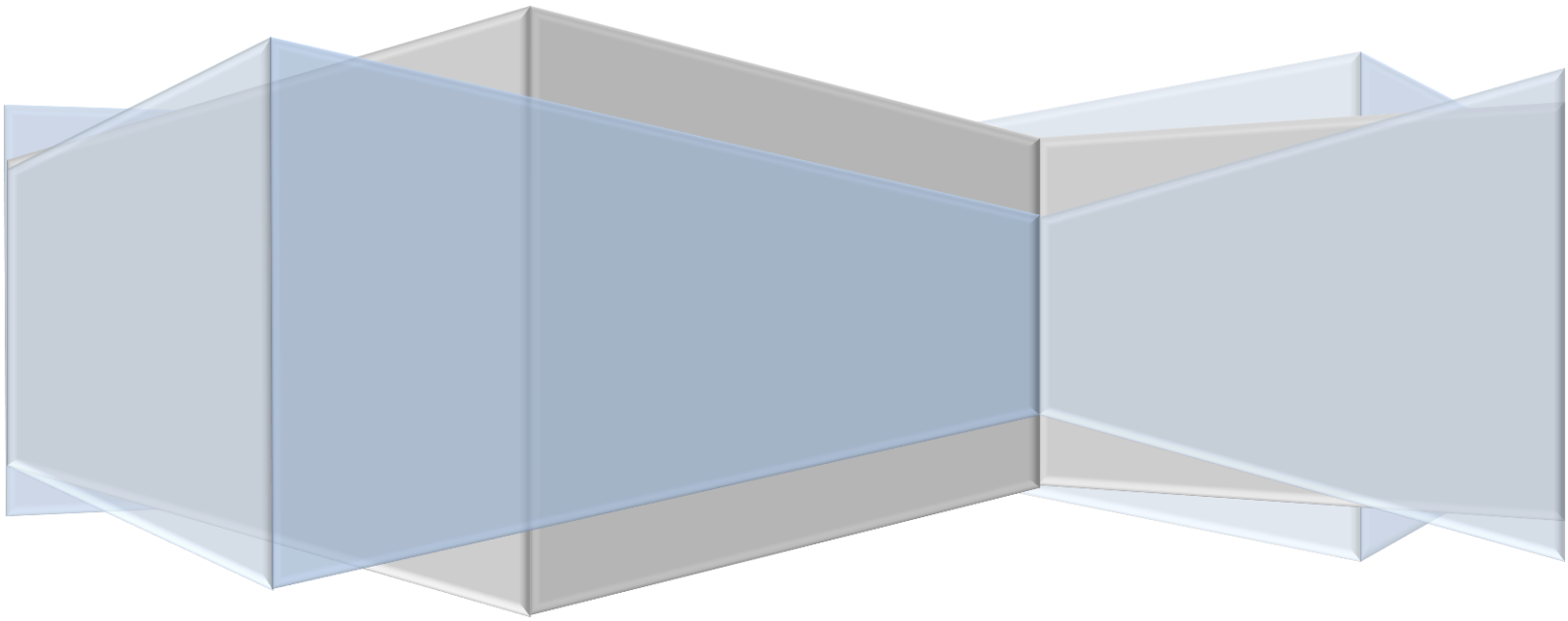
South Fork Research, Inc.

CHaMP / ISEMP Data Analysis User's Guide

For Spsurvey R-Package Analysis Functions

Matt Nahorniak
matt@southforkresearch.org
(541) 740-5487



Contents

1. Introduction

The Integrated Status and Effectiveness Monitoring Program (ISEMP) and its companion project, the Columbia Habitat Monitoring Program (CHaMP), provide long term monitoring of anadromous salmonid population and habitat status in the Columbia River Basis. These monitoring programs extensively utilize probability based survey designs, such as Generalized Random Tessellation Stratified (GRTS) designs, that enable efficient sampling of a spatially correlated resource, such as fish density by location in a stream network. The R package **spsurvey** includes a set of analysis tools created specifically for design based analysis of probability samples such as those obtained by GRTS designs.

This manual is intended as a user's guide, specifically geared toward users of ISEMP / CHaMP data sets, for using the spsurvey tools to analyze and draw inference from sampled data. It is assumed that the readers, while having some experience using R, are not R experts, nor statisticians.

There are excellent user's guides and examples already available for using spsurvey tools [3, 7]. This document, however, is geared specifically toward ISEMP /CHaMP data users. As such, it will attempt to address specific issues and points of confusion commonly encountered by these users, as well provide some guidance that may be more accessible to non-statisticians than that typically found in on-line documentation of R packages. In addition, the R code provided by this guide may be useful as template for those analyzing ISEMP / CHaMP data. Rather than create scripts from scratch, users may wish to simply modify the code included here.

The spsurvey package includes functions to assist in both generating probability based sampling plans as well as functions used to analyze data collected from such plans. This document will focus on use of the analysis functions used after a sampling is completed. The main spsurvey functions that CHaMP / ISEMP users will use to analyze data are **cont.analysis** and **cat.analysis**. The function **cont.analysis** is used to analyze continuous variables (i.e. estimate population means, quantiles, corresponding confidence intervals, etc.), while **cat.analysis** is used to analyzing categorical response variables. In addition to estimating distributions of site level metrics, these tools can be used to estimate population totals for metrics such as total number of fish across the sample frame, or subgroup of the sample frame.

The R package spsurvey is authored by Tom Kincaid and Tony Olsen, with contributions from Don Stevens, Christian Platt, Denis White, and Richard Remington. It is currently maintained and updated by Tom Kincaid. For up to date details, refer to the cran.r-project.org page at: http://cran.r-project.org/web/packages/spsurvey/index.html [1].

2. General Background Statistics

While not intended as a full treatise on sampling, there are a few basic concepts that are useful for the spsurvey user to review. These concepts are key to understanding the advantages and limits of probability based sampling analysis.

Design Based Inference versus Model Based Inference

Design based inference, in the context of sampling, seeks to describe and draw conclusions about a population based on the elements of a sample. Conversely, model based inference seeks to estimate parameters that describe the relationship between variables in an assumed or derived model. Probability surveys, such as GRTS sampling, and the spsurvey functions described here, serve to facilitate *design based inference*.

In terms of CHaMP / ISEMP data, for metrics such as fish density (measured by site, within a stream network), design based inference will yield estimates and uncertainties of fish densities across all sites within the sampling frame. Additionally, it can be used to estimate the total abundance over the entire sampling frame, or within subsets of sites.

The spsurvey tools do NOT provide tools for model based inferences. For example, a researcher may posit a model suggesting fish abundance is a linear function of stream power, habitat characteristics, and water temperature. Model based inferential tools are needed to estimate model coefficients for such a model. The spsurvey functions do NOT enable estimations of coefficients for such models.

While GRTS samples and spsurvey tools are geared toward design based analysis, that does not preclude a design based analysis of "meta-data" that are actually model based parameter estimates from individual sites. For example, a researcher may be interested in the change in fish density over time. The spsurvey functions (or design based analyses in general) do not allow for the direct estimate of a slope parameter for the change in fish density per unit of time. However, if multiple years of data exist at each site measured, a model based estimate of this slope could be calculated for each site. The set of slopes, by site, could then be examined via a design based analysis using the spsurvey analysis tools, just as if slope (change in fish abundance over time) were a single measurement at each site.

Additional Definitions

- **Probability Based Sampling**: Random sampling, where each item in a sampling frame has an a-priori probability of being selected in the sample. In simple random sampling,

all elements in a sampling frame have equal probability of selection. In stratified, or variable probability sampling, elements within a sampling frame may have differing probability of being sampled. All are nevertheless examples of probability based sampling.

- **Sampling Frame:** A sampling frame is the source containing all elements from which samples may be drawn. Ideally, the sampling frame contains each element within a population of interest, and contains no additional elements. In practice, a perfect sampling frame is rarely achieved. In CHaMP / ISEMP sampling, a list of all stream segments in all streams in a watershed, often in the form of a shapefile, is often used as the sampling frame.

- **Sample:** The sample refers to the elements of the sampling frame that were selected via a probability based sampling method. In CHaMP / ISEMP samples, the sample is simply the list of sites that are selected for measurements. These are generally sampled using a GRTS sampling procedure.

3. R Basics

Included below is a brief description of how to find R for download and how at install the spsurvey package of functions, as well as a brief description of data types commonly used in the spsurvey functions (as well as in most R functions). For beginner R users, it is recommended that the user walk through one of the tutorials available at the link below.

Installing R

The spsurvey analysis tools are available to be used within the R programming language. R is a free, open source programming language designed specifically for statistical computing and graphics. The software and documentation are available free at: http://www.r-project.org/.

Common R Data Structures: Vectors, Lists, and Data frames

It is assumed that the reader has some rudimentary experience programming in R and using basic R functions and data structures. If not, it is suggested to review some of the on-line documentation and examples available. Nevertheless, it may be useful here to describe to data structures used extensively in the spsurvey tools: vectors, lists, and data frames.

Vectors

Vectors, as data structures in R, are simply ordered lists of data. The can be created using the combine function "c", as in:

My.vector= c(5, 7, 33).

Optionally, items in vectors can be named using "names" function or by assigning names within combine function:

My.vector=c ("y1" = 5, "y2" = 7, "y3"=33)

Components of a vector are accessed using square brackets after the vector name, with the index position in the bracket. For example, if "My.vector" is as above, then "My.vector[3]" returns the value 33.

Lists

A list is similar to a vector in R, the main difference that the components of a list need not be single values, but can be data structures of any sort. For example, a list might contain a list of named vectors, or even a list of lists. It addition, components of a list are typically named, as in the following:

My.list = list("one"=1, "two"=2, "three"=3)

Components of a list can be accessed as are vectors using square brackets and an index number, or by using the "$" symbol to specify the name of the component within the list being accessed. For example, "*My.list$two*" returns the numeric value 2.

