Statistical inference of the sufficiency of spatial data

Damian Satterthwaite-Phillips

Lead Scientist

Northwest Arctic Borough, Alaska

Subsistence Mapping Project

Abstract:

Using land-use data from a sample of interviewees documenting their locations used for hunting, fishing, gathering, and trapping subsistence resources, we provide methods for estimating how close the sample data come to representing the population data. We demonstrate that the amount of novel information provided decreases as the number of interviewees increases. Furthermore, the rate at which new information decreases can be effectively modeled as *new information* = $\left(\frac{[number\ of\ interviewees]+a}{b}\right)^{c}$, using nonlinear regression methods to estimate the parameters. Although we use subsistence data gathered from interviewees in this paper, the same methods may be more broadly applied to similar spatial data, including, e.g., species distributions as estimated from multiple fly-over counts, or a sample of radio-collared individuals. While the data are not confined to Arctic research, as research in the Arctic continues to increase, these methods may be applied to better estimate the sufficiency of the data gathered to represent the population statistics.

Introduction

The methods described in this paper were developed as part of the subsistence mapping project conducted by the Northwest Arctic Borough, AK (Satterthwaite-Phillips, et al. in press). A primary goal of this project was to document current land-use by the residents in the study area for obtaining subsistence resources. We gathered data through semi-directed interviews about individuals' land use for subsistence hunting, fishing, gathering and trapping. Being unable to conduct a census of all residents in the study area, we aimed to infer the population-level behavior from a sample of residents. Ideally, the goal would be to have comprehensive documentation of land use, but due to time and financial limitations, only a sample of individuals could be interviewed. Hence we wish to quantify how close the sample comes to being comprehensive.

Intuitively, because there is frequently considerable overlap in individuals' land use—e.g., many individuals hunt caribou at the same river crossings, or net salmon at important locations along their migrations—there are "diminishing returns" on the amount of new information provided by each additional interviewee. Simply put, if the first ten interviewees have already provided data on the vast majority of locations used by the community, the eleventh interviewee will have little new information to add. Below we formalize this intuitive notion. The question we are addressing here is: "of all the land used by the community for subsistence, how close are we to documenting all of it from our sample?" That is, the focus is on whether or not a given region was documented, and not on the relative intensity of use in different regions. There are established methods for inferring the latter, which we address briefly below. In addition to a description of the methods, this paper provides an R (R Core Team, 2015) script for implementing both the analysis of the completeness of a data set, as well as for quantifying intensity of use at each location.

Methods

Measuring the area covered by each interviewee—

For the purpose of this analysis the amount of "information" obtained from each interviewee was quantified by the area that individual uses for subsistence practices, where area is approximated to an arbitrary degree of precision using rasterized versions of each individual's data. For each interviewee, a GIS shape file was constructed out of the union of all of that interviewee's individual polygons. This shape file was then converted into a raster file—or a pixelated equivalent—in which, for our study, each pixel represented 1 km^2. Within the extent of the mapped area then, each pixel is either defined as part of that individual's total subsistence area, or not. The raster file was converted into an ASCII file, which is simply a textual representation of the pixelated data, in which each pixel is assigned a value: 1 if it was included in the interviewee's total area, -9999 if it was not. The ASCII files were further converted by changing the -9999 values to 0, and representing the data as a matrix of 1s and 0s. The matrices were then further manipulated in R (version 3.2.0; R Core Team 2015). The total area for each individual was calculated simply by summing the value of all cells in the matrix. In other words, if an interviewee's area included 935 pixels, each of value 1, the sum is simply

935. Because each pixel is 1 km^2, the total number of pixels is equivalent to the total number of square km.

Determining the amount of new *information—*

For this analysis, we were concerned not with how much *total* information the interviewee provides, but how much *new* information he or she provides. This is defined here simply as that area included in the interviewee's subsistence area *that was not included in any previous interviewee's area*. When dealing with matrices, this is straightforward to calculate. For the first interviewee, all data must be "new information." For the second interviewee, we sum the number of pixels that are 1 in the new matrix and 0 in the previous interviewee's matrix. Because the matrices are binary, we can use mathematical logic to perform these calculations:

(1) New information = $$\sum_i (newMatrix_i \wedge \neg previousMatrices_i)$$

where *i* represents the individual cells in the matrix. For the third interviewee, we count all data not found in either of the previous interviews. For this purpose, it is helpful to store a matrix that is the union of all previous interviewees' data (i.e., a cell is given the value 1 if *any* previous interviewee had a 1, and 0 otherwise). If we call the matrix that represents the union of all previous data, the cumulative matrix, then:

(2) Cumulative matrix$_i$ = previousMatrix$_i$ $\vee$ newMatrix$_i$,

where the *i*s again represent the individual cells of the matrix. If the cumulative matrix is initialized as all 0s, then equation (1) may be used for every interviewee considered, where *previous matrices* are represented by the cumulative matrix.

Following this procedure, the amount of new information may be plotted as a function of the number of interviewees, as in Figure 1.

Randomizing the data—

Figure 1 was constructed from our data gathered in the village of Selawik. In Figure 1, the first interviewee's total area included nearly 1500 km^2. The second interviewee contributed just over 3,500 km^2 of *new* data *not* provided by the first interviewee. The third interviewee provided about 500 km^2 of novel data, and the fourth provided almost no new data, meaning that all of his or her search areas were already covered by previous interviewees. In general, as more and more interviewees are added, the amount of land available that has not already been reported becomes less and less. On average, we expect the new information to decrease as a function of the number of interviewees. In this example, we see that most of the later interviewees provided relatively little new information. However, the exact rate at which the information drops off is dependent on the order in which the interviewees are added. If, by chance, the first interviewee had an unusually abundant amount of information, additional interviewees will not contribute much novel information. Conversely, if most interviewees contain only a small amount of data, and the last individual chosen happens to have a lot of data, we will observe a spike at the far right of the graph. In both cases, this is really just an artifact of the chance order in which interviewees were chosen. To illustrate this, in Figure 2 below, the same Selawik data from Figure 1 are replotted, but with the interviewees in a different order. What we want, then, is the *mean*, or *expected* behavior of how the function behaves. To determine the mean value, we randomly permute the interviewees, and replot the data. We repeat the process repeatedly, randomly permuting the interviewees at each iteration, and averaging the results from the two trials. As the iterations increase, the average value of the function will begin to converge. Figure 3 shows the Selawik data after 10 iterations of random sampling and averaging the results. Finally, Figure 4 shows the average value of 1,000 iterations. At 1,000 iterations, it is apparent that the mean behavior of the function—indicated by the thick black line—is approaching a curve, in which, as expected, the amount of new information decreases as a function of the number of interviewees.

