Point and interval estimates of abundance using multiple covariate distance sampling: An example using great bustards

Eric Rexstad*
Research Unit for Wildlife Population Assessment
University of St. Andrews

January 2007

1 Summary

Description of computations to produce sex-specific estimates of density from a multiple-covariate distance sampling analysis. Program Distance 5.0 has limited capacity to bootstrap certain types of analytical situations (e.g., cluster size as a covariate). Herein I describe steps and code to perform an analysis of this sort. Possible ways to adapt this code for similar analyses are described.

---

2 Introduction

Analyzing distance sampling surveys with covariates in addition to distance is an analytical technique that is seeing increasing use as surveys and analyses become more sophisticated. The entire range of capabilities for such analyses are not incorporated into Distance version 5.0 Release 2. Here I demonstrate the use of R to facilitate a fairly sophisticated analysis. The reader of this document is assumed to be conversant and able to program in R.

*The code described herein is modified in part from code developed in 2005 by Tiago Marques. His code, in turn, is borrowed, in part, from code written by Len Thomas. Their hard work made this programmming exercise much less daunting. Errors contained in this technical report, however, are my responsibility.

2.1 Basic analytical situation

Field data on great bustards *(Otis tarda)* were collected, with group being the unit of data collection. Groups were identified as being either male or female in composition. Covariates were added to the detection function (without adjustment terms) to improve the performance of the model. The resulting model was

$$\text{detection probability} = fn(\text{distance, sex, cluster size})$$

The difficulties with the result provided by Distance for this analysis are

- there are no sex-specific density estimates provided, and
- there are no measures of precision associated with the sex-specific estimates.

This technical report describes how to create those point and interval estimates.

A companion piece of R code, written by Tiago Marques, is available online, and performs bootstrap variance estimation for an analysis incorporating covariates in the detection function, and geographic strata. The setup for that analysis is very similar to what is described here, but the R code that sets up the bootstrap resample is quite different because of the stratification. The quantity being estimated by Tiago's code is also quite different, so the statistics being saved in each bootstrap differ from those reported here.

3 Computing set-up

A Fortran program (MCDS.EXE) lives underneath Distance 5.0, and it is this code that will do most of the hard work in this analysis. MCDS.EXE expects to find instructions in a text file, and it will deliver results to a different text file called the `stats file`. From these building blocks, we will build a mechanism to derive the estimates we desire. So, we must perform a bit of preliminary work before embarking upon our analysis.

Create a directory and populate it with MCDS.EXE To keep all of the bits and pieces associated with this analysis, create a separate directory for this project. Find the MCDS.EXE program within `C:/Program Files/Distance 5/` and copy it into the directory you have created.

Create datafile and command file Distance 5.0 acts as an interpreter for you sending instructions to MCDS and processing results. This will be done by the instructions contained herein. To relieve Distance 5.0 of this responsibility, begin by running Distance 5.0 in DEBUG mode. Examine Distance 5.0 Users Guide (Chapter 10 RUNNING THE MRDS ANALYSIS ENGINE FROM OUTSIDE DISTANCE) regarding setting up DEBUG mode.

Edit command file Most of the information contained in the command file (reproduced below) will not require alteration. However, you should examine the file names (the first 6 lines of the file), and the file name following the string 'Infile=' (see Appendix A). A correspondence will be maintained between these file names and the remainder of the code you will use.

The filenames are comprised of a 'base' and an extension. In the example below, the 'base' is 'boot06' and the extensions are 'out.txt,' 'plot.txt,' etc. The analysis here requires information contained in the `stats file` 'boot06.stat.txt' in our example. The remainder of the files are extraneous to our purposes.

4 Basic mechanics

The basic concept is that from the orginal data, a bootstrapping routine resamples with replacement from the dataset (we have the simplification here because there are no strata in the original survey design). Each resample is written to a file [file.base + 'data.txt' in our example], so it may be read into MCDS.EXE. The results the analysis of each resample is sent to the stats file [file.base + 'stat.txt']. The format of the `stats file` is described in Distance 5.0 Users Guide APPENDIX - MCDS ENGINE REFERENCE, and statistics of interest can be harvested from the stats file. Consult the `'read.stats.file'` (see Appendix F) function to pluck the necessary statistics from the `stats file`.

5 Sex-specific density estimates

For purposes of the great bustard analysis, interest centred on sex-specific estimates of density, so we wish to produce those estimates for each sex. Those estimates need to be computed from the estimated coefficients and the values of the covariates for each detected object (sex and group size in our situation).

