PARAMETRIC ENTROPY BASED DENSITY ESTIMATION IN DISTANCE SAMPLING

KEVIN JOYCE

1. Introduction

The problem of interest in this work is to estimate the density of objects from distance data collected along line-transects within a given area. This problem has been extensively studied, and much work involves estimation based on the density estimator

$$\widehat{D}_i = \frac{y_i \widehat{f}(0)}{2L_i} \tag{1}$$

where y_i are the total observation counts for the ith transect, L_i is the length of the ith transect, and $\widehat{f}$ is an estimated density function modeling detectability based on perpendicular distance from the transect. In particular, f is such that the proportion of objects observed within a distance of x of any given transect is $\int_0^x f(t)\,dt$. The density estimator can be derived solely from data that records an observation's perpendicular distance from the transect. For a derivation see [1] [2]. By far, the most involved part of the estimation is obtaining $\widehat{f}(0)$.

In [1], Buckland highlights this difficulty with distance data obtained by Anderson and Pospahala measuring distances of duck nests from a single transect in the Monte Vista National Wildlife Refuge in Colorado.

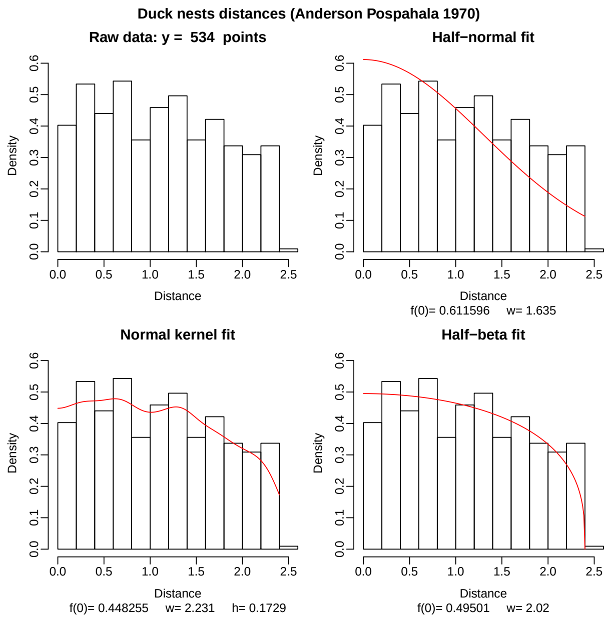

Figure 1. Distance data for duck nests in the Monte Vista National Wildlife Refuge in Colorado.

In Figure 1, a histogram of the distance data is displayed with a parametric half-normal density fit, a non-parametric normal kernel fit, and a density fit using the method described in this work. Note that the abrupt drop in detectability proves problematic for the half-normal method. The non-parametric kernel fitting method fits the density closely, however as we will see in this paper, this method tends to overfit the data resulting in instability in the estimator. This work is aimed at finding a robust parametric method for estimating $\widehat{f}$ where detectability is platykurtic or having a heavy central weighting within a fixed region.

Date: 12 December 2014.

2. Entropy Estimation and the Beta Distribution

A well-known result from mathematical statistics states that if a random variable X is distributed with a cumulative distribution function (CDF) $F(x)$, then the random variable $U = F(X)$ is distributed uniformly over $[0,1]$. Perhaps less well known is that of all density functions supported on $[0,1]$, say f, the one that maximizes the continuous entropy statistic

$$H(X) = -\int_0^1 f(x)\log f(x)\,dx \tag{2}$$

is the density for a uniform random variable on $[0,1]$ [4] (Note that $H(X) \geq 0$ since $\log f(x) < 0$.) Hence, if $X_i = x_i$ are realizations of a random variable, maximizing an estimate of the entropy from $F(x_i)$ with respect to a family $\mathcal{F}$ of densities is optimal in the information theoretic sense.

