DFO Atlantic Fisheries
Research Document 95/ 99

MPO Pêches de l'Atlantique
Document de recherche 95/ 99

Programs to Simulate Catch Rate Estimation in a Roving Creel Survey of Anglers

by

Colin J. Greene, John M. Hoenig, Nicholas J. Barrowman
Science Branch
Department of Fisheries and Oceans
P.O. Box 5667
St. John's NF A1C 5X1

Kenneth H. Pollock
Department of Statistics, Box 8203
North Carolina State University
Raleigh, NC 27695-8203, USA

ABSTRACT

This technical report describes two programs (i.e., functions) written in S-PLUS for simulating a roving creel survey used for estimating catch rate and total catch. The first program, **makeanglers()**, is used to create an angler population. The program output gives, for each anger, the location around the shoreline of a lake, the fishing start time and the trip length for one fishing day. Also given by this program is the true effort during this day (i.e., the sum of all the anglers' trip lengths). The second program, **gettotalvalues()**, is used to simulate each angler's catch for that day, and then sample the data on the basis of a clerk travelling through the fishery and interviewing these anglers (i.e., a roving creel survey). The outputs from this program are the true total catch for the fishing day, along with the estimates of the total catch from the creel survey using the ratio of means estimator and six mean of ratios estimators (i.e., with interview times of less than 0, 1, 5, 15, 30 or 60 minutes discarded). With some minor changes to the **gettotalvalues()** program, several different scenarios can be created, such as a scenario where the anglers become smarter after they catch a fish (i.e., their catch rate improves) or a scenario where different bag limits are imposed on the anglers. The code for these programs is given, as well as the modifications needed for three different scenarios, along with an example of the output from each of the two programs.

RÉSUMÉ

Le présent rapport technique décrit deux programmes (fonctions) S-PLUS de simulation d'une enquête par interrogation des pêcheurs, servant à estimer le taux de prises et les prises totales. Le premier programme, **«makeanglers()»**, sert à créer une population de pêcheurs sportifs. Son extrant fournit pour chaque pêcheur, l'emplacement de celui-ci sur les rives d'un lac, l'heure de début de la pêche et la durée de la sortie pour une journée de pêche. Il fournit également l'effort véritable durant cette journée (soit, la somme de la durée des sorties de tous les pêcheurs). Le second programme, **«gettotalvalues()»**, permet de simuler les prises de chaque pêcheur pour la journée considérée, puis d'échantillonner les données à la manière dont le ferait un agent qui interviewerait les pêcheurs (comme dans une enquête auprès des pêcheurs). Les extrants de ce programme donnent les prises totales réelles pour la journée de pêche considérée, ainsi que les estimations des prises totales des pêcheurs interrogés, fondées sur le quotient de l'estimateur des moyennes et six estimateurs de la moyenne des quotients (c.-à.-d. avec rejet des périodes d'entrevues inférieures à 0, 1, 5, 15, 30 ou 60 minutes). En apportant certains changements mineurs au programme **«gettotalvalues»**, on peut produire différents scénarios, par exemple un scénario dans lequel les pêcheurs deviennent plus efficaces après qu'ils capturent un poisson (c.-à.d. que leur taux de prises augmente) ou un scénario dans lequel différentes limites de prises sont imposées aux pêcheurs. On fournit le code de ces programmes, ainsi que les modifications à apporter pour obtenir trois scénarios différents, et des exemples d'extrant pour chacun des deux programmes.

Introduction

The two programs (`makeanglers()` and `gettotalvalues()`), written in **S-PLUS** (1993) version 3.2, are used to simulate a creel survey. The use of **S-PLUS** has simplified the program through certain predefined commands, as will be explained in the program summary. The simulation is conducted in two steps. First, the function `makeanglers()`, found in Appendix 1, is run in order to create an angler population. This program returns the:

- location of each angler, evenly spaced around the shoreline of a lake with a perimeter of one unit,
- start time of each angler, all of which are currently set to one hour into an eight hour day, and
- trip length of each angler, currently alternating between 3 and 6 hours.