We start with the scale parameter (σ^2) of a half-normal detection function,

specific to each detected object.

$$\sigma_i^2 = \begin{cases} \left(\alpha_1 \cdot e^{\alpha_3 \cdot CS_i}\right)^2 & \text{for males, and} \\ \left(\alpha_1 \cdot e^{\alpha_2 + \alpha_3 \cdot CS_i}\right)^2 & \text{for females.} \end{cases} \quad (1)$$

The detection function $g(y, \underline{z}_i)$ given the scale parameter σ^2

$$g(y, \underline{z}) = exp\left(\frac{-y^2}{2\sigma^2(\underline{z})}\right) \quad (2)$$

However, we need the probability density function ($f(y)$ with covariates $f(y|\underline{z})$

$$f(y|\underline{z}) = \begin{cases} \frac{g(y)}{\int_0^w g(y|\underline{z})dy} & \text{for line transects, and} \\ \frac{y \cdot g(y)}{\int_0^w y \cdot g(y|\underline{z})dy} & \text{for point transects.} \end{cases}$$

see eqn 3.8 of Buckland et al. (2001)

Evaluating the probability density function at distance 0 for line transect estimators, this results in

$$f(0|\underline{z}_i) = \frac{1}{\int_0^w exp\left(\frac{-y^2}{2\sigma_i^2}\right) dy} \quad (3)$$

whereas for point transects, this is

$$f_i'(0|z_i) = \frac{-1}{\left[\sigma_i^2 exp\left(\frac{-y^2}{2\sigma_i^2}\right)\right]_0^w} = \frac{1}{\sigma_i^2\left(1 - exp\left(\frac{-w^2}{2\sigma_i^2}\right)\right)}$$

see 3.45 of Buckland et al.(2001)

With estimates of the detection function (or its derivative) evaluated at distance 0, this estimated parameter can then be placed into a Horvitz-Thompson-like estimator for density

$$\hat{D} = \begin{cases} \frac{1}{2L} \sum_{i=1}^n s_i \cdot \hat{f}(0|\underline{z}_i), & \text{for line transects, or} \\ \frac{1}{2K\pi} \sum_{i=1}^n s_i \cdot \hat{f}'(0|\underline{z}_i), & \text{for point transects.} \end{cases}$$

3.32 and 3.45 Marques and Buckland (2004)

6 Programming issues

The nature of the estimation task for the great bustard dataset can be decomposed into the following list, with functions performing the tasks

sex-specific point estimates from raw data `point.estimate` (Appendix C), that calls compute.sigma.f0.Dht (this is where the math takes place);

resample original data create.data.file.bootstrap, which fundamentally is a call to the R function sample (with replacement) (Appendix E), and writing the result to a file that can be recognized by MCDS.EXE;

produce point estimates from resamples bootstrap.data, (see Appendix B) that calls compute.sigma.f0.Dht (Appendix D). This is essentially the calling routine that subsequently invokes the remaining functions;

harvest interesting statistics and estimates from MCDS 'stats file' read.stats.file, (see Appendix F) with help from split.line (Appendix G); and

running MCDS.EXE and file management this is the job of cds, which makes use of run.cds, and get.cds.file.names (Appendix G) to complete its job.

7 Miscellaneous items

Because calculations are taking place without the assistance of Distance, there are some small, but critical details that must be handled by the code we write. If there are differences in measurement units between measurements of effort and detection distance, or if you wish the resulting densities to be reported in different units; unit conversion factors need to be defined. In the case of the great bustards, line length and detection distances were measured in metres, but densities were to be reported in km^{-2}, so a conversion factor of 1,000,000 was incorporated.

8 Code modification for other uses

It is difficult to imagine all the different scenarios that this coding situation embedding MCDS.EXE within R scripts might be used. Hence this is not code that can be applied to many situations without some modification. The one nod to flexibility I have made is to offer 'method' as an argument to the functions described. In this way, either line transect analyses or point transect analyses can be analyzed without need to modify the R scripts.

Beyond, that small piece of flexibility, all of the computations described herein are stem from the detection function being a half-normal with no adjustment terms. If this is not the appropriate form of the detection function, effectively all of `'compute.sigma.f0.Dht'` would need to be rewritten. Further

modifications to `'compute.sigma.f0.Dht'` and `'read.stats.file'` would need to be made if here were more covariates involved in fitting the detection function; in the case of the great bustards, there was a coefficient for the 'intercept' of the detection function, and a coefficient for sex of the detected group, and group size.

