Maximum Entropy Bootstrap for Time Series: Toy Example Exposition

Hrishikesh D. Vinod
Fordham University

April 2, 2023

Toy Example

The Maximum Entropy Bootstrap is illustrated with a small example. Let the sequence $x_t = (4, 12, 36, 20, 8)$ be the series of data observed from the period $t = 1$ to $t = 5$ as indicated in the first two columns in Table 1. We jointly sort these two columns on the second column and place the result in the next two columns (Table 2 columns 3 and 4), giving us the ordering index vector in column 3.

Next, the four intermediate points in Column 5 are seen to be simple averages of consecutive order statistics. We need two more (limiting) "intermediate" points. These are obtained as described in Step 3 above. Using 10% trimming, the limiting intermediate values are $z_0 = -11$ and $z_T = 51$. With these six z_t values we build our five half open intervals:

$$U(-11, 6] \times U(6, 10] \times U(10, 16] \times U(16, 28] \times U(28, 51]$$

The maximum entropy density of the ME bootstrap is defined as the combination of T uniform densities defined over (the support of) T half open intervals.

Time	x_t	Ordering vector	Sorted x_t	Intermediate points	Desired means	Uniform draws	Preliminary values	Final replicate
1	4	1	4	6	5	0.12	5.85	5.85
2	12	5	8	10	8	0.83	6.70	13.90
3	36	2	12	16	13	0.53	13.90	23.95
4	20	4	20	28	22	0.59	15.70	15.70
5	8	3	36		32	0.11	23.95	6.70

Table 1: Example of the ME bootstrap algorithm.

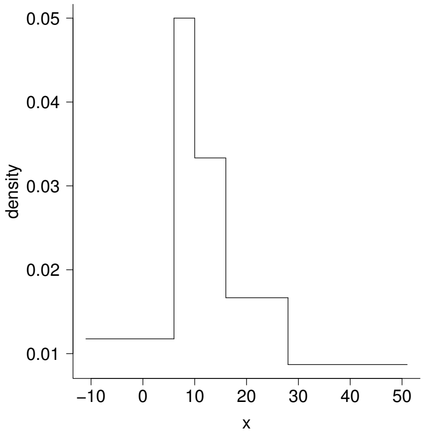

Figure 1: Maximum entropy density for the $x_t = 4, 12, 36, 20, 8$ example.

```
> xx <- c(4,12,36,20,8) #original time series up and down shape
> trimprop <- 0.10 #trimming proportion
> reachbnd <- FALSE #reaching the bound of the range forced or  not?
> # uniform draws used as an example in Table 1 of the paper
>
> p <- c(0.12, 0.83, 0.53, 0.59, 0.11)
> n <- length(xx)
> x <- sort(xx)
> ordxx <- sort(xx, index.return=TRUE)
> print(c("ordxx=",ordxx)) #without the dollar ix appending
```

```
[[1]]
[1] "ordxx="

$x
[1]  4  8 12 20 36

$ix
[1] 1 5 2 4 3
```

```
> print(c("ordxx$ix=",ordxx$ix))
```

```
[1] "ordxx$ix=" "1"         "5"         "2"         "4"         "3"
```

Index Return = TRUE using sort command

The above use of the sort command with index.return=TRUE is worth learning. It will be used later to map from numerical magnitudes (values) domain to the time domain. The use of sort with option index.return=TRUE allows us to avoid explicit use of sorting on two columns of data.

```
> x <- sort(xx)
> x #sorted magnitudes original xx data in values domain


[1]  4  8 12 20 36


> #embed good for getting a matrix with lagged values in the second column
> embed(1:4,2) #allows no worry about missing values with lags


     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]    4    3


> embed(x, 2) #apply embed to our sorted xx


     [,1] [,2]
[1,]    8    4
[2,]   12    8
[3,]   20   12
[4,]   36   20


> z <- rowMeans(embed(x, 2))
> z #these are intermediate values


[1]  6 10 16 28


> dv <- abs(diff(xx))
> dv #vector of absolute differences


[1]  8 24 16 12


> dvtrim <- mean(dv, trim=trimprop)
> dvtrim #trimmed mean of dv
```

```
[1] 15
```

```
> xmin <- x[1]-dvtrim
> xmax <- x[n]+dvtrim
> xmin #ultimate minimum for resampled data gives z_0
```

```
[1] -11
```

```
> xmax #ultimate maximum for resampled data gives z_T
```

```
[1] 51
```