Estimating the entropy statistic has been studied [4], and we employ a straightforward histogram based estimator

$$\widehat{H} = -\sum_{\text{bins}} f(n_i)\log f(n_i). \tag{3}$$

This estimator is based constructing a histogram of the data (n_i are counts within a given histogram bin) and computing a discrete estimate by estimating the entropy within each bin. This is known to be a maximum likelihood estimator and is relatively stable when for large univariate data [4]. Moreover, it is conveniently implemented in `R` in the freely available `CRAN` library `entropy`. This leads to the following general optimization problem for estimating a symmetric density supported on a finite interval:

$$\widehat{F} = \underset{F\in\mathcal{F}}{\arg\min}\,\widehat{H}(F(x_i))^{-1} \tag{4}$$

where $\mathcal{F}$ is a family of CDFs for densities supported on the range of the distance data.

Of distributions supported on the interval $[0,1]$, symmetric beta distributions form a flexible parametric family whose densities are given by

$$f(x) = B(\alpha)x^{\alpha}(1-x)^{\alpha}. \tag{5}$$

In particular, the family provides a flexible collection of functions that have sufficiently low kurtosis to fit the problematic distance data given in the introduction.

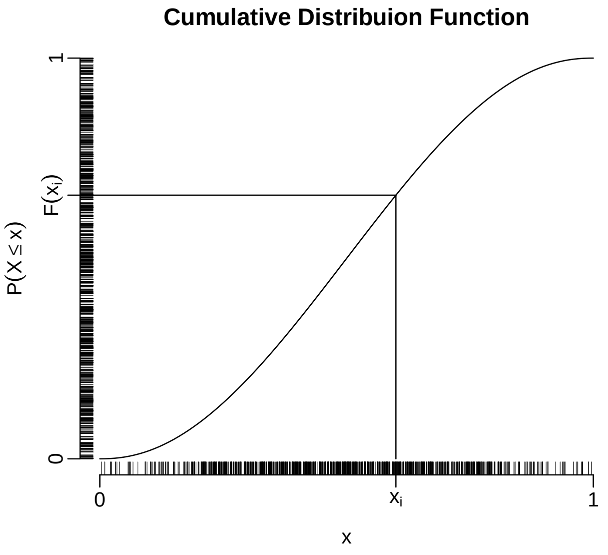

Figure 2. Finitely supported density data mapped through a beta CDF that maximizes entropy.

To fit arbitrary distance data supported on sets larger than $[0,1]$ to a beta distribution, we reflect the empirical data about zero to satisfy symmetry and rescale to $[0,1]$. The transformed data is fitted to a symmetric beta distribution, and the estimate f is half of the distribution transformed back to the scale of the data. This is similar to what is done to fit a kernel estimate [2]. We will refer to this as a half-beta distribution. The fourth plot in Figure 1 gives an example of this method fitted to the Anderson and Pospahala duck nest data.

3. Simulation Comparison

To quantify a comparison between methods, we simulate systematic transect data on 500 randomly placed points in a 1×1 square. To mimic Anderson and Pospahala's data, we simulated detection dependent on the perpendicular distance to a transect by Bernoulli trials with probabilities

$$p(d) = \begin{cases} 1 & |d| < e \\ (1-d)^p & |d| < b\,e. \end{cases} \tag{6}$$

In this particular simulation, we choose parameters $p = .65$ and $b = 1.3$ which produces distance samples sufficiently platykurtic. See Figure 4. Since the region has unit area, estimating the density estimates the total $\tau = 500$. We conducted 500,000 systematically sampled ($n = 4$ equally spaced) ideal bootstraps, and estimated using half-normal estimation, normal kernel estimation with the recommended kernel width [2], and EM half-beta estimation with histogram bin sizes given by Sturge's method which is the default histogram break selection in R. An example of a sample is illustrated in Figure 5. The results of the simulation are illustrated in Figure 6. R codes implementing this simulation can be found at the end of this paper.

