ESTIMATING SHANNON ENTROPY

Let's be interested in guess the Shannon entropy from a sample drawn from a population of one continuous random variable.[Beirlant2001.],[Duncan2004],[Brissaud2005]

This estimation can be accomplished with a variation of the Box-Counting algorithm. This algorithm is a standard for the estimation of the fractal dimension of phenomena [TÉL1989],[Saa2007],[Lopes2009],[Hausser2009].

In short, hopefully

$$Sh = D_1 \ln(d_N) + H0 \tag{1}$$

holds on some range of d_N, where:

d_N, is the size of the bins when we took N bins over the sample. That is $d_N = \frac{Max(sample) - min(sample)}{N}$

Sh is the Shannon's entropy estimation for N bins. That is $Sh = \sum p_i \ln(p_i)$ and p_i is the relative frequency of the bin *i.*

It is direct that *Sh=H0* for d_N*=1*, that is for the measure in the unitary scale.

SIMULATIONS

All simulations was carried out with R version 3.0.1 (2013-05-16) -- "Good Sport" [R] and Rstudio Version 0.97.551 [RStudio].

All the described code are in the R scripts *ebc.R*[1] and *ebc_demo.R*[2]

Let's draw a sample of size 100 from a normal population with mean 0 and entropy 3.5 nats and test it with all methods available in the entropy package [entropy].

```
par(mfrow=c(3,3))

sample=get_sample(N=100,dist='normal',Sh=3.5,okgraph=T)


Sh=c()

for (met in
     c('ML','MM','Jeffreys','Laplace','SG','minimax','CS','shrink')){

    Sh=append(Sh,ebc_sample(sample,method=met,
     bins=set_bins('dyadic',1e5),okplot=T,npts=6))

}

summary(Sh)
```

This results in :

[1] https://docs.google.com/file/d/0B6ZuqpeSKSqcWFlCN2hqT2tGSEk/edit?usp=sharing

[2] https://docs.google.com/file/d/0B6ZuqpeSKSqcWXl6OFp6c1JrNHM/edit?usp=sharing

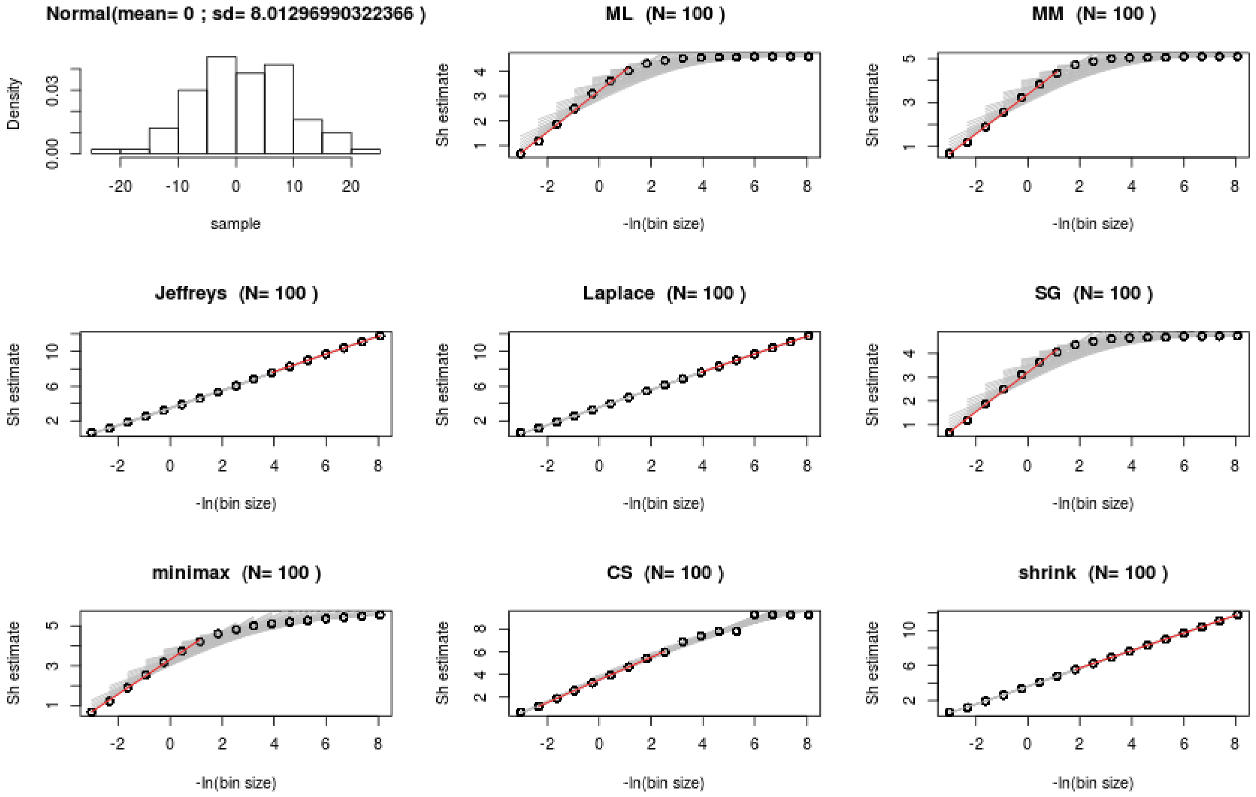


Ilustración 1: A sample (upper left) and the result of the application of the ebc algorithm. The size of the sample was 100. The red lines represent the linear fit selected from the models tried (gray lines). The Shannon's entropy estimation is the interception of the model selected. It's worth to say that ln(100)= 4.6052 and it seems to be a limit for ML, MM, SG and minimax methods an in some extension for CS method.