All of the above settings can be changed relatively easily, but because of the outlay of `makeanglers()`, they are more easily changed within the program, as oppossed to being passed as arguments. Also returned by this function is the true effort for this fishing day, which is simply the sum of the trip lengths of all the anglers.

The second function `getttotalvalues()`, found in Appendix 2, performs the following tasks:

1. Generate a catch history for each angler.
2. Sum the catches over all the anglers to get the true total catch.
3. Simulate a clerk travelling through the fishery and interviewing the anglers, where the data to be recorded will only include information about the angler's catch before the individual's time of interview.
4. Estimate the cate rate and total catch from the data collected by the clerk using the ratio of means estimator.
5. Estimate the cate rate and total catch from the data collected by the clerk using six mean of ratios estimators (i.e., with interview times of less than 0, 1, 5, 15, 30 and 60 minutes removed).

Note that the only information returned by this function will be the true total catch and the seven estimates of the total catch. A more theoretical explanation of the equations to be used for the estimators can be found in Hoenig et al. (in review).

These programs were generally used to simulate 10000 replicate days with 50 anglers, which appeared to be sufficient. The `makeanglers()` function need only be run once to define the number of anglers, and then the `gettotalvalues()` function can be repeated as many times as needed, once for each simulationed day. This repeatition can be carried out in several ways, but the `gettotalvalues()` function, and it's input, will not have to change for each run.

Program Summary

Anyone familiar with **S-PLUS** may wish to simply read the commented programs `makeanglers()` and `gettotalvalues()` in Appendix 1 and Appendix 2 respectively, but for those that are unfamiliar with the language, here is a step-by-step explanation.

`makeanglers()`

The only argument needed for this function is the number of anglers, given by the variable `nanglers`. The values computed for the anglers will be stored in a list called `anglers`, which is an **S-PLUS** object that has the following components:

`loc` vector containing the anglers' locations,

`starttime` vector containing the anglers' start times, and

`triplength` vector containing the anglers' trip lengths.

These are written as `anglers$loc`, `anglers$starttime`, and `anglers$triplength`.

The location of each angler is calculated by evenly spacing the anglers around the shoreline of a lake with a perimeter of one unit. This is done by:

```
startpos <- 0.004
spacing  <- 1/nanglers
anglers$loc <- c(seq(from=startpos, by=spacing, length=nanglers))
```

Here `1/nanglers` gives the distance between the anglers. The location of each angler is then determined by the vector containing the sequence starting at postion 0.004, and being incremented by the given spacing, until each angler has a location assigned (i.e., the length of the vector is equal to the number of anglers).

The next step is to assign each angler a start time. This is done by repeating the value 1.0 until a vector of length equal to the number of anglers is produced.

```
anglers$starttime <- c(rep(1.0, nanglers))
```

The next step is to assign a trip length to each angler. This is done by repeating the values 3 and 6, `nanglers/2` times. If the number of anglers is odd, more care will have to be taken in how this vector is created.

```
anglers$triplength <- c(rep(c(3,6), nanglers/2))
```

Finally, the true effort is calculated by summing up all of the angler trip lengths for the day.

```
trueeffort <- sum(anglers$triplength)
```

These values are returned as permanent **S-PLUS** objects so that they can be accessed for use in the simulations of the `gettotalvalues()` function.

`gettotalvalues()`

The arguments for this function will be obtained from the permanent objects created by the `makeanglers()` function. That is, the 'list' `ang` will be given by the pemanent object `anglers`, the value `teffort` will be given by the permanent object `trueffort` and the value `nanglers` will be given by the length of `anglers$loc` (which would be equivalent to getting the length of `anglers$starttime` or `anglers$triplength`).