Data frames

Data frames are general and flexible data structures useful for passing large amounts of related information to and from functions in R. A data frame is similar to a list, except that the components can be of multiple types within the same data frame. For example, a single data frame could contain a vector, a list, an single (atomic) variable, and even data frames within the data frame. To access items within a data frame, the "$" symbol is used, as in accessing components of a list by name, shown above.

Note: the "attach" function can be used to bring components of a data frame into directly accessible memory. For example, if a data frame called "my.data.frame" contained a vector "my.vector", and a list "my.list", they could be accessed either as: *my.data.frame$my.vector* and *my.data.frame$my.list*, or by "*attach(my.data.frame)* and then directly accessing "*my.vector*" and

"*my.list*". While this can be useful when items in a data frame are accessed repeatedly, caution should be used with the attach function, as using it multiple times, or failing to "detach" data, can results in data masking issues and often causes great confusion. When in doubt, avoid using the "attach" command.

4. Spsurvey Package Overview

General Description

The spsurvey package is one of numerous add-on packages available within the R environment.

Installing spsurvey

To download and install spsurvey, open R and perform the following steps: click on "install package" and select a CRAN mirror site (any site should work; it may be most efficient to pick one geographically close to your current location). Scroll down and select the package "spsurvey", after which installation will take place automatically. Installation of this package need only be done once for the computer on which R is loaded.

Sampling Design Functions

While this document covers the analysis functions in spsurvey, a brief note on the survey design tools in spsurvey is warranted. CHaMP / ISEMP surveys are usually designed using a GRTS (Generalized Random Tessellation Stratified) sampling process [6]. GRTS sampling is a probability based sampling technique that provides a spatially balanced sample, which is useful for efficiently assessing a population distribution.

The "grts" function in spsurvey is provided to assist the user in generating such designs. Arguments to the "grts" function include the sampling frame (often in the form of a shapefile), and a design argument which describes the sampling probability assigned to each member of the sampling frame. Oversampling needs are also input as arguments to the grts function. A list of sites to sample, the weights assigned to each site, and over-sample sites are returned from the grts function.

The "adjwgt" is used when oversample sites are used. Oversample sites will be used in the event that one or more sites selected in the initial sample cannot be used for some reason. An advantage of GRTS sampling is that it enables the user to replace sampled sites that cannot be sampled with over-sampled sites, while maintaining the spatial balance of the original sample. When over-sample sites are used, the weights must be adjusted on all sampled sites, such that the relative weights are maintained and the total weights add up to the total resource size. In the case of stream networks (a linear resource), the total resource size is the total length of stream

included in the sampling frame. Adjusted weights are simply the initial weights times the frame size divided by the sum of the initial weights:

$$adjwgt_i = wgt_i * \frac{Frame\ Size}{\sum_{j=1}^{n} wgt_j}$$

Where the frame size is the total resource size included in the sample frame (the total length of stream sites in the frame if sampling a linear stream network); wgt_i are the initial weights for each site i, and n is the number of sites sampled.

This document is geared toward using the analysis functions. As such, for the remainder of the document it will be assumed that the user has data available from a properly conducted probability sample, and that the adjusted weights (or initial weights if no oversampling was utilized) are correct. It is critical, however, that this is indeed the case in practice. Incorrect weights will lead to incorrect inferences regarding the population.

Once a GRTS sample (or any other probability sample) has been completed, the primary spsurvey functions used to analyze the data and draw inference about the population, or subgroups within the populations, are **cat.analysis** and **cont.analysis**. The function **cat.analysis** is used to analyze categorical variables, while **cont.analysis** is used to analyze continuous variables, as well as non-continuous ordinal variables. (A non-continuous ordinal variable is a response variable that can take on only a discrete set of quantitative values, either infinitely or finitely. For example, the number of fish observable in a stream segment is infinitely discreet, in the set {0, 1, 2, 3, …}. While this isn't a true continuous response variable, it would nevertheless be analyzed using **cont.analysis**.

There are numerous additional functions available in spsurvey, some of which are used to generate plots and conduct tests for differences among subpopulations, and will be discussed below. Other (in fact most) functions in spsurvey are most often called from other functions, rather than directly by the user. These will not be discussed in this guide.

5. Data Needed for spsurvey Analysis Functions

As with any "off the shelf" R function, the spsurvey functions require specific inputs provided in specific formats. In general, data collected during sampling is available in a database or spreadsheet in a form not directly usable as an input to spsurvey functions. The user will need to prepare the external data in a manner that it can be read into R, then, within R, the user must manipulate the data in order to create input data that can be passed into the spsurvey functions. There are many variations on the way this can be done. Expert R users may wish to import data

into R in a "raw" form and perform data manipulations within R, while other users may choose to do a majority of data preparation in an external program, such as Excel, and import data into R that requires a minimum amount of R manipulation. This user's guide will present only one example, where a ".csv" file of raw data is created, read into R, and data is manipulated in R and passed into the spsurvey functions. It is hoped that this process will be useful to users of ISEMP / CHaMP data. But keep in mind this is by no means the only process that could be followed.

The following describes the inputs that are required to be part of the data passed to the spsurvey analysis tools, as well as a subset of the inputs likely to be applicable to ISEMP / CHaMP data users. Note that the specific names of data columns is not yet critical.

Note: spsurvey tools require that only a single row of data exist for each site sampled. This, of course, doesn't preclude multiple measurements of a response variable at each site, it simply requires that the user choose to select, or aggregate data, into a single response. For example, if the number of fish per meter at a site is measured five times, the user may wish to use the average of these five measurements, or perhaps the median, as the single response for fish per meter passed into the spsurvey arguments. If time series data is available, the change in the response over time, or the slope of the response regressed on time, may be used as the a single response for each site passed into the spsurvey analysis functions. Multiple observations per site may also have the advantage of reducing the measurement variability, which will be important if measurement noise is not small relative to the signal being measured.

Required Data:
The following data are required inputs to the spsurvey analysis tools, for any analysis. The user should ensure that all required data are available prior to beginning the analysis:

Site ID: A unique identifier for each site sampled in the survey. This can be a numeric or string id.

X and Y Coordinates. These are the coordinates of each site. These can be in latitude / longitude initially, but will later need to be converted to Albers coordinates to avoid distortive affects in the spsurvey analysis. For the examples provided in this document, we'll start with latitude/longitude, and translate to Albers coordinates using spsurvey tools provided, in R.

Weights: These are the final, adjusted (if necessary) sampling weights (i.e. inverse sampling probabilities) derived from the sample design. Adjusted weights are necessary if there were sites not sampled due to access or other non-response issues, if over sampled data sites were used, etc. Getting the correct weights is critical in accurately analyzing the data. Erroneous weights will lead to biased and potentially misleading results. If there are any questions about the validity of

the weights, it is appropriate to revisit their calculation and/or seek assistance from a knowledgeable source before proceeding.

Note that weights typically have units of measure associated with them. For a linear resource such as a stream network, these units are units of length, such as km. It is critical that the user know what the units on weight are, especially if estimates of response totals, such as number of fish in an entire stream network, are to be calculated. It is recommended that the units on length be included in column names in any spreadsheet or database holding the raw data.

Response Variable Data: Any number of response variables may be included in the dataset, and analyzed concurrently. Responses, in terms of spsurvey tools, are either categorical or continuous.

Categorical responses are responses that can take on non-ordinal values such as "red, blue, or green". Habitat classification may be such a response, taking on values such as "forested, ranch land, clear cut forest, urban". The key feature of categorical responses is that they do not have an intrinsic ordering. Note: merely assigning a number to a value (1=forested, 2=clear cut, 3=ranch land, etc.) does not make a categorical into an ordinal response, as the order is not indicative of a change in the magnitude of a response.

Continuous response variables are, traditionally, thought of as response variables that can take on an infinite number of values within a set. For example, a weight of a fish can vary from near zero to some maximum possible value, and could theoretically take on any value within that range. For spsurvey analysis, we may think of discreet, but ordinal responses, such as the number of fish counted (which can take on only zero or positive integer values) as "continuous", as any ordinal response variables will be analyzed using the spsurvey function **cont.analysis**.

Optional Data

The following input data are not required for all spsurvey analysis, but may be needed to address specific user questions and output requirements.

Stratification Levels

During the sample design stage, strata may be identified over which the researcher may deem it advantageous to define differences in sampling probabilities. In the data analysis phase, this variable probability is reflected in the design weights. Therefore, in many cases it is unnecessary to explicitly define the strata during the analysis phase. The user may in fact note that specifying the strata at the analysis phase usually appears to have no impact on the results. Strata may be

included, however, if the user wishes to utilize the popsize argument and has total resource size information across stratum/subgroup levels [3].

In general, the user will not specify strata at the analysis phase, even if stratification is used in the sampling design. All information regarding strata is reflected in the adjusted weights. (The spsurvey documentation is not entirely clear on why stratum may be an input, yet have no effect on the response. In some versions of the code, the calculation of local variance estimates may have been adjusted somewhat based on user defined strata, while the version as of this writing does not. In older versions, user defined strata may have had a small effect on results).

Subgroup Levels

Subgroups and Strata are terms sometimes used interchangeably by users; however, these are not the same: strata refer to groups made during the sample plan design phase, over which differential sampling probabilities are assigned. Subgroups are defined at the data analysis phase, and merely describe the way or ways that the user wishes to "cut" or "roll up" the data. Of course, the user may wish to define subgroups by the same categories as strata were defined, though this is not necessary. The user may define any subgroups at the analysis stage, regardless of whether such subgroups were considered at the design stage, or if any design stratification was used with respect to subgroups.

Multiple subgroup classifications may be used during an analysis. For example, a user may wish to analyze a response variable by stream Strahler order, and also by another response such as a management reporting unit category.