Table 1: Shannon's entropy estimation with different methods from a sample with H0=3.5

ML	MM	Jeffreys	Laplace	SG	minimax	CS	shrink
3.19537	3.37812	3.62403	3.68405	3.21396	3.31513	3.50257	3.70253

An estimate about the method performance would be done with the code

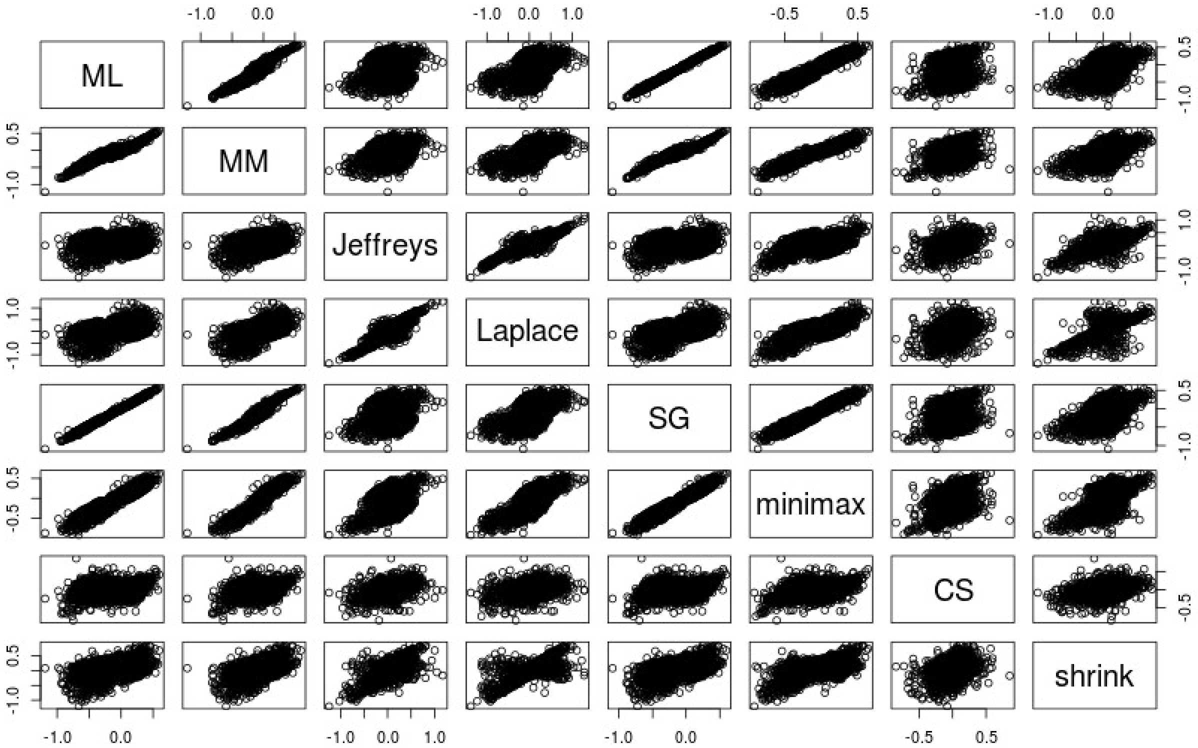
```
global=data.frame()
#entropy estimation methods to test
methods= c('ML','MM','Jeffreys','Laplace','SG','minimax','CS','shrink')
#distributions to test
distributions=c('normal','exp','uniform')
#dyadic bins for algorithm
bins=set_bins('dyadic')
#sample size to test
ext=c(30,50,75,100,1000)
#entropies to test
Sh_ref=c(-0.88,0.01,1.577, 2.153, 3.3682, 3.7181)
```

```
par(mfrow=c(2,3))
par(oma=c(0,0,3,0))
#Simulation

for (dst in distributions){
    for (Sh in rep(Sh_ref,4)){

        for (N in rep(ext,4)){

            sample=get_sample(N,dist=dst,Sh=Sh)
            r=c()
            for (met in methods){

     r=append(r,ebc_sample(sample,bins=bins,method=met,okplot=F)[1])
            }
            results=data.frame(dist=dst,N=N,H0=Sh,rbind(r))
            global=rbind(global,results)

        }
    }
    boxplot(global[,4:ncol(global)]-global[,3],ylab='dSh')
}

names(global)=c('dist','N','H0',methods)
global_0=global

perf=global_0
v_met=c(4:(ncol(perf)))
perf[,v_met]=perf[,v_met]-perf$H0
perf[abs(perf$H0)>0.5,v_met]=perf[abs(perf$H0)>0.5,v_met]/perf[abs(perf$H
     0)>0.5,3]
```

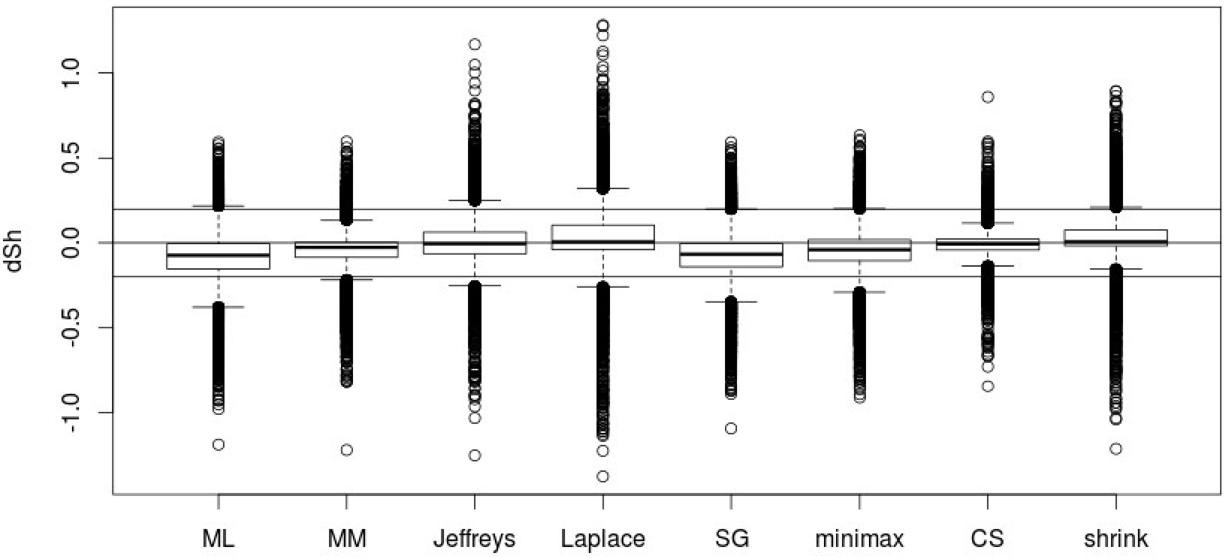
The dataframe *perf* contains the *dSh* for each method and for each combination of parameters: *N*, the sample size, *H0* the Shannon entropy of the sample and *dist* the distribution of the sample. $dSh=\begin{cases} \frac{Sh-H0}{H0} & if\,|H0|>0.5 \\ Sh-H0 & otherwise \end{cases}$,that is the relative error.

