Twelth R Practice exercise using Monte Carlo simulation to investigate a statistical quantity

Alan E. Berger March 6, 2022 minor edits March 10 in text and comments, none are in active R code

available at

https://github.com/AlanBerger/Practice-programming-exercises-for-R

Introduction and Background

This article will give an example of using a suitable random number generator and associated R code to "explore" and approximate a statistical quantity. Using an appropriate random number generator to simulate statistical data and thereby obtain a good approximation for a statistical quantity is a versatile technique. This is often referred to as doing a **Monte Carlo** (**MC**) **simulation** or doing **Monte Carlo sampling**.

The example to be used was recently presented as a puzzle in the article **Probability puzzle: How many balls in the bag?** by Michael Fletcher on page 8 of the February 2022 issue of **Significance**.

The statistical question can be phrased as: Given N marbles numbered from 1 through N in a container, and having randomly taken out k of the marbles; estimate what N is from the numbers on those k marbles. A more formal statement is: Given k independent random samples $s_1, \ldots, s_k$ taken **without replacement** from the integers 1,2, ... , N (i,e., the same number can be selected at most once); estimate N from the k values $s_1, \ldots, s_k$.

This question is commonly referred to as the "German Tank Problem" (**GTP**), which involved estimating the number of German tanks being manufactured during World War II given the serial numbers from captured/destroyed tanks. This is discussed in many articles including **Great moments in statistics** by Carlos Gómez Grajalez, Eileen Magnello, Robert Woods, Julian Champkin, **Significance**, December 2013, see page 28, freely available at https://doi.org/10.1111/j.1740-9713.2013.00706.x The paper **Lessons from the German Tank Problem**, George Clark, Alex Gonye and Steven J. Miller, **The Mathematical Intelligencer** V43, 2021, pages 19-28, is freely available in preprint form at https://arxiv.org/abs/2101.08162v2 (version 2, last revised 21 Jan 2021). This paper gives a nice statement and discussion of the problem, and a thorough derivation of the well known (frequentist) estimate for N. Letting L denote the maximum of the k samples $s_1, \ldots, s_k$, this estimate is

$$N \approx L\left(\frac{k+1}{k}\right) - 1 \qquad \text{equation (1)}$$

Here I have used L for the maximum (instead of m as done in the references) since I will let m denote the mean in the R code below.

This paper also gives a nice discussion of using Monte Carlo simulation and regression modeling to explore how to best estimate N from the k samples s_1, ... , s_k.

The online article **How many tanks? MC testing the GTP** by Matt Asher, https://statisticsblog.com/2010/05/25/how-many-tanks-gtp-gets-put-to-the-test/ points out the usefulness of Monte Carlo sampling for testing a statistical estimate in the setting of the German Tank Problem.

The Wikipedia article **German tank problem** https://en.wikipedia.org/wiki/German_tank_problem also includes Bayesian analysis for estimating N.

The specific case given in the **Probability puzzle: How many balls in the bag?** article by Michael Fletcher is $k = 4$, with the 4 sampled values being 24, 87, 14 and 35. Equation (1) "tells us" to estimate N by 87 * 5 / 4 - 1 which is 107.75, which, rounding to the nearest integer, gives 108.

The conceptual example of Monte Carlo Sampling that will be presented here

Regression modeling, when judiciously implemented as in the George Clark, Alex Gonye and Steven J. Miller paper, can do quite well in obtaining a good approximation for the result in equation (1). Here, to concentrate on the conceptual idea of Monte Carlo sampling, we will outline and implement an R function to see how well using a "natural function" of the mean and of the median of the k samples does in comparison to equation (1) for estimating N from the values s_1, ... , s_k, using the length of the 0.95 confidence interval (**CI**) about N as the "goodness" criterion (this CI is defined as the smallest integer interval

$$[j_1, j_2]$$

that contains N and 95% of the predictions that were done). The "natural function" is 2 * mean - 1 and 2 * median - 1 (rounded to the nearest integer), since if k were equal N (all the numbers were sampled), then 2 * mean - 1 and 2 * median - 1 are both equal N. One could immediately do better, for example by taking maximum(2 * mean - 1, the largest of the sampled values), and similarly for the median, since clearly N must be at least as large as the maximum of the sampled values, but here I will keep things simple to concentrate on the conceptual idea of Monte Carlo sampling.

R code to implement Monte Carlo Sampling for testing how well using 2 * mean - 1 and 2 * median - 1 do in estimating N

Note if you want to copy code lines for use in R, it is best to copy from the .Rmd file, since sometimes text in the pdf file can contain formatting characters that R does not accept.

Note before doing a set of calculations using a random number generator, one should always first set the random number seed to some arbitrary recorded integer value (use the **set.seed** function), so that one can exactly reproduce the results.

The idea of Monte Carlo sampling in the context of this example (the German Tank Problem with a chosen value of N and of k), is to do steps 1. and 2. below a large number of times (denoted in the code by numMCsamples) (variables used in the R code will now be displayed in regular type, not mathematical font):

1. Use the **sample** function to generate k independent random samples s_1, ... , s_k (without replacement) from the integers 1 through N; place the sample values in the vector V1. Set m = mean(V1) and med = median(V1). It will be convenient in some of the discussion of the results below to let V equal sort(V1), i.e., V is V1 with the entries sorted in ascending order.

2. Define the predicted value from the mean to be Nm = round(2*m - 1), and define the predicted value from the median to be Nmed = 2*med - 1 (the round function rounds a number to the nearest integer). Nm and Nmed will always be integers between 1 and 2*N (including the endpoints 1 and 2*N).