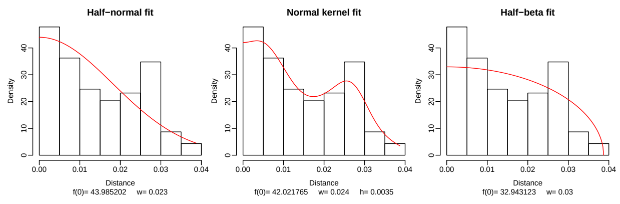

FIGURE 4. Simulated detected distance data with various density fits.

4. Conclusion

Each estimator has a bimodal distribution to varying degrees. This source of this is unknown, but it may be due to the piecewise method for defining detectability in (6). It may also be inherent to the density estimation, and more work needs to be done to address this question. The half-normal estimator has a large positive bias. This is likely due to overestimating $\widehat{f}$ near zero, and underestimating in the tails. See Figure 1 and Figure 4. The normal kernel estimate is skewed right, but appears to be relatively unbiased in estimating the total. However, it suffers from instability (in terms of mean squared error) likely due to over-fitting as hinted in the introduction. The EM half-beta estimator has a significantly lower variance than the normal kernel estimator, but appears to have a somewhat significant positive bias. Yet, in terms of mean squared error, it still outperforms the non-parametric kernel based estimator.

Note that the EM half-beta curve is constrained to have zero probability at the maximum data value sampled. This could be the source the bias in the estimation, and more investigation could reduce this effect and, subsequently, the mean squared error of the estimator.

In conclusion, the techniques presented in this work give a method for estimating low-kurtosis densities that have potentially lower mean squared errors than two of the most common parametric and non-parametric methods. Although biased, further work modeling EM density estimators may result in attractively stable and accurate density estimators.