Generate Catch History

To begin with, generate a catch rate parameter for each angler. Assume that the catch of each angler follows a Poisson process, therefore it will be required to generate the Poisson success parameter λ for each angler. Let λ follow a gamma distribution: $\lambda \sim$ G(1,2). This is done with the following command:

```
lambda <- rgamma(nanglers,1)*2
```

This creates a vector of length equal to the number of anglers containing a lambda value that corresponds to each angler. The mean of a gamma distribution with parameters α and β is $\alpha \times \beta$, so the average of the values in the vector should be close to $1 \times 2 = 2$. Also, the variance of the values is $\alpha \times \beta^2 = 1 \times 4 = 4$.

For each individual angler a catch history must now be created, which is accomplished by chronologically storing, in a vector, the times that each fish is caught. Therefore, the number of fish caught would be given by the length of the vector. All of these vectors were stored in a 'list' `catch`, and could be accessed by a command such as `catch[[i]]`, refering to the catch vector for angler `i`. The following code is used:

```
for(i in 1:nanglers){
  time <- ang$starttime[i]
  time <- time + rexp(1, rate=lambda[i])
  while(time <= ang$starttime[i] + ang$triplength[i]){
      catch[[i]] <- c(catch[[i]],time)
      time <- time + rexp(1, rate=lambda[i])
  }
}
```

At the begining of each step through the `for` loop, the variable `time` becomes initialized to the the corresponding angler's start time. Then time is incremented by the random exponential number calculated by `rexp()` with one of its parameters being the individual angler's `lambda` value. This will give the time when the first fish is supposed to be caught. However, if the `time` value is greater than the angler's start time plus the trip length (i.e., the time of day when the angler leaves), then the time is not recorded and the loop moves on to the next angler (i.e., no fish were caught). If the `time` value is less than or equal to the angler's start time plus trip length then the `while` loop is entered and this first time is recorded in the catch vector. Now the `time` value is incremented by a second random exponential number with the same parameters as before. If this value is less than or equal to the angler's start time plus trip length this time will be recorded as the time when the second fish is caught, otherwise the second value is not recorded and the loop moves on to the next angler. This process is continued until the `time` value is greater than the angler's start time plus trip length, which gives a complete catch history for all the anglers.

True Total Catch

This step is simply summing up all the catches over all the anglers. **S-PLUS** (1993) supplies a function `sapply()` which applies a function to each element of a list. The

list is the first argument of `sapply()` and another function is the second argument. In this case, the second argument is the `length()` function which returns each angler's total catch. So the sum of the `sapply()` results is the total catch for the fishing day.

```
totalcatch <- sum(sapply(catch,length))
```

Simulate Interviews

To begin, a starting postion for the survey agent must be chosen. This is done with a random uniform number between 0 and 1.

```
startloc   <- runif(1)
```

The speed of the agent will be a constant of .125 units (circuits) per hour, such that the agent travels a full circuit in 8 hours (i.e., once around the lake).

```
agentspeed <- .125
```

A time switch is nesseccary for the simulation of the circular lake, with a perimeter of 1.0, where the positions 0.0 and 1.0 are equivalent on the lake representation. At this point a "jump" must be made, which is accomplished by the time-switch:

```
timeswitch <- (1-startloc)/agentspeed
```

The use of this time switch will be explained below.

The above information is used to calculate for each angler the length of the fishing trip at the time of the interview, the number of fish caught at the time of this interview and the catch rate for that individual angler. These three calculations are done in a `for` loop over all the anglers as follows.

To calculate the time of day when the survey agent reaches the location of angler `i`, the following code is used:

```
if( (startloc < ang$loc[i]) & (ang$loc[i] < 1) ){
   timeint <- (ang$loc[i] - startloc) * 8 }
else if( (0 < ang$loc[i]) & (ang$loc[i] < startloc) ){
   timeint <- ang$loc[i] * 8 + timeswitch
}
```

Note that position 0.0 is equivalent to position 1.0; see Figure 1 below.