To conform to the specific nature of the dataset being analyzed, the reading of the data (done both in `'bootstrap.data'` and `'point.estimate'` by the 'read.table' function) would need to be amended accordingly.

The other item to point out is that the set of R functions described here does not contain any code for processing the bootstrap results. There are many functions in R that can summarize the product of these functions, I have made no attempt to provide a universal summary routine.

9 Literature cited

- Buckland, S.T., D.R. Anderson, K.P. Burnham, J.L. Laake, D.L. Borchers, and L. Thomas. 2001. Introduction to distance sampling. Oxford University Press.

- Marques, F.F.C., and S.T. Buckland. 2004. Covariate models for the detection function. Pages 31-47 in S.T. Buckland, D.R. Anderson, K.P. Burnham, J.L. Laake, D.L. Borchers, and L. Thomas (eds.) Advanced distance sampling. Oxford University Press.

- Thomas, L., Laake, J.L., Strindberg, S., Marques, F.F.C., Buckland, S.T., Borchers, D.L., Anderson, D.R., Burnham, K.P., Hedley, S.L., Pollard, J.H., Bishop, J.R.B. and Marques, T.A. 2006. Distance 5.0. Release 2. Research Unit for Wildlife Population Assessment, University of St. Andrews, UK. http://www.ruwpa.st-and.ac.uk/distance/

A Command file to guide MCDS.EXE

Contents of command file generated by running Distance 5.0 in debug model note particularly first six lines and Infile= argument

```
boot06.out.txt
boot06.plot.txt
boot06.stat.txt
boot06.log.txt
boot06.boot.txt
boot06.prog.txt
Options;
Type=Line;
Length /Measure='Kilometre';
Distance=Perp /Measure='Metre';
Area /Units='Square kilometre';
Object=Cluster;
SF=1;
Selection=Specify;
Confidence=95;
Print=All;
End;
Data /Structure=Flat;
Fields=STR_LABEL, STR_AREA, SMP_LABEL,
       SMP_EFFORT, DISTANCE, Sex, Cluster size;
Factor /Name=Sex /Levels=2 /Labels=F, M;
SizeC=Cluster size;
Infile=boot06.data.txt /NoEcho;
End;
Estimate;
Distance /Width=500;
Density=All;
Detection=All;
Size=All;
Estimator /Key=HN /Adjust=CO /NAP=0
          /Covariates=Sex, Cluster size;
Monotone=None;
Pick=AIC;
GOF;
VarN=Empirical;
End;
```

B Bootstrap.data function listing

Function 'bootstrap.data'

```
bootstrap.data<-function(B, datafile="original.data.txt", file.base="boot06", method="point"){
library(svMisc)  #  for progress function to track progress
#                    DEFINING THE FILE NAMES
data.file.name<-paste(file.base,".data.txt",sep="")
#                    read the data in
data<-read.table(datafile,header=FALSE,na.strings="",sep = "",fill=TRUE)
#           create a vector of zeros to fill the data.frame with the results from this function
fill.in<-rep(0,B)
res.boot<-data.frame(
               Dhat.grp=fill.in, Dhat.ind=fill.in, mu=fill.in,
               Es=fill.in, LnL=fill.in, AIC=fill.in,
               N=fill.in, a1=fill.in, a2=fill.in, a3=fill.in,
               status=fill.in, n.detects=fill.in, D.ht.male=fill.in,
               D.ht.female=fill.in, sex.ratio=fill.in)
m <- 0
errors <- NULL
for(r in 1:B) {
    progress(r, B)
    create.data.file.bootstrap(data,data.file.name)
    results <- cds(file.base=file.base,ext.files=TRUE,bootstrap=TRUE)
    if(results$status>2) {
        m <- m+1
        errors[m] <- results$status
    } else {
#               Reread bootstrap data file, keep only detections,
#                 hang onto covariate coefficients
      boot.rep <- read.table(file=paste(file.base,".data.txt",sep=""), sep="\t", header=FALSE,
                        col.names=c("strata","area","label","effort",
                                    "distance","sex","cluster.size"),as.is=TRUE)
      boot.estimates <- compute.sigma.f0.Dht(results, boot.rep, method=method)
      for ( i.fields in 1:length(results)) {
        res.boot[r,i.fields] <- results[[i.fields]]
        }
      res.boot[r, i.fields+1] <- boot.estimates$num.detects
      res.boot[r, i.fields+2] <- boot.estimates$point.male
      res.boot[r, i.fields+3] <- boot.estimates$point.female
      res.boot[r, i.fields+4] <- boot.estimates$point.ratio
#         remove any temp files
      bootstrap <- TRUE
      file.names<-get.cds.file.names(file.base)
      if(bootstrap==FALSE) {
        if(is.null(file.base)) remove.files(file.names)
      }
       else {#if inside a bootstrap routine, delete intermediate files, except command file
         remove.files(file.names,bootstrap=bootstrap)
         }
      }
  }
return(list(res.boot=res.boot,errors=errors))
}
```

