Recommended Methods for Outlier Detection and Calculations of Tolerance Intervals and Percentiles – Application to RMP data for Mercury-, PCBs-, and PAH-contaminated Sediments

Final Report

May 27th, 2011

Prepared for:

Regional Monitoring Program for Water Quality in San Francisco Bay, Oakland CA

Prepared by:

Dr. Don L. Stevens, Jr. Principal Statistician
Stevens Environmental Statistics, LLC
6200 W. Starr Road
Wasilla, AK 99654

Introduction

This report presents results of a review of existing methods for outlier detection and for calculation of percentiles and tolerance intervals. Based on the review, a simple method of outlier detection that could be applied annually as new data becomes available is recommended along with a recommendation for a method for calculating tolerance intervals for population percentiles. The recommended methods are applied to RMP probability data collected in the San Francisco Estuary from 2002 through 2009 for Hg, PCB, and PAH concentrations.

Outlier detection

There is a long history in statistics of attempts to identify outliers, which are in some sense data points that are unusually high or low**.** Barnett and Lewis (1994) define an outlier in a set of data as "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data. This captures the intuitive notion, but does not provide a constructive pathway. Hawkins (1980) defined outliers as data points which are generated by a different distribution than the bulk observations. Hawkins definition suggests something like using a mixture distribution to separate the data into one or more distributions, or, if outliers are few relative to the number of data points, to fit the "bulk" distribution with a robust/resistant estimation procedure, and then determine which points do not conform to the bulk distribution. This approach assumes that a suitable parametric distribution can be determined.

Previous work attempted to separate some contaminant distributions into "ambient" and "impacted" components using a mixture distribution approach. Unfortunately, the distributions did not resolve into something that could clearly be interpreted as an ambient and an impacted distribution. Furthermore, none of the data sets evaluated (Hg, PCB, and PAH) exhibited distributions that could be reasonably modeled by parametric methods. Most outlier detection methods either rely on a parametric distribution (e.g., assume underlying normality) or rely on visual inspection and interpretation of graphical representations (e.g., Tukey's box plots (Tukey, 1997)). Inasmuch as the aim of the current exercise is to identify a procedure that can be applied to future data sets with minimal human intervention, the more common outlier detection methods do not seem suitable.

Last & Kandel (2001) proposed using an approach based on fuzzy set theory (Zadeh, 1985). Their approach identifies data points that are separated from the main body of the data. It seems particularly suited for the present application, where we wish to identify and remove unusually large values. Their method uses a measure of conformity that compares the distance between a point and its next smaller neighbor to the average distance between next *m* smaller points. This is essentially a comparison of local point densities that identifies abrupt changes in point density.

The measure of conformity for value v_j used by Last and Kandel is

$$\rho_j = \frac{2}{1 + \exp\left(\frac{\beta m (v_j - v_{j-1})}{n_j (v_{j-1} - v_{j-m-1})}\right)}$$

where n_j is the number of times value v_j occurs, and β is a user-defined parameter that controls sensitivity. The default value of β is set to detect a relative difference of 10, i.e., the distance between a value and its next lower neighbor is 10 times the average distance between the next lower m values. The parameter m should be set so that a reasonably stable average density is obtained. With the data sets in hand, I set m to include 2.5% of the data or at least 12 points. A point is deemed to be non-conforming if ρ is small. Here I used the criterion $\rho \leq 0.05$.

The appendix includes R code (R Development Core Team (2009)) that implements Last & Kandel auto detect algorithm.

Percentiles and Tolerance Intervals

A percentile is a number such that a specified proportion of the population has values equal to or less than that number. As with outlier detection, there are both parametric and non-parametric methods to estimate percentiles. Because the contaminant concentrations are not easily fit with a parametric model, non-parametric methods are preferable. The simplest non-parametric estimator of a percentile is simply to sort the data in ascending order, calculate an index for the j^{th} order point given by $\frac{100j}{N}$, and take the first point with an index exceeding the target percentile as the estimate.

The above procedure works so long as the data can be considered a simple random sample from the population, but does not take into account weights that result from more complex survey designs. The procedure implemented by the USEPA's EMAP draws on an estimate of the cumulative distribution function to estimate percentiles (Diaz-Ramos, et al.,1996). The CDF is essentially a complete collection of percentiles, with a percentile being calculated for each unique data value. Specific percentiles are calculated by interpolation if they do not happen to coincide with a data value. This is the recommended procedure, because the CDF as calculated by the R survey analysis package *spsurvey* (Kincaid, et al., 2010) is appropriate for complex survey designs as well as simple random sampling.

A tolerance interval is essentially a confidence interval on a specified proportion of a population distribution. An upper tolerance limit is a number such that there is a specified level of confidence that a specified proportion of the population has values at or below that number. Figure 1 illustrates the distinction between the confidence interval around a cumulative distribution function and tolerance intervals using the data for Hg concentration. The CDF gives the proportion of the population with Hg concentrations less than or equal to the values on the x-axis. The 95% confidence limits give bounds on that estimated proportion. A 95% tolerance limit, on the other hand, is a concentration such that some specified proportion of the population is less than or equal to that concentration with 95% confidence. For example, we can estimate a 95% tolerance limit on the 90th percentile of Hg concentration by drawing a line parallel to the x-axis at the level where the CDF = 0.9, finding the intersection of that line with the lower 95% confidence limit on the CDF, and then dropping down to the corresponding Hg concentration (in this case, 0.340). This is essentially the manner in which the USEPA's EMAP estimates confidence limits on percentiles. It is the method implemented in the R survey analysis package *spsurvey* (Kincaid, et al., 2010). Because it is based on *spsurvey*'s estimate of the CDF and confidence limits, it is appropriate for complex as well as simple survey designs.

There are several other non-parametric methods available for estimating tolerance intervals (Hahn and Meeker, 1991; Wald, 19143; Wilks, 1941). These are based on the binomial distribution and assume simple random sampling. They also require large data sets to work well, especially for high confidence on extreme percentiles. Although I recommend EMAP's procedure, the Hahn & Meeker estimator (implemented in the R package *tolerance* (Young, 2009)) was also calculated for comparison. For the most part, the two estimators were in good agreement; differences showed up primarily for high confidence or high percentile tolerance limits. Only the results for the *spsurvey* method are presented here.

Results

Outlier Results

One or more outliers were identified in each of the three contaminant data sets. Three outliers were identified for Hg: site codes (CB016S), (SPB018S), and (CB044S) with values 0.610, 0.780 and 0.942, respectively. Figure 2 is a histogram of the Hg distribution with the outliers identified. Three outliers were identified for PCB: site codes SPB018S, CB034S, and SB011S with values 25.1293, 26.5817, and 29.8293, respectively. Figure 3 is a histogram of the PCB distribution with the outliers identified. Only one outlier was identified for PAH: site code CB044S with 43046.9. Figure 4 is a histogram of the PAH distribution with the outlier identified.