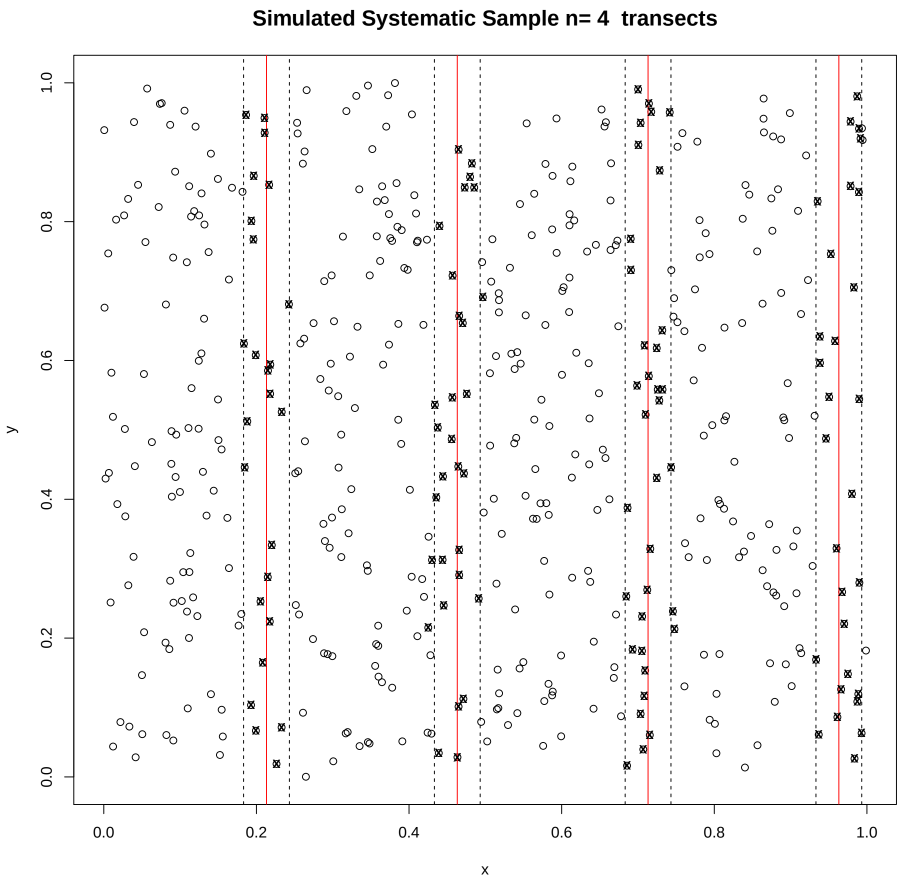

FIGURE 5. Simulated systematic sample of size $n = 4$ from randomly placed objects in a 1×1 square.