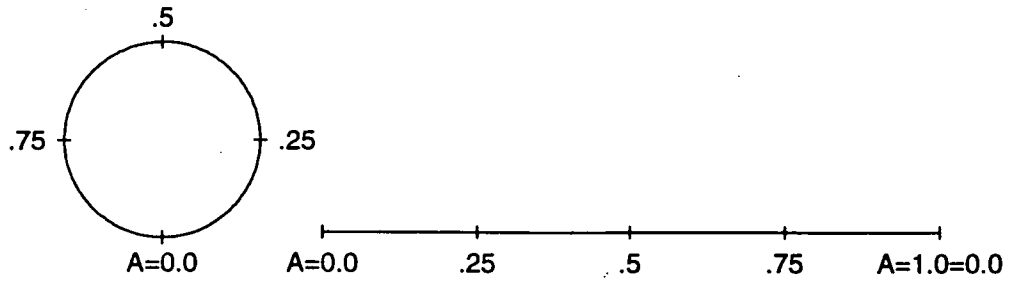

Figure 1. The circular and straight line representation of the perimeter lake.

The first part of the `if` statement says that if the angler's location is between the surveyor's location and the position of 1.0 on the lake, then use the equation,

$$\text{time of interview} = \text{distance surveyor has travelled so far} \times 8 \text{ hours}$$

recalling that it takes 8 hours for the surveyor to travel one full circuit. Otherwise (`else if`), the angler's location is between the position of 0.0 on the lake and the surveyor's location. This means the surveyor has travelled past the position of 1.0 ($\equiv$ 0.0) and therefore the equation to use is now

$$\begin{aligned}\text{time of interview} \;=\; &(\text{distance surveyor travels past position } 0.0 \times 8 \text{ hours}) + \\ &\text{time travelled before position } 1.0\end{aligned}$$

where time travelled before position 1.0 is given by the time switch.

The actual fishing effort for the interview (i.e., length of trip up to time of interview) will be stored as either zero or the time of interview minus the angler's start time. A zero will be stored if the interview time happens before the angler was set to arrive during the day, or if it occurs after the angler is set to leave the fishery. This is done with:

```
inteffort[i] <- 0
if( (ang$starttime[i] < timeint) &
          (timeint < ang$starttime[i] + ang$triplength[i]) ){
   inteffort[i] <- timeint - ang$starttime[i]
}
```

Therefore, `inteffort[i]` gives the interview effort for angler `i`.

Secondly, the catch at the time of the interview is calculated with the code:

```
intcatch[i] <- 0
if( length(catch[[i]]) > 0 ){
  for(k in 1:length(catch[[i]])){
     if((catch[[i]][k] < timeint) &
              (timeint < ang$starttime[i]+ang$triplength[i])){
        intcatch[i] <- intcatch[i] + 1
     }
  }
}
```

This sets the interview catch to zero unless the full catch for the angler is greater than zero, with at least one of the catch times being less than the interview time (and the interview time being less than the time when the angler leaves the fishery). The `for` loop runs through all the catch for the day, incrementing the `intcatch` by one only if the time the fish was caught was before the time of interview. Hence, `intcatch[i]` gives the number of fish caught by angler `i` at the time of the interview.

Each angler now has the number of fish caught and the fishing effort at the time of the interview recorded, such that the catch rate can be calculated for each of these anglers as follows:

```
if( inteffort[i] > 0) cr[i] <- intcatch[i] / inteffort[i]
else cr[i] <- NA
```

This calculates,

$$\text{catch rate} = \frac{\text{interview catch}}{\text{interview effort}} .$$

However, it assigns the `NA` (i.e., Not Available) value to the catch rate if no interview took place (i.e., the interview effort is equal to zero).