embed command

R function embed Embeds the time series x into a low-dimensional Euclidean space. We are using dimension=2 here. It gives lagged values in second column of a matrix without worrying about missing values.

dv denotes the absolute difference between consecutive sorted values.

z denotes intermediate values needed for defining half-open intervals $I_t = (z_{(t-1)}, z_t]$.

Unfortunately, the xmin ($z_0 = -11$) and xmax ($z_T = 51$) do not appear in the published Table.

Mass and Mean Preserving Constraints satisfy ergodic theorem

A fraction $1/T$ of the mass of the probability distribution must lie in each interval. meboot requires each half open interval I_t to have an equal chance being included in the resample.

$\Sigma x_t = \Sigma x_{(t)} = \Sigma m_t$, where m_t denote the mean of $f(x)$ within the interval I_t. **mean preserving** constraint.

$$f(x) = 1/(z_1 - z_0), \quad x \in I_{(1)}, \quad m_1 = 0.75x_{(1)} + 0.25x_{(2)}, \tag{1a}$$

$$f(x) = 1/(z_k - z_{k-1}), \quad x \in (z_k - z_{k-1}], \tag{1b}$$

$$\text{with mean} \quad m_k = 0.25x_{(k-1)} + 0.50x_{(k)} + 0.25x_{(k+1)} \tag{1c}$$

$$\text{for} \quad k = 2, \dots, T-1, \tag{1d}$$

$$f(x) = 1/(z_T - z_{T-1}), \quad x \in I_{(T)}, \quad m_T = 0.25x_{(T-1)} + 0.75x_{(T)}. \tag{1e}$$

The weights for the two observations at the left end interval are (0.75, 0.25)

Note that the weights are (0.25, 0.50, and 0.25) for all intermediate intervals We have $T = 5$ here leading to three intervals needing these weights.

The weights for the two observations at the right end interval are (0.25, 0.75)

Properties of uniform density

If the range of continuous uniform random variable are `a` to `b` the density of uniform is f(1/(b-a))

Mean of uniform is (a+b)/2

We have used maximum entropy principle to say that the densities between the intermediate points z_0 to z_T are all uniform.

The desired means for our toy example with order `stats`=(4,8,12,20,36) are (6+10)/2=8, (10+16)/2=13, (16+28)/2=22 for intermediate intervals

For the left extreme we use $0.75 * x_{(1)} + 0.25 * x_{(2)}$=0.75*4+0.25*8=5

For the right extreme interval desired mean is $0.25 * x_{(T-1)} + 0.75 * x_{(T)}$ 0.25*20+0.75*36=32

see last table column entitled `desintxb` with entries (5,8,13,22,32)

`embed` helps achieve desired means m_t of the T intervals

It is worth learning how three dimensional `embed` function of R works with this toy example where we are considering mean of 3 consecutive values, except for the two intervals at the two ends of the series.

```
> embed(1:5,3) #embeding 1:5 gives 3 by 3 matrix
```

```
     [,1] [,2] [,3]
[1,]    3    2    1
[2,]    4    3    2
[3,]    5    4    3
```

```
> #Note j-th column has lag=j-1 values.  Col.2 has lag 1
> #Note embed retains only non-missing lag values
> x
```

```
[1]  4  8 12 20 36
```

```
> t(embed(x,3))# transpose embed matrix
```

```
     [,1] [,2] [,3]
[1,]   12   20   36
[2,]    8   12   20
[3,]    4    8   12
```

```
> t(embed(x, 3))*c(0.25,0.5,0.25) #multiply by weights
```

```
     [,1] [,2] [,3]
[1,]    3    5    9
[2,]    4    6   10
[3,]    1    2    3

> t(t(embed(x, 3))*c(0.25,0.5,0.25)) #transpose twice to get back

     [,1] [,2] [,3]
[1,]    3    4    1
[2,]    5    6    2
[3,]    9   10    3
```