Percentiles & Tolerance Interval Results

All outliers were removed before this part of the analysis. Also, all non-detect values were replaced with the detection limit. Because the focus is on the upper percentiles, the actual value used for the non-detects is immaterial: it has no effect on upper percentile calculation so long as it is small.

Percentiles were calculated using the interpolation algorithm from *spsurvey*. Tolerance intervals were calculated using *spsurvey* methodology. That is, the tolerance intervals were based on cdf's and confidence limits calculated using survey weights and the variance estimator developed for Generalized Random Tessellation Stratified (GRTS) designs (Stevens & Olsen, 2003). Results for multiple percentiles and tolerance levels are presented in Tables 1 through 3. Figures 5 through 7 are histograms with outliers removed, and the with the median and 90% tolerance limit on the 90^{th} percentile identified. (NB: These histograms are based a counts, not survey weights, so they are not an unbiased representation of the population distribution. They are provided to illustrate where the tolerance limit lies relative to the sample data. The medians and tolerance limits were estimated using the survey weights.)

Table 1: Upper tolerance limits for Hg						
Percentile Level	Percentile Estimate	Confidence Level				
		80	85	90	95	99
80	0.300	0.301	0.302	0.302	0.305	0.310
85	0.309	0.313	0.314	0.317	0.321	0.328
90	0.328	0.332	0.333	0.334	0.340	0.343
95	0.347	0.351	0.351	0.352	0.357	0.364
99	0.440	0.468	0.470	0.472	0.474	0.478

Table 2: Upper tolerance limits for PCB						
Percentile Level	Percentile Estimate	Confidence Level				
		80	85	90	95	99
80	9.6	10.0	10.2	10.2	10.3	11.0
85	10.7	11.6	11.8	12.0	12.1	12.4
90	12.4	13.7	14.0	15.7	15.8	15.9
95	16.6	18.0	18.1	18.2	18.3	18.5
99	19.0	19.4	19.4	19.5	19.5	19.6

Table 3: Upper tolerance limits for PAH						
Percentile Level	Percentile Estimate	Confidence Level				
		80	85	90	95	99
80	3488	3517	3517	3531	3540	3815
85	3828	3904	3963	4072	4182	4357
90	4476	4556	4690	4847	5062	5276
95	6203	6483	6643	6837	7742	9155
99	12461	16594	16822	17057	17332	17695

References

Diaz-Ramos, S., D.L. Stevens, Jr., and A.R. Olsen. (1996). *EMAP Statistical Methods Manual.* EPA/620/R-96/XXX. Corvallis, OR: U.S. Environmental Protection Agency, Office of Research and Development, National Health Effects and Environmental Research Laboratory, Western Ecology Division.

Hahn, G. J. and Meeker, W. Q. (1991), Statistical Intervals: A Guide for Practitioners, Wiley-Interscience.

Kincaid, Tom, Tony Olsen with contributions from Don Stevens, Christian Platt, Denis White and Richard Remington (2010). spsurvey: Spatial Survey Design and Analysis. R package version 2.1.2. http://www.epa.gov/nheerl/arm/

Last, A, and M. Kandel (2001) Automated Detection of Outliers in Real-World Data. Proc. of the Second International Conference on Intelligent Technologies, pp292-301

R Development Core Team (2009). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN3-900051-07-0, URL http://www.R-project.org.

Stevens, Jr., D.L., and A.R. Olsen (2003). Variance estimation for spatially balanced samples of environmental resources. Environmetrics **14**: 593–610

Tukey J (1977). Exploratory Data Analysis, Reading, MA: Addison-Wesley.

USEPA (1990) Statistical Analysis of Groundwater Monitoring Data At RCRA Facilities: Unified Guidance. USEPA Office of Resource Conservation and Recovery EPA 530/R-09-007

Wald, A. (1943), An Extension of Wilks' Method for Setting Tolerance Limits, The Annals of Mathematical Statistics, 14, 45–55.

Wilks, S. S. (1941), Determination of Sample Sizes for Setting Tolerance Limits, The Annals of Mathematical Statistics, 12, 91–96.

Young, D.D. (2009). tolerance: Functions for calculating tolerance intervals.. R package version 0.1.0. http://CRAN.R-project.org/package=tolerance

Zadeh, L.A. (1985). Syllogistic Reasoning in Fuzzy Logic and its Application to Usuality and Reasoning with Dispositions. IEEE Transactions on Systems, Man, and Cybernetics, SMC-15, 6, 754-763.