Estimate Total Catch by Ratio of Means

An explanation of the usefulness (or lack of it) of the estimators given in this and the next section can be found in Hoenig et al. (in review), while the calculations used are as follows:

```
crrom <- sum(intcatch) / sum(inteffort)
totalcrom <- crrom * teffort
```

This gives the catch rate estimated by the ratio of means estimator as,

$$\frac{\sum_{\text{all anglers}} \text{interview catch}}{\sum_{\text{all anglers}} \text{interview effort}}$$

To find the estimate of the total catch by the ratio of means estimator, simply multiply the catch rate estimate by the true effort, which was given by the `makeanglers()` function.

Estimate Total Catch by Mean of Ratios

The mean of ratios estimator is given as,

$$\frac{\sum_{\text{all anglers}} \text{individual's catch rate}}{\text{number of anglers}}$$

Here we are interested in the individual catch rates recorded with interview efforts greater than 0, 1, 5, 15, 30, and 60 minutes respectively. To accomplish this the following code was used:

```
cr0 <- mean(cr[ inteffort > 0 ])
totalc0 <- cr0 * teffort

cr1 <- mean(cr[ inteffort > (1/60) ])
totalc1 <- cr1 * teffort

cr5 <- mean(cr[ inteffort > (5/60) ])
totalc5 <- cr5 * teffort

cr15 <- mean(cr[ inteffort > (15/60) ])
totalc15 <- cr15 * teffort

cr30 <- mean(cr[ inteffort > (30/60) ])
totalc30 <- cr30 * teffort

cr60 <- mean(cr[ inteffort > 1 ])
totalc60 <- cr60 * teffort
```

For example, the catch rate for the mean of ratios estimator calculated with interview efforts greater than 5 minutes is given by,

$$\frac{\sum_{\text{all anglers with inteffort} > 5 \text{ minutes}} \text{individual's catch rate}}{\text{number of anglers with interview effort} > 5 \text{ minutes}}$$

For all of these, multipling the catch rate estimate by the true effort gives the corresponding total catch estimates.

The final step is to return the true total catch along with the seven estimates of total catch. Note that for the mean of ratios estimator, any number of time ranges can be excluded if more information is needed from the function.

Modification for Three Scenarios

Three additional scenarios that we ran included an effort dependant model, a learner model and a bag limit model. Along with these, the trip durations were altered in the `makeanglers()` function.

Effort Dependant Model

The effort dependant model involved doubling the `lambda` value for anglers with larger trip lengths (i.e., in the case of the trip lengths alternating between 3 and 6 hours, the anglers with an effort of 6 hours had their `lambda` doubled). This was achieved by:

```
lambda[anglers$triplength==6] <- 2*lambda[anglers$triplength==6]
```

This statement can be placed directly after the calculation of the original `lambda` values.

Learner Model

The learner model involved the doubling of each angler's `lambda` after the angler caught the first fish (i.e, after an angler catches a fish, he or she should have learned where the fish are and the catch rate should improve). This is a simple modification because of the way the program generates a catch history for each angler. As can be seen below, changing the `rate` in the calculation of the random exponential within the `while` loop solves the problem. This loop is only entered after the first fish is caught, and the doubled `lambda` is used for any calculations after the first fish.

```
for(i in 1:nanglers){
  time <- ang$starttime[i]
  time <- time + rexp(1, rate=lambda[i])
  while(time <= ang$starttime[i] + ang$triplength[i]){
```

```
        catch[[i]] <- c(catch[[i]],time)
        time <- time + rexp(1, rate=2*lambda[i])
    }
}
```

Bag Limit Model

For the bag limit model, several modifications must be made. First, a bag limit (in this case, a limit of three) must be set using:

```
baglimit   <- 3
```

and this can be placed before the first `for` loop. The true total catch should now be calculated within the first `for` loop using:

```
if( length(catch[[i]]) >= baglimit ){
    totalcatch <- totalcatch + baglimit  }
else if( length(catch[[i]]) > 0 ){
    totalcatch <- totalcatch + length(catch[[i]])  }
```

This method increments the true total catch by the value of the bag limit if the number of fish the angler would have caught without a bag limit is greater than or equal to the bag limit itself. However, if the number of fish the angler would have caught without a bag limit is less than the bag limit then the total catch is incremented by the number of fish caught.