C Point estimate function listing

Function 'point.estimate'

```
point.estimate <- function(cmdfile="point06.cmd.txt",
                           statfile="point06.stat.txt",
                           datafile="bust06.trunc.data.txt",method="line")
{
  point <- run.cds(cmdfile)
  results <- read.stats.file(statfile)
  real.data <-  read.table(file=datafile, sep="\t", header=FALSE,
                      col.names=c("strata","area","label","effort",
                                  "distance","sex","cluster.size"),as.is=TRUE)
  point.values <- compute.sigma.f0.Dht(results, real.data, method="line")
  return(list(point.male=point.values$point.male,
              point.female=point.values$point.female,
              point.ratio=point.values$point.ratio))
}
```

D Compute.sigma.f0.Dht function listing

Function 'compute.sigma.f0.Dht' and 'g.of.y'

```
compute.sigma.f0.Dht <- function(stats.file.results, datafile.covariates, method="line")
{
  detects <- subset(datafile.covariates, !is.na(distance))
  n.detects <- length(detects[,1])
  coeff1 <- stats.file.results[[8]]
  coeff2 <- stats.file.results[[9]]
  coeff3 <- stats.file.results[[10]]
#               Hard-wired truncation distance and number of points surveyed for bustards
  w <- 500
  k.points <- 133
  total.effort <- 1130700
  sq.meter.to.sq.km <-  1000 * 1000
  sum.male <- 0
  sum.female <- 0
  for (i.detect in 1: n.detects) {
    this.cluster <- detects$cluster.size[i.detect]
    if (detects$sex[i.detect] == "M") {
      sigma.sq.male <- (coeff1 * exp(coeff3 * this.cluster )) ^2
#               multiply h(0,i) or f(0,i) by s(i) a la eqn 3.32 Marques & Buckland(2004)
      if (method == "point" ) {
        sum.male <- sum.male +  this.cluster /
                              ( sigma.sq.male * ( 1 - exp(-w^2/(2*sigma.sq.male))))
      } else {
        sum.male <- sum.male + this.cluster /
                               integrate(g.of.y, 0, w, sigma.sq=sigma.sq.male)$value
          }
    } else {
      sigma.sq.female <- (coeff1 * exp(coeff2 + coeff3 * this.cluster )) ^2
      if (method == "point" ) {
          sum.female <- sum.female + this.cluster /
                              ( sigma.sq.female * ( 1 - exp(-w^2/(2*sigma.sq.female))))
      } else {
         sum.female <- sum.female + this.cluster /
                                    integrate(g.of.y, 0, w, sigma.sq=sigma.sq.female)$value
        }
    }
    }
#         equations coded here approximate 3.45 removing A to estimate density
    if (method == "point" ) {
        D.ht.male <- 1/(2*k.points*pi) * sum.male  * sq.meter.to.sq.km
        D.ht.female <- 1/(2*k.points*pi) * sum.female  * sq.meter.to.sq.km
    } else {
         D.ht.male <- 1/(2*total.effort) * sum.male * sq.meter.to.sq.km
         D.ht.female <- 1/(2*total.effort) * sum.female * sq.meter.to.sq.km
    }
    sex.ratio <- D.ht.male / D.ht.female
  return(list(num.detects=n.detects,
              point.male=D.ht.male,
              point.female=D.ht.female,
              point.ratio=sex.ratio))
#-------------------------------------
g.of.y <- function(y, sigma.sq){
  exp(-y^2/(2*sigma.sq))
}
}
```