Running the previous code until achieve 112 observation per combination of parameters it yields a total of 10080 observations[3].

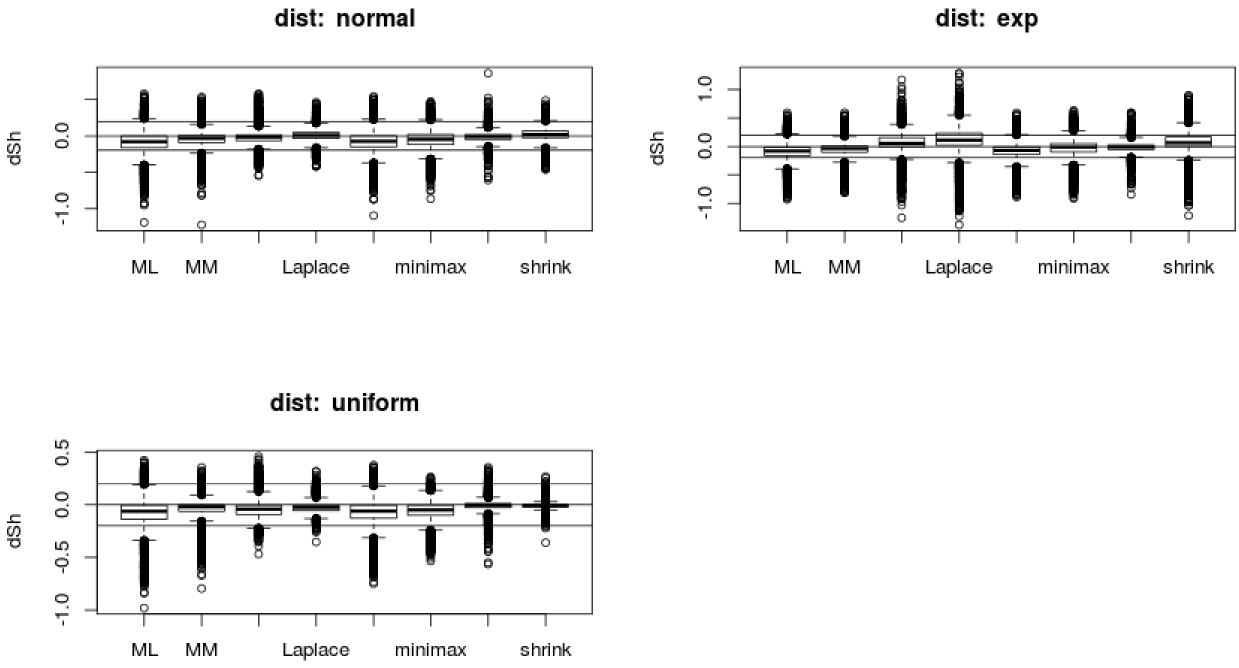
[3] https://docs.google.com/file/d/0B6ZuqpeSKSqcaWp5VVhpOVVKcEU/edit?usp=sharing



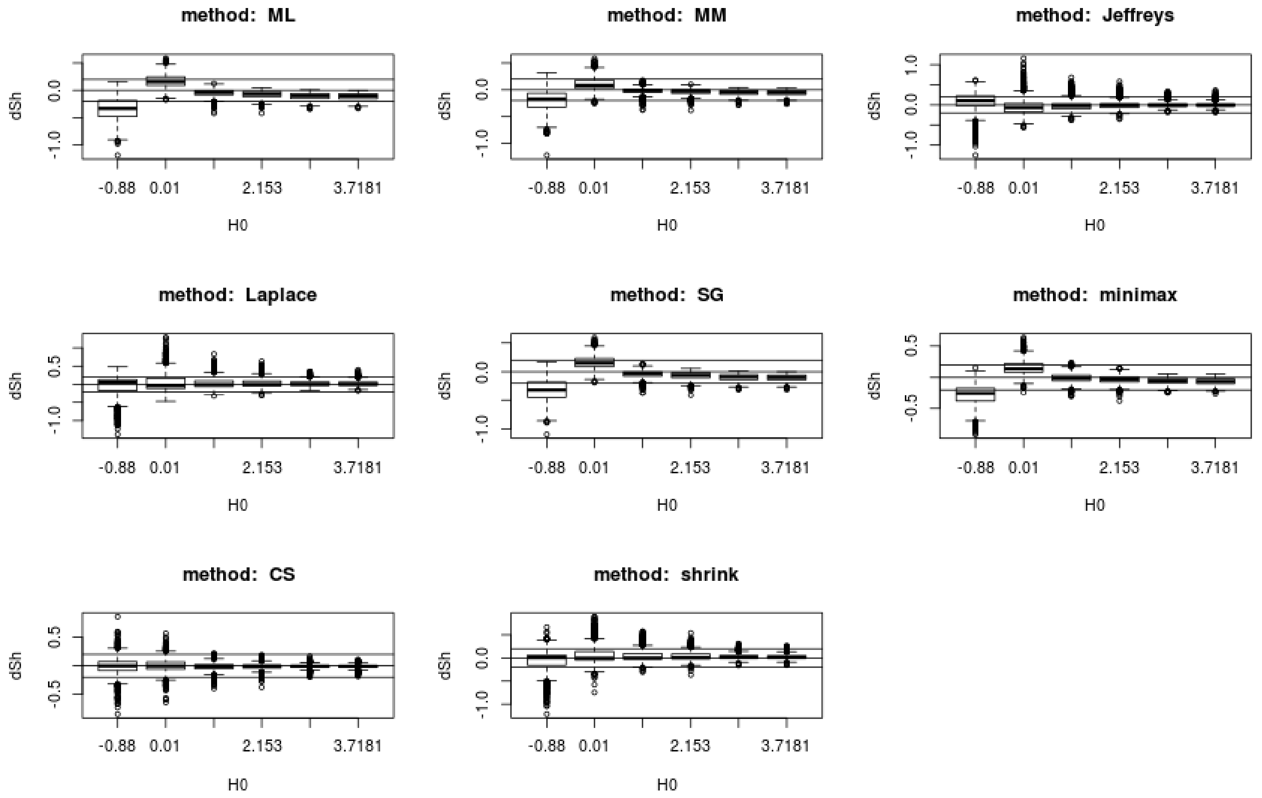
Ilustration 1: Regression plot between methods.