Describing the function mathematically—

In order to be able to predict how much new information the n^{th} interviewee might be expected to provide, we need to be able to describe the function mathematically. The exponential function,

$$\text{(3a)} \qquad y = ae^{-bx},$$

or, in the case of our data,

(3b) $new\ information = ae^{-b(number of\ interviewees)}$

where *a* and *b* are constant parameters that determine the shape of the curve (i.e., how rapidly it drops off), and *e* is Euler's constant (2.71828…), intuitively seems like a good choice, and in the case where land use is truly random, the fit is indeed good. However, when there exists preference for certain regions over others, the fit tends to be poor in the limits. In Figure 4 we provide the best fitting exponential function, and for the region above ten interviewees, the fitted data are systematically lower than the observed data, which lead to an erroneous conclusion that the data are more comprehensive than they truly are. In both the random and non-random case, an inverse function,

(4) $y = \left(\frac{x+a}{b}\right)^{c}$; or

(4b) $new\ information = \left(\frac{[number\ of\ interviewees]+a}{b}\right)^{c}$,

where *a*, *b*, and *c* are again constant parameters that determine the shape and location of the curve. As *c* will be a negative value, we refer to this as the "inverse function". R provides a function, nls (nonlinear least squares; see Bates and Watts 1988, Bates and Chambers 1992), that may be used to find the values of *a*, *b*, and *c* that best fit the data (minimizing the sum of squares of the error). This inverse function provides a very good approximation of the data. Compare the exponential fit in Figure 4 with the inverse fit in Figure 5. The fit is clearly better, and lies completely within the 95% confidence interval for the mean (dotted black lines). We provide a script (Appendix 2) that allows users to input their own data and to determine the function fitted to their data.

While not the primary focus of these methods, to further aid the user, the script (Appendix 2) also allows for estimation of the proportion of the population using each given cell of the rasterized data, along with the upper and lower limits of a user-specified confidence interval. For these, the maximum likelihood estimate for the proportion at any locus is simply the observed proportion in the sample at each locus, and the standard error is computed as:

$$(5)\ \mathrm{SE} = \sqrt{\frac{p(1-p)}{n}},$$

where *p* is the observed proportion, and *n* is the number of interviewees. Because our data are not yet available for public release, example output of this part of the script is provided using simulated data (for which the script is also available; Appendix 1) in Figure 6.

Although we discuss the above methods in terms of our own data, they may be applied more broadly to other data that estimate spatial data from multiple data sources, including, for example, estimates of an animal species distribution based on multiple fly-overs, or a sample of radio-collared individuals. These methods may be applied to future research conducted in the Arctic (and elsewhere) to better quantify the completeness of the data gathered.

Acknowledgements:

This research was conducted for the Northwest Arctic Borough, Alaska, and was made possible, in part, with funding provided by Royal Dutch Shell, Oak Foundation, and the Coastal Impact Assistance Program (CIAP) administered by the United States Fish and Wildlife Service. CIAP provides federal grant funds derived from federal offshore lease revenues to oil producing states for conservation, protection, or restoration of coastal areas including wetlands; mitigation of damage to fish, wildlife, or natural resources; planning assistance and the administrative costs of complying with these objectives; implementation of a federally-approved marine, coastal, or comprehensive conservation management plan; and mitigation of the impact of outer Continental Shelf activities through funding of onshore infrastructure projects and public service needs. CIAP funds are sourced from the Outer Continental Shelf (OCS) Lands Act, as amended; 31 U.S.C. 6301-6305.

References

Bates, D. M. and Watts, D. G. 1988. Nonlinear Regression Analysis and Its Applications, John Wiley & Sons.

Bates, D. M. and Chambers, J. M. 1992. Nonlinear models. In: Chambers, J. M. and Hastie, T. J., eds. Statistical Models in S. Boca Raton, FL: Chapman & Hall / CRC. 421-454.

R Core Team. 2015. R: a language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org/

Satterthwaite-Phillips, D., Krenz, C., Gray, G. and Dodd, L. In press. *Iñuuniaḷiqput iḷiḷugu nunaŋŋuanun (Documenting our way of life with maps): Northwest Arctic Borough subsistence mapping project.* Vol. 1. Kotzebue, AK: Northwest Arctic Borough.

Figure 1. Plotting "new information" as a function of the number of interviewees—arbitrary ordering of interviewees