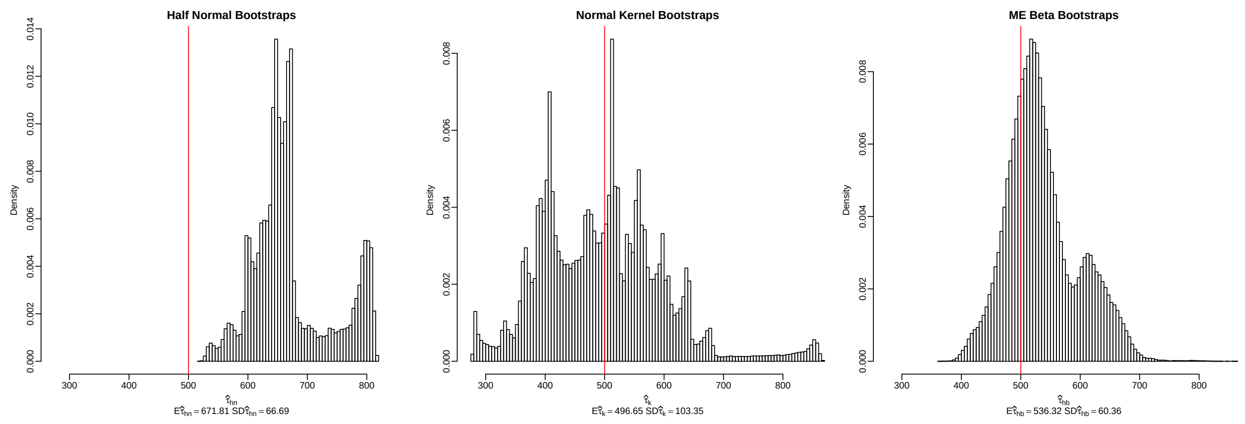

FIGURE 6. Density estimation on 500,000 ideal bootstraps on simulated data.

```
library('entropy')

# This function takes a list of positive distances
# and uses quantil maximum entropy to fit a
# half symmetric beta disribution.  The return value
# density is the density function
me_beta.density = function(d,breaks = NA) {
  # These are helper functions to beta_max_entropy
  beta_cdf = function(x,a) { pbeta(x,a,a) } # Fit data to a symmetric beta
      distribution
  scale_beta_pdf = function(s,a,max_s) { # This scales beta to [0,max(x)]
    dbeta( s/(2*max_s) + .5,a,a) / max_s
  }
  x = d / (2*max(d))
  x = c(-x,x) + .5
  if(is.na(breaks)) breaks = nclass.Sturges(x) # Use the Sturges algorithm for
       deciding the number of breaks for the entropy estimator
  cost = function(a) {1/entropy(discretize(beta_cdf(x,a),breaks)) } # Define 1
      /entropy based cost function
  opt_param = optimize(f=cost,interval=c(0,6)) # values over 5 sometimes lead
      to Inf values
  return( function(s) scale_beta_pdf(s,opt_param$minimum,max(d)) )
}

# These functions were culled from DistanceFunctions.R
halfnormal.density = function(x) {
  what.hn <- sqrt((pi/2) * mean(x^2)) # effective half-width for half-normal
  return( function(xx) exp(( - pi * xx^2)/(4 * what.hn^2))/what.hn )# half-
      normal fit
}

# h is the window width for the normal kernel method.
# If h is not given then formula (17.12) is used to calculate a window
# width from the data: h=0.9ay^(-1/5) where a = min(s,IQR/1.34).
# kernel.density returns a discretization of the density with n points (
    default 200)
kernel.density = function(x,h=NA,n=200) {
  y <- length(x)
  xx <- seq(0, max(x), length = n)
  a <- min(sd(x), IQR(x)/1.34)
  if(is.na(h))
    h <- (0.9 * a)/y^0.2  # default bandwidth
  f.ker <- rep(0, n)
  for(i in 1:n) f.ker[i] <- f.ker[i] <- (1/(y * h)) * (1/sqrt(2 * pi)) * sum(
      exp(-0.5 *
      ((xx[i] - x)/h)^2) + exp(-0.5 * ((xx[i] + x)/h)^2))
  return( f.ker )
}

# This plots densities overlayed on a histogram of the data
# and is also culled from DistanceFunctions.r
compare_three_densities = function(x,h=NA,n=200,breaks=NA) {
  if(is.na(breaks)) { breaks = nclass.Sturges(x) }
  xx = seq(0,max(x),length=n)
  f.hn = halfnormal.density(x)
  f.beta = me_beta.density(x)
  f.ker = kernel.density(x,h=h,n=n)
  if(is.na(h)){
    a <- min(sd(x), IQR(x)/1.34)
    h <- (0.9 * a)/length(x)^0.2  # default bandwidth
  }

  oldpar <- par(mfrow = c(2, 2), mgp = c(2, 0.5, 0),mar = c(4, 3, 2, 0) + 0.1,
      oma=c(0,0,1.1,0))
  ymax <- max(c(f.beta(xx)[!is.infinite(f.beta(xx))], f.hn(xx), f.ker, hist(x,
      plot = F,breaks = breaks)$density))
  hist(x + 0.0001 * max(x), col = 0, breaks = breaks, freq = F,
       ylim = c(0, ymax), main = paste("Raw data: y = ", length(x),
        " points"), xlab = "Distance")
  hist(x + 0.0001 * max(x), col = 0, breaks = breaks, freq = F,
        ylim = c(0, ymax), main = "Half-normal fit", sub = paste(
        "f(0)=", round(f.hn(0), 6), "    w=", round(1/f.hn(0),
        3)), xlab = "Distance")
  lines(xx, f.hn(xx),col='red')
  hist(x + 0.0001 * max(x), col = 0, breaks = breaks, freq = F,
        ylim = c(0, ymax), main = "Normal kernel fit", sub =
        paste("f(0)=", round(f.ker[1], 6), "    w=", round(
        1/f.ker[1], 3), "    h=", round(h, 4)), xlab = "Distance")
  lines(xx, f.ker,col='red')
  hist(x + 0.0001 * max(x), col = 0, breaks = breaks, freq = F,
        ylim = c(0, ymax), main = "Half-beta fit", sub =
        paste("f(0)=", round(f.beta(0), 6), "    w=", round(
        1/f.beta(0), 3)), xlab = "Distance")
  lines(xx, f.beta(xx),col='red')
}
```