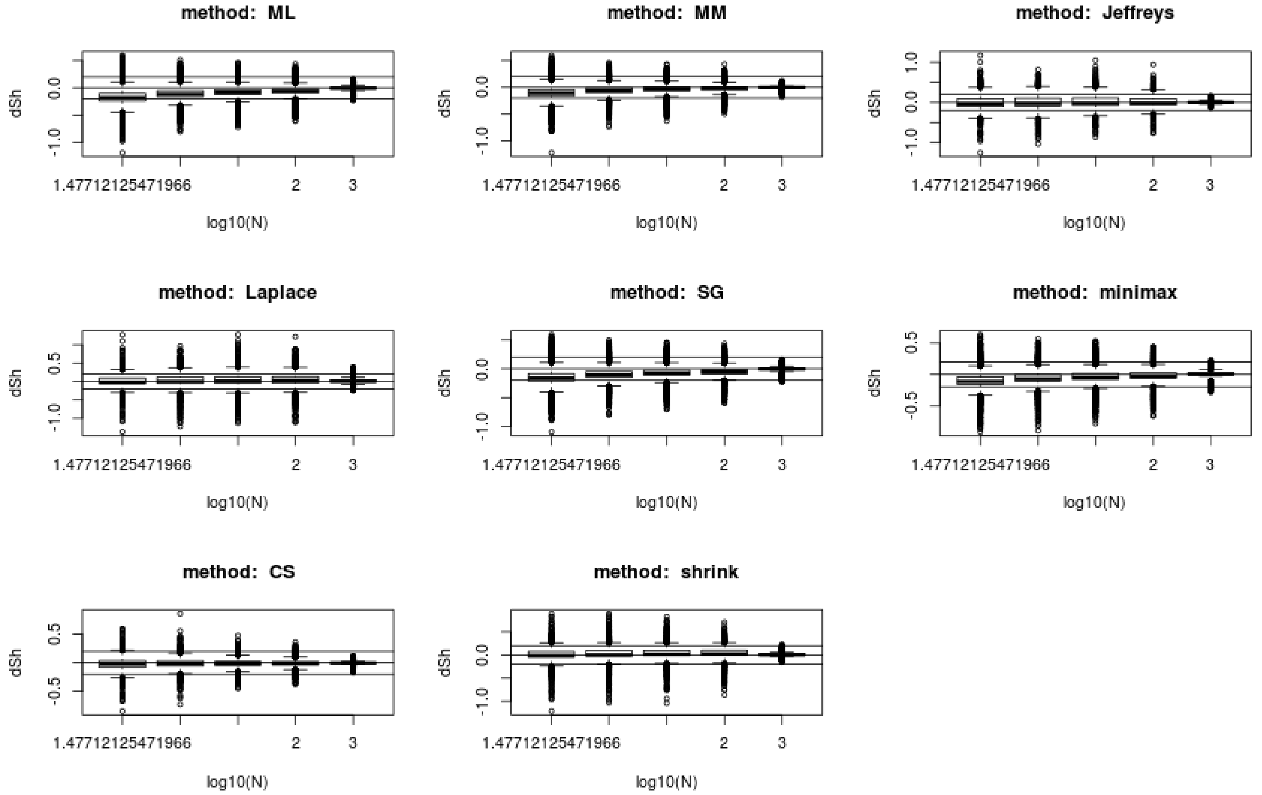
Ilustration 2: Global performance. The horizontal lines at {-0.2;0.2}