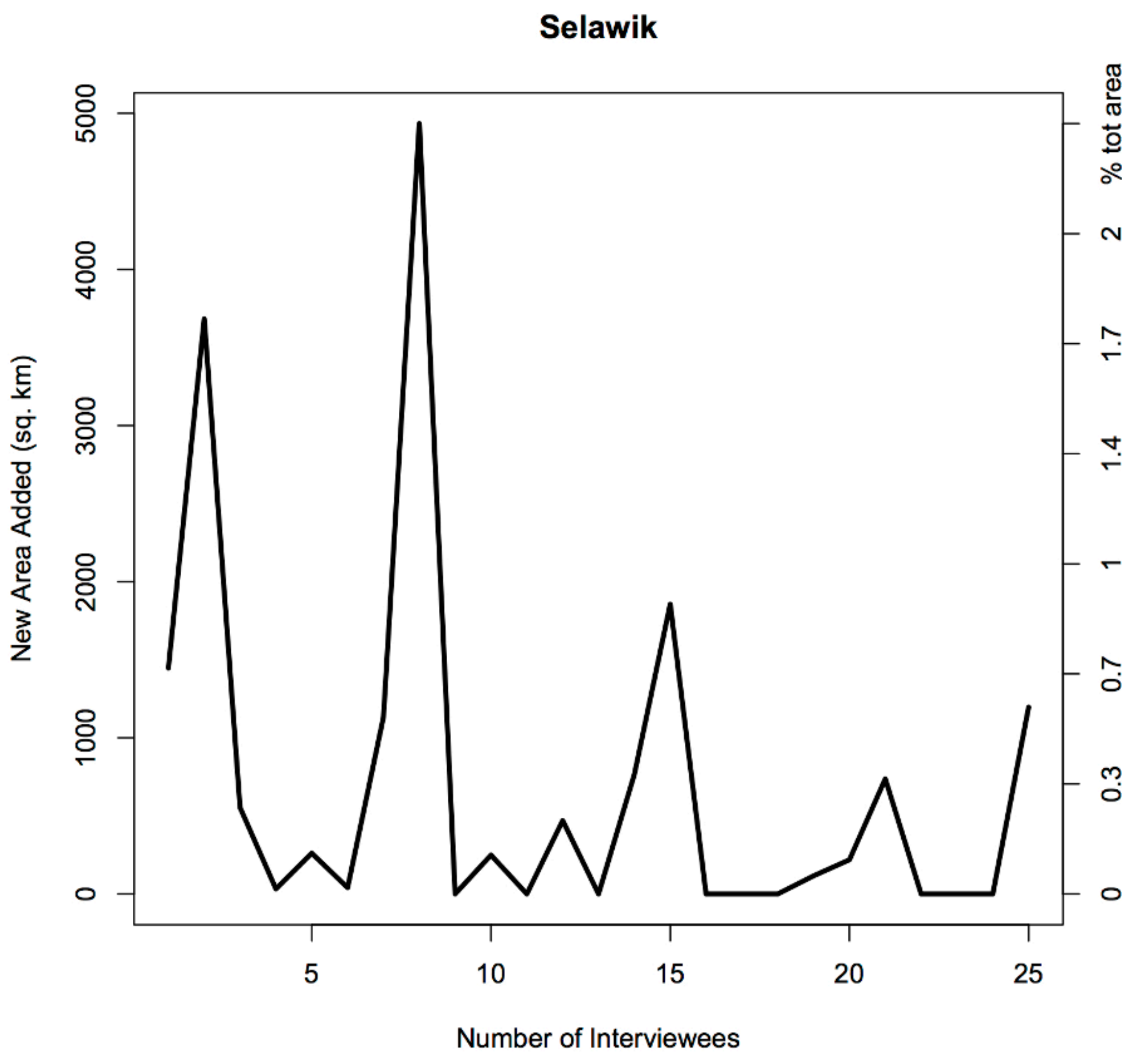


Figure 2. "New information" as a function of the number of interviewees is subject to the ordering of the data—arbitrary ordering of interviewees

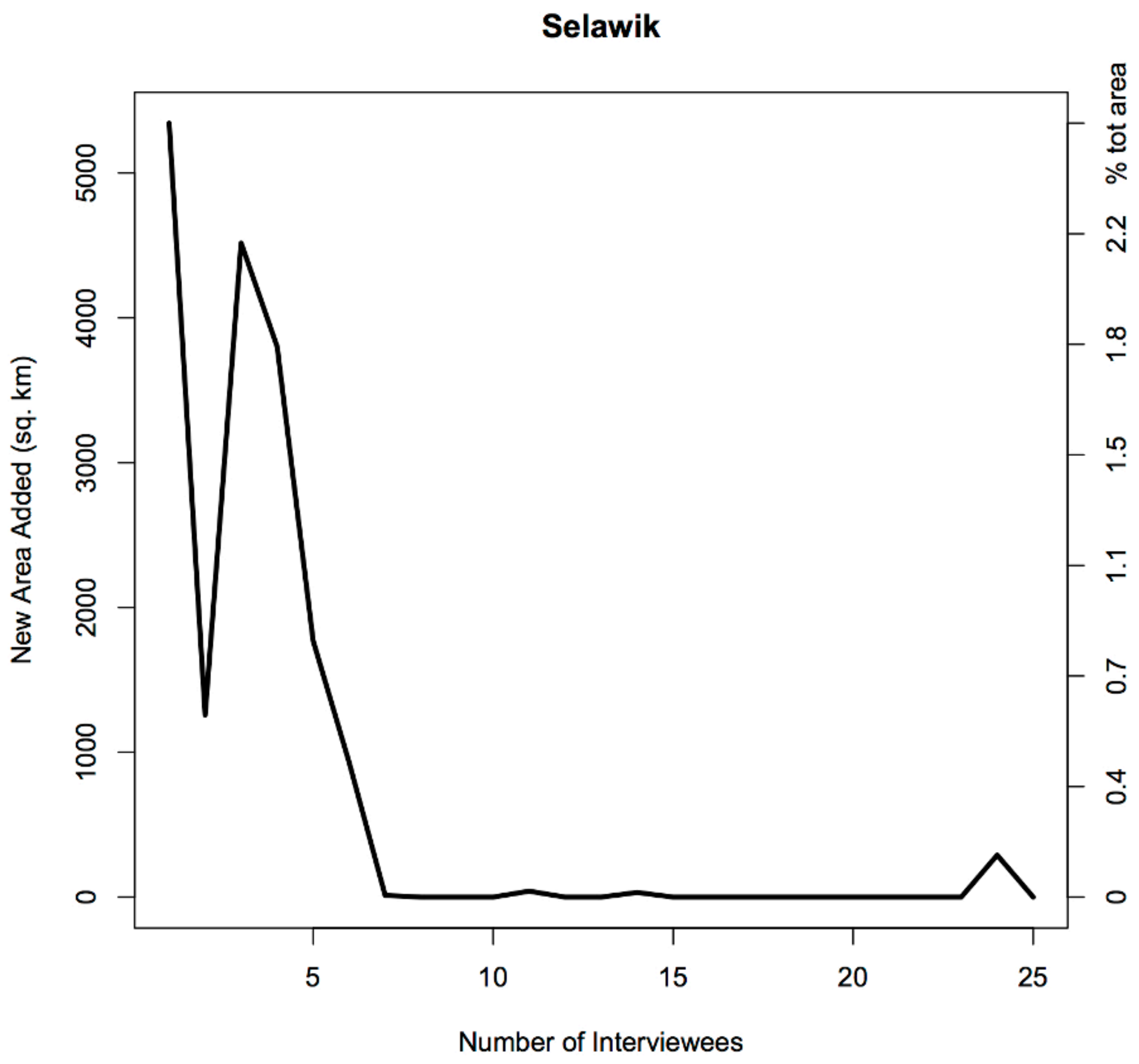


Figure 3: Average function of new information after 10 random samplings.