Figure 1: CDF of Hg with Lower 95% Confidence Limit

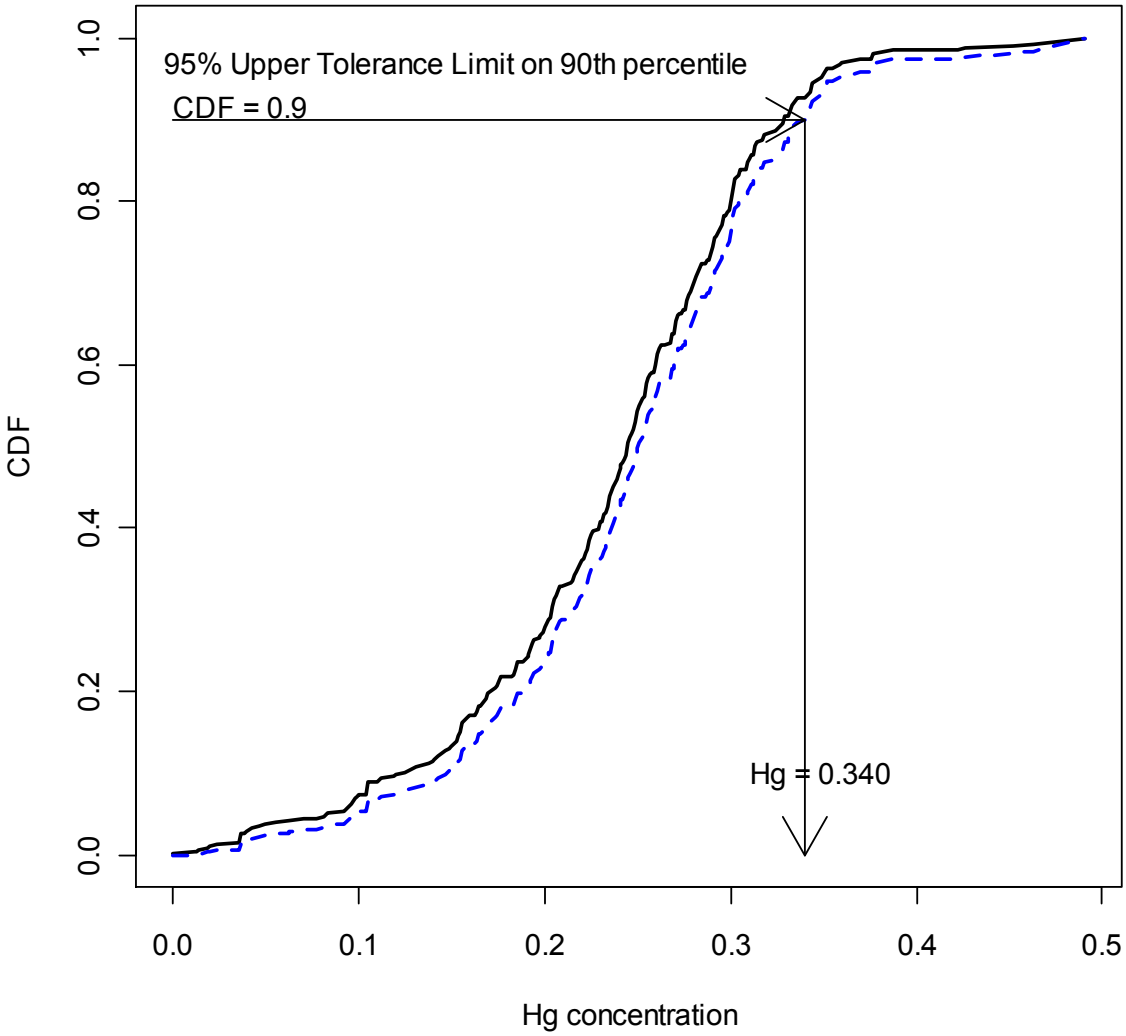


Figure 2: Histogram of Hg with outliers identified

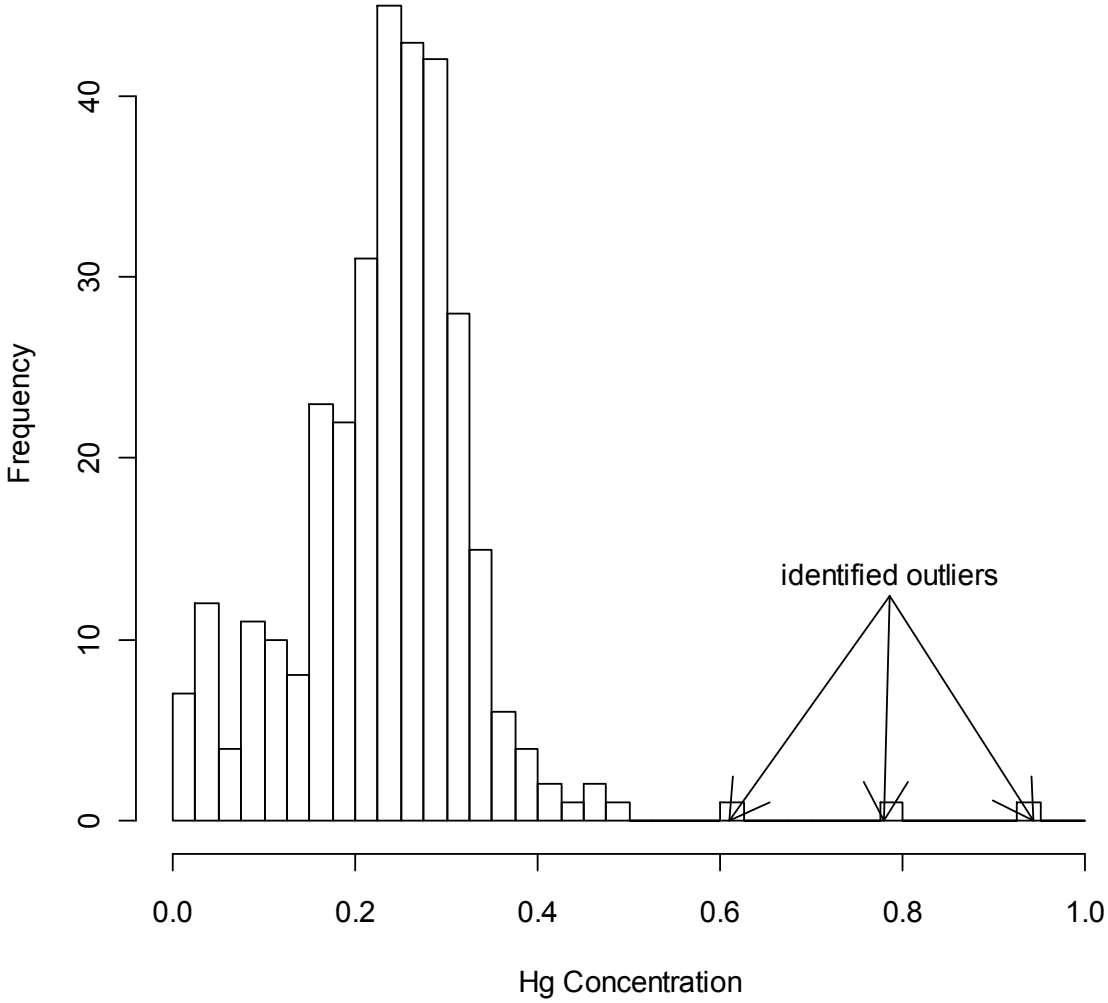