Ilustration 3: Performance vs. distribution



Ilustration 4: Performance vs. H0



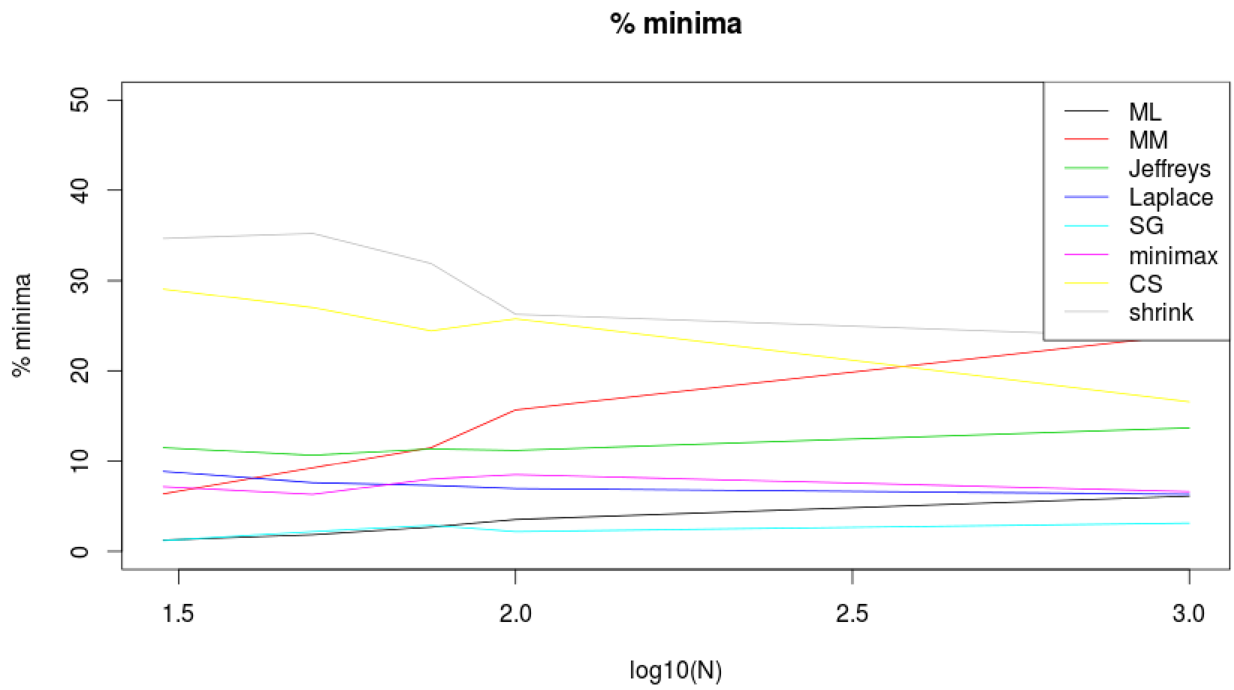
Ilustration 5: Performance vs. log10(N)

Table 2: Counting of times each method is the best. A method is the best in a row if its |dSh| is the minimum of the row.

Method	ML	MM	Jeffreys	Laplace	SG	minimax	CS	shrink
Times meth. is best	310	1346	1178	745	233	736	2475	3057
Prop. meth. is best	0.0308	0.1335	0.1169	0.0739	0.0231	0.0730	0.2455	0.3033

Table 3: Proportion times each method is best vs. sample size

N	ML	MM	Jeffreys	Laplace	SG	minimax	CS	shrink
30	0.0124	0.0635	0.1151	0.0883	0.0119	0.0714	0.2907	0.3467
50	0.0184	0.0923	0.1066	0.0759	0.0218	0.0630	0.2698	0.3522
75	0.0268	0.1151	0.1136	0.0729	0.0288	0.0799	0.2440	0.3189
100	0.0352	0.1567	0.1121	0.0694	0.0218	0.0848	0.2574	0.2624
1000	0.0610	0.2401	0.1369	0.0630	0.0313	0.0660	0.1657	0.2361



Ilustration 6: % times a method is the best vs. log10(sample size)

So it seems the most useful methods are MM, CS and shrink.

MIXED SAMPLES

Suppose two populations *Q* and *R* represented by continuous random variables. These populations have entropies H_Q and H_R. Suppose the samples *q*, of size m_q and *r* of size m_r. If we took *N* bins over the samples, and q_i is the number of elements of *q* in the i-th bin (the mass of *q* in this bin), then $\sum^{N} q_i = m_q$ and $H_Q = \ln(m_q) - \frac{1}{m_q} \sum^{N} q_i \ln(q_i)$

Let's create a new sample *p* as the union of *q* and *r*. So $m_p = m_q + m_r$ and $H_P = \ln(m_q + m_r) - \frac{1}{m_q + m_r} \sum^{N} p_i \ln(p_i)$

The easy case for the relationship between p_i , q_i and r_i is when *q* and *r* are disjoint. That is when all the elements in the *i* bin of *p* are from *q* or *r* but not from both. In this case we can write $H_p = \ln(m_q + m_r) - \frac{1}{m_q + m_r} \left(\sum^{N_q} q_j \ln(q_j) + \sum^{N_r} r_k \ln(r_k) \right)$ Let's define $\tau = \frac{m_q}{m_r}$ and $\theta = \frac{\tau}{\tau + 1}$ and with some algebra we get

$$H_{NO} = \frac{m_q H_q + m_r H_r}{m_q + m_r} + \ln\left(\frac{\tau + 1}{\tau^{\theta}} \right) \tag{2}$$

Taking a look to the more general case of two overlapping samples, we'll arrive to

$$H_p = H_{NO} + \sigma \tag{3}$$

Where H_{NO} is the entropy when there is no overlapping, as equation (2), and σ is an overlapping factor