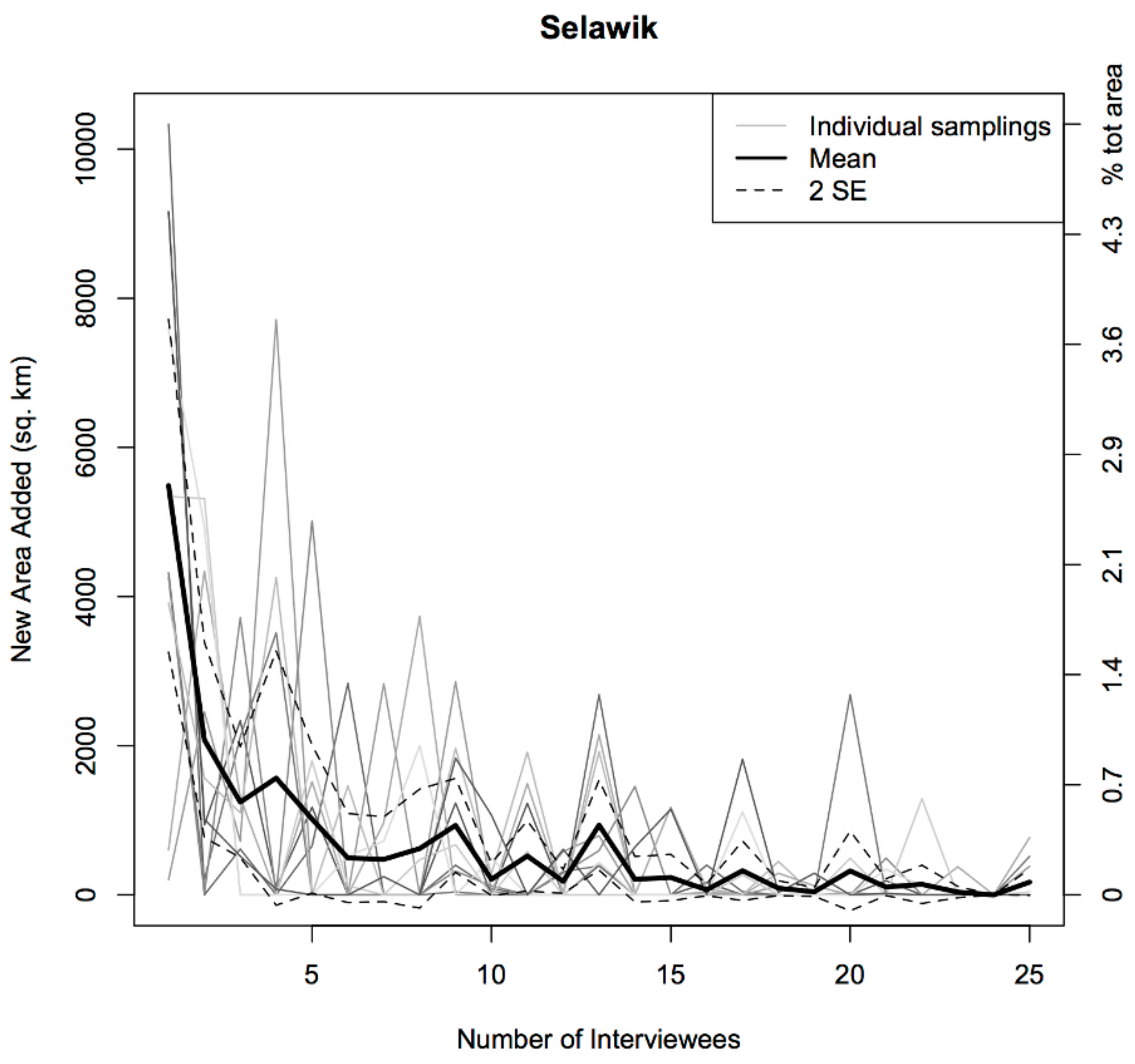


Figure 4: Average function of new information after 1,000 random samplings; exponential model fitted to data

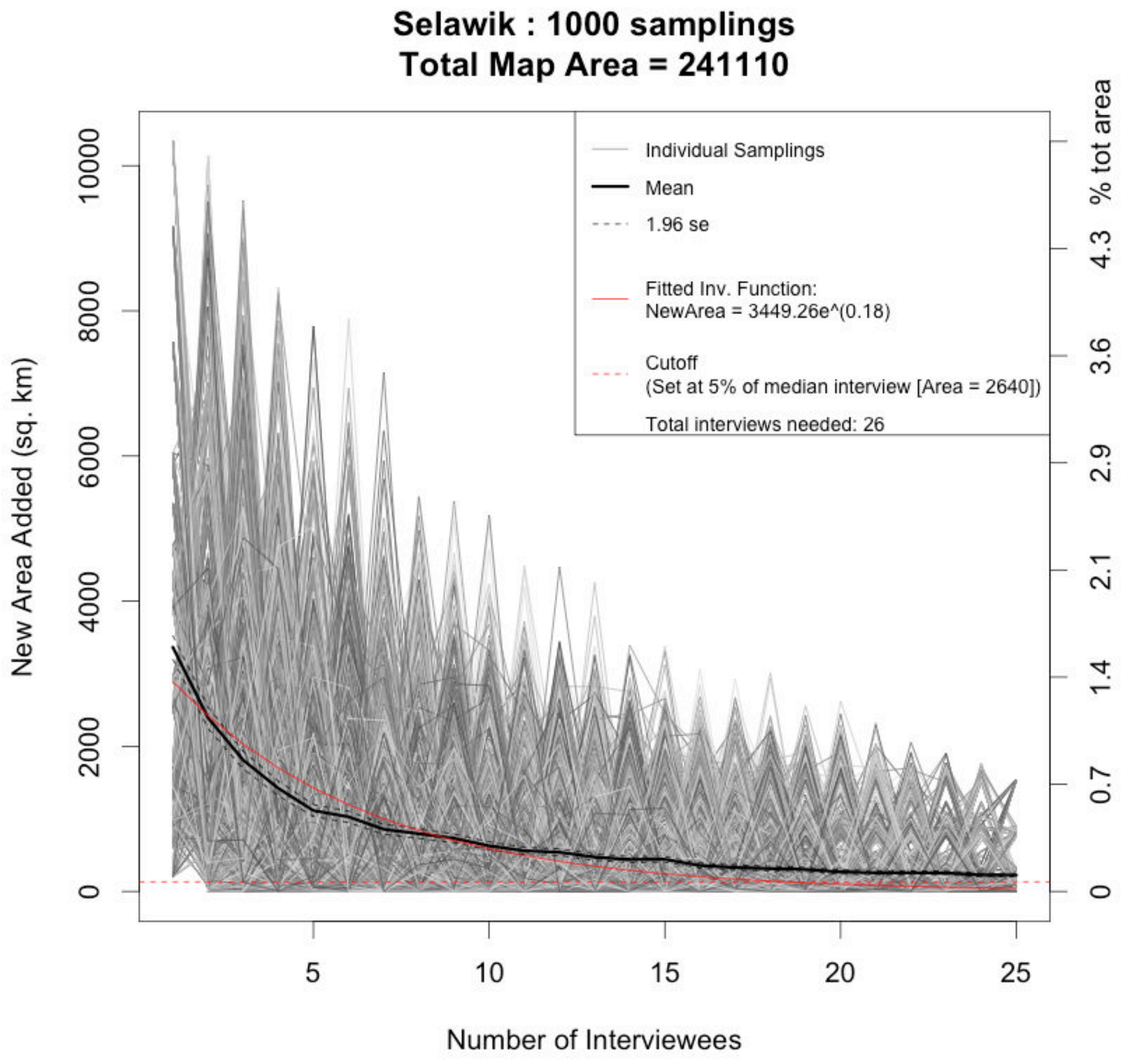


Figure 5: Average function of new information after 1,000 random samplings; inverse model fitted to data