Resource Size (Total Stream Length) for Sample Frame and by Subgroup

In analysis of a categorical response, a typical result would estimate the proportion of the population of subgroup in each category. For example, the results may estimate that sites within a sample frame are 35% forested, 25% clear cut, 38% ranch lands, and 2% urban lands (and of course with each estimate there would be reported a standard error, confidence intervals, and/or some level of uncertainty). However, instead of percentages, the user may wish to report estimates and uncertainties in kilometers of stream length instead of percent of stream length estimated in each category. To do this, information on the total stream length included in the sample frame must is required. If the user wishes to provide this information by subgroup, then the total stream length by subgroup must be provided.

For CHaMP / ISEMP data, this information is available via the sampling frame used to create the sample design. This may be in the form of a spreadsheet, or a shape files from which the GRTS sample plans were created.

Measurement Variance

The measurement noise of an individual metric may be important to consider, unless the measurement noise is small relative to the signal being measured. Measurement variance can be included in the spsurvey analysis for each continuous response measured. It is assumed in the analysis that the measurement variance is uniform across sites for a given metric, though in reality this may be a poor assumption. If the user is unsure if measurement noise ought to be included in the analysis, or if site to site variation in measurement noise is excessive, it is recommended that a statistician be consulted.

6. Example spsurvey Analysis: Lemhi Watershed 2010 Data

An analysis example for both categorical and continuous responses is presented here to illustrate the details of using the spsurvey analysis tools in R. This example may serve as a template for users constructing their own analysis scripts. The full R-script used in this analysis is provided in the appendix. 2010 data, from the Lemhi watershed, is used for this example. Valley class will be considered as a categorical response variable, while Chinook and Steelhead (O. Mykiss) abundance will be analyzed as continuous response variables. Results will be considered at two spatial scales: the watershed level, and reporting units within the watershed.

Figure 1 shows the first 18 rows of a dataset generated from a GRTS sample on the Lemhi watershed in 2010.

Figure 1. Example Data from Lemhi Watershed

siteid	CHaMPstudy frame	SpatialStrata	ReportingUnit	SSFrameWeight	lat	long	ValleyClass	ChinPerMeter	OmykissPerMeter
LEM-000463	no	Canyon	Upper Lemhi	6.50	44.78	-113.25	Transport		
LEM-001487	Yes	Big Timber	Upper Lemhi	15.50	44.66	-113.38	Depositional	0	0.15
LEM-001487	Yes	Big Timber	Upper Lemhi	15.50	44.66	-113.38	Depositional	0	0.4
LEM-004959	no	Agency	Lower Lemhi	6.40	44.98	-113.50	Source	0	0.5671875
LEM-005039	no	Hayden	Hayden Creek	27.75	44.72	-113.76	Source		
LEM-006575	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.75	-113.48	Depositional	0.434375	0.4335938
LEM-006575	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.75	-113.48	Depositional	0.2000541	0
LEM-006575	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.75	-113.48	Depositional	0.5683594	0
LEM-006575	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.75	-113.48	Depositional	0.213695	0.4592319
LEM-007007	Yes	Wimpey	Lower Lemhi	11.00	45.11	-113.69	Source	0	1.016904
LEM-009055	no	Hawley	Upper Lemhi	7.67	44.71	-113.13	Source	0	0.0521682
LEM-010703	Yes	Hawley	Upper Lemhi	7.67	44.67	-113.15	Source	0	0.0170381
LEM-019295	Yes	Agency	Lower Lemhi	6.40	44.96	-113.54	Transport	0	0.226967
LEM-020943	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.73	-113.44	Depositional	0.2788727	1.1098958
LEM-020943	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.73	-113.44	Depositional	1.0063894	1.1518927
LEM-026031	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.87	-113.62	Depositional	1.5505069	0.7456916
LEM-026031	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.87	-113.62	Depositional	0.8957419	0.4592319
LEM-026031	Yes	Lemhi Mainstem	Upper Lemhi	3.43	44.87	-113.62	Depositional	0.5547184	1.3595338

Additional details on the variables are as follows:

- **Siteid:** unique identifier for each site
- **CHaMPstudyframe**: yes/no variable indicating whether row should be included in the analysis
- **SpatialStrata**: Stratification group used in design, reflected in differing sample probabilities and resulting weights
- **ReportingUnit**: subgrouping over which to analyze data
- **SSFrameWeight**: adjusted sample weights
- **lat**: latitude
- **long**: longitude
- **ValleyClass**: categorical response indicating valley sediment transport classification of site
- **ChinPerMeter**: Number of Chinook salmon per meter measured at site
- **OmykissPerMeter**: Number of Steelhead per meter measured at site

Note that, in the dataset, there are multiple rows for the same siteid. The spsurvey tools require that only one row per unique site be used. This will be dealt with in R for this example.

For this example, the excel file shown above is saved as a .csv file, which can be easily read using standard R functions.

The following steps illustrate how to create an R script to analyze and create some standard output plots for the dataset and analysis goals described above.

Step 1. Create an R Script

Start R, and change the working directory to the directory in which the data file is stored, by clicking on the R console window, then selecting "file", "change dir…", and then navigating to the folder in whish the data file exists.

Next, begin a new R script by selecting "file", "new script". A blank script will appear. You can add comments to the script by preceding them with a "#" symbol. Add some basic info at the top of the script, ssuch as:

```
# Example of analysis of GRTS data using spsurvey
# functions "cat.analysis", "cont.analysis"
```

 # September 2012

Then save the script under a filename of your choice, using the "file" – "save as" options. Continue to save often as you build your script!

Step 2. Load the spsurvey library

Begin the script by loading the spsurvey functions into memory library (assuming the spsurvey package is installed. If not, see section 4) by entering:

```
# load spsurvey function
library(spsurvey)
```

Note that in R, the "#" symbol is used to enter comments into the code. Anything entered to the right of the "#" symbol is a comment and will be ignored by the actual code.

Step 3. Read Data from the data file

Next, read the data from the .csv file using the "read.csv" command, into a data frame called "data":

```
data <- read.csv("Lemhi2010Data.csv", 1)
```

Recall that this data set has, for some sites listed, more than one row of data. For the spsurvey analysis functions, we must have only one row of data per site. The following line of code removes all but the first row listed for each site (though the user, in reality, may wish to use other strategies to choose the best row to keep, or to aggregate multiple rows of data into a single row):

```
# Remove duplicate site entries - keep only the first site listed.
data <- data[!duplicated(data$siteid), ]
```

The coordinates in the in raw data are in latitude/longitude format. The spsurvey functions require a projected coordinates system such as albers coordinates. Included in the spsurvey package is the "geodablers" function that will convert latitude/longitude coordinates into albers coordinates. Code to do this, and add the updated x and y coordinates to the "data" data frame as vectors "xalbers" and "yalbers" is shown below:

```
latlon <- geodalbers(data$lon, data$lat)
```

```
data$xalbers <- latlon$xcoord
data$yalbers <- latlon$ycoord
```

Step 4 (Optional). Overlay Sample Points on Shapefile

If a shapefile covering the sampling frame is present, it is quite straightforward to produce a plot of the stream network, with sampled sites overlaid on the sampling frame. This provides a quick check of data integrity. For this example, we will color code the shapefile and sample points by reporting unit and spatial strata, respectively. The user could easily specify colors by the response level or magnitude, or any number of other ways, to visually present data.

First, read in the shapefile using the "read.shape" function (which is contained with the spsurvey package). In this case, the shape file is titled "LemhiFrameFinal_20120527.shp". Note that there are typically several files, under the same name but with different extensions (.shp, .dbf .prg) that should all be located in the same folder. For the read.shape function, specifiy the ".shp" file, as shown below:

```
# Read shapefile for Lemhi watershed
shape = read.shape(filename =
        "LemhiFrameFinal/LemhiFrameFinal_20120527.shp")
```

Now the shape file information is stored in the data frame "shape". The plot command recognizes shapefiles, and can be used to directly plot the shapefile, as in:

```
plot(shape)
```

However, it may be useful to specify unique colors for some subgroup. For this example, we'll specify a line color for each reporting unit, using the "match" function:

```
# Set color levels by reporting unit
WSlevels = c("Hayden Creek", "Lower Lemhi River", "Upper Lemhi River")
WScolors = match(Rep.unit, WSlevels)
```

Next, we'll check the maximum and minimum x and y locations, so we can adjust the area of plotting (leaving room for a legend):

```
# See what the max and min values are
max(data$xalbers)
min(data$xalbers)
```

After some trial and error on nice looking plot boundaries, we'll set the x plot boundaries using the xlim argument. The following plots the shapefile using a unique color for each reporting unit:

```
# plot the shapefile
    plot(shape, col=1+WScolors, xlim=c(-1400000,-1300000))
```

The "1+" was added simply to change the colors to avoid using black (col=1) as one of the colors, purely a matter of preference.

Next, a legend can be added:

```
# make a legend
legendtext = WSlevels
legend("topright", legendtext, col=(1+seq(1:length(WSlevels))),
 ncol=, title="Reporting Unit",lty=1)
```

Next, we'll plot the individual points included in the sample, overlaid on the map of the watershed. We'll specify a unique color and symbol for each category in "SpatialStrata".

```
# For the individual sample points, set Color by Spatial Strata,
# and add these points to the plot.
strat_levels = levels(data$SpatialStrata)
strat_levels
colors=match(data$SpatialStrata, strat_levels)+1
 symbol = colors+10

# Add the points
points(latlon, col=colors,
  pch=symbol, main = "Spatial Location of Sites, by SpatialStrata",
  xlab="longitude", ylab="latitude")

# specify legend colors
    legend_col = seq(1:length(strat_levels))+1
legend_symbol = legend_col+10

# specify legend text
legendtext = strat_levels
legend("bottomrig", legendtext, col=legend_col, pch=legend_symbol,
```

, ncol=1, title="Spatial Strata")

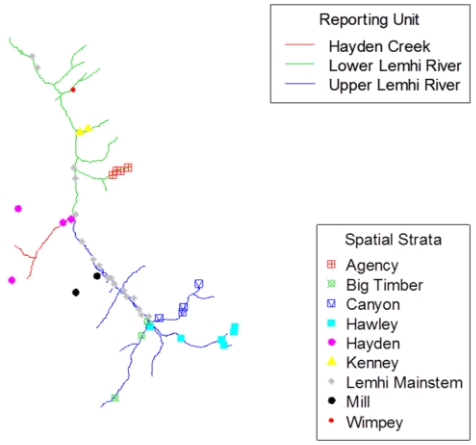
The resulting plot is shown by Figure 2. Note that there are two points in the Hayden creek spatial strata that are disconnected from the stream network. This is indicative of an issue where the shape file does not match, exactly, the sampling frame from which the data were taken. Thus total stream length should not be calculated from this shapefile, but rather a corrected shapefile or from another, correct source. Plots such as these can be quite useful in identifying such inconsistencies.

