given below.

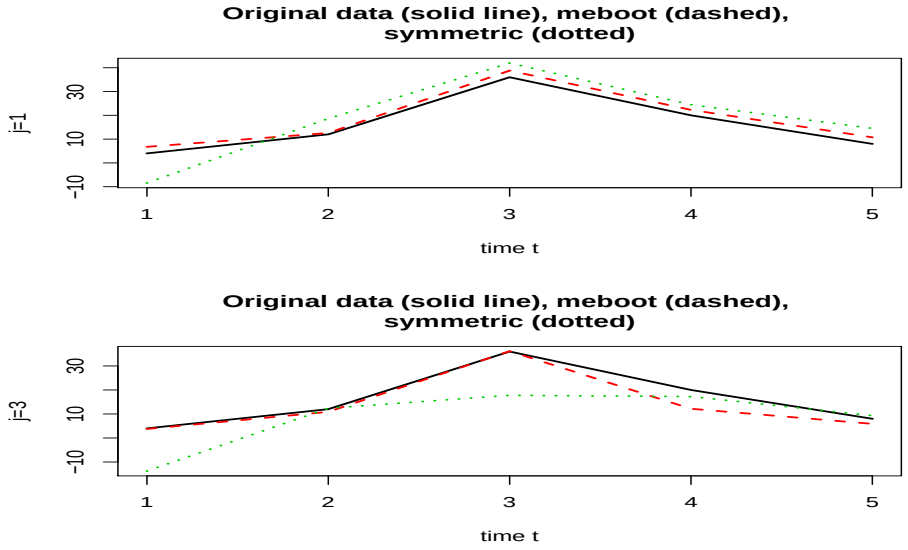
```
par(mfrow=c(2,1))
my3=cbind(x, out1$ens[,1], out3$ens[,1])
matplot(my3, typ="l",xlab="time t", ylab="j=1", lwd=2)
title("Original data (solid line), meboot (dashed),
symmetric (dotted)")
my3=cbind(x, out1$ens[,3], out3$ens[,3])
matplot(my3, typ="l",xlab="time t", ylab="j=3", lwd=2)
title("Original data (solid line), meboot (dashed),
symmetric (dotted)")
```

Figure 3 plots the x_t of the toy example (solid line) along with two realizations $j = 1, j = 3$ respectively in the upper and lower panels. The dashed line refers to the usual meboot resamples (without any enhancement) and dotted line refers to its symmetric version in both panels.

In conclusion, it is not obvious exactly when the enhancements suggested in this note are needed and worthwhile. Of course, it would be interesting to check them in the context of simulation designs in Vinod (2012) and Vinod (2010) where they already support traditional meboot. The enhancements do seem to be intuitively useful in many situations. I invite readers to send me applications where traditional meboot provides too narrow confidence intervals, where the rescaling enhancement may be worth a try. Also, when the sampling distribution of the statistic under study is known to be symmetric, the symmetrizing enhancement is worth a try.

In one large experiment involving inference regarding the standard errors of regression coefficients, I found that the symmetric version of meboot continues to outperform the moving block bootstrap. Simulation details are omitted here since the experiment requires too much space to describe and does not explicitly compare the symmetrizing enhancement with the traditional bootstrap. While I am planning to perform such simulations in the future, my purpose here to provide usable code to potential users.

Figure 3: Resamples from toy example meboot and its symmetric version



.

References

Theil, H. (1980), "The Symmetric Maximum Entropy Distribution," *Economics Letters*, 6, 53–57.

Theil, H. and Laitinen, K. (1980), "Singular Moment Matrices in Applied Econometrics," in "Multivariate Analysis – V," , ed. Krishnaiah, P., New York, USA: North USA-Holland Publishing Co., pp. 629–649.

Vinod, H. D. (2006), "Maximum entropy ensembles for time series inference in economics," *Journal of Asian Economics*, 17(6), 955–978.

— (2008), *Hands-on Intermediate Econometrics Using R: Templates for Extending Dozens of Practical Examples*, Hackensack, NJ: World Scientific, iSBN 10-981-281-885-5, URL `http://www.worldscibooks.com/economics/6895.html`.

— (2010), "New Solution to Time Series Inference in Spurious Regression Problems," *SSRN eLibrary*, URL `http://ssrn.com/paper=1560074`.

— (2012), "Constructing Scenarios of Time Heterogeneous Series for Stress Testing," *SSRN eLibrary*, URL `http://ssrn.com/paper=1987879`.

Vinod, H. D. and López-de-Lacalle, J. (2009), "Maximum Entropy Bootstrap for Time Series: The meboot R Package," *Journal of Statistical Software*, 29, 1–19, URL `http://www.jstatsoft.org/v29/i05/`.