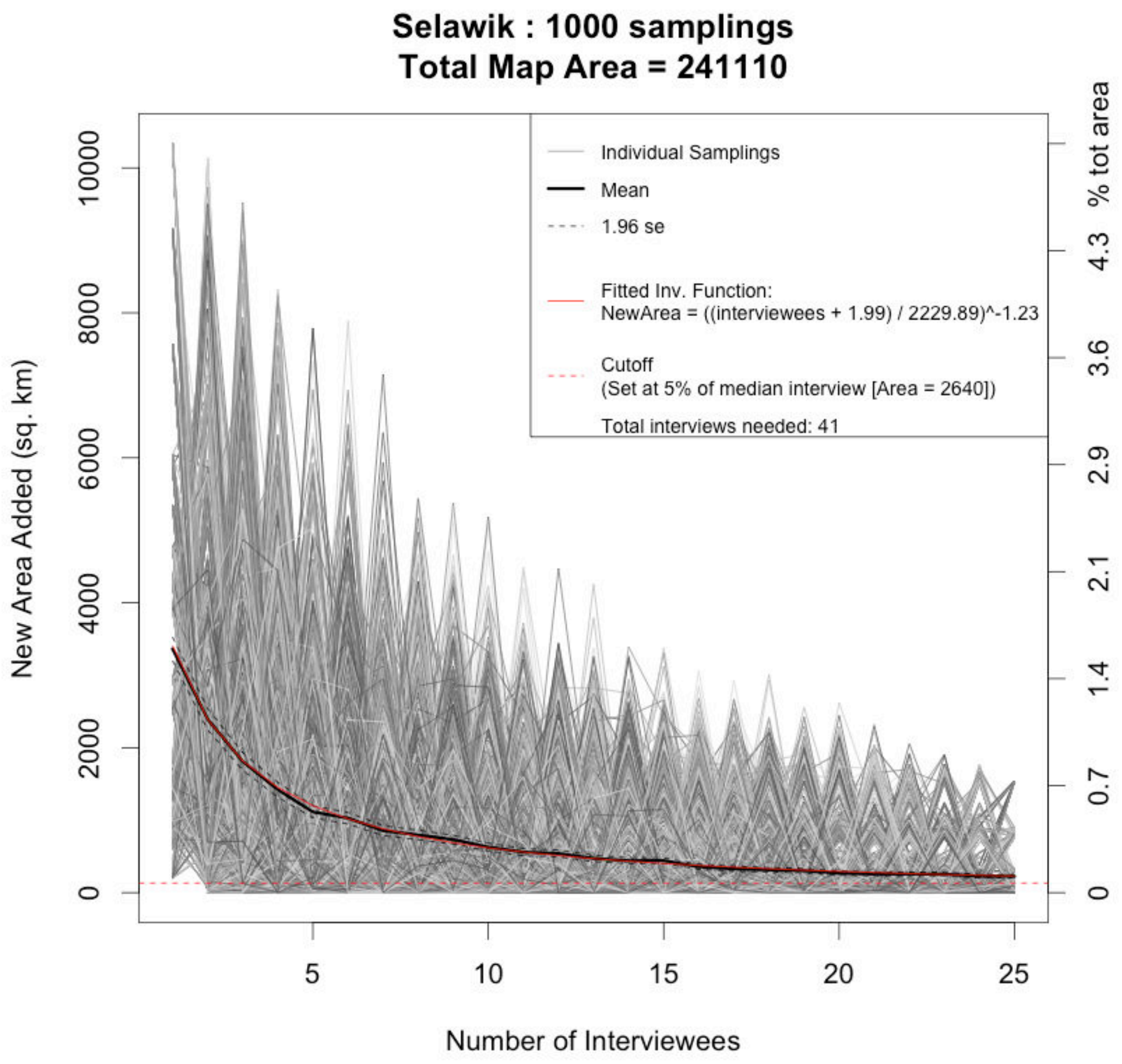
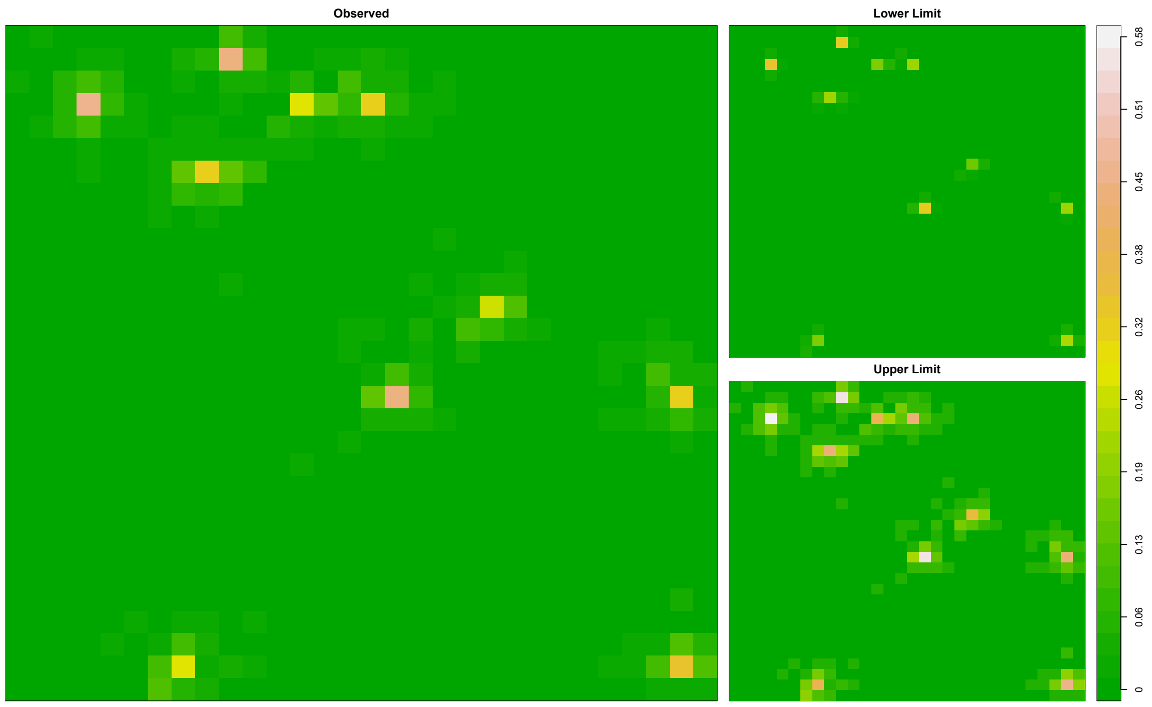


Figure 6: Simulated output of estimated proportion of the population using each grid cell within the region, along with upper and lower limits of an arbitrary (user-specified) confidence interval.