The other modifications should be made within the second `for` loop. For each of the `i` anglers, the time that the last fish was caught needs to be calculated using:

```
numcaught <- min(length(catch[[i]]),baglimit)
if(numcaught==baglimit){
      timelastcaught <- catch[[i]][baglimit] }
else  timelastcaught <- ang$starttime[i] + ang$triplength[i]
```

Here, the variable `numcaught` will be the value of the original number of fish caught if this value is less than the bag limit, otherwise the value of `numcaught` will be given the value of the bag limit. Then the `if` statement records the time the last fish was caught in the variable `timelastcaught`. This variable is then used in attaining the interview effort and the interview catch, where the `if` statements used to obtain these values must be changed to include `(timeint < timelastcaught)` as can be seen below.

```
if( (ang$starttime[i] < timeint) &
            (timeint < ang$starttime[i] + ang$triplength[i]) &
                (timeint < timelastcaught) )

if((catch[[i]][k] < timeint) &
            (timeint < ang$starttime[i]+ang$triplength[i]) &
                (timeint < timelastcaught) )
```

These modifications achieve the effect of a bag limit where one should note that, if the bag limit is three, then the third fish should never be recorded by the surveyor, because the angler should leave as soon as his bag limit is reached.

Reference

Hoenig, J.M., K.H. Pollock, C.M. Jones, D.S. Robson, C.J. Greene (in review). Catch Rate Estimation for Roving and Access Point Surveys of Anglers. Available from J. Hoenig, Department of Fisheries and Oceans, P.O. Box 5667, St. John's, NF, A1C 5X1, Canada.

Statistical Sciences, *S-PLUS User's Manual*, Version 3.2, Seattle: StatSci, a division of MathSoft, Inc., 1993.

Appendix 1

An example of this program's output is given at the end of this appendix.

Code for `makeanglers()`

```
makeanglers <- function(nanglers=50){

# This function sets up an angler population by giving each angler a
# location, a starting time, and a fishing trip length. The number of
# anglers is given by nanglers, which is defaulted to 50.

  anglers <- list()  # The anglers location, start-time and triplength are
     # stored in a list, as three separate vectors, each of
     # length equal to the number of anglers (nanglers).

 ####################################>>>>>>>>>>>>>>>
 # Position the anglers around a lake with perimeter of length = 1 at
 # evenly spaced intervals.
 ####################################>>>>>>>>>>>>>>>

   startpos <- 0.004
   spacing  <- 1/nanglers
        # This will give an even spacing for the nanglers

   anglers$loc <- c(seq(from=startpos, by=spacing, length=nanglers))

 ####################################>>>>>>>>>>>>>>>
 # Give all the anglers a start time representing 1.0 hour
 # into the fishing day.
 ####################################>>>>>>>>>>>>>>>

   anglers$starttime <- c(rep(1.0, nanglers))

 ####################################>>>>>>>>>>>>>>>
 # Assign each angler a triplength, where the duration of the trip will
 # alternate between 3 and 6 hours as the anglers alternate.
 ####################################>>>>>>>>>>>>>>>
```

```
  anglers$triplength <- c(rep(c(3,6), nanglers/2))

  # NOTE: nanglers/2 = 25. If the value of nanglers is an odd number, the
  # number of repetions given by rep() would be one number short. For our
  # purposes only even numbers were used.

#####################################>>>>>>>>>>>>>>>
# Add up the amount of fishing to get the true effort.
#####################################>>>>>>>>>>>>>>>

  trueeffort <- sum(anglers$triplength)


  permanent <- function(name, value){
       assign(name, value, where = 1)
       cat("Creating permanent S-PLUS data object:", name, "\n")
       return(invisible(NULL)) }

  permanent("anglers",anglers)   # Make these values permanent for use
  permanent("trueeffort",trueeffort)   # in the simulations.
}
```