Next we compute the row sum of above and call it a vector `aux`. This applies to intermediate intervals not the extreme intervals a the bottom end and at the top end, where one needs to average only two intermediate values with weights 0.75 and 0.25.

```
> aux <- rowSums( t( t(embed(x, 3))*c(0.25,0.5,0.25) ) )
> aux #these are only 3

[1]  8 13 22

> #append the means of two extreme intervals at the two ends
> desintxb <- c(0.75*x[1]+0.25*x[2], aux, 0.25*x[n-1]+0.75*x[n])
> desintxb# des=desired, int=interval, xb=xbar=means

[1]  5  8 13 22 32

> desintxb  #desired means  5  8 13 22 32,  Now 5 as desired

[1]  5  8 13 22 32

> print( "mean(xx),mean(desintxb),mean(x)") #all=16

[1] "mean(xx),mean(desintxb),mean(x)"

> print(c(mean(xx),mean(desintxb), mean(x)))

[1] 16 16 16
```

The above shows that the **mean preserving constraint** is satisfied, since the mean of data and mean of desired means equal the same number 16. This is no accident, but achieved by designed weights which force the desired means of each interval to be based on the x_t data. This helps ensure that the ergodic theorem is satisfied by our resamples.

Drawing random quantile $q_t \in [z_0, z_T]$ from empirical cumulative ME density$\in [0, 1]$

Given empirical cdf of ME density consisting of uniform patches, we just draw 999 realizations of iid uniform in the values domain. For example, a random draw of uniform between 0 to 1 illustrated in the toy example is:

p=c(0.12, 0.83, 0.53, 0.59, 0.11)

I wish I had included an additional column for sorted uniform draws pp=(0.11, 0.12, 0.53, 0.59, 0.83) in the published paper for clearer exposition. A complete table is included toward the end of this document.

```
> q=rep(0,n) #place holder for q
> pp=sort(p) #sorted random draws
> print(c("sorted random draws", pp))
```

```
[1] "sorted random draws" "0.11"                "0.12"
[4] "0.53"                "0.59"                "0.83"
```

In traditional iid bootstrap each x_t has 1/T (if we have T observations) chance of being included in the resample. Of course, some x_t might repeat and some may not be present in some individual realizations of the random draws from the uniform density. Imposing similar requirement in meboot algorithm is called satisfying **mass preserving constraint** in the paper.

If the uniform random variable is defined over the range [a,b], then the its mean is (a+b)/2. The first interval is $(-11, 6]$ with width 17 and mean $-5/2$. Since we have T=5 observations in the toy example, each interval should have 1/5 =0.2 probability of being included in the resample.

The sorted random draws are (0.11, 0.12, 0.53, 0.59, 0.83). the numbers having the pp values less than 0.2 (=1/n or 1/T) are two numbers: 0.11 and 0.12. First we use the approx function to interpolate in I_1 interval to get the corresponding two interpolated value qq$=-1.65, -0.8$. These do **not** satisfy the mean preserving constraint. They need to be adjusted by adding the adjustment 7.5 (calculated above) to yield 5.85 and 6.7 as the two quantiles of the ME density associated with the first two sorted random draws 0.11 and 0.12.

```
> z[1] #first intermediate value
```

```
[1] 6
```

```
> xmin #smallest
```

```
[1] -11
```

```
> 0.5*(z[1]+xmin) #average for the first interval

[1] -2.5

> desintxb[1] #des=desired, int=interval, xb=xbar=mean

[1] 5

> desintxb[1]-0.5*(z[1]+xmin)#adjustment for first interval

[1] 7.5
```

Thus the first interval adjustment 7.5 must be added so that the mean equals the desired value so that eventually we satisfy mean preserving constraint.