Figure 3: Histogram of PCB with outliers identified

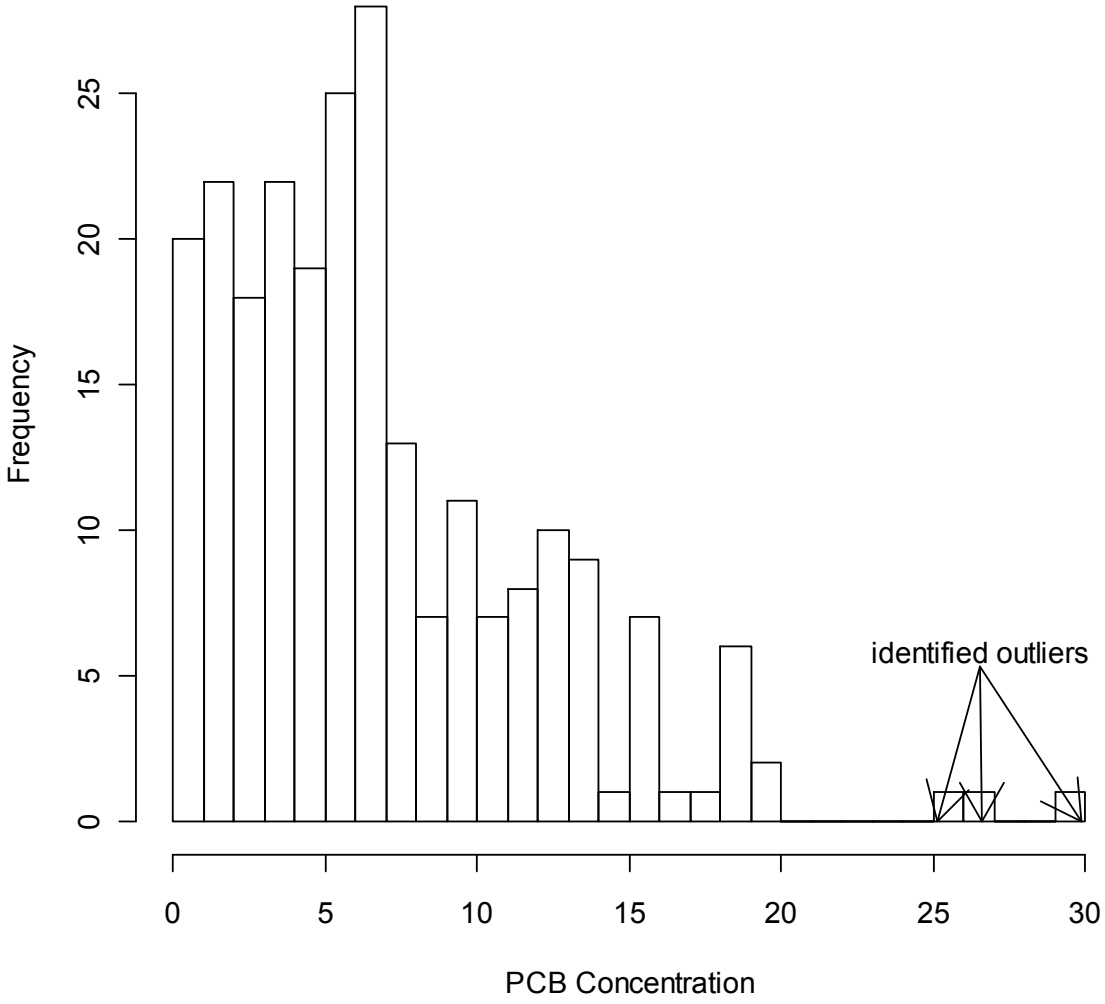


Figure 4: Histogram of PAH with outlier identified

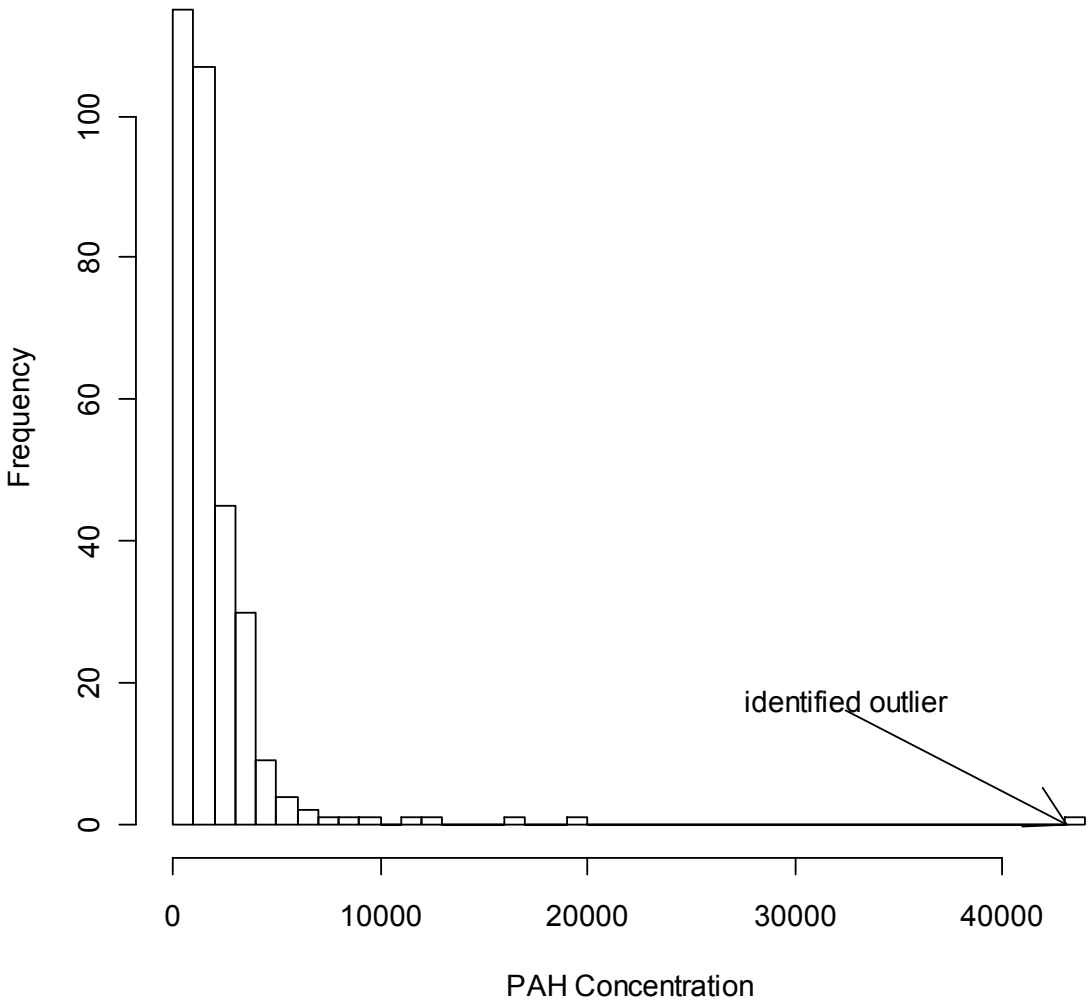
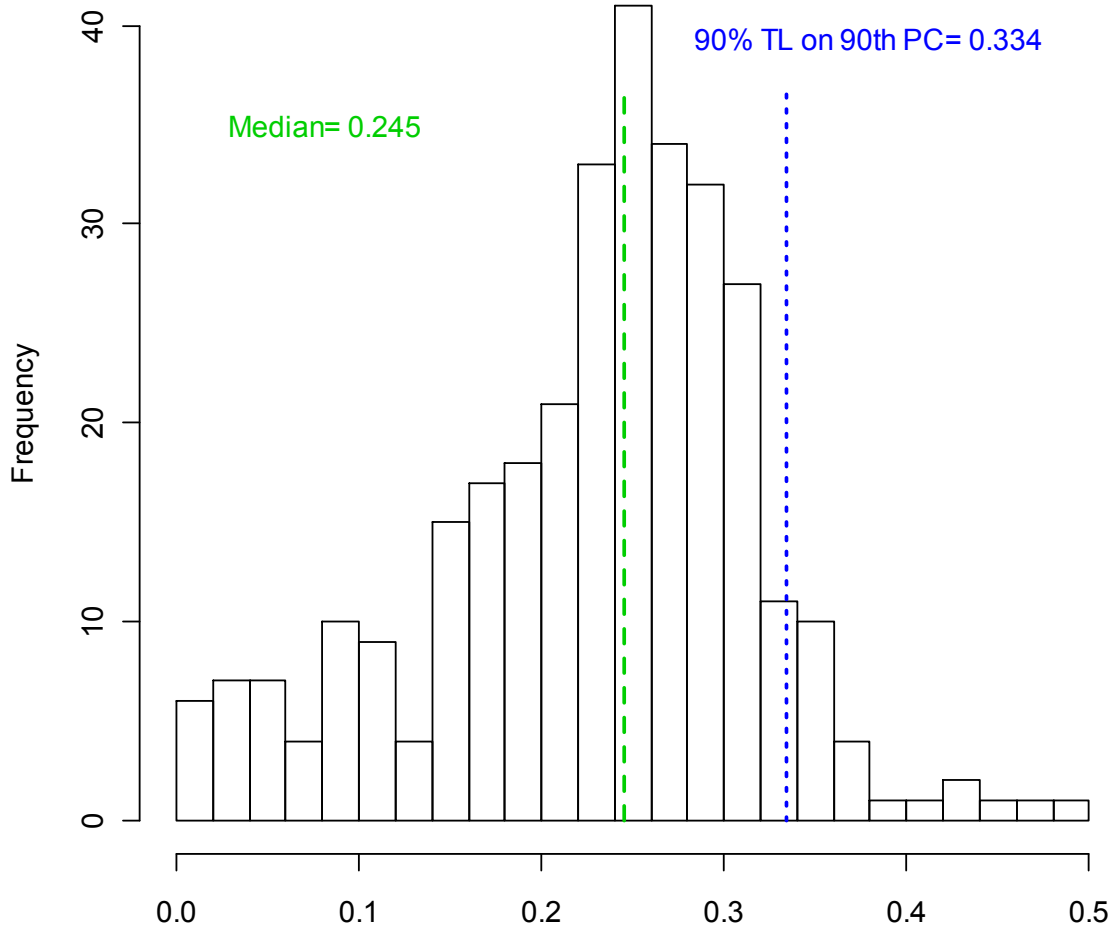