Output for `makeanglers()`

```
anglers$loc:
 [1] 0.004 0.024 0.044 0.064 0.084 0.104 0.124 0.144 0.164 0.184 0.204 0.224
[13] 0.244 0.264 0.284 0.304 0.324 0.344 0.364 0.384 0.404 0.424 0.444 0.464
[25] 0.484 0.504 0.524 0.544 0.564 0.584 0.604 0.624 0.644 0.664 0.684 0.704
[37] 0.724 0.744 0.764 0.784 0.804 0.824 0.844 0.864 0.884 0.904 0.924 0.944
[49] 0.964 0.984

anglers$starttime:
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1

anglers$triplength:
 [1] 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6 3 6
[39] 3 6 3 6 3 6 3 6 3 6 3 6

trueeffort
 [1] 225
```

Appendix 2

An example of this program's output is given at the end of this appendix.

Code for gettotalvalues()

```
gettotalvalues <- function(ang=anglers,teffort=trueeffort,
                                  nanglers=length(anglers$loc)){

# Simulate a creel survey and compute estimates of total catch

#########################################>>>>>>>>>>>>>>>
# Generate a catch history for each angler.
#########################################>>>>>>>>>>>>>>>
 # Start by obtaining a random catch rate parameter for each angler following
 # a Poisson process
 #########################################>>>>>>>>>>>>>>>

   lambda <- rgamma(nanglers,1)*2

 #########################################>>>>>>>>>>>>>>>
 # Note: departure time = start time + trip length
 # Simulate each angler's fishing day by first calculating the time at which an
 # initial fish is to be caught. If this time is less then the anglers departure
 # time, save the time value in a vector, and calculate the next time when a
 # fish is to be caught. If this second time is less then the anglers departure
 # time, save the time value in the vector, and calculate a third time when a
 # fish is to be caught. Continue on in this manner until the calculated time
 # of a fish being caught is greater then the angler's departure time. In this
 # way we have all the times that an angler catches a fish, and the function
 # length() gives a convenient way of obtaining the number of fish caught.
 #########################################>>>>>>>>>>>>>>>

   catch <- vector("list",nanglers)

   for(i in 1:nanglers){   # Do this for each angler.

    # Time of day when the angler arrives.
```

```
    time <- ang$starttime[i]

   # If the time calculated below falls within the trip duration, this time
   # will be recorded as the instant when the first fish was caught.

    time <- time + rexp(1, rate=lambda[i])

   # At the beginning of each loop through the while statement, check to see if
   # the current fish capture time falls within the duration of the trip.

    while(time <= ang$starttime[i] + ang$triplength[i]){

       ### NOTE: the number of fish caught is given by length(catch[[i]]).
        catch[[i]] <- c(catch[[i]],time)

       # Calculate the time when the next fish is to be caught.

        time <- time + rexp(1, rate=lambda[i])

    } # end of while loop
  } # end of i for loop

#######################################>>>>>>>>>>>>>>>
# Sum up all the catches over the anglers.
#######################################>>>>>>>>>>>>>>>

  totalcatch <- sum(sapply(catch,length))

#######################################>>>>>>>>>>>>>>>
# Obtain a starting postion for the survey agent.
#######################################>>>>>>>>>>>>>>>

  startloc   <- runif(1)  # Start postion of surveyor
  agentspeed <- .125   # Speed of the surveyor in circuits per hour. It will
                       # take the surveyor 8 hours to make a full circuit
                       # (once around the lake) at this speed.

  timeswitch <- (1-startloc)/agentspeed
```

```
      # This time switch is nesseccary for the simulation of the circular
      # lake, with a perimeter of 1.0, where the positions 0.0 and 1.0
      # are equivalent on the lake representation. At this point a
      # "jump" must be made, which is accomplished by our timeswitch.

inteffort  <- intcatch <- cr <- vector("numeric",length=length(catch))

# For each ofto sample the anglers

for(i in 1:nanglers){

  # Calculate the time of each interview

  if( (startloc < ang$loc[i]) & (ang$loc[i] < 1) ){
     timeint <- (ang$loc[i] - startloc) * 8 }
  else if( (0 < ang$loc[i]) & (ang$loc[i] < startloc) ){
     timeint <- ang$loc[i] * 8 + timeswitch
  }

  # Calculate the fishing effort at the time of each interview

  inteffort[i] <- 0
  if( (ang$starttime[i] < timeint) &
             (timeint < ang$starttime[i] + ang$triplength[i]) ){
     inteffort[i] <- timeint - ang$starttime[i]
  } # else if no interview took place leave inteffort at default 0

  # Determine the number caught by the time of the interview

  intcatch[i] <- 0
  if( length(catch[[i]]) > 0 ){
    for(k in 1:length(catch[[i]])){
       if((catch[[i]][k] < timeint) &
                (timeint < ang$starttime[i]+ang$triplength[i])){
          intcatch[i] <- intcatch[i] + 1
       }
    }
  } # else if no fish were caught leave intcatch at default 0
```

```
    # Calculate catch rate

    if( inteffort[i] > 0) cr[i] <- intcatch[i] / inteffort[i]
    else cr[i] <- NA

  } # end of i for loop

#####################################>>>>>>>>>>>>>>>
# Catch rate and total catch estimated using ratio of means (or totals)
#####################################>>>>>>>>>>>>>>>

  crrom <- sum(intcatch) / sum(inteffort)
  totalcrom <- crrom * teffort


#####################################>>>>>>>>>>>>>>>
# Catch rate and total catch estimated using mean of ratios
#####################################>>>>>>>>>>>>>>>

  cr0 <- mean(cr[ inteffort > 0 ])
  totalc0 <- cr0 * teffort

  cr1 <- mean(cr[ inteffort > (1/60) ])
  totalc1 <- cr1 * teffort

  cr5 <- mean(cr[ inteffort > (5/60) ])
  totalc5 <- cr5 * teffort

  cr15 <- mean(cr[ inteffort > (15/60) ])
  totalc15 <- cr15 * teffort

  cr30 <- mean(cr[ inteffort > (30/60) ])
  totalc30 <- cr30 * teffort

  cr60 <- mean(cr[ inteffort > 1 ])
  totalc60 <- cr60 * teffort

 return(c(totalcatch,totalcrom,totalc0,totalc1,
                totalc5,totalc15,totalc30,totalc60))  }
```