E Create.data.file.bootstrap function listing

Function 'create.data.file.bootstrap'

```
create.data.file.bootstrap<-function(data,filename="boot.data.txt"){
num.points <- max(data$V3)
resample <- sample(1:num.points, num.points, replace=TRUE)
for (write.i in 1:num.points)
  {
  id <- resample[write.i]
  if (is.na(data[data$V3==id, 7])) {
    write.record <- paste(paste(data[data$V3==id,1], ".",sep=""), data[data$V3==id,2],
                          paste(write.i, ". ", write.i, sep=""),
                          data[data$V3==id,5], " ", " ", " ", sep="\t")
    } else {
    write.record <- paste(paste(data[data$V3==id,1], ".",sep=""), data[data$V3==id,2],
                          paste(write.i, ". ", write.i, sep=""),
                          data[data$V3==id,5], data[data$V3==id,6], data[data$V3==id,7],
                          data[data$V3==id,8], sep="\t")
    }
  cat(write.record, file=filename, sep="\n", append=TRUE)
  }
return()
}
```

F Read.stats.file function listing

Function 'read.stats.file'

```
read.stats.file<-function(stat.file.name) {
#       read the file in and test that it has something in it
  lines.v<-readLines(stat.file.name)
  n.lines<-length(lines.v)
  if(n.lines==0) {
    stop ("Nothing to read!")
  }
#   go through each line, looking for the results we want to store
  aic<-NULL; Dhat.grp<-NULL; Dhat.ind<-NULL; mu<-NULL; LnL<-NULL
  Es<-NULL; N<-NULL; a1<-NULL; a2<-NULL; a3<-NULL
  for (line in 1:n.lines) {
    parsed.line<-split.line(lines.v[line])
    if(parsed.line$ok) {
       if(parsed.line$module==2 & parsed.line$statistic==6){#effective strip (half) width line
         if(parsed.line$stratum==0) {mu[length(mu)+1]<-parsed.line$value}
            else{mu[parsed.line$stratum]<-parsed.line$value}
       }
       if(parsed.line$module==2 & parsed.line$statistic==7){#Akaike Information Criterion line
          aic<-parsed.line$value
       }
       if(parsed.line$module==2 & parsed.line$statistic==9){#log-likelihood line
          LnL<-parsed.line$value
       }
       if(parsed.line$module==2 & parsed.line$statistic==101){#covariate coefficient 1
         a1<-parsed.line$value      }
       if(parsed.line$module==2 & parsed.line$statistic==102){#covariate coefficient 2
         a2<-parsed.line$value      }
       if(parsed.line$module==2 & parsed.line$statistic==103){#covariate coefficient 3
         a3<-parsed.line$value      }
       if(parsed.line$module==3 & parsed.line$statistic==1){#mean cluster size line
#              using mean cluster size for now (statistic==1) -
#              if switch to cluster size regression will want statistic==3
         if(parsed.line$stratum==0)  {Es[length(Es)+1]<-parsed.line$value}
           else {Es[parsed.line$stratum]<-parsed.line$value}
       }
       if(parsed.line$module==4 & parsed.line$statistic==1){#density of groups line
         if(parsed.line$stratum==0) {Dhat.grp[length(Dhat.grp)+1]<-parsed.line$value}
           else {Dhat.grp[parsed.line$stratum]<-parsed.line$value}
         }
         if(parsed.line$module==4 & parsed.line$statistic==2){#density of individuals line
           if(parsed.line$stratum==0) {Dhat.ind[length(Dhat.ind)+1]<-parsed.line$value}
             else {Dhat.ind[parsed.line$stratum]<-parsed.line$value}
         }
         if(parsed.line$module==4 & parsed.line$statistic==3){#number of animals line
           if(parsed.line$stratum==0) {N[length(N)+1]<-parsed.line$value}
             else {N[parsed.line$stratum]<-parsed.line$value}
         }
    }
  }
  return(list(aic=aic, Dhat.grp=Dhat.grp, Dhat.ind=Dhat.ind, mu=mu,
              LnL=LnL, Es=Es,N=N, a1=a1, a2=a2,a3=a3))
 }
```