Another useful plot for data exploration is a boxplot. An example boxplot, showing steelhead density, in fish/meter, by reporting unit, is shown by Figure 3. Code to create this boxplot is as follows:

boxplot(data$OmykissPerMeter ~ data$SpatialStrata, main= "Steelhead Per Meter, by SpatialStrata", las=2, col=5)

Additional exploratory plots, including histograms, are created by the example code provided as an appendix to this document.

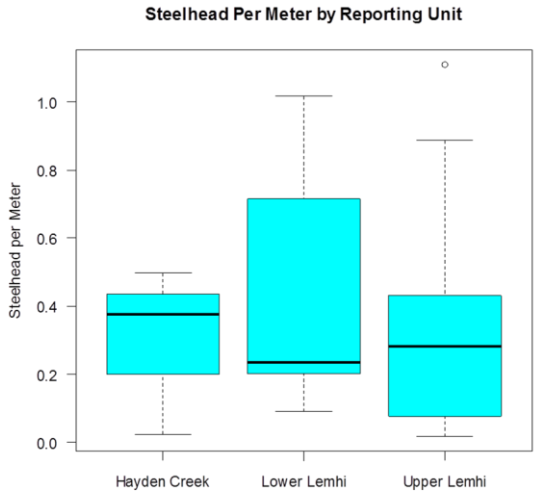
Figure 2. Lemhi Watershed Shapefile, with Sample Points



Step 5. Build Data Frames used by cat.analysis and cont.analysis.

The function **cat.analysis** requires, as input, the following data frames: sites, subpop, design, and data.cat. While not required for all analyses, we'll also create the "popsize" data frame. Specific items included in each data frame will be shown below. When constructing these data frames, it's good practice to give them unique names (different from "site", "subpop", etc) and then pass them into the functions as will be shown below.

Figure 3. Steelhead per Meter, by Reporting Unit



The function **cont.analysis** requires the same sets of inputs as **cat.analysis**, except that "data.cat" dataframe will be replaced by "data.cont".

Sites Data Frame

The sites data frame contains a vector called "siteID" with a single unique ID for each site included in the survey (i.e. one site ID for each row of data). It also must contain a vector called "Use" which lists which rows of data are to be included in the analysis. For this example, we'll use the column "CHaMPstudyframe" to indicate whether to include a given row. Note that we need to turn this into a vector Boolean TRUE/FALSE values in the code below, by using the logical "==" (double equal sign) test. We'll call this data frame "my.sites".

```
# Create the sites data frame, which identifies sites to use in the analysis
my.sites <- data.frame(siteID=data$siteid, Use=(data$CHaMPstudyframe=='Yes'))
```

Design Data Frame

Next, we'll create the design data frame. Like all data frames used, siteID must be included. The next component is "wgt", which is used to specify the weights, obtained from the GRTS design, adjusted for any updates to the sampling frame after using the adjwgt function. Entering proper weights is crucial, as the sampling design probabilities, stratification levels, etc. are reflected in the design weights and are not otherwise specified in the analysis phase. Also required in the design data frame are vectors specifying x coordinates and y-coordinates, on an appropriate (projected) scale, such that x and y distances are in equivalent projected units.

Recall that weights have units of length when analyzing a linear resource. It may be necessary to ensure that units on weights and a response (say fish per meter) are consistent. In this case, the weights were provided in km, and a response of interest is fish per meter, so we need to proactively adjust units be converting weights to meters. (Note this is only required if we are to sum a response, such as fish/meter, over the entire length of the sample frame, or some sub-grouping of the total, as will be demonstrated below).

```
# Input the adjusted weights
data$final.wgt <- data$SSFrameWeight*1000 # convert to meters
```

An effective and easy check on the weights can be done by summing all weights together. The sum of all weights should be equal to the total resource length. For a stream network, the sum of

weights should equal the total length of all streams included in the sampling frame. The sum function can be used as follows:

```
sum(data$final.wgt)
```

The output from above can be compared to the known total stream length from the sampling frame, or from that calculated from the shapefile. In the shapefile associated with this stream network, the vector "FrameLengt" contained stream segment lengths for all sites included in the sampling frame.

```
sum(shape$FrameLengt)
```

If the outputs from the above two summations aren't equal (or at least quite close), the user should double check for errors in the weights or sample frames. (Note, in this example, the included shape file is NOT the shapefile from which the GRTS sample was generated, so there is a small differences between the sums listed above (as well as the inconsistency noted in Figure 2). We'll ignore this inconsistency for the remainder of this example, assuming the weights are correct and the shape file is not an accurate account of the sampling frame used). In practice all such assumptions should be verified.

At this point we can create a design data frame, which we'll call "my.design", as follows:

```
# Create the design data frame, which identifies the stratum code, weight,
# x-coordinate, and y-coordinate for each site ID
my.design <- data.frame(siteID=data$siteid,wgt=data$final.wgt,
        xcoord=data$xalbers, ycoord=data$yalbers)
```

Subpop Data Frame

The subpop data frame specifies the level(s) to which the user would like to aggregate the results. In many cases, the user may wish to simultaneously aggregate to multiple levels. If the subpop argument is omitted, the analysis will simply cover all sites included, with no subgroupings.

In this example, we wish to analyze the data over all sites, while also looking at results by reporting unit. To do this simultaneously, we will a) created a dummy vector of subgroups called "All.Sites" which has only one level: the level of "All.Sites" for each row of data is the same, "All.Sites". Treating this as a subgroup is analogous to not using subgroups, as all data is

in the same subgroup. At the same time, we'll specify that reporting unit is to be used as an additional level of subgroup.

The code below illustrates the subpop data frame for this example. Note the "rep" command is used to create a vector called "All.Sites" where each element is simply "All.Sites". "nr" is a variable containing the number of rows in the data set.

```
# Create the subpop data frame, which defines populations and subpopulations for
# which estimates are desired
my.subpop <- data.frame("siteID"=data$siteid,
                        All.Sites= rep("All.Sites",nr),
                        ReportingUnit = data$ReportingUnit)
```

data.cat and/or data.cont data frames

The data frame used to pass the respons data into **cat.analysis** and **cont.analysis** are straightforward to create. Both will contain a vector of site ID's (as do the design, subpop, and sites data frames), plus a vector for each response variable. Data frames for the categorical analysis variable "ValleyClass" and for the categorical response variables "OmykissPerMeter" and "ChinPerMeter" are shown below:

```
# Now make the "data" data frame for our categorical variable
my.cat.data <- data.frame(siteID=data$siteid, ValleyClass=data$ValleyClass)

# Now make the "data" data frame for our continuous variable
my.cont.data <- data.frame(siteID=data$siteid,
OmykissPerMeter=data$OmykissPerMeter, ChinPerMeter = data$ChinPerMeter)
```

Popsize Argument

The popsize argument is used to specify the total resource size, by subgroup. For a stream network, the resource size is the total stream length, in the same units as the weights in the design data frame. This is an optional input, and it used when the user wants to convert a proportion estimate to an estimate in units of the resource size. In this example, we'll consider valley class as a categorical response. Default results will include estimates of the percent of each valley class category in each subgroup specified. If, however, the user would like estimates for the total length of stream in each valley class category, by subgroup, the popsize argument can be used to specify the total stream lengths, by subgroup, such that the total of each valley class category is forced to sum to the total stream length.

The structure used to enter the popsize argument can be troublesome, and may require some trail and error on the part of the user. For this example, we're specifying two "levels" of subgroups. The "top" level includes all data points and is specified by the "All.Sites" label in the subgroups data frame; while the next level divides this into three subgroups, by reporting unit, as described above. The popsize argument must specify popsize for each level of subgroup(s) specified. In this case, we must specify popsize for "All.Sites", as well as for each level of the subgroup variable. Subgroup names within the popsize argument must match the subgroups names exactly. The structure for the popsize argument is a list (a data structure within R that allows for named elements) of elements at each level of subgroup. If there are multiple levels of subgroup, then the lower levels are specified as lists within the broader list.

Note: If stratification is used and the user wishes to specify popsize by stratum/subgroup levels, the argument takes on an additional level of complexity. Often, though stratification is used in the design, it may not be necessary to track strata at the analysis stage, as the weights take into account the effects of design stratification. However, it may be useful when aggregating by subgroup, in some cases [3].