Figure 5: Histogram of Hg with outliers removed



Median and 90% Tolerance Limit on 90th Percentile are identified

Figure 6: Histogram of PCB with outliers removed

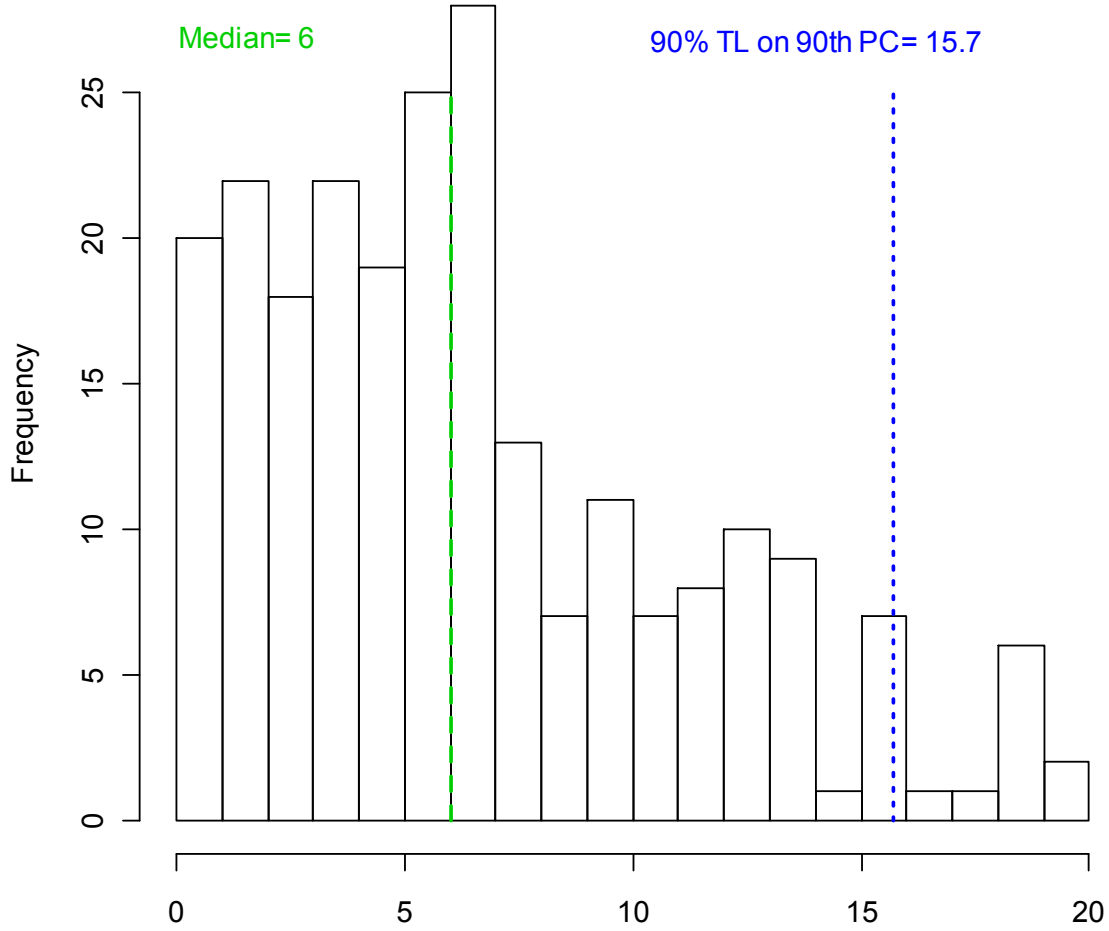
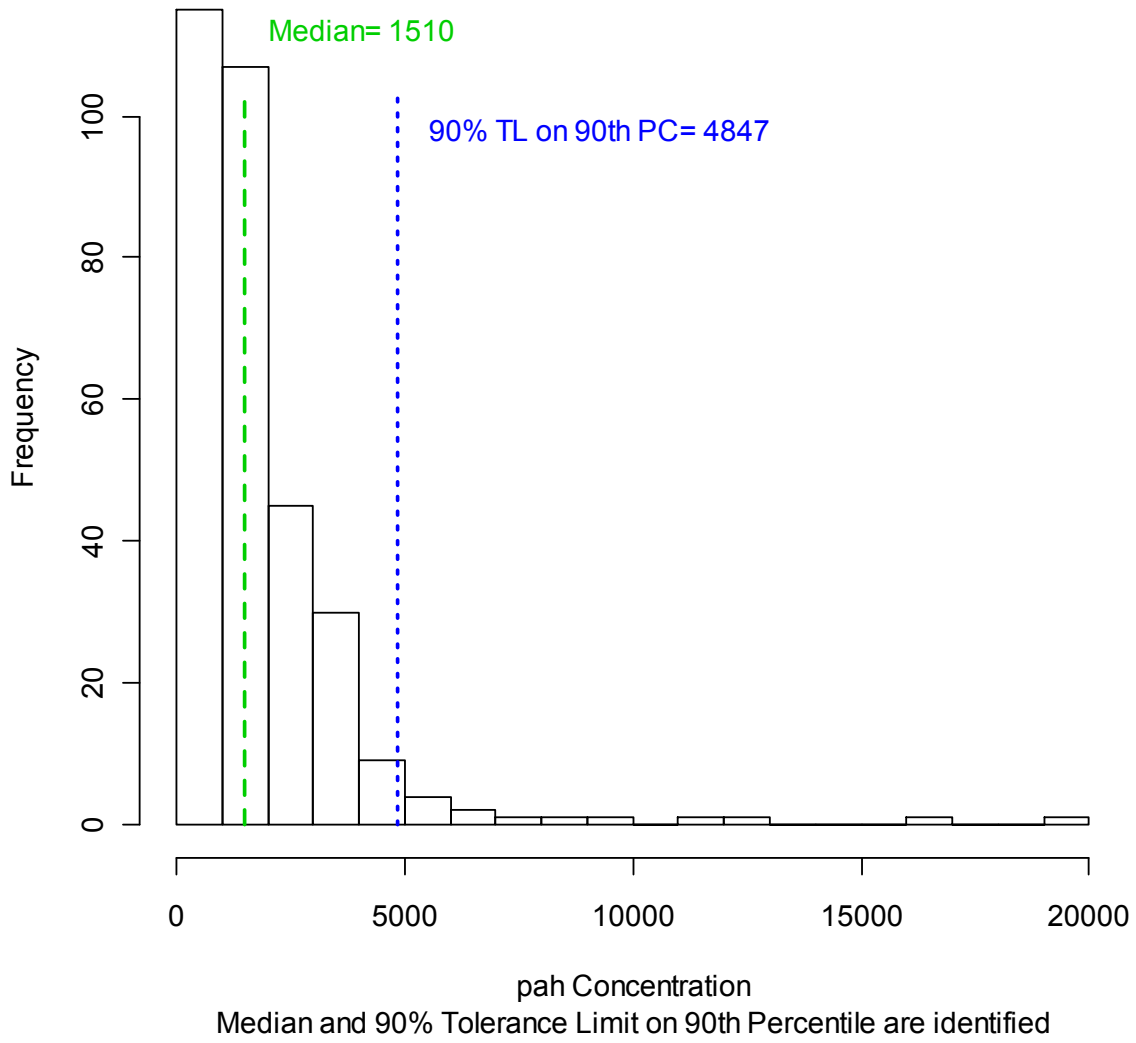