Appendix 1: R source code for simulated data.

```
#=================================================================#
#                                                                 #
# SIMULATE SPATIAL DATA:                                          #
# This script generates simulated data for both randomly          #
# distributed data,and data showing a clear preference for certain #
# regions over others, and allows the user to input different     #
# parameters according to underlying assumptions about the data   #
# she wishes to simulate.                                         #
#  @author: Damian Satterthwaite-Phillips <damiansp@gmail.com>    #
#  @version: 26 Oct 2015                                          #
#                                                                 #
#=================================================================#



# Generate a random matrix intended to show an individual's land use
# @param cellsPerSide: the number of rows and columns of the matrix
# @param meanLocations: the mean number of unique search areas in the
#                            population; the number of actual areas will
#                            be randomly generated as a Poisson process
#                            with lambda = meanLocations
# @param graph: if true a graphical output of the matrix wil be
#                   generated
# @return: a matrix with 1s in the simulated "use areas" and 0s
#              elsewhere
rMatrix = function(cellsPerSide = 10, meanLocations = 10, graph = T) {
    m = matrix(0, nrow = cellsPerSide, ncol = cellsPerSide)

    # Determine the number of use locations as a Poisson random
    # variable
    n = rpois(1, meanLocations)

    # randomly distribute the n locations in the matrix
    locations = sample(1:cellsPerSide^2, n)
    m[locations] = 1

    if (graph == T) {
        # set graph to no margins
        par(mar = c(0, 0, 0, 0))
        image(m)
    }

    return (m)
}

# Helper functions:
# calculate euclidean distance between two cells
# @params: i1, j1 = [i, j] indices of cell one
# @params: i2, j2 = [i, j] indices of cell two
# @return: euclidean distance between cells in cell units
distance = function(i1, j1, i2, j2) {
    return(sqrt((i1 - i2)^2 + (j1 - j2)^2))
}

# Generate a matrix intended to show an individual's land use, with a
# greater probability of using "hot spots"
# @params cellsPerSide, meanLocations, graph: as above
# @param falloff: the rate at which preference for a hotspot decreases
#                    with distance (should be a value between 0 and 1);
#                    see below for details.
# @param hotSpotLocations: the cell locations indicating the hotspots
#                              (this must be specified so that it is
#                              consistent for all simulated individuals);
#                              this is expressed as a vector of length
#                              between 1 and cellsPerSide^2
hotspotProbMatrix = function(cellsPerSide = 10, falloff = 0.7,
                              hotSpotLocations = list(c(1, 1), c(3, 3),
                                                        c(5, 7)),
                               graph = T) {
    m = matrix(0, nrow = cellsPerSide, ncol = cellsPerSide)

   # Initiate a matrix of probabilities with a value of 1 at hotspots,
   # and 0 elsewhere
   mProbs = m
   for (hs in hotSpotLocations) {
      mProbs[hs[1], hs[2]] = 1
   }

   # For any given cell, [i, j], determine the distance to the nearest
   # hotspot
   distToNearestHotspot = function(i, j) {
       minDist = Inf
       for (hs in hotSpotLocations) {
           dist = distance(i, j, hs[1], hs[2])
           if (dist < minDist) {
               minDist = dist
           }
```

```
        }

        return(minDist)
    }

    # Assign weights to all cells in matrix.
    # For each unit cell away from a hot spot, the weight is scaled by a
    # factor of <falloff>, e.g., hs = 1, then at a distance 1, weight =
    # 1*<falloff>, at distance 2, weight = 1*<falloff>^2, or more
    # generally, weight = <falloff>^distance
    for (i in 1:cellsPerSide) {
        for (j in 1:cellsPerSide) {
            dist = distToNearestHotspot(i, j)
            mProbs[i, j] = falloff^dist
        }
    }

    # Scale mProbs so all cells sum to 1
    mProbs = mProbs / sum(mProbs)

    if (graph == T) {
        par(mar = c(0, 0, 0, 0))
        image(mProbs)
    }

    return (mProbs)
}
```

```
# Tests:
#hspM = hotspotProbMatrix()
#hspM = hotspotProbMatrix(
#    cellsPerSide = 20, falloff = 0.95,
#    hotSpotLocations = list(
#        c(1, 1), c(2, 3), c(5, 7), c(11, 13), c(17, 19)
#    )
#)



# Simulate one individuals data using a given hotspot probability
# matrix
# @param hsProbMatrix: the output matrix from hotspotProbMatrix()
# @param meanLocations: the population mean number of locations; the
#                       number for any individual run will be a Poisson
#                       random variable with mean (lambda) =
#                       meanLocations
# @graph: if TRUE, graphic output will be generated
# @return: a matrix of the individuals simulated use areas
hotspotMatrix = function(hsProbMatrix, meanLocations = 10,
                         graph = T) {
    # Number of unique locations for a single simulated individual
    n = rpois(1, meanLocations)
    m = hsProbMatrix * 0

    locations = sample(1:prod(dim(m)), n, prob = hsProbMatrix)

    m[locations] = 1
    if (graph == T) {

        # set graph to no margins

        par(mar = c(0, 0, 0, 0))

        image(m)

    }


    return (m)

}





# Create n = 50 random matrices and 20 "hot spot" matices, and store

# each to a list.  Each hot spot matrix will have hs = 10 hotspots, and

# indiviuals will have meanLocs = 10 locations on average. Matrices

# will be 30 x 30

set.seed(123)

n = 50

hs = meanLocs = 10

mdim = 30


rMatrices = list()

hsMatrices = list()


# randomly generate hotspots

hsLocations = list()

for (i in 1:hs) {

    hsLocations[[i]] = sample(30, 2, T)

}
```

```
hsProbMatrix = hotspotProbMatrix(cellsPerSide = mdim, falloff = 0.2,
                                 hotSpotLocations = hsLocations)



for (i in 1:n) {
    mName = paste('m', i, sep = '')
    rMatrices[[mName]] = rMatrix(cellsPerSide = mdim,
                                 meanLocations = meanLocs, graph = F)
    hsMatrices[[mName]] = hotspotMatrix(hsProbMatrix = hsProbMatrix,
                                        meanLocations = meanLocs,
                                        graph = F)
}

# Plot the sum of all matrices in each group.
rMatrixSum = rMatrices[[1]]
hsMatrixSum = hsMatrices[[1]]

for (i in 2:n) {
    rMatrixSum = rMatrixSum + rMatrices[[i]]
    hsMatrixSum = hsMatrixSum + hsMatrices[[i]]
}

# Plot the sum of all individuals, simulating the "population" use area
quartz()
par(mfrow = c(1, 2))
par(mar = c(0, 0, 1, 0))
image(rMatrixSum, main = 'Random Areas')
image(hsMatrixSum, main = 'Hot Spots', yaxt = 'n')
```

```
#================== END OF SCRIPT =================#
```