Popsize is illustrated by the example below. From the sampling frame, the total resource length was calculated to be 349830 meters. We wish to estimate total resource lengths, by valley class, for the entire watershed. In addition, we wish to estimate resource length, by valley class, within each reporting unit. Thus included in the main list is "Reporting Unit", which is itself a list specifying the total stream length for each reporting unit within the broader watershed. Note that the lengths within the "ReportingUnit" list should add up to the length specified in the "All.Sites" group.

```
# Create the popsize data frame, based on the calculation done earlier.
# The numbers represent the total stream length by subgroups
        my.popsize = list(All.Sites = 349830,
                ReportingUnit=list("Hayden Creek"=64558,
                                   "Lower Lemhi"= 121266,
                                   "Upper Lemhi" = 163153))
```

Step 6. Running the cat.analysis function

Now that we've prepared the input data frames and other arguments, it is straightforward to run the spsurvey function **cat.analysis** and **cont.analysis**. The **cat.analysis** function is shown below. Note that it is not necessary to used the form "site=my.sites, subpop = my.subpop", etc. The user may simply arrange the arguments in the correct order. However, including the structure as

shown ensures that the correct arguments are passed correctly to the function, regardless of the order in which they are passed. This practice can make debugging much easier.

```
# Now, let's use "cat.analysis" to analyze the categorical data
Results.Cat <- cat.analysis(site=my.sites,
        subpop = my.subpop, design = my.design,
        data.cat = my.cat.data, popsize=my.popsize)
```

"Results.Cat" is now a data frame containing the results of the **cat.analysis**. The "names" function can be used to view the names of the results contained in "Results.Cat". Descriptions of each item in the output are listed below:

names(Results.Cat)

Figure 4: Output from names(Results.Cat):

```
[1] "Type" "Subpopulation" "Indicator"     "Category"      "NResp"         "Estimate.P"    "StdError.P"
 [8] "LCB95Pct.P"    "UCB95Pct.P"    "Estimate.U"    "StdError.U"    "LCB95Pct.U"    "UCB95Pct.U"
```

Descriptions of each element of the cat.analysis output are as follows:

- **Type:** indicates which subgroup category of results is listed in a given row; in this case it will be either "All.Sites", or "ReportingUnit".
- **Subpopulation:** lists which category of "All.Site" or "ReportingUnit" is listed.
- **Category:** indicates for which level of the categorical variable results are shown
- **NResp:** indicates the number of response in a given subgroup / level of categorical variable.
- **Esimate.P:** The estimated percentage of stream length, across all sites in the sample frame / subgroup combination, of the given level of the subgroup.
- **StdError.P, LCB95Pct.P, UCB95Pct.P:** standard error and corresponding upper and lower 95% confidence limits for the percentage estimates in Estimate.P.
- **Estimate.U:** The estimate of total stream length (in the units used by popsize), across all sites in the sample frame / subgroup combination. This is only accurate if the popsize argument is used. The "U" refers to "units" (meters, in this case).
- **StdError.U, LCB95Pct.U, UCB95Pct.U:** standard error and corresponding upper and lower 95% confidence limits for the percentage estimates in Estimate.U.

To see the actual results, simply type the name of the output data frame "Results.Cat". To write the data to a .csv file, use the command "write.csv", as shown below, along with a table of results (formatting added in excel).

Results.Cat
 # Write results to a file
write.csv(Results.Cat, "Results.Cat.csv")

Figure 5. Results from "cat.analysis" function

Type	Subpopulation	Indicator	Category	NResp	Estimate.P	StdError.P	LCB95Pct.P	UCB95Pct.P	Estimate.U	StdError.U	LCB95Pct.U	UCB95Pct.U
All.Sites	All.Sites	ValleyClass	Depositiona	19	50.47071	7.826746	35.130571	65.81085	176561.69	27380.31	122897.28	230226.1
All.Sites	All.Sites	ValleyClass	Source	6	18.402	5.530085	7.563229	29.24076	64375.7	19345.9	26458.443	102292.96
All.Sites	All.Sites	ValleyClass	Transport	9	31.12729	7.879645	15.683472	46.57111	108892.61	27565.36	54865.491	162919.73
All.Sites	All.Sites	ValleyClass	Total	34	100	NA	NA	NA	349830	NA	NA	NA
ReportingUnit	Hayden Creek	ValleyClass	Depositiona	2	100	0	100	100	64558	0	64558	64558
ReportingUnit	Hayden Creek	ValleyClass	Total	2	100	NA	NA	NA	64558	NA	NA	NA
ReportingUnit	Lower Lemhi	ValleyClass	Depositiona	4	21.66772	9.430861	3.183572	40.15187	26275.58	11436.43	3860.591	48690.56
ReportingUnit	Lower Lemhi	ValleyClass	Source	3	37.74479	15.280938	7.794701	67.69488	45771.6	18530.58	9452.322	82090.87
ReportingUnit	Lower Lemhi	ValleyClass	Transport	4	40.58749	13.461307	14.203816	66.97117	49218.83	16323.99	17224.4	81213.26
ReportingUnit	Lower Lemhi	ValleyClass	Total	11	100	NA	NA	NA	121266	NA	NA	NA
ReportingUnit	Upper Lemhi	ValleyClass	Depositiona	13	43.34053	10.223316	23.303196	63.37786	70711.37	16679.65	38019.864	103402.88
ReportingUnit	Upper Lemhi	ValleyClass	Source	3	16.83367	5.883447	5.302322	28.36501	27464.63	9599.021	8650.898	46278.37
ReportingUnit	Upper Lemhi	ValleyClass	Transport	5	39.82581	12.077888	16.153581	63.49803	64977	19705.44	26355.052	103598.94
ReportingUnit	Upper Lemhi	ValleyClass	Total	21	100	NA	NA	NA	163153	NA	NA	NA

The above table indicates, for example, that over all sites, an estimates 50.47% of stream length is valley class "Depositional" (95% confidence bound 35.1% to 65.8%). Instead of %, we can look at the ".U" columns to see that this corresponds to 176562 meters of Depositional valley class (95% confidence interval 122897 m to 230226 m). Considering only sites in the Lower Lemhi Valley class, we see, for example, that there is an estimated 49219 meters of valley class "Transport" (95% confidence interval 17224 meters to 81213 meters). Note that for the Hayden Creek reporting unit, we have only two responses, both in the Depositional valley class. With only two responses, estimates and confidence bounds have little meaning.

Step 7. Running the cont.analysis function, and Estimating Total Population Abundances

Running the **cont.analysis** function is nearly identical to running the **cat.analysis** function, except that we now pass the continuous data frame "my.cont.data" in "data.cont", instead of the categorical data passed into **cat.analysis**. All other data inputs can be used for both functions without alteration.

In addition, for this example we will include the optional "total=TRUE" argument. This argument allows, in this case, for an estimate of the total Steelhead and Chinook abundance, by Subgroups. The estimate for total population is calculated as follows:

$$Total = \sum_{i=1}^{n} Y_i * W_i$$

Where n is the number of samples included in the data set, Y_i is the ith observation of the response, and W_i is the weight of the i_{th} observation. It is important when using the "total = TRUE" argument to check (and re-check) the units on your response and your weights. In this example, the units for "OmykissPerMeter" are number of Steelhead per meter, and the weights are in meters, thus the units on "Total" are number of Steelhead. Similarly, for "ChinPerMeter", the units on "Total" are number of Chinook.

NOTE: If the units of the response variable are, say "number of Steelhead per site", but the weights are in "meters", the "total=TRUE" **argument will not return meaningful information**.

The following shows the call for the **cont.analysis** function for this example:

```
# Analyze Continuous Data usisng "cont.analysis" function
Results <- cont.analysis(sites=my.sites, subpop = my.subpop,
            design = my.design, data.cont = my.cont.data,
            total=TRUE,
          popsize=my.popsize)
```

The output for this example is stored in the data frame "Results". Typing "*names(Results)"* shows that there are four items within this dataframe: CDF, Pct, CDF.D, and Pct.D. CDF contains results for the cumulative distribution function estimates for all response variables, by subgroup(s).

Pct contains estimates for various percentiles of the cumulative distribution function for each variable analyzed, as well as estimates of the mean and variance of the distributions. The default percentiles are: 5, 10, 25, 50, 75, 90, and 95. These defaults can be user defined if desired (see step 11). In addition to these estimates, confidence intervals for each estimates are provided, as well as standard errors for the estimates of the mean and variance parameters.

If the "total=TRUE" argument was passed to **cont.analysis**, then the Pct data frame will also contain an estimate, standard error, and confidence interval for the sum total of a resource (in this case, number of fish) for each subgroup specified.

The CDF data frame contains results formatted so as to enable plotting of the estimated CDF curve and corresponding confidence intervals, as will be seen blow.

CDF.D and Pct.D are NULL (empty) for this example. These return information only if measurement variance is considered in the analysis. An example of this will be shown in sections to follow.

First, we'll examine the results in Pct. Results can be displayed to the screen by entering Results$Pct. Alternately, for easier viewing, write results to a .csv file with the command:

```
#Write results to file(s)
write.csv(Results$Pct, "ResultsPCT.csv")
```

A table of results contained in the PCT data frame is shown by Figures 6a (for Steelhead) and 6b (for Chinook).

Figure 6a. Steelhead Abundance Results for All sites from "ResultsPCT.csv" (formatted)

Type	Subpopulation	Indicator	Statistic	NResp	Estimate	StdError	LCB95Pct	UCB95Pct
All.Sites	All.Sites	OmykissPerMeter	5Pct	1	0.019165		0.017038	0.036446
All.Sites	All.Sites	OmykissPerMeter	10Pct	2	0.031376		0.017038	0.08993
All.Sites	All.Sites	OmykissPerMeter	25Pct	6	0.12371		0.037162	0.234504
All.Sites	All.Sites	OmykissPerMeter	50Pct	18	0.301396		0.213359	0.367265
All.Sites	All.Sites	OmykissPerMeter	75Pct	25	0.445728		0.323722	0.956224
All.Sites	All.Sites	OmykissPerMeter	90Pct	29	0.865367		0.478554	1.051564
All.Sites	All.Sites	OmykissPerMeter	95Pct	31	0.978292		0.745533	1.109896
All.Sites	All.Sites	OmykissPerMeter	Total	33	87297.84	14550.69	58779.01	115816.7
All.Sites	All.Sites	OmykissPerMeter	Mean	33	0.362397	0.041084	0.281874	0.44292
All.Sites	All.Sites	OmykissPerMeter	Variance	33	0.082051	0.018302	0.04618	0.117922
All.Sites	All.Sites	OmykissPerMeter	Std. Deviat	33	0.286446	0.031946	0.223832	0.34906

From the above table, we see, for example, that the total Steelhead (O.Mykiss) population over all sites is 87297 fish, with a 95% confidence interval of 5879 to 115817 fish. The mean density of steelhead per meter in the watershed is .362 steelhead per meter (95% confidence interval .28 to .44 fish per meter). The median density (50^{th} percentile) is .301 fish per meter.