Figure 7: Histogram of pah with outliers removed



Appendix: R code for outlier detection & tolerance interval calculation

A data file must be in the directory in which R was opened. Data can be downloaded from the Web Query Tool: http://www.sfei.org/tools/wqt.

Select the following options from the Web Query Tool interface

Search Parameters:
Test Material:
Sediment
Program/Project:
Regional Monitoring Program
Start Year:
2002
End Year:
2009

Then either:

Parameter Type:		
Trace Elements	Polychlorinated Biphenyls (PCB)	Polycyclic Aromatic Hydrocarbons (PAH)
Parameter:		
Mercury	Sum of 40 PCBs (SFEI)	Sum of PAHs (SFEI)

Save the file as "Hg_Sediment_2002-2009_out.xls", or a similar name, in the same directory as the source code. Open to the sheet "Results – flat file" and save the worksheet as a ".csv" by the same name. R can read Excel files directly, but it's easier to first save as a .csv file.

The lines below constitute a script in R – they can be copied directly into the command line interface. Beforehand, the custom functions must be saved in the working directory with the file names "auto_detect_outlier.R" and "albers_proj.R", as indicated. The packages *spsurvey* and *tolerance* (and any packages they are dependent on) must be installed as well. Parameters above the "Do Not Adjust Code Below this Line" mark should be adjusted as necessary.

```
##################### R code below ######################

# Script: Calculation of Ambient Sediment Thresholds

# based on code from Don Stevens' report: "Recommended Methods for Outlier
# Detection and Calculations of Tolerance Intervals and Percentiles -
# Application to RMP data for Mercury-, PCBs-, and PAH-contaminated Sediments"

# created May 2011
# revised 6/6/2011

# load packages needed for the custom functions
require("spsurvey", quietly=TRUE)
require("tolerance", quietly=TRUE)
```

```
# set working directory
setwd('S:\\RMP Documents\\Ambient Sediment Conditions discussion\\thresholdCalculation')

# load custom functions
source('auto_detect_outlier.R')
source('albers_proj.R')

# load in data
sed.data <- read.csv("Hg_Sediment_2002-2009_out.csv")
# The value of the contaminant concentration is in the field named Result

# set user parameters
analyte_name <- 'Hg' # options = 'Hg' or 'PCB' or 'PAH'

# set the percentile levels and the confidence intervals
tolval <- c(80, 85, 90, 95, 99)
conf <- c(80, 85, 90, 95, 99)

#---- Do Not Adjust Code Below this Line------#
# Set analyte names from data sets based on user parameter
if (analyte_name == "Hg") {
        analyte = 'Mercury'
} else if (analyte_name == "PCB") {
        analyte = c('Sum of PCBs (SFEI)', 'Sum of 40 PCBs (SFEI)')
  # Sum of 209 PCBs should not be included
  # prior to 2009, all sums of PCBs were sum of 40 PCBs
} else if (analyte_name == "PAH") {
        analyte = 'Sum of PAHs (SFEI)'
} else {
        analyte = NULL
}

# convert results to numeric if data is not read in as "numeric"
if (!is.numeric(sed.data$Result)) {
        lev <- sub(",","",levels(sed.data$Result)[as.integer(sed.data$Result)]); # take out
commas from the results
        sed.data$Result <- as.numeric(lev);
}

# extract year from Cruise Number
ychar <- substr(levels(sed.data$Cruise.Number)[as.integer(sed.data$Cruise.Number)],1,4);
sed.data$Year <- as.numeric(ychar);

# remove 2002 from PCB data - data is not compatible
if (analyte_name == "PCB") {
        idx <- which(sed.data$Year != "2002")
        sed.data <- sed.data[idx,]
}
```

```
 # remove nontarget sample frames
tst <- sed.data$Region == 'Southern Sloughs';
sed.data <- sed.data[!tst,];
tst <- sed.data$Region == 'Rivers';
sed.data <- sed.data[!tst,];
tst <- sed.data$Region == 'Carquinez Strait';
sed.data <- sed.data[!tst,];

# remove historical stations (based on B... site code)
hist <- substr(levels(sed.data$Site.Code)[as.integer(sed.data$Site.Code)],1,1);
tst <- which(hist == "B");
if (length(tst) != 0) sed.data <- sed.data[-tst,];

# remove non-target parameters
contam <- levels(sed.data$Parameter)[as.integer(sed.data$Parameter)]
tst <- which(contam == analyte[1]| contam == analyte[2]);
sed.data <- sed.data[tst,];

# find number of data points
nr <- nrow(sed.data)

                # define frame area for sediment in sq km
frameArea <- c('Lower South Bay'=7.642,
                                        'South Bay'=185.171,
        'Central Bay'=396.442,
                                        'San Pablo Bay'=226.821,
        'Suisun Bay'=80.357);
                        # 'Carquinez Strait'=21.289, 'Southern Sloughs'=1.733,'Rivers'=16.478);

# compute the weight for each sample, based on frame area
idx.region <- match(sed.data$Region, names(frameArea))
sed.data.num <- table(sed.data$Region)
sed.region <- match(sed.data$Region,names(sed.data.num) )
sed.wt <- frameArea[idx.region]/sed.data.num[sed.region]


  # Convert lat/long to equal-area projection.  (Albers, in this case)
clon <- -122
clat <- 38
sp1 <- 37
sp2 <- 40
sed.xy <- albxy(sed.data$Actual.Latitude, sed.data$Actual.Longitude,
    clon = clon, clat=clat, sp1 = sp1, sp2 = sp2)

# Find the non-detects, and replace with MDL
idx <- which(is.na(sed.data$Result))