$$\sigma = \sum_{O_{min}}^{O_{max}} f_i \ln(\eta_i) \tag{4}$$

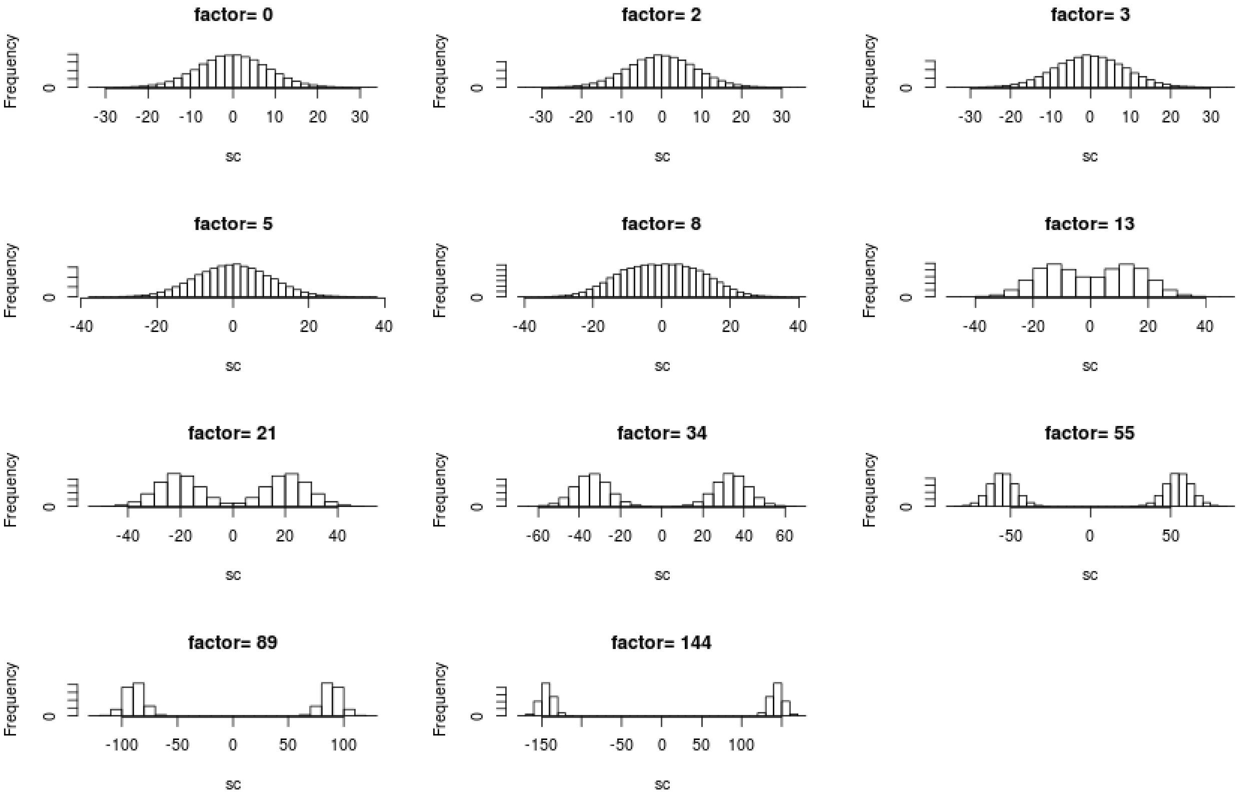
Where f_i is the relative frequency of the i-th bin, O_{min}, O_{max} are the lowest and the highest bin numbers that defines the overlapping region, and $\eta_i = \frac{q_i}{m_i}$ is the proportion of the i-th bin mass that comes from the *q* sample.

Simulate the process of mixing to samples from populations iid is easy if we take a sample, *a* and a factor, *f*. We can make up the mixed sample as the union $\{a+f\} \cup \{a-f\}$ In this case $\tau = 1$, $\theta = \frac{1}{2}$ and $H_q = H_r = H0$ so $H_{NO} = H0 + \ln(2)$

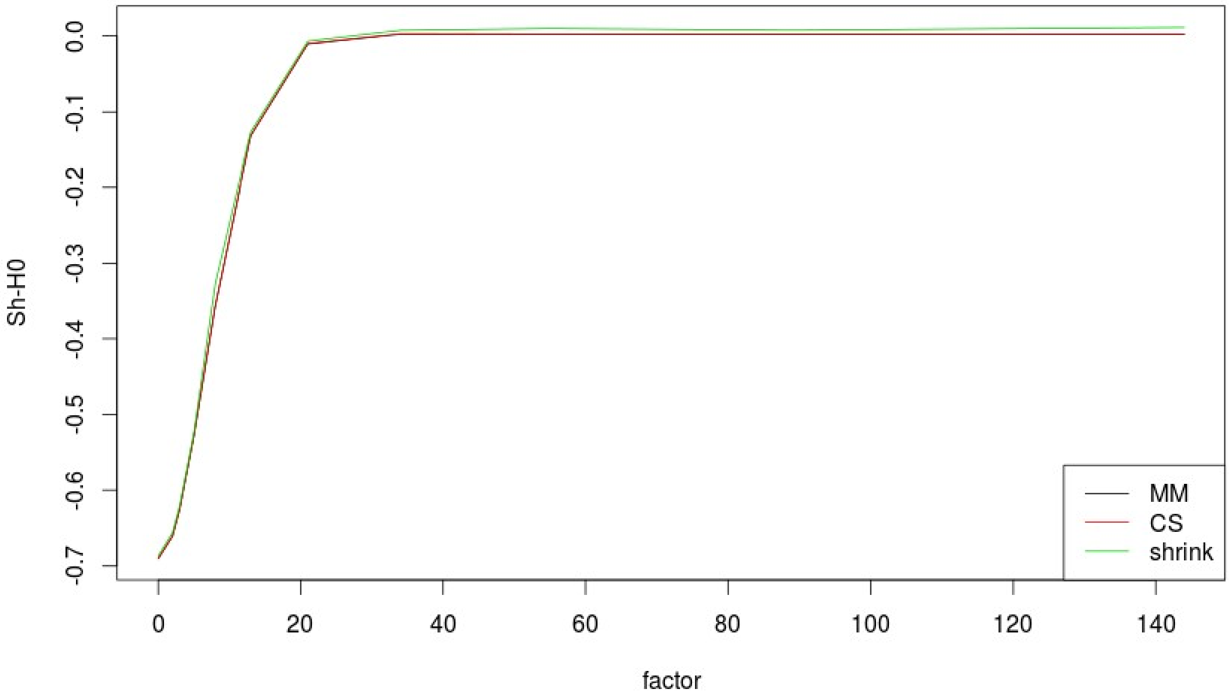
This is accomplished with the following code:

```
methods=c('MM','CS','shrink')
par(mfrow=c(4,3))
H0=3.5
dist='normal'
```

```
p1=50
base=1e5
factor=set_bins('fib',100)
res=data.frame()
sa=get_sample(base,dist=dist,Sh=H0)
sb=get_sample(base,dist=dist,Sh=H0)
for (f in append(0,factor)){
    sc=append((sa-f),(sa+f))
    hist(sc,breaks=30,main=paste('factor=',f))
    H=H0+log(2)
    print(paste('H_esp=',H))
    v=c()
    for (met in methods){
        a=ebc_sample(sc,method=met,okplot=F)
        v=append(v,a[1])
    }
    res=rbind(res,data.frame(factor=f,H0=H,rbind(v)))

}
names(res)[3:5]=methods
res[,3:5]=res[,3:5]-res$H0
par(mfrow=c(1,1))
plot(res$factor,res$factor,type='l',col='white',ylim=c(min(res[,3:5]),
     (max(res[,3:5]))),xlab='factor',ylab='Sh-H0')
lines((res$MM)~res$factor,col=1)
lines((res$CS)~res$factor,col=2)
lines((res$shrink)~res$factor,col=3)
legend(x='bottomright',legend=names(res)[3:5],col=c(1,2,3),lty=1)
summary(res[,3:5])
res
```