G Listing of miscellaneous functions

Miscellaneous functions including 'cds,' 'get.cds.file.names,' 'run.cds,' 'split.line,' and 'remove.files'

```
cds <- function (key, adj, L, w, A=NA, xi, zi, file.base,ext.files=TRUE,bootstrap=TRUE) {
#    Purpose: Driver function to run the mcds.exe engine from R
#     get input and output file names
  file.names<-get.cds.file.names(file.base)
#    if external not provided, create data file
  if(ext.files==FALSE) {
      create.data.file(file.names$data.file, L, A, xi, zi)
  }
#   if external not provided,create command file
  if(ext.files==FALSE) {
      create.command.file(file.names, key, adj, w)
  }
#     call cds engine
  run.status<-run.cds(file.names$cmd.file)
#    harvest results from stats file
  res <- read.stats.file(file.names$stat.file)
#      do not clean out temporary files here because they are needed later in the
#         computation of the HT estimators of stratum-specific estimates
#    return results
  if(is.na(A))  {
#       If no Area was provided
    return(list(Dhat.grp=res$Dhat.grp, Dhat.ind=res$Dhat.ind,
                mu=res$mu,  Es=res$Es, LnL=res$LnL, AIC=res$aic, N=res$N,
                a1=res$a1, a2=res$a2, a3=res$a3, status=run.status))
    } else {
#       If Area was provided
    return(list(Nhat.grp=res$Dhat.grp*A, Nhat.ind=res$Dhat.ind*A,
                mu=res$mu, Es=res$Es, LnL=res$LnL, AIC=res$aic,
                a1=res$a1, a2=res$a2, a3=res$a3, status=run.status))
                }
}
#-------------------------------------------------------------------------------
get.cds.file.names<-function(file.base=NULL) {
#         Purpose: returns a list of filenames given the file.base
   files.mid<-c("cmd","data","out", "log","stat","boot","plot")
   files.suffix<-"txt"
   if(is.null(file.base)) {
#         get temp file names
     file.names<-tempfile(files.mid)
   } else {
      file.names<-rep("",length(files.mid))
#         and add the file base
     for(i in 1:length(files.mid)) {
       file.names[i]<-paste(file.base,".",files.mid[i],sep="")
     }
   }
#        add the appropriate suffix
   for(i in 1:length(files.mid)) {
     file.names[i]<-paste(file.names[i],files.suffix,sep=".")
   }
   return(list(cmd.file=file.names[1], data.file=file.names[2], out.file=file.names[3],
     log.file=file.names[4], stat.file=file.names[5], boot.file=file.names[6],
     plot.file=file.names[7]))
}
#-------------------------------------------------------------------------------
run.cds<-function(cmd.file.name) {
#Purpose: runs the MCDS.exe engine and waits for it to finish
#Inputs:
#  cmd.file.name - name of the command file to run
#Returns:
#  A status integer - 1=OK, 2=warnings, 3=errors,
                      4=file errors, 5=some other problem (e.g., program crash)

 command <- paste("mcds 0, \"",cmd.file.name,"\"",sep="")
 res<-system(command, intern=TRUE, invisible=TRUE)
 res<-as.integer(res)
 if(is.na(res)) res<-5
 return(res)
```

```
}
#-------------------------------------------------------------------------------
split.line <- function(line) {
#takes each line and returns the different values for stratum, samp, estimator, ...
  if(nchar(line)<6) stop ("This isn't a stats file!")
  stratum<-as.integer(substr(line,1,6))
  if(is.na(stratum)) {
    #this means that
    ok=FALSE
    samp<-NULL
    estimator<-NULL
    module<-NULL
    statistic<-NULL
    value<-NULL
    cv<-NULL
    lcl<-NULL
    ucl<-NULL
    degrees.freedom<-NULL
  } else {
    ok=TRUE
#       Parse the rest of the line when ok is true
    samp <- as.integer(substr(line, 8, 13))
    estimator<-as.integer(substr(line,14,15))
    module<-as.integer(substr(line, 16,17))
    statistic<-as.integer(substr(line,18,20))
    value<-as.numeric(substr(line, 21,36))
    cv<-as.numeric(substr(line, 37,45))
    lcl<-as.numeric(substr(line, 46,60))
    ucl<-as.numeric(substr(line, 61 ,75))
    degrees.freedom<-as.integer(substr(line, 76,91))
   }
  return(list(ok=ok,stratum=stratum,samp=samp,
              estimator=estimator, module=module, statistic=statistic,
              value=value, cv=cv, lcl=lcl, ucl=ucl, degrees.freedom=degrees.freedom))
}
remove.files<-function(file.names,bootstrap=F) {
    start <- 1
    if(bootstrap==TRUE) {
        start <- 2          #if used inside bootstrap loop, the command file is not deleted
    }
   for (i in start:length(file.names)) {
     if (file.exists(file.names[[i]])) file.remove(file.names[[i]])
   }
}
```