Output for `gettotalvalues()`

The vectors returned for ten executions of this function are:

```
[1] 463.0000 507.3501 489.8762 489.8762 489.8762 445.3025 480.9267 505.6241

[1] 509.0000 382.7631 549.4431 549.4431 446.8467 463.3966 481.2195 498.3395

[1] 427.0000 444.4938 408.0655 408.0655 408.0655 423.1791 457.0334 462.3741

[1] 502.0000 537.2506 587.7818 587.7818 587.7818 562.9954 585.5152 523.0208

[1] 603.0000 683.4448 561.3052 561.3052 561.3052 604.4826 628.6619 565.1250

[1] 500.0000 518.8297 530.8509 530.8509 530.8509 496.8322 450.0611 460.7152

[1] 431.0000 344.4543 321.5798 321.5798 321.5798 333.4902 360.1694 398.2353

[1] 414.0000 491.0840 434.0926 434.0926 434.0926 450.1701 486.1837 439.3496

[1] 510.0000 557.2905 459.9650 459.9650 477.0008 453.8516 472.0057 500.2122

[1] 580.0000 665.4135 569.2529 569.2529 589.5834 611.4198 634.9359 645.4074
```

These vectors contain the value given for the true total catch, the value given for the estimate of the total catch by ratio of means, and the six estimates of total catch by mean of ratios with 0, 1, 5, 15, 30 and 60 minutes discarded respectively for the last six vector values.