Ilustration 7: The mixing process as described in the text



Ilustration 8:estimation of the overlapping factor, as described in the text,

Tabla 4: Values of Sh-H0 for the mixing process

factor	H0	MM-H0	CS-H0	Shrink-H0
0	4.19315	-0.69024	-0.69033	-0.68663
2	4.19315	-0.66078	-0.66069	-0.65608
3	4.19315	-0.62542	-0.62539	-0.62078
5	4.19315	-0.52780	-0.52748	-0.52347
8	4.19315	-0.35565	-0.35557	-0.32643
13	4.19315	-0.13154	-0.13154	-0.12648
21	4.19315	-0.01052	-0.01008	-0.00690
34	4.19315	0.00236	0.00295	0.00746
55	4.19315	0.00234	0.00243	0.00982
89	4.19315	0.00205	0.00245	0.00758
144	4.19315	0.00237	0.00249	0.01134

FUNCTIONS

GET_SAMPLE

Returns a random sample. Parameters:

- N: the sample size.
- dist: one of normal, exponential, uniform or gamma
- Sh: The Shannon entropy of the population from which the sample is drawn.
- p1, p2: additional parameters.

Distribution	p1	p2
Normal	mean	sd
exponential	rate	--
uniform	min	max
gamma	shape	rate

- okgraph: plots the sample histogram

```
get_sample<-function(N,dist='normal',Sh=NaN,p1=NaN,p2=NaN,okgraph=FALSE){
    problem=TRUE
    if (okgraph) {x=c(0:1e4)/100}
    if (dist=='normal'){
        if (okgraph) {x=x-50}
        if (is.nan(p1)) {p1=0}
        if (! is.nan(Sh)) {
            p2=exp(Sh-log(sqrt(2*pi*exp(1))))

            problem=FALSE
        }
        else {
            if (! is.nan(p2)) {
                Sh=log(sqrt(2*pi*exp(1))*p2)

                problem=FALSE
            }
        }
        if (!problem) {
            print(paste('N(mean=',p1,',sd=',p2,'); Sh0=',Sh))
            sample=rnorm(N,mean=p1,sd=p2)
            if (okgraph) {
                pdf=dnorm(x,mean=p1,sd=p2)
                tit=paste('Normal(mean=',p1,'; sd=',p2,')')
            }

            #return(sample)
        }

    }
    if (dist=='exp') {
        if (! is.nan(Sh)) {
            p1=exp(1-Sh)

            problem=FALSE
        }
        else {
            if (! is.nan(p1)) {
                Sh=1-log(p1)

                problem=FALSE
            }
        }
        if (!problem) {
            print(paste('Exp(rate=',p1,'); Sh0=',Sh))
            sample=rexp(N,rate=p1)
            if (okgraph) {
                pdf=dexp(x,rate=p1)
                tit=paste('Exp(rate=',p1,')')
            }
            #return(sample)
        }
    }
    if (dist=='uniform') {
        if (! is.nan(Sh)) {
            if (is.nan(p1)) {p1=0}
            p2=exp(Sh)+p1

            problem=FALSE
        }
        else {
            if (! (is.nan(p1) & is.nan(p2))) {
                if (is.nan(p1)) {p1=0}
                if (is.nan(p2)) {p2=0}
                Sh=log(p2-p1)

                problem=FALSE
            }
        }
        if (!problem) {
            print(paste('Uniform(min=',p1,',max=',p2,'); Sh0=',Sh))
            sample=runif(N,min=p1,max=p2)
            if (okgraph) {
                pdf=dexp(x,mean=p1,sd=p2)
                tit=paste('Uniform(min=',p1,';max=',p2,')')
            }
            #return(sample)
        }
    }
    if (dist=='gamma') {
        if (!is.nan(Sh)) {
            if (is.nan(p1)) {
                p1=1
                p2=exp(1-Sh)
            }
            else{
                aux=p1+log(gamma(p1))+(1-p1)*digamma(p1)
                p2=exp(aux-Sh)
            }



            problem=FALSE
        }
        else {
            if (!is.nan(p1)) {
                if (is.nan(p2)) {p2=1}
                Sh=p1-log(p2)+log(gamma(p1))+(1-p1)*digamma(p1)

                problem=FALSE
            }
        }

        if (!problem) {
            print(paste('Gamma(shape=',p1,',rate=',p2,'); Sh0=',Sh))
            sample=rgamma(N,shape=p1,rate=p2)
            if (okgraph) {
                pdf=dgamma(x,shape=p1,rate=p2)
                tit=paste('Gamma(shape=',p1,';rate=',p2,')')
            }
            #return(sample)
        }
    }
    if (problem) {
        print('Some kind of problem with:')
        print(paste('Distr: ',dist))
        print(paste('N=',N))
        print(paste('Sh=',Sh))
        print(paste('p1=',p1))
        print(paste('p2=',p2))
        return(NaN)
    }
    else {
        if (okgraph) {
            #plot(x,pdf,type='l',main=tit)
            hist(sample,breaks=10,main=tit,freq=F)
            #lines(pdf~x, col = 2, add = TRUE)
        }
        return(sample)
    }
}
```