Now we turn to random draw(s) which happen to be less than or equal to (1/T=1/5), which will come from the first half open interval $I_1 = (z_0, z_1]$.

R commands `approx` (linear interpolate) and `which`

```
> ref1 <- which(pp <= (1/n)) #how many are less than or equal to 1/5 if n=T=5
> ref1

[1] 1 2

> # approx. returns list of points which linearly interpolate given data points,
> #first interval ref1
>
> if(length(ref1)>0){
+   qq <- approx(c(0,1/n), c(xmin,z[1]), pp[ref1])$y
+   qq #interpolated values
+ adj= desintxb[1]-0.5*(z[1]+xmin)
+ print(c("qq=",qq,"adj=",adj))
+   q[ref1] <- qq
+   if(!reachbnd)  q[ref1] <- qq + desintxb[1]-0.5*(z[1]+xmin)
+ }

[1] "qq="                "-1.65"              "-0.800000000000001"
[4] "adj="               "7.5"

> print(c("qq=",qq))

[1] "qq="                "-1.65"              "-0.800000000000001"

> print(c("q",q))

[1] "q"    "5.85" "6.7"  "0"    "0"    "0"
```

Second, Third and Fourth intervals

In our example sorted random draws are pp =(0.11 0.12 0.53 0.59 0.83) and the relevant range limits are (0, 0.2, 0.4, 0.6, 0.8, 1.0). Clearly the first two pp values are in the first interval 0 to 0.2 discussed above.

The second interval is $I_2 = (6, 10]$ with width 4 and mean 8 for sorted pp in (1/T, 2/T]. None of our pp=(0.11 0.12 0.53 0.59 0.83) is between 0.2 and 0.4.

Two pp values 0.53 and 0.59 are both in the range 0.4 to 0.6 from which there is no random draw. The next second range of probabilities is 0.2 to 0.4 and no draw in the range 06 to 0.8.

The third interval is $I_3 = (10, 16]$ with width 6 and mean 13 for sorted pp in (2/T, 3/T]. After interpolation and adjustment, corresponding two quantile values are 13.9 and 15.7.

The fourth interval is $(16, 28]$ with width 12 and mean 22 for sorted pp in (3/T, 4/T]. No random draw here.

```
> for(i1 in 1:(n-2)){
+   ref2 <- which(pp > (i1/n))
+ print(c("ref2",ref2,"pp[ref2]", pp[ref2]))
+   ref3 <- which(pp <= ((i1+1)/n))
+ print(c("ref3",ref3))
+   ref23 <- intersect(ref2, ref3)
+ print(c("ref23",ref23,"sorted draw pp[ref23]=",pp[ref23]))
+
+   if(length(ref23)>0){
+     qq <- approx(c(i1/n,(i1+1)/n), c(z[i1], z[i1+1]), pp[ref23])$y
+ print(c("interpolated value qq=",qq))
+ adj= desintxb[-1][i1]-0.5*(z[i1]+z[i1+1])
+ print(c("qq=",qq,"adj=",adj))
+     q[ref23] <- qq + desintxb[-1][i1]-0.5*(z[i1]+z[i1+1])
+ print(c("q",q))
+   }
+ }
```

```
[1] "ref2"     "3"        "4"        "5"        "pp[ref2]" "0.53"     "0.59"
[8] "0.83"
[1] "ref3" "1"    "2"
[1] "ref23"                  "sorted draw pp[ref23]="
[1] "ref2"     "3"        "4"        "5"        "pp[ref2]" "0.53"     "0.59"
[8] "0.83"
[1] "ref3" "1"    "2"    "3"    "4"
[1] "ref23"                  "3"                      "4"
[4] "sorted draw pp[ref23]=" "0.53"                   "0.59"
[1] "interpolated value qq=" "13.9"                   "15.7"
[1] "qq="  "13.9" "15.7" "adj=" "0"
[1] "q"    "5.85" "6.7"  "13.9" "15.7" "0"
```

```
[1] "ref2"     "5"        "pp[ref2]" "0.83"
[1] "ref3" "1"    "2"    "3"    "4"
[1] "ref23"                  "sorted draw pp[ref23]="
```

We find that the adjustment to interpolated value above is zero.