sed.data$Result[idx]<- sed.data$MDL[idx] # replace non detects with MDL

# find the outliers
o_idx <-auto_detect_outlier.fcn(x=sed.data$Result)
```

```
# "o_idx" contains the indices of any points identified as outliers (or is NULL).

# Now calculate cdf & spsurvey-type tolerance limits for confidence intervals and percentiles as
set above
if (!is.null(o_idx)) {
                data.cdf.tol <- cdf.tol.est.fcn(sed.data$Result[-o_idx],conf=conf,
                   tolval = tolval, vartype = "Local",
   x = sed.xy[-o_idx,1], y = sed.xy[-o_idx,2], wt = sed.wt[-o_idx])
} else {
                data.cdf.tol <- cdf.tol.est.fcn(sed.data$Result,conf=conf,
                   tolval = tolval, vartype = "Local",
   x = sed.xy[,1], y = sed.xy[,2], wt = sed.wt)

}
data.cdf <- data.cdf.tol$cdf

# For reference, calculate the Hahn-Meeker tolerance limits
data.hm.tol <- matrix(0,nrow = length(tolval),ncol = length(conf) +1)
data.hm.tol[,1] <- data.cdf.tol$tol[,1,1]
pctval <- as.character(tolval/100)
dimnames(data.hm.tol) <- list( pctval,c("PCT", conf))
if (!is.null(o_idx)) {
   for(j in 1:length(conf)) {
      for(i in 1:length(tolval)) {
         tmp <- nptol.int(sed.data$Result[-o_idx],P=tolval[i]/100, alpha = 1-conf[j]/100,
         method="HM")[[4]]
         data.hm.tol[i,j+1] <- tmp
   }}
 } else {
   for(j in 1:length(conf)) {
      for(i in 1:length(tolval)) {
         tmp <- nptol.int(sed.data$Result,P=tolval[i]/100, alpha = 1-conf[j]/100,
         method="HM")[[4]]
         data.hm.tol[i,j+1] <- tmp
   }}
 }
 data.hm.tol

# This table has the percentile level in the first column, the percentile value
# in the second column, and upper tolerance limits in the succeeding columns
# for the confidence levels in the top row.
# print out 2 significant figures for regulatory threshold
signif(cbind(data.cdf.tol$tol[,1,1],data.cdf.tol$tol[,2,]),digits =2)
# round to the appropriate number of digits for easy viewing
ndig <- switch(analyte_name, "Hg" = 3, "PCB" = 1, "PAH" = 0)
round(cbind(data.cdf.tol$tol[,1,1],data.cdf.tol$tol[,2,]),ndig)
```

```
 #auto_detect_outlier.R
auto_detect_outlier.fcn <- function(x,m =NULL,alpha =0.05, beta = NULL, dif.detect = 10) {
# detect outliers in the vector v by comparing lag 1 difference to
#   lag m difference
# dif.detect controls sensitivity to the relative distance magnitude. Default
#  value of 10 detects a relative magnitude of 10, e.g., a difference that is
#  10 times the local average difference.
#
# alpha controls the level of conformity that is deemed to be outlying. Lower
#  values will cause fewer values to be recognized as outliers.
# default value for m is at least 12 or ceiling(length(x)*0.025),
#  i.e., about 2.5% of data
# function returns the indices of high outliers, or NULL if none are detected

 if(is.null(m)) m <- max(12, ceiling(length(x)*0.025))
 if(is.null(beta)) beta <- log(2/alpha -1)/dif.detect
 ord <- order(x)
 sx <- x[ord]
 tst <- tapply(sx, sx)
 tbx <- table(x)
 v<- unique(sx)
 nv <- length(v)
 nv1<- nv-1
 nm <- nv-m
 cfl <- cfh <- rep(1, nv)
 difl <- diff(v)
 difm <- (v[-(1:(m))]-v[1:nm]) /m
 cfh[(m+2):nv] <- 2/(1+exp(beta*difl[(m+1):nv1]/(tbx[(m+2):nv]*difm[-nm])))
 idx <-which(cfh < alpha)
 if(length(idx)==0) return(NULL) else return(ord[match(min(idx):nv,tst)])

 }
# CDF, percentile, & tolerance interval calculation

cdf.tol.est.fcn <-function(z, conf=95,tolval=95,wt=NULL,vartype = "SRS",
  zrng=NULL,x=NULL, y=NULL ) {
# z   vector of observed values
# conf a single value or a vector of confidence levels
# tolval a single value or vector of percentile levels
# wt  a vector of same length as z with survey weight values. The default
#  value NULL results in equal weighting
# vartype specifies type of variance calculation. Default uses the SRS
#  variance estimator (see package spsurvey documentation for more details)
#  the alternative is "Local" which uses the local variance estimator. If
#  the local estimator is used, x and y coordinates must be supplied.
# zrng is vector of values at which the cdf is estimated. Default uses
#  the sorted unique values of z
# x, y are coordinates of the z observations. Only needed if vartype = "Local"
#
```

```
#  gets estimate of the cumulative distribution function, its standard deviation,
#  and 1-sided lower confidence limits.
# Also estimates percentiles and upper tolerance limits
# confidence limits will be estimated for all levels specified in conf
# Returned value is a list with components "CDF" and "tol". CDF is a matrix
#   with values of the cdf and upper confidence limits; tol is a three dimensional
#   array row = percentile, column = tolerance limits, and sheet = confidence
#
 if(vartype =="Local" & (is.null(x) | is.null(y) )) {
   return("x & y coordinates must be supplied for local variance estimator")
   }
 conf <- conf/100
 tolval <- tolval/100
 n <- length(z)
 if(is.null(zrng)) zrng <- sort(unique(z))
 m <- length(zrng)
 ym <- matrix(rep(zrng, n), nrow = n, byrow = T)
 zm <- matrix(rep(z, m), nrow = n)
 if(is.null(wt)) wt <- rep(1, length(z))
 wm <- matrix(rep(wt, m), nrow = n)
 cdf <- apply(ifelse(zm <= ym, wm, 0), 2, sum)/sum(wt)