CODE LISTING 1. estimator_functions.r

```
source('estimator_functions.r')
duck = read.csv('ducknests.csv')
crab = read.csv('crabbieMCDS.csv')
compare_three_densities(duck$distance)
title('Duck nests distances (Anderson Pospahala 1970)',outer=T)
#dev.copy(pdf,'duck_nest_fits.pdf',width=7,height=7)
#dev.off()

scale_fun = function(x) {
  x = x / (2*max(x))
  x = c(-x,x) + .5
}
x = scale_fun(crab$distance)

beta_cdf = function(x,a) { pbeta(x,a,a) } # Fit data to a symmetric beta
    distribution
scale_beta_pdf = function(x,a,max_x) { # This scales beta to [0,max(x)]
  dbeta( x/(2*max_x) + .5,a,a) / max_x
}
```

```
cost = function(a) {1/entropy(discretize(beta_cdf(x,a),bins)) } # Define
   entropy based cost function
opt_param = optimize(f=cost,interval=c(0,4))

# Example of quantile function
f = function(x)pbeta(x,opt_param$minimum,opt_param$minimum)
plot(f,
     ylab = expression(P(X <= x)),
     xlab = expression(x),
     main = "Cumulative Distribuion Function",
     axes = F
     )
axis(side=1, at=c(0,.6,1), labels=c(0,expression(x[i]),1))
axis(side=2, at=c(0,f(.6),1), labels=c(0,expression(F(x[i])),1))
idx = seq(1,length(x),4)
rug(x[idx])
rug(pbeta(x[idx],opt_param$minimum,opt_param$minimum),side=2)
segments(c(.6,.6), c(0,f(.6)), c(.6,-.1), c(f(.6),f(.6)))
dev.copy(pdf,'cdf_example.pdf',width=5,height=5)
dev.off()
```