SET_BINS

Returns a series, the last element of this series is the first element greater than limit.

base: could be 'dyadic', then the series is $2^{(1,2,3,...)}$, 'fib', then the series is the Fibonacci series or a number, then the series is $base^{(1,2,3,.....)}$

```
set_bins <- function(base=dyadic,limit=1e4){
    if (base=='dyadic') {
        exponent=ceiling(log(limit)/log(2))
        sample=2^c(1:exponent)
    }
    else {
        if (base=='fib') {
            a=1
            b=1
            c=a+b
            sample=c(2)
            while (c<limit){
                a=b
                b=c
                c=a+b
                sample=append(sample,c)
            }

        }
        else {
            if (base=='factorial'){
                cont=2
                f=2
                sample=c(1,2)
                while (f<limit){
                    cont=cont+1
                    f=f*cont
                    sample=append(sample,f)
                }
            }
            else {
                exponent=ceiling(log(limit)/log(base))
                sample=base^c(1:exponent)
                sample=as.integer(sample)
                sample=as.integer(names(table(sample)))
            }
        }
    }
    return(sample)
}
```

EBC_SAMPLE

Returns the entropy estimate. It calls to *evaluate* function. Parameters:

- sample: the sample
- method: one of the entropy package methods. This method is used to calculate the entropy for each number of bins.
- bins: a series with the number of bins to be considered.
- npts: minimum number of points to fit the linear model

```
ebc_sample<-
     function(sample,method='MM',bins=set_bins('dyadic',1e4),okplot=FALSE
     ,npts=5){



    size=-log((max(sample)-min(sample))/bins)
    v=c()
    for (i in bins) {
        tries=discretize(sample,i)
        v=append(v,entropy(tries,method=method))

    }
    tit=paste(method,' (N=',length(sample),')')
    a=evaluate(v,size,npts=npts,plot=okplot,title=tit)
   return(a[1])



}
```

EVALUATE

Returns the model selected as a dataframe with interception, slope, adjusted R^2 and F value. The model is $ML \sim -D \ln(\text{size}) + H0$ The model selected is the one with the *D* nearest 1, max adjusted R² and max F value and max number of points. Parameters:

- ML: a vector with the entropy estimates from *ebc_sample*
- size: a vector with the values -ln(d_N) from *ebc_sample*, as described in eq. (1)
- npts: minimum number of consecutive points to fit the model.

```
evaluate<- function(ML,size,npts=5,plot=FALSE,title='') {
    ok=TRUE
    bottom=1
    top=length(ML)
    ntps_max=min(c(12,top))
    minmax=c(1,1,0)
    if (plot) { plot(size,ML,xlab='-ln(bin size)',ylab='Sh
     estimate',main=title)}
    for (n in c(ntps_max:npts)){
        for (bottom in c((length(ML)-n):1)){
            top=bottom+n
            model=lm(ML[bottom:top]~size[bottom:top])
            a=summary(model)
            if(!(is.nan(a$fstatistic[1]) | is.nan(a$coefficients[2]) |
     is.nan(a$adj.r.squared))){
                poll=0
                if (abs(1-a$coefficients[2])<minmax[1]) {poll=poll+0.3}
                if (a$fstatistic[1]>minmax[2]) {poll=poll+0.2}
                if (abs(a$adj.r.squared)>minmax[3]) {poll=poll+0.2}
                if (poll>=0.5){



     inf.a=data.frame(H0=a$coefficients[1],slope=a$coefficients[2],

     R2=a$adj.r.squared,F=a$fstatistic[1],
                                     bottom=bottom,top=top)
                    minmax=c(abs(1-inf.a$slope),inf.a$F,inf.a$R2)
                    #print(inf.a)
                }
            }
            #v_par=rbind(v_par,inf.a)
            if (plot){
                points(size,ML)
                lines(model$fitted.values~size[bottom:top], col='grey')
            }

        }

    }
    if (plot) {
        bottom=inf.a$bottom
        top=inf.a$top
        model=lm(ML[bottom:top]~size[bottom:top])
        lines(model$fitted.values~size[bottom:top], col='red')
    }
    a=as.numeric(inf.a)
    return(a)
}
```

Bibliography

Beirlant2001.: Beirlant, J., Dudewicz, E.J., Györfi, L. & van der Meulen, E.C. , Nonparametric entropy estimation: an overview,NATO Research Grant No. CRG 931030 , ,,2001

Duncan2004: Duncan, T.L., The Deep Physics Behind the Second Law: Information and Energy As Independent Forms of Bookkeeping,Entropy , 6,21-29,2004

Brissaud2005: Brissaud, J., The meanings of entropy,Entropy , 7[1],68-96,2005

TÉL1989: TÉL, T., FÜLÖP, Á. & VICSEK, T., DETERMINATION OF FRACTAL DIMENSIONS FOR GEOMETRICAL MULTIFRACTALS .,Physica A, 159,155-166,1989

Saa2007: Saa, A., Gascó, G., Grau, J.B., Antón, J.M. & Tarquis, A.M., Comparison of gliding box and box-counting methods in river network analysis, Nonlin. Processes Geophys., 14,603–613,2007

Lopes2009: Lopes, R. & Betrouni, N. , Fractal and multifractal analysis: A review ,Medical Image Analysis, 13,634–649,2009

Hausser2009: Hausser, J. & Strimmer, K. , Entropy Inference and the James-Stein Estimator, with Application to Nonlinear Gene Association Networks, Journal of Machine Learning Research, 10,1469-1484,2009

R: R Core Team (2013), R: A language and environment for statistical computing.,R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/., ,,

RStudio: RStudio Team (2012), RStudio: Integrated Development for R,RStudio, Inc., Boston, MA,http://www.rstudio.com/, ,,

entropy: Jean Hausser and Korbinian Strimmer, entropy: Estimation of Entropy, Mutual Information and Related Quantities,R package version 1.2.0,http://CRAN.R-project.org/package=entropy, ,,2013