Figure 6b. Chinook Abundance Results for the Lemhi Reporting Unit from "ResultsPCT.csv" (formatted)

Type	Subpopulation	Indicator	Statistic	NResp	Estimate	StdError	LCB95Pct	UCB95Pct
ReportingUnit	Lower Lemhi	ChinPerMeter	5Pct	7	0		0	0
ReportingUnit	Lower Lemhi	ChinPerMeter	10Pct	7	0		0	0
ReportingUnit	Lower Lemhi	ChinPerMeter	25Pct	7	0		0	0
ReportingUnit	Lower Lemhi	ChinPerMeter	50Pct	7	0		0	0
ReportingUnit	Lower Lemhi	ChinPerMeter	75Pct	7	0		0	0.267113
ReportingUnit	Lower Lemhi	ChinPerMeter	90Pct	9	0.256711		0	0.423125
ReportingUnit	Lower Lemhi	ChinPerMeter	95Pct	10	0.311784		0.19915	0.423125
ReportingUnit	Lower Lemhi	ChinPerMeter	Total	11	3150.318	1363.479	477.9494	5822.687
ReportingUnit	Lower Lemhi	ChinPerMeter	Mean	11	0.049752	0.025087	0.000584	0.098921
ReportingUnit	Lower Lemhi	ChinPerMeter	Variance	11	0.013367	0.006641	0.000351	0.026383
ReportingUnit	Lower Lemhi	ChinPerMeter	Std. Deviat	11	0.115615	0.02872	0.059325	0.171906

From the above table we see, for example, that the estimated total Chinook population in the Lower Lemhi for is 3150 fish, with a 95% confidence interval of (1363, 5822).

To obtain a table showing only total abundance, for both steelhead and Chinook salmon, the following code can be used:

```
# Write results to file(s)
write.csv(Results$Pct[Results$Pct$Statistic=="Total",], "ResultsPCT.Total.csv")
```

Total abundance estimates are no longer in units of "fish per meter", but rather simply number of fish. Figure 7 shows total fish abundance estimates for steelhead and Chinook Salmon, over the entire watershed and by reporting unit, as well as 95% confidence limits.

Figure 7: Total Fish Abudance Estiamtes from cont.analysis functions

Type	Subpopulation	Indicator	Statistic	NResp	Estimate	StdError	LCB95Pct	UCB95Pct
All.Sites	All.Sites	OmykissPerMeter	Total	33	87298	14551	58779	115817
ReportingUnit	Hayden Creek	OmykissPerMeter	Total	2	24243	3383	17613	30873
ReportingUnit	Lower Lemhi	OmykissPerMeter	Total	11	31663	9376	13286	50041
ReportingUnit	Upper Lemhi	OmykissPerMeter	Total	20	31391	3812	23919	38863
All.Sites	All.Sites	ChinPerMeter	Total	33	108124	42113	25583	190665
ReportingUnit	Hayden Creek	ChinPerMeter	Total	2	74741	35959	4264	145219
ReportingUnit	Lower Lemhi	ChinPerMeter	Total	11	3150	1363	478	5823
ReportingUnit	Upper Lemhi	ChinPerMeter	Total	20	30232	9242	12119	48346

From Figure 7 we can see the estimates for total number of steeleahd and Chinook, for each reporting unit as well as over all sites, and corresponding standard errors and confidence intervals.

Step 8. Making CDF Plots.

There are two options available for making CDF plots showing the distribution estimates resulting from **cont.analysis**. You can use the spsurvey function **cont.cdfplot**, which will automatically generate cdf plots from the CDF data frame output from **cont.analysis**. It will plot estimates for the cdf and corresponding confidence limited, for all continuous response variables included in the analysis, at all subgroup levels analyzed in **cont.analysis**. This is easy to do, as is shown below:

```
# Use function "cont.cdfplot"
cont.cdfplot("Example_CDF.pdf", Results$CDF)
```

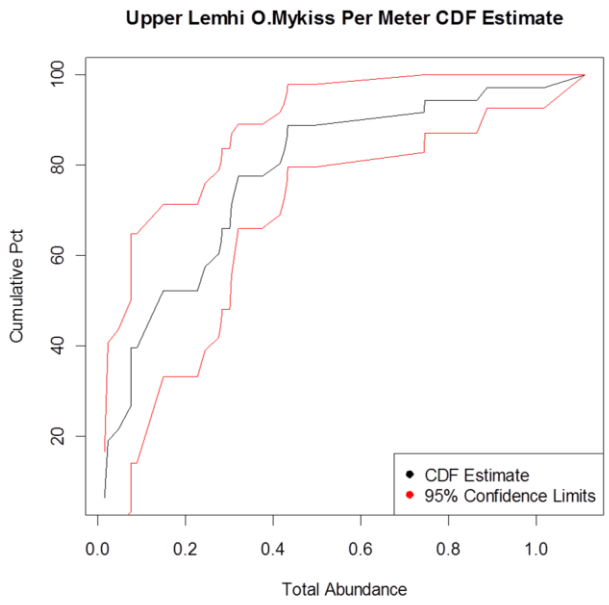
The code above will create an output file called "Example_CDF.pdf.

Alternately, you can access the results in the "CDF" data frame directly to create your own CDF plots. The advantage of doing this is greater flexibility in customizing the graph(s) to your liking. The following code is used to create the CDF plot shown by Figure 7 for the distribution of steelhead abundance in the upper Lemhi reporting unit:

```
# Select Subpopulation for plotting cdf plots to screen
Plot.Subpop= Results$CDF[Results$CDF$Subpopulation=="Upper Lemhi",]
# Manually make some plots
title = Plot.Subpop$Subpopulation[1]
metric = Plot.Subpop$Indicator[1]

# Generate a CDF Plot for Estimate.P
with(Plot.Subpop, {
plot(Value[Indicator=="OmykissPerMeter"],Estimate.P[Indicator==
        "OmykissPerMeter"],
 type="l",       main = paste(title, "O.Mykiss Per Meter CDF Estimate"),
xlab="Total Abundance",      ylab="Cumulative Pct")

# Add confidence bounds
lines(Value[Indicator=="OmykissPerMeter"],LCB95Pct.P[Indicator==
        "OmykissPerMeter"],
        col="red")
```

```
lines(Value[Indicator=="OmykissPerMeter"],UCB95Pct.P[Indicator==
        "OmykissPerMeter"],
        col="red")
})

# add a legend
legend.text = c("CDF Estimate", "95% Confidence Limits")
legend("bottomright", legend.text,col = c("black", "red"), pch=19)
```

Figure 7. CDF Plot for Steelhead density (fish/m) for the upper Lemhi Reporting Unit



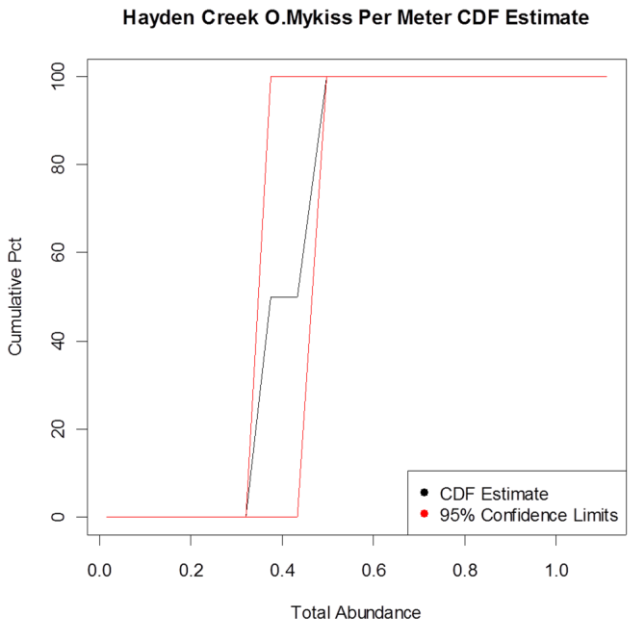
The CDF plot in Figure 7 is exactly equivalent to the corresponding plot created by the function **cont.cdfplot** function. The only difference is in coloring, but the user is free to change title, axis labels, plot height and width, etc., using any of the standard R graphing tools.

Note that when interpreting the confidence limits on CDF plots, the vertical distance between confidence limits represents the confidence interval, rather than the horizontal distance. Visually, CDF confidence limits can be misleading in cases where the vertical width is large, but the horizontal limits are narrow. An example of this can be seen in figure 8, which shows the CDF estimate and 95% confidence interval for steelhead in the Hayden creek reporting unit. Because the sample size in this reporting unit is quite small, the 95% confidence interval is quite wide. Note that the wide confidence limits are seen only when examining the vertical distance between the estimates and the confidence limits. Examining the horizontal distance between lines, one could easily make the incorrect conclusion that the confidence limits are narrow and that we have high precision in our estimates. This is NOT correct!

Step 9 (Optional). Testing for Differences Between Sub-Populations

In some cases, the user may wish to test for statistical evidence of differences among subpopulations. The spsurvey function **cont.cdftest** can be used to test for differences in distribution between subgroups in a population. Note that, as a test for differences in distribution, this is a more general test than, for example, a test for differences in means. The function **cont.cdftest** uses a WALD test for differences between populations [1].