CODE LISTING 2. duck_fits.r

```
rm(list=ls())
tau = 500  # Population total
n = 4 # Number of transects
systematic_length = 1/n # Number in systematic sample
sight_radius = .03 # distance from transects that gaurantees counting
buf = 1.3 # Buffer parameter for detectability function
pow = .65 # Power on 1-distance for detectability function

x = runif(tau) # Generate random coordinates
y = runif(tau)

transect_sample = function(x,y) {
#transects = runif(n) # Sample from the bottom of the square  # Randomly
   placed transects
  transects = (runif(1) + (1:n-1)*systematic_length) %% 1 # A systematic
     sample
  distance = sapply(transects,function(t)abs(t - x)) # Calculate the distance
     of every point to each transect
  idx1 = distance < sight_radius # Include all points within sight_radius
  idx2 = (distance < buf*sight_radius) & # Detectability is random within
     buffer zone
   (matrix(runif(tau*n),tau,n) < (1-distance/(buf*sight_radius))^pow) # This
      implements Bernouli trials
  idx = idx1 | idx2

  xx = matrix(rep(x,n),tau,n,byrow=F) # Replicate x to match dimensions of
     distance matrix
  yy = matrix(rep(y,n),tau,n,byrow=F) # Replicate y to match dimensions of
     distance matrix
  tt = matrix(rep(1:n,tau),tau,n,byrow=F) # Replicate transects to match
      dimensions of distance matrix

  samp = data.frame(
      x = as.vector(xx[idx]),
      y = as.vector(yy[idx]),
      transect = as.vector(tt[idx]),
      transect_x = transects[as.vector(tt[idx])],
      distance = as.vector(distance[idx])
  )
}
samp = transect_sample(x,y)

# Plot the sample from the simulation
plot(x,y,main=paste('Simulated Systematic Sample n=',n,' transects'),cex.main
    =1.4)
transects = unique(samp$transect_x)
out = sapply(transects,function(x)abline(v=x,col='red'))
out = sapply(transects,function(x)abline(v=x-sight_radius,lty='dashed'))
out = sapply(transects,function(x)abline(v=x+sight_radius,lty='dashed'))
out = sapply(1:n,function(i)points(samp$x,samp$y,pch='x'))
#dev.copy(pdf,'simulated_sample.pdf',width=10.2,height=10)
#dev.off()

dev.new()
source('estimator_functions.r')
compare_three_densities(samp$distance)
#dev.copy(pdf,'density_curve_fits.pdf')
#dev.off()

estimate_total = function(samp,f0) {
  yi = tapply(samp$distance,samp$transect,length)
  Li = tapply(samp$distance,samp$transect,function(x)1)
  Di = yi*f0/(2*Li)
  return( mean(Di) )
}

f.hn = halfnormal.density(samp$distance)
f.beta = me_beta.density(samp$distance)
f.ker = kernel.density(samp$distance)

(tauhats = sapply(c('half-normal'=f.hn(0),'half-beta'=f.beta(0),'kernel'=f.ker
    [1]), function(f0)estimate_total(samp,f0)))

bootstrapper = function(x,y,pb,i) {
  setTxtProgressBar(pb,i)
  samp = transect_sample(x,y)
  f.hn = halfnormal.density(samp$distance)
  f.beta = me_beta.density(samp$distance)
  f.ker = kernel.density(samp$distance)
  sapply(c('half-normal'=f.hn(0),'half-beta'=f.beta(0),'kernel'=f.ker[1]),
      function(f0)estimate_total(samp,f0))
}
```

```
#N = 5e5 # This took a few hours with my PC
#pb = txtProgressBar(min=1,max=N,style=3)
#bootstraps = sapply(1:N,function(i)bootstrapper(x,y,pb,i))
#close(pb)
#save.image('500k_bootstrapped_simulations.RData')
load('500k_bootstrapped_simulations.RData')

dev.new()
par(mfrow = c(1, 3), mgp = c(2, 0.5, 0),mar = c(4, 3, 2, 0) + 0.1)
xl = c(min(bootstraps),max(bootstraps))
breaks = min(sapply(1:3,function(i)nclass.scott(bootstraps[i,])))
hist(bootstraps['half-normal',],
     xlab=expression(widehat(tau)[hn]),
     xlim=xl,
     sub=bquote(paste(
         plain(E),  widehat(tau)[hn]==.(round(mean(bootstraps['half-normal',])
             ,2)),"  ",
         plain(SD), widehat(tau)[hn]==.(round(sd(  bootstraps['half-normal',])
             ,2))
        )),
     main='Half Normal Bootstraps',
     cex.main = 1.3,
     cex.sub  = 1,
     cex.lab  = 1,
     breaks=breaks,
     freq=F
     )
abline(v=tau,col='red')
hist(bootstraps['kernel',],
     xlab=expression(widehat(tau)[k]),
     xlim=xl,
     sub=bquote(paste(
         plain(E),  widehat(tau)[k]==.(round(mean(bootstraps['kernel',]),2)),"
              ",
         plain(SD), widehat(tau)[k]==.(round(sd(  bootstraps['kernel',]),2))
        )),
     main='Normal Kernel Bootstraps',
     cex.main = 1.3,
     cex.sub  = 1,
     cex.lab  = 1,
     breaks=breaks,
     freq=F
     )
abline(v=tau,col='red')
hist(bootstraps['half-beta',],
     xlab=expression(widehat(tau)[hb]),
     xlim=xl,
     sub=bquote(paste(
         plain(E),  widehat(tau)[hb]==.(round(mean(bootstraps['half-beta',])
             ,2)),"  ",
         plain(SD), widehat(tau)[hb]==.(round(sd(  bootstraps['half-beta',])
             ,2))
        )),
     main='ME Beta Bootstraps',
     cex.main = 1.3,
     cex.sub  = 1,
     cex.lab  = 1,
     breaks=breaks,
     freq=F
     )
abline(v=tau,col='red')
#dev.copy(pdf,'500K_bootstrap_comparison.pdf',width=15,height=5)
#dev.off()
```

CODE LISTING 3. `simulate_sample.r`

REFERENCES

[1] S.T. Buckland. *Introduction to Distance Sampling: Estimating Abundance of Biological Populations*. Oxford University Press, 2001.

[2] S.K. Thompson. *Sampling*. CourseSmart Series. Wiley, 2012.

[3] DR Anderson and RS Pospahala. Correction of bias in belt transect studies of immotile objects. *The Journal of Wildlife Management*, pages 141–146, 1970.

[4] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. A Wiley-Interscience publication. Wiley, 2006.