Interval called `ref4` if `pp` exactly equals 4/5 (n-1)/n is empty.

```
> ref4 <- which(pp == ((n-1)/n))
> print(c("ref4", ref4))
```

```
[1] "ref4"
```

```
> if(length(ref4)>0)
+   q[ref4] <- z[n-1]
> q
```

```
[1]  5.85  6.70 13.90 15.70  0.00
```

Last interval, fifth here, is called `ref5`

Note that the last interval interpolated value `qq` is 31.45 and we adjust it by `adj=-7.5` to yield 23.95. Recall that the adjustment is designed to ensure the **ergodic theorem** is numerically satisfied by the meboot algorithm.

```
> ref5 <- which(pp > ((n-1)/n))
> if(length(ref5)>0){
+ print(c("ref5",ref5,"pp[ref5]=",pp[ref5]))
+   qq <- approx(c((n-1)/n,1), c(z[n-1],xmax), pp[ref5])$y
+ print(c("interpolated value qq in last interval",qq))
+   q[ref5] <- qq   # this implicitly shifts xmax for algorithm
+ adj=desintxb[n]-0.5*(z[n-1]+xmax)
+ print(c("qq=",qq,"adj=",adj))
+
+   if(!reachbnd)  q[ref5] <- qq + desintxb[n]-0.5*(z[n-1]+xmax)
+ }
```

```
[1] "ref5"      "5"         "pp[ref5]=" "0.83"
[1] "interpolated value qq in last interval"
[2] "31.45"
[1] "qq="   "31.45" "adj="  "-7.5"
```

Now wrap up the entire calculation of meboot for toy example

Following code maps the q vector from values domain to the time domain by using the sort function with the option index.return=TRUE noted above.

We set q[ordxx$ix] as sorted q denoted by qseq in the values domain.

```
> prel=q #preliminary quantile values
> qseq <- sort(q)
> print(c("sorted q",qseq))


[1] "sorted q" "5.85"     "6.7"      "13.9"     "15.7"     "23.95"


> q[ordxx$ix] <- qseq
> print(c("after mapping to time domain",q))


[1] "after mapping to time domain" "5.85"
[3] "13.9"                         "23.95"
[5] "15.7"                         "6.7"


> print(q)


[1]  5.85 13.90 23.95 15.70  6.70
```

Now we produce the table.

```
> Tim=1:5
> xt=xx #notation xt for original data
> xordstat=x #order stats
> ord1=ordxx$ix #output of sort
> intermed=c(z,xmax)  #these are zt
> prel


[1]  5.85  6.70 13.90 15.70 23.95


> qseq #sorted quantiles


[1]  5.85  6.70 13.90 15.70 23.95


> final=q #final quantiles of ME density
> cb=cbind(Tim,xt,xordstat,ord1,intermed,desintxb, p,pp,prel,final)
> require(xtable)
> options(xtable.comment = FALSE)
> print(xtable(cb))
```

	Tim	xt	xordstat	ord1	intermed	desintxb	p	pp	prel	final
1	1.00	4.00	4.00	1.00	6.00	5.00	0.12	0.11	5.85	5.85
2	2.00	12.00	8.00	5.00	10.00	8.00	0.83	0.12	6.70	13.90
3	3.00	36.00	12.00	2.00	16.00	13.00	0.53	0.53	13.90	23.95
4	4.00	20.00	20.00	4.00	28.00	22.00	0.59	0.59	15.70	15.70
5	5.00	8.00	36.00	3.00	51.00	32.00	0.11	0.83	23.95	6.70

I thank Fred Viole, director of the consulting firm OVVO Financial Systems specializing in analysis of stock market data for vastly improving an earlier draft version of this vignette.