Figure 8. CDF Plot for Steelhead density (fish/m) for the Hayden Creek Reporting Unit



In this example, we will test for differences in the density of fish across the three reporting units, for both Steelhead and Chinook salmon. Code to conduct this test, as well as write the output to the screen and to a filed called "CDF_Tests_Example.csv" file, is as follows:

```
# Conduct a WALD test for differences among subpopulations
        CDF_Tests <- cont.cdftest(sites=my.sites, design= my.design,
             data.cont= my.cont.data,
            subpop = data.frame("SiteID"=data$siteid,
"ReportingUnit"=data$ReportingUnit))
          print(CDF_Tests)
          write.csv(CDF_Tests, "CDF_Tests_Example.csv")
```

Note that we didn't use our original "mySubpop" data frame for cont.cdftest. This is because "my.subpop" contained two levels of subgroups: "All.data" and "ReportingUnit". "All.data" was a single subgroup, for which a test for differences among subgroups would be meaningless. Thus the subpop data frame was re-created for cont.cdftest, to include only the ReportingUnit. Also note that, in this case, the subpop data frame was constructed directly within the function call. This can be done with any functions, though it is often easier to construct the data frames external to the function call, in order to keep function calls short and simple.

The results from this function are shown in Figure 9:

Figure 9. Test for Differences in Distribution of Fish Density, by Reporting Unit

Type	Subpopulation_1	Subpopulation_2	Indicator	Wald_F	Degrees_of_Freedom_1	Degrees_of_Freedom_2	p_Value
ReportingUnit	Hayden Creek	Lower Lemhi	OmykissPerMeter	2.695518	2	10	0.115787
ReportingUnit	Hayden Creek	Upper Lemhi	OmykissPerMeter	13.62432	2	19	0.000214
ReportingUnit	Lower Lemhi	Upper Lemhi	OmykissPerMeter	1.038816	2	28	0.367114
ReportingUnit	Hayden Creek	Lower Lemhi	ChinPerMeter	105.6784	2	10	1.88E-07
ReportingUnit	Hayden Creek	Upper Lemhi	ChinPerMeter	163.8178	2	19	1.05E-12
ReportingUnit	Lower Lemhi	Upper Lemhi	ChinPerMeter	0.710444	2	28	0.500071

From the above results, we see that there is significant evidence (p-value < .05) of differences in the distribution of steelhead per meter (O.mykiss per meter) between the Hayden Creek and Upper Lemhi reporting units. There is also evidence of differences in Chinook salmon densities between Hayden Creek and the Lower Lemhi, and Hayden Creek and the Upper Lemhi reporting units. Keep in mind that, in cases where p-values do not provide evidence of differences, that does NOT necessarily imply that the distributions are the same. Rather, this suggests that one of the following is true: the distributions are nearly the same OR the differences in distributions are too small to be detected with the sample size. It is a common, but serious mistake to interpret a p-value > .05 as evidence of "no differences".

Step 10 (Optional). Including Measurement Noise in the Analysis

In many cases, measurement error (a.k.a. measurement "noise") may be large relative to the signal being measured. Spsurvey analysis tools provide a mechanism to include measurement variance in the analysis in order to provide deconvoluted cdf estimates. Deconvoluted, in this case, means that the variance of the population will be estimated separately from the measurement variance. Note that when measurement variance is not included in the analysis, all variance is assumed to be due to actual variation in the response being measured.

Note also that measurement variance will not (in spsurvey) affect the estimated distributions, but rather it will affect the uncertainty of those estimates, as indicated by standard errors, confidence limits on cdf plots, etc. Including measurement variance will generally increase the uncertainty of the response estimates. This "correction" to the results is indicative of the fact that, due to uncertainty in the data, the user actually has less information about the response than he/she would if the response variable were measured without uncertainty.

It is not always obvious when it is necessary to include measurement variance in an analysis. As a rule of thumb, it may be appropriate to ignore measurement variance if the magnitude of the measurement variance is much less (i.e. and order of magnitude) below the variance of the signal being measured. However, assuming measurement variances are known, there is little downside to including them, with the exception of adding some complexity to the analysis. The upside of including them is a more accurate description of uncertainty in population distribution estimates.

One potential issue when including measurement error is the need for measurement variance to be distributed equally across all sites. For CHaMP / ISEMP data, this may be problematic for at least two reasons: not all data is obtained via the same measurement techniques (i.e. fish density may be estimates by mark-recapture techniques, electrofishing, etc); secondly, measurement variance may increase as the signal being measured increases (i.e. the measurement variance for a site with 10 fish per meter may be higher than for a site with 1 fish per meter). The spsurvey functions do not enable specification of measurement uncertainty on a per site level, but rather assume that the measurement variance is constant over all sites for a given metric.

Finally, note that high measurement variances on a site level do not necessarily doom the analysis to poor estimates of populations aggregated to higher spatial scales. The more sites aggregated together in an analysis, the greater the precision of total population estimates relative to the size of the signal being measured.

To include measurement variance in R, we need to first define a vector of measurement variance (although missing values are allowed for responses for which measurement error is unknown). Please note that the function requires variance estimates, NOT standard deviation estimates. (Variance equals the standard deviation squared). This can be somewhat confusing in spsurvey due to the fact that the argument name for measurement variance is "sigma" rather than something like "sigma_squared" , a more typical nomenclature for a variance component. (Note: this has been confirmed with the authors of spsurvey).

For this example, the measurement variance was defined, and the continuous variable analysis re-run, as follows:

```
Measurement.variance = c("OmykissPerMeter" = .25, "ChinPerMeter" = .2)
Results.D <- cont.analysis(sites=my.sites, subpop = my.subpop,
                           design = my.design, data.cont = my.cont.data,
                          total=TRUE,
                    popsize=my.popsize,
                   sigma= Measurement.variance)
```

The data frame Results.D now includes the same data frames as were produced when using **cont.analysis** without measurement error. However, now the data.frames CDF.D and Pct.D are included. These data frames contain the deconvoluted results. These data frames now contain more accurate descriptions of uncertainty in results, and should be used in lieu of the CDF and Pct dataframe. CDF plots using the corrected (deconvoluted) estimates can be obtained as follows:

```
#Create a PDF file containing plots of the CDF estimates--the function generates the
#cdf plots from the data in Results$CDF.D
cont.cdfplot("Results.D.CDF.D.pdf", Results.D$CDF.D)
```

Note that the results data frames in CDF and Pct are identical to those produced when not including measurement variance in the analysis. The results in the dataframes CDF.D and Pct.D have the modified estimates.

Step 11: Other Options for Analysis

Additional options exist for using the spsurvey analysis function **cat.analysis** and **cont.analysis**. Some of those that may be of interest to CHaMP / ISEMP users include:

- **conf**: Change the Confidence Level from the default 95%
- **pct.val**: change the values at which the percentiles are estimates (from the default {5, 10, … , 90, 95})
- **var.sigma:** Include variance of the measurement variance estimate in the analysis.

Details on using these are available in the spsurvey documentation, or by typing *"?cat.analysis*" or *"?cont.analysis*" in R.

7. Summary

The spsurvey functions **cat.analysis** and **cont.analysis**, and related functions, provide tools for analysis of probability survey data, including the CHaMP / ISEMP datasets obtained via GRTS sampling. This document provides an additional resource, geared specifically toward CHaMP / ISEMP scientists, to aid in the analysis and interpretation of these datasets, using the spsurvey analysis tools.

While this document provides a basic introduction and a framework that may be applicable to most analyses, this is by no means an exhaustive review of the spsurvey tools, nor analysis

methods for probability based designs in general. Users are encouraged to seek additional resources and/or seek the assistance of the author and other statisticians as questions and issues arise. It is hoped that such requests may contribute to updates and improvements in future revisions of this document.

8. References

1. Kincaid, Thomas, and Olsen, Tony. 2012. R-package spsurvey on-line documentation: Spatial Survey Design and Analysis, Version 2.4. http://cran.r-project.org/web/packages/spsurvey/spsurvey.pdf

2. Kincaid, Thomas, 2009. User Guide for spsurvey, version 2.1 Probability Survey Design and Analysis Functions. http://www.epa.gov/nheerl/arm/documents/design_doc/UserGuide%20for%20spsurvey%202.1.pdf.

3. Kincaid, Thomas, 2012: Analysis of a GRTS Survey Design for a Linear Resource. http://cran.r-project.org/web/packages/spsurvey/vignettes/Linear_Analysis.pdf

4. Schreuder, H.T., T.G. Gregoire, and J.P. Weyer (2001).For what applications can probability and non-probability sampling be used? *Environmental Monitoring and Assessment* 66: 281291.

5. Stevens, D. L., Jr. and Olsen, A. R. Variance Estimation for Spatially Balanced Samples of Environmental Resources. Environmetrics 14:593-610. Abstract (html 14KB)Document (pdf 387KB)

6. Stevens, D. L., Jr. and A. R. Olsen (2004). "Spatially-balanced sampling of natural resources." Journal of American Statistical Association 99(465): 262-278. Abstract (pdf 9KB) Document (pdf 1.4MB)

7. United States E.P.A. Aquatic Resources Monitoring Home Page: http://www.epa.gov/nheerl/arm/designpages/monitdesign/monitoring_design_info.htm