Appendix 2: R source code for analysis of completeness of data

```
#==============================================================#
#                                                              #
#  ANALYSIS OF COMPLETENESS:                                   #
#  This script runs the analysis of completeness described in Methods #
#  Section 2.1.1.6.2 “Analysis of Data Sufficiency”            #
#  @input: a series of .csv files (with no headers or row names),     #
#          each equivalent to a raster representing the union of all  #
#         polygons for a given interviewee, and for which 1          #
#         indicates the area enclosed in the polgyon, and 0          #
#         indicates area outside.                                     #
#  @output: a vector of values c(totalArea, a1, a2, ..., an), where   #
#           total area indicates the total number of cells in the     #
#         matrix, a1 is the area of the polygons enclosed by the      #
#         first randomly chosen matrix, a2 = the area enclosed by     #
#         the second randomly chosen matrix minus the area of a1      #
#         (or the new area found in the matrix), a3 = randomly        #
#         selected 3rd matrix - a1 - a2 (or the new area found in     #
#         the matrix), etc.                                           #
#              An optional graphical output may also be generated,    #
#         showing the best fitting model.                             #
#  @author: Damian Satterthwaite-Phillips <damiansp@gmail.com>       #
#  @version: 26 Oct 2015                                             #
#                                                                    #
#==============================================================#


# Data: supply paths to source files (e.g.,
# '~/Desktop/data/interviewee1.csv'; on Windows machines backslashes
# may be necessary instead:
```

```
# 'C:\Desktop\data\interviewee1.csv'



# NOTE: If using real data begin here. If using the
# simulated data, skip ahead to the line: “siteList = rMatrices” below

# Source file 1 (a single interviewee’s data):
s1 = as.matrix(read.csv('Path to file for first interview',
                        header = F))
# In some cases this results in a final column of NAs being added;
# check with:
head(s1)

# If so, remove the NA column:
#s1 = s1[,-dim(s1)[2]]

# Source file 2
s2 = as.matrix(read.csv('Path to file for second interview',
                        header=F))
#head(s2)

# If final NA column present, remove:
#s2 = s2[,-dim(s2)[2]]

# ...repeat for all input files for a given study site

#siteList = list(s1,s2, # Enter all such matrices
#             )
# For the purposes of the simulated data, we skip the above steps
#(commented out), and instead set our siteList to either rMatrices or
# hsMatrices
siteList = rMatrices



# The following code assumes that all input matrices are of the same
# dimension. If not, new input files will have to be created to correct
# for this.
# Check that all are of the same dimension:
for (i in 1:(length(siteList))) {
    print(dim(siteList[[i]]))
}

# The following are helper functions necessary to perform the
# computations.

# Determine total number of cells in matrix
totalArea = function(M) {
    dim(M)[1] * dim(M)[2]
}

# Calculate the number of cells with value 1 in newM that are not
# already in existingM
newArea = function(existingM, newM) {
    sum(newM & !existingM)
}

# Create a matrix that is the union of the existingM and the newM: any
# cell that is 1 in either existingM or newM (or both) is 1; any cell
# that is 0 in both remains 0
union = function(existingM, newM) {
    1 * (existingM | newM)    # Multiplication by 1 coerces output to
                              # numeric instead of boolean
}

# Function to calculate standard error for a sample mean
se = function(x) {
    sd(x) / sqrt(length(x))
}

# Create output of new areas added from each matrix in listOfMs
# @output: c(totalArea, a1, a2, ..., an) as explained above
newAreas = function(listOfMs, graph = T) {
    # Create an initial all-0 matrix
    m0 = listOfMs[[1]] - listOfMs[[1]]

    # Initialize an output placeholder
    out = c(totalArea(m0))

    # Initialize a matrix to be the union of all matrices added
    unionM = m0

    # Create a placeholder to store the union at each step
    unionList = list(unionM)

    for (i in listOfMs) {
        # Calculate the new area contributed by the next matrix
        out = c(out, newArea(unionM, i))


        # Update unionM by adding next matrix; update unionList
        unionM = union(unionM, i)
        unionList = list(unionList, unionM)
    }


    cumArea = cumsum(out[2:(length(out))])
    if (graph == T) {
        plot(cumArea, type='l', ylab = 'Cumulative Area',
              xlab = 'Number of Maps Added')
        abline(h = out[1], col=2)
        legend('topleft', lty = 1, col = 2, legend = 'Total Map Area',
                bg = 'white')
    }


    list(mapArea = out[1], addedArea = out[2:(length(out))],
         cumArea = cumArea, combinedMaps = unionList)
}


# Main function--the completeness analysis: Repeat the newAreas()
# function <iterations> times, randomizing the list each time.
# Write cumArea values to a data.frame for analysis.
# Plot all cumArea plots along with mean and 2se.
# Determine the best exponential decay model, and plot.
# @param locale: name of the location
# @param listOfMs: a list of matrices, each of which should have the
#               same dimensions
# @param iterations: number of random permutations over which to
#                  average the behavior--recommend a small number
#                  (~10) for initial testing
# @param graph: if true, displays graphical output
# @params cStart, dStart, eStart: initial coefficients to pass to the
#                                   nls function; some trial and error
#                                   may be needed to get the function to
#                                   converge.
# @param limitPercent: Cutoff threshold (see text).  Existing code
#                    represents this as a percent of the information
#                    in the median input matrix.
# NOTE: Although not appropriate for data that are not randomly
# distributed, the exponential function is typically easier to fit to
# the data, particularly when suitable parameter estimates cannot be
# found to get the nls function to converge. Hence the exponential
# model is also provided below (commented out).
completeness = function(locale = '', listOfMs, iterations, graph = T,
                        limitPercent = 1, cStart = 5, dStart = NULL,
                        eStart = -1) {

    # Create placeholder for output
    out.df = data.frame()

    for (i in 1:iterations) {
        # Print progress after every 10 iterations
        if (i %% 10 == 0) {
            cat('Progress: ', 100 * i / iterations, '%\n', sep = '')
        }

        # Randomize list
        currentList = sample(listOfMs)


        # Run newAreas

        nextData = newAreas(currentList, graph = F)


        # Append cumArea data to out.df

        out.df = rbind(out.df, c(nextData$mapArea,

                                 nextData$addedArea))

    }


    nCols = dim(out.df)[2]

    means = apply(out.df, 2, mean)

    ses = apply(out.df, 2, se)

    totalArea = out.df[1,1]

    meanCumArea = sum(out.df[,2:nCols]) / iterations

    cutoff = (limitPercent / 100) * median(out.df[,2])


    # Model the information added as an exponential function of the

    # number of interviewees

    interviewees = 1:(nCols - 1)

#    exp.mod = nls( means[2:nCols] ~ a * exp(-b * (interviewees)),

#                     start = list(a = means[2], b = 0.2) )


#    a = coef(exp.mod)[1]

#    b = coef(exp.mod)[2]



    # Model the information as an inverse function

    # initialize dStart such that y-intercept = means[2]
    if (is.null(dStart)) {