 tw2 <- (sum(wt))^2
 im <- ifelse(matrix(rep(z, m), nrow = n) <= matrix(rep(zrng, n), nrow = n,
   byrow = T), 1, 0)
 rm <- (im - matrix(rep(cdf, n), nrow = n, byrow = T)) * matrix(rep(wt, m),
   nrow = n)
 if (vartype == "Local") {
   weight.lst <- localmean.weight(x, y, 1/wt)
   varest <- apply(rm, 2, localmean.var, weight.lst)/tw2
   } else {
   varest <- n * apply(rm, 2, var) / tw2
    }
 sd <- sqrt(varest)
 mult <- qnorm(conf)
 cint <- matrix(0,nrow =m,ncol=length(mult))
 for(i in 1:length(mult)) {
  cint[,i] <- pmax(0,cdf - sd*mult[i])
  }
 CDF <- cbind(cbind(zrng, cdf, sd, cint) )
 dnm <- paste(100*conf, "%UCB",sep = "")
 dimnames(CDF) <- list(NULL, c("Value", "CDF", "SD",as.vector(t(dnm))))
 tol <- array(0, c(length(tolval), 2,length(conf)))
 dimnames(tol) <- list(100*tolval, c("PCT","UPPER TL"),100*conf)
 for (j in 1:length(conf)) {
   tol[,,j] <- pctol.est.fcn(cbind(zrng, cdf,cint[,j]),tolval)
  }
 list(cdf =CDF, tol=tol)
 }
```

```
pctol.est.fcn <- function(cdfest, tolpct) {
# calculates percentile & upper tolerance liimit
# input is estimated cdf with upper confidence limit, and vector of percentiles
rslt <- matrix(0, nrow=length(tolpct),ncol=2)
for(i in 2:3) {
 for (j in 1:length(tolpct)) {
 hdx <- which(cdfest[,i] >= tolpct[j])
 high <- ifelse(length(hdx) >0, min(hdx), NA)
 ldx <- which(cdfest[,i] <= tolpct[j])
 low <- ifelse(length(ldx) >0, max(ldx), NA)
 if (is.na(high)) {
  rslt[j,i-1] <- NA
       }  else if (is.na(low)) {
  rslt[j,i-1] <- cdfest[high,1]
   } else {
     if (high > low)
      ival <- (tolpct[j] - cdfest[low,i])/ (cdfest[high,i] - cdfest[low,i])
       else ival <- 1
      rslt[j,i-1] <- ival * cdfest[high,1] + (1 - ival) * cdfest[low,1]
       }
}
}
 rslt
 }
```

```
# albers_proj.R

  albxy <<- function(lat, lng,sph = "Clarke1866", clon = -96, clat = 23, sp1 = 29.5,
    sp2 = 45.5)
{
    if (sph == "Clarke1866") {
        a <- 6378206.4
        b <- 6356583.8
    }
    else if (sph == "GRS80") {
        a <- 6378137
        b <- 6356752.31414
    }
    else if (sph == "WGS84") {
        a <- 6378137
        b <- 6356752.31424518
    }
    else {
        stop("\nSpheroid does not match available options")
    }
    RADDEG <- (180/pi)
    DEGRAD <- (pi/180)
#
                #  ec = eccentricity = sqrt(1-(b/a)^2)
                #
                ec <- sqrt(1-(b/a)^2)
                dgrd <- pi/180.
                ph0 <- clat * dgrd
                ph1 <- sp1 * dgrd
                ph2 <- sp2 * dgrd
                l0 <- clon * dgrd
                q0 <- alb.que(ph0,ec)
                q1 <- alb.que(ph1,ec)
                q2 <- alb.que(ph2,ec)
                m0 <- alb.em(ph0,ec)
                m1 <- alb.em(ph1,ec)
                m2 <- alb.em(ph2,ec)
                lat <- lat * dgrd
                lng <- lng * dgrd
                q <- alb.que(lat, ec )
                m <- alb.em(lat, ec)
                n <- (m1^2. - m2^2.)/(q2 - q1)
                cn <- m1^2. + n * q1
                r0 <- (a * sqrt(cn - n * q0))/n
                th <- n * (lng -l0)
                r <- (a * sqrt(cn - n * q))/n
                x <- r * sin(th)
                y <- r0 - r * cos(th)
                cbind(x, y)
}
```

```
alb.em <- function(z, ec )
{
		cos(z)/sqrt(1. - (ec * sin(z))^2.)
}


alb.que <- function(z, ec )
{

		snlt <- sin(z)
		esnlt <- ec * snlt
		(1. - ec^2.) * (snlt/(1. - esnlt^2.) - logb((1. - esnlt)/(1. + esnlt))/(2. * ec))
}


 albersgeod <<-
function (x, y, sph = "Clarke1866", clon = -96, clat = 23, sp1 = 29.5,
   sp2 = 45.5)
{
   if (sph == "Clarke1866") {
      a <- 6378206.4
      b <- 6356583.8
   }
   else if (sph == "GRS80") {
      a <- 6378137
      b <- 6356752.31414036
   }
   else if (sph == "WGS84") {
      a <- 6378137
      b <- 6356752.31424518
   }
   else {
      stop("\nSpheroid does not match available options")
   }
   RADDEG <- (180/pi)
   DEGRAD <- (pi/180)
   clat <- clat * DEGRAD
   clon <- clon * DEGRAD
   sp1 <- sp1 * DEGRAD
   sp2 <- sp2 * DEGRAD
   e2 <- 1 - (b * b)/(a * a)
   e4 <- e2 * e2
   e6 <- e4 * e2
   e <- sqrt(e2)
   t1 <- 1 - e2
   t2 <- 1/(2 * e)
   sinlat <- sin(clat)
   t3 <- 1 - e2 * sinlat * sinlat
   q0 <- t2 * log((1 - e * sinlat)/(1 + e * sinlat))
```

```
    q0 <- t1 * (sinlat/t3 - q0)
    sinlat <- sin(sp1)
    t3 <- 1 - e2 * sinlat * sinlat
    q1 <- t2 * log((1 - e * sinlat)/(1 + e * sinlat))
    q1 <- t1 * (sinlat/t3 - q1)
    m1 <- cos(sp1)/sqrt(t3)
    sinlat <- sin(sp2)
    t3 <- 1 - e2 * sinlat * sinlat
    q2 <- t2 * log((1 - e * sinlat)/(1 + e * sinlat))
    q2 <- t1 * (sinlat/t3 - q2)
    m2 <- cos(sp2)/sqrt(t3)
    n <- (m1 * m1 - m2 * m2)/(q2 - q1)
    C <- m1 * m1 + n * q1
    rho0 <- a * sqrt(C - n * q0)/n
    rho <- sqrt(x * x + (rho0 - y) * (rho0 - y))
    theta <- atan(x/(rho0 - y))
    q <- (C - (rho * rho * n * n)/(a * a))/n
    lon <- clon + theta/n
    lat <- asin(q/(1 - (t1/(2 * e)) * log((1 - e)/(1 + e))))
    s2 <- sin(2 * lat) * (e2/3 + 31 * e4/180 + 517 * e6/5040)
    s4 <- sin(4 * lat) * (23 * e4/360 + 251 * e6/3780)
    s6 <- sin(6 * lat) * (761 * e6/45360)
    lat <- lat + s2 + s4 + s6
    data.frame(lon = lon * RADDEG, lat = lat * RADDEG)
}
```