9. Appendix: R-code Used For Example Analysis in Section 6

```
# Example of analysis of GRTS data using spsurvey
# functions "cat.analysis", "cont.analysis"
# Matt Nahorniak, August 2012
#
# To be used in conjunction with data file "Lemhi2010Data.csv"


# Load Required Libraries
        library(spsurvey)

# Read the data file
        data <- read.csv("Lemhi2010Data.csv", 1)

# Remove duplicate site entries - keep only the first site listed.  This
# is for example only... user needs to determine what to do with duplicate
# measurements and if/how to use them.  cont.analysis and cat.analysis take
# only one set of measurements per site as inputs.
        data <- data[!duplicated(data$siteid), ]

# Take a look at the first few rows of data
        head(data)

# create Albers xy coords for use in analysis
# spsurvey functioins need lat/lon in this format
# and add the albers x and y coords to the "data" dataframe
        latlon <- geodalbers(data$lon, data$lat)
        data$xalbers <- latlon$xcoord
        data$yalbers <- latlon$ycoord


##################################################
# Make Some simple Exploratory Plots

# Read shapefile for Lemhi watershed
        shape = read.shape(filename =
                "LemhiFrameFinal/LemhiFrameFinal_20120527.shp")

# Do a quick check on the total frame length in the shape file
sum(shape$FrameLengt)

# Create Rep.unit in a way that I think matches the data,
# so I can get total frame lengths
```

```
        Rep.unit= shape$HU_10_NAME
        levels(Rep.unit)
        Rep.unit[Rep.unit=="Hawley Creek"] = "Upper Lemhi River"
        Rep.unit[Rep.unit=="Middle Lemhi River"] = "Lower Lemhi River"
        Rep.unit[Rep.unit=="Timber Creek"] = "Upper Lemhi River"
        Rep.unit[Rep.unit=="Texas Creek"] = "Upper Lemhi River"
        Rep.unit[Rep.unit=="Eighteenmile Creek"] = "Upper Lemhi River"


########################################
# Do some preliminary Graphical Analysis
# Make a plot using the shapefile, and overlay sample
# points on top.

# Set color levels by reporting unit
        WSlevels = c("Hayden Creek", "Lower Lemhi River", "Upper Lemhi River")
        WScolors = match(Rep.unit, WSlevels)

# See what the max and min values are
        max(data$xalbers)
        min(data$xalbers)

# plot the shapefile
        plot(shape, col=1+WScolors, xlim=c(-1400000,-1300000))

# make a legend
        legendtext = WSlevels
        legend("topright", legendtext, col=(1+seq(1:length(WSlevels))),
       ncol=1, title="Reporting Unit",lty=1)


# For the individual sample points, set Color by Spatial Strata,
# and add these points to the plot.
        strat_levels = levels(data$SpatialStrata)
        strat_levels
        colors=match(data$SpatialStrata, strat_levels)+1
    symbol = colors+10
    legend_col = seq(1:length(strat_levels))+1
        legend_symbol = legend_col+10
        points(latlon, col=colors,
        pch=symbol, main = "Spatial Location of Sites, by SpatialStrata",
        xlab="longitude", ylab="latitude")
        legendtext = strat_levels
        legend("bottomrig", legendtext, col=legend_col, pch=legend_symbol,
      , ncol=1, title="Spatial Strata")


####################################
```

```
# Make some other plots

        hist(data$OmykissPerMeter, main="O.MykissPerMeter", xlab="O.MykissPerMeter", col=5)

        boxplot(data$OmykissPerMeter ~ data$SpatialStrata, main=
          "Steelhead Per Meter, by SpatialStrata", las=2, col=5)

        boxplot(data$OmykissPerMeter ~ data$ReportingUnit, main=
          "Steelhead Per Meter by Reporting Unit", las=1, col=5,
      ylab="Steelhead per Meter")


        hist(data$ChinPerMeter, main="Chinook Per Meter", xlab="O.MykissPerMeter", col=5)

        boxplot(data$ChinPerMeter ~ data$SpatialStrata, main=
          "Chinook Per Meter, by SpatialStrata", las=2, col=5)

        boxplot(data$ChinPerMeter ~ data$ReportingUnit, main=
          "Chinook Per Meter by Reporting Unit", las=1, col=5)



#####################################################
#
# Start the analysis with spsurvey functions here!
#
#####################################################

# Start Buidling dataframes for cont.analsys and cat.analysis
# First time through, we'll analyze as though sample plan was
# unstratified


# get number of rows
        nr <- nrow(data)

# Input the adjusted weights
        data$final.wgt <- data$SSFrameWeight*1000 # convert to meters
        sum(data$final.wgt)

# Create the sites data frame, which identifies sites to use in the analysis
        my.sites <- data.frame(siteID=data$siteid, Use=(
        data$CHaMPstudyframe=='Yes'))

# Create the subpop data frame, which defines populations and subpopulations for
# which estimates are desired

        my.subpop <- data.frame("siteID"=data$siteid,
                                All.Sites= rep("All.Sites",nr),
```

```
                ReportingUnit = data$ReportingUnit)


# Create the popsize data frame, based on the calculation done earlier.
# The numbers represent the total stream length by subgroups

# missing length belongs to Hayden Creek / Hayden
        sum(data$final.wgt)
        sum(shape$FrameLengt)
        sum(data$final.wgt)-sum(shape$FrameLengt)
        24417+40141.38

        my.popsize = list(All.Sites = 349830,
          ReportingUnit=list(
                          "Hayden Creek"=64558,
           "Lower Lemhi"= 121266,
           "Upper Lemhi" = 163153))


# Create the design data frame, which identifies the stratum code, weight,
# x-coordinate, and y-coordinate for each site ID
        my.design <- data.frame(siteID=data$siteid,
              wgt=data$final.wgt,
               xcoord=data$xalbers,
               ycoord=data$yalbers)



# Now make the "data" data frame for our categorical variable
        my.cat.data <- data.frame(siteID=data$siteid, ValleyClass=data$ValleyClass)

# Now make the "data" data frame for our continuous variable
        my.cont.data <- data.frame(siteID=data$siteid,
                    OmykissPerMeter=data$OmykissPerMeter,
                  ChinPerMeter=data$ChinPerMeter)



#####################################
# Now, let's use "cat.analysis" to analyze the categorical data
        Results.Cat <- cat.analysis(site=my.sites,
                  subpop = my.subpop, design = my.design,
                  data.cat = my.cat.data, popsize=my.popsize)


# And see the results
        names(Results.Cat)
# Print all results
        print(Results.Cat)
```

```
# Write results to a file
        write.csv(Results.Cat, "Results.Cat.csv")

# Print results in percentiles
        print(Results.Cat[,c(1:9)])

# Print results scaled to total lengths
        print(Results.Cat[,c(1:4,10:13)])



#####################################################################
# Analyze Continuous Data usisng "cont.analysis" function

        Results <- cont.analysis(sites=my.sites, subpop = my.subpop,
                design = my.design, data.cont = my.cont.data,
                total=TRUE,
                                        popsize=my.popsize)


# And see the results
        Results$CDF
        Results$Pct


# Write results to file(s)
        write.csv(Results$Pct, "ResultsPCT.csv")
        write.csv(Results$Pct[Results$Pct$Statistic=="Total",], "ResultsPCT.Total.csv")


# print some results to the screen
        names(Results)
        with(Results$CDF, data.frame(Type,
                 Subpopulation, Indicator, NResp, Estimate.P,LCB95Pct.P, UCB95Pct.P, Estimate.U,
LCB95Pct.U, UCB95Pct.U))

#############################################################################
# Make a CDF plot for a subpopulation of choice
# for steelhead abundance.

# Select Subpopulation for plotting cdf plots to screen
# (Pick one of the four options below)
        Plot.Subpop= Results$CDF[Results$CDF$Subpopulation=="All.Sites",]
        Plot.Subpop= Results$CDF[Results$CDF$Subpopulation=="Lower Lemhi",]
        Plot.Subpop= Results$CDF[Results$CDF$Subpopulation=="Upper Lemhi",]
        Plot.Subpop= Results$CDF[Results$CDF$Subpopulation=="Hayden Creek",]
```

```
# Manually make some plots
        head(Plot.Subpop)
        title = Plot.Subpop$Subpopulation[1]
        metric = Plot.Subpop$Indicator[1]


# Generate a CDF Plot for Estimate.P
        with(Plot.Subpop, {
        plot(Value[Indicator=="OmykissPerMeter"],Estimate.P[Indicator=="OmykissPerMeter"],
         type="l",          main = paste(title, "O.Mykiss Per Meter CDF Estimate"),
        xlab="Total Abundance", ylab="Cumulative Pct")

        lines(Value[Indicator=="OmykissPerMeter"],LCB95Pct.P[Indicator=="OmykissPerMeter"],
                col="red")
        lines(Value[Indicator=="OmykissPerMeter"],UCB95Pct.P[Indicator=="OmykissPerMeter"],
                 col="red")
        })

# add a legend
        legend.text = c("CDF Estimate", "95% Confidence Limits")
        legend("bottomright", legend.text,col = c("black", "red"), pch=19)
###############################################################################


# Use function "cont.cdfplot"
# Create a PDF file containing plots of the CDF estimates--the function
# generates the cdf plots from the data in Janish_CDF_Estimates$CDF
        cont.cdfplot("Example_Omykiss_CDF.pdf", Results$CDF)


# Conduct a WALD test for differences among subpopulations
        CDF_Tests <- cont.cdftest(sites=my.sites, design= my.design,
      data.cont= my.cont.data,
     subpop = data.frame("SiteID"=data$siteid, "ReportingUnit"=data$ReportingUnit))

# Print results and write results to a file
          print(CDF_Tests)
          write.csv(CDF_Tests, "CDF_Tests_Example.csv")


#################################################################
# Analyze again, but include measurement variance
# Note: These measurement variances are "made up" for the sake
# of this example

        Measurement.variance = c("OmykissPerMeter" = .3, "ChinPerMeter" = .3)
        Results.D <- cont.analysis(sites=my.sites, subpop = my.subpop,
                design = my.design, data.cont = my.cont.data,
```

```
        total=TRUE,
                                popsize=my.popsize,
                                sigma= Measurement.variance)

    names(Results.D$CDF)
    Results.D$CDF[Results.D$CDF$Subpopulation=="All.Sites",1:9][1:10,]
    Results.D$CDF.D[Results.D$CDF.D$Subpopulation=="All.Sites",1:9][1:10,]


# Use function "cont.cdfplot"
#Create a PDF file containing plots of the CDF estimates--the function generates the cdf plots from the data in
Janish_CDF_Estimates$CDF
    cont.cdfplot("Results.D.CDF.pdf", Results.D$CDF)
    cont.cdfplot("Results.D.CDF.D.pdf", Results.D$CDF.D)
```