        dStart = cStart / means[2]^(1 / eStart)
    }

    inv.mod = nls(means[2:nCols] ~ I(((interviewees + c) / d)^e),
                  start = list(c = cStart, d = dStart, e = eStart),
                  trace = T)

    c = coef(inv.mod)[1]
    d = coef(inv.mod)[2]
    e = coef(inv.mod)[3]


    i = 1
#    newAreaAdded = a * exp(-b * 1)
 newAreaAdded = ((i + c) / d)^e

#    while (newAreaAdded > cutoff) {
#         i = i + 1
#         newAreaAdded = a * exp(-b * i)
#    }

    while (newAreaAdded > cutoff) {
     i = i + 1
          newAreaAdded = ((i + c) / d)^e
    }


    interviewsNeeded = i
    if (graph == T) {
        # Create a palette of grey-tones to use to plot individual
        # samples
        palette = c('#555555', '#666666', '#777777','#888888','#999999',
                    '#AAAAAA', '#BBBBBB', '#CCCCCC', '#DDDDDD')

        mainTitle = paste(locale, ':', iterations,
                          'samplings\nTotal Map Area =', totalArea)

        # Plot individual samplings
        matplot(t(out.df[,2:nCols]), type = 'l', lty = 1, col = palette,
                ylim = c(0, max(out.df[,2:nCols])),
                xlab = 'Number of Interviewees',
                ylab = 'New Area Added (sq. km)', main = mainTitle )

        # Add lines for mean behavior and 96% CI for mean
        lines(means[2:nCols], lwd = 3)
        lines(means[2:nCols] + 1.96 * ses[2:nCols], lty = 2)
        lines(means[2:nCols] - 1.96 * ses[2:nCols], lty = 2)

        # Plot exponential function
#         lines(interviewees, a * exp(-b*(interviewees)), col = 2)

        # Plot inverse function
        lines(interviewees, ((interviewees + c) / d)^e, col = 2)
        # Add threshold line
        abline(h = cutoff, lty = 2, col = 2)


        # Add legend
        # Exp. mod.
#         legend( 'topright', lty=c(1, 1, 2, 1, 1, 1, 2, 1),
#                   lwd=c(1, 3, 1, 1, 1, 1, 1, 1),
#               col = c('#999999', 1, 1, 'white', 2, 'white', 2,
#                       'white'),
#               legend = c('Individual Samplings', 'Mean', '1.96 se',
#                          '',
#                          paste("Fitted Inv. Function:\nNewArea = ",
#                                round(a, 2), 'e^(',
#                                round(b, 2), ')', sep=''),
#                          '',
#                          paste('Cutoff \n(Set at ', limitPercent,
#                                '% of median interview [Area = ',
#                                median(out.df[,2]),'])', sep=''),
#                          paste('\nTotal interviews needed: ',
#                                interviewsNeeded, sep='') ),
#               bg = 'white', cex = 0.7 )


        # Inverse mod.
        legend('topright', lty=c(1, 1, 2, 1, 1, 1, 2, 1),
               lwd=c(1, 3, 1, 1, 1, 1, 1, 1),
               col = c('#999999', 1, 1, 'white', 2, 'white', 2,
                       'white'),
               legend = c('Individual Samplings', 'Mean', '1.96 se', '',
                          paste("Fitted Inv. Function:\nNewArea = ",
                              '((interviewees + ', round(c, 2),
                              ') / ',
                              round(d, 2), ')^', round(e, 2), sep=''),
                         '',
                         paste('Cutoff \n(Set at ', limitPercent,
                              '% of median interview [Area = ',
                              median(out.df[,2]),'])', sep=''),
                              paste('\nTotal interviews needed: ',
                                   interviewsNeeded, sep='') ),
          bg = 'white', cex = 0.7 )

    # Add an axis on the right side indicating area as a % of the
    # total area
    axis(side = 4, at = seq(0, max(out.df[,2:nCols]),
                              length.out = 8),
         labels = c(
           as.character(round(
                seq(0, 100*max(out.df[,2:nCols]) / totalArea,
                   length.out = 7),
                1)),
           '% tot area') )

}

names(out.df)[1] = 'totalArea'
names(out.df)[2:nCols] = paste('a', 1:(nCols - 1), sep = '')

list(additionalData = out.df, meanAdded = means[2:nCols],
     fit = summary(inv.mod), totalInterviewsNeeded = i )
```

```
}


# Run the analysis:
set.seed(123)
completeness(locale = 'Random Sites', listOfMs = rMatrices,
             iterations = 100, limitPercent = 5, cStart = 1000,
             eStart = -30)




# Repeat with hotspot matrices
quartz()
completeness(locale = 'Hot Spots', listOfMs = hsMatrices,
             iterations = 100, limitPercent = 5, cStart = 1000,
             eStart = -30)



#======================================================================#
#                                                                      #
# Code to show intensity of use in each raster cell, along with upper #
# and lower confidence intervals                                       #
#                                                                      #
#======================================================================#

# The following code plots the collective land use of the entire
# sample, along with the empirical confidence interval.
# Helper function: determine the standard error of a percentage of
# successesgiven n samples
# @param p: probability of success
# @param n: number of samples
# @return: the standard error of the estimate
# NOTE: estimates will not be reliable if the observed p is 0
se = function(p, n) {
    return (sqrt(p * (1 - p) / n))
}


# The estimated percentage of the population using any single cell
# Determines the percentage of users at each cell in the grid, along
# with the empirical confidence interval at each location.
# @param ci: the desired confidence interval; e.g. 0.95 = 95%
#             confidence interval
# @param ms: list of matrices of individual uses in the sample
# @param graph: if TRUE, provides visual output
# @param col: the color palette and number of unique colors to use in
#             the output graph
# @return: matrices of the observed percentages of use for the entire
#             sample, along with the upper and lower confidence levels
ciMatrices = function(ci = 0.95, ms, graph = T,
                      col = terrain.colors(100)) {
   # Obtain Z values for quantiles for the given CI
   seQ = qnorm(p = c((1 - ci) / 2, (1 + ci) / 2))


   # sum matrices
   mSum = ms[[1]]
   for (i in 2:length(ms)) {
       mSum = mSum + ms[[i]]
   }


   # create a matrix of proportions for each cell (= p)
   mProp = mSum / length(ms)


   # and SEs for each cell
   mSE = se(mProp, length(ms))


   # lower and upper limits of confidence interval
   mLL = mProp + seQ[1] * mSE
   mLL[mLL < 0] = 0
   mUL = mProp + seQ[2] * mSE
   mUL[mUL > 1] = 1
   zl = range(mLL, mUL)


   if (graph == T) {
       n = dim(mProp)[1]


       par(mar = c(1, 1, 2, 1))
       layout(matrix(c(1, 1, 1, 1, 2, 3, 4, 4), ncol = 4),
              widths = c(1, 1, 1, 0.13), heights =  c(1, 1, 1, 1))
       image(mProp, xaxt = 'n', yaxt = 'n', main = 'Observed',
             col = col, zlim = zl)


       par(mar = c(0, 0, 2, 1))
       image(mLL, xaxt = 'n', yaxt = 'n', main = 'Lower Limit',
             col = col, zlim = zl)
       par(mar = c(1, 0, 2, 1))
```

```
        image(mUL, xaxt = 'n', yaxt = 'n', main = 'Upper Limit',
              col = col, zlim = zl)

        # Create a key for color values
        vals = seq(min(zl), max(zl), length = n)
        keyM = matrix(vals, nrow = 1)
        par(mar = c(1, 0, 2, 2))
        image(keyM, col = col, zlim = zl, xaxt = 'n', yaxt = 'n')
        ticsAt = seq(0, 1, length = 10)
        ticVals = round(seq(min(zl), max(zl), length = 10), 2)
        axis(side = 4, at = ticsAt, labels = ticVals)
    }

    return (list(observed = mProp, lower = mLL, upper = mUL))

}



# Examples
ciMatrices(0.9, rMatrices, T)
ciMatrices(0.95, hsMatrices, T)

#================== END OF SCRIPT =================#
```