3. (done while doing steps 1. and 2.) For each integer j between 1 and 2*N, count (in vectors NmVector and NmedVector of size (length) 2*N) the number of the times (out of the numMCsamples times one does steps 1. and 2. above) that Nm and Nmed, respectively, is equal to j. Also "collect" (save) all the numMCsamples values of Nm and of Nmed (the predictions for N) that were generated in steps 1. and 2., in the vectors Nvalues.mean and Nvalues.median, respectively (having size (length) numMCsamples).

4. Use the counts in NmVector and NmedVector, and the values in Nvalues.mean and Nvalues.median, to gauge how well using the mean and median in this way predicts the value of N. One can do scatterplots using NmVector/numMCsamples and using NmedVector/numMCsamples plotted vs. the integers 1:(2*N), and compute confidence intervals using Nvalues.mean and Nvalues.median (this is done in R code below).

Here is an R function (with extensive explanatory comments on the Monte Carlo simulation and the R coding) implementing the steps 1. - 4. above. There will also be a function provided below called obtain.conf.interval.from.vector.vec.about.value.y.4March2022 that will calculate confidence intervals. If one wants to try programming part of this oneself, one could read and use the part of the code below up to the line filled with ######### and then think about how one would do the Monte Carlo steps, and form the items to be returned by this function (in a list).

```
mean.or.median.of.k.samples.for.finding.N <-
                        function(N, k, Lrepl, numMCsamples, CIfraction) {
# March 4, 2022   Alan E. Berger
#
# Given a positive integer N and k independent random samples from the
# integers 1 through N; without replacement if Lrepl = FALSE,
# with replacement if Lrepl = TRUE,
# will try to estimate N from the k random samples using their mean and
# using their median as detailed below.

# Will do numMCsamples Monte Carlo simulations to see how well this does.
# For each the mean and the median: return a data frame
# containing, for each integer j in 1:2N, the fraction of the simulations
# for which the estimated value of N was equal to j (sorted high to low).
# Also return the CIfraction confidence interval (CI) about N,
# e.g., CIfraction equal 0.95,
# meaning the smallest range of integers [j1, j2] such that N is in
# this interval AND at least
#        ceiling(numMCsamples * CIfraction)
# of the estimated values for N from the
# numMCsamples simulation runs are between j1 and j2 (including
# the endpoints j1 and j2).
# Also return CIlen equal j2 - j1
# This confidence interval might not be symmetric about N. Its length
# will be used as a measure of the quality of the method for estimating N.
# Also return the number of times the estimate is correct (exactly N).
#
#
# Note before doing one or more simulation runs, should use set.seed
# to set the random number generator seed to some recorded integer so
# will be able to reproduce the result.


# numMCsamples times: use the sample function to select k (approximately)
# independent random samples from 1:N, with or without replacement
# governed by Lrepl,
# then take the mean (m) and median (med) of the k samples.

# If k were set to N, then with a bit of algebra one can check that:
# one would recover N exactly by doing:
# for the mean:  Nm = 2*m - 1   # N estimated from m
# for the median: Nmed = 2*med - 1   # N estimated from med
#                                      Nmed will always be an integer
# Will use these formulas: Nm = 2*m - 1     Nmed = 2*med - 1
# to estimate N
#
# ******* note if k were odd, then med would always be an integer
# *******      so Nmed would always be an odd integer
#
# For actual mean (m) values from k samples,
# we will use the round function to get the nearest integer to Nm
# these rounded estimated values for N will be denoted by:   rNm

# Note the mean and median of the k samples are
# at least 1 and at most N,
# so rNm and Nmed are between 1 and 2*N

# **** note round(a positive integer K + 0.5) will be the EVEN integer
# that is closest to K
# For example, round(2.5) is 2 and round(3.5) is 4
# This will have a noticeable effect if, e.g., k is 4


# This R function will return two data frames,
# one from using Nm and one from using Nmed to estimate N.
# Column 2 of each data frame will contain the fraction of
# the numMCsamples runs for which round(Nm), Nmed, respectively, was
# equal to the integer j, for j = 1, 2, ... , 2*N
# Column 1 will be the integers 1 through 2*N
# Each data frame will be sorted from high to low based on column 2.


# simple checks (a "library" level code would check the validity of
# all the function arguments)
if(k < 1) stop("k is < 1")
if(N < k) stop("N is < k")


# Initialize vectors NmVector and NmedVector of length 2*N containing
# all 0's, and then for each of the numMCsamples simulation runs,
# add 1 to index rNm of NmVector, and add 1 to index Nmed of NmedVector
# Also initialize other required vectors:

Nvector <- 1:N  # will do k samples from Nvector,
# with replacement governed by Lrepl
#

jVector <- 1:(2*N)
NmVector <- vector(mode = "integer", length = 2*N)
NmedVector <- vector(mode = "integer", length = 2*N)

# initialize vectors to contain all the numMCsamples estimated
# values for N (will use them to obtain confidence intervals)

Nvalues.mean <- numeric(numMCsamples)
Nvalues.median <- numeric(numMCsamples)


# Do the numMCsamples Monte Carlo sampling runs using basic R commands
# and "collect" the results in NmVector, NmedVector,
#                                Nvalues.mean, Nvalues.median
# Then form the two data frames Nm.df and Nmed.df:
# for the mean; from jVector and NmVector / numMCsamples
# for the median; from jVector and NmedVector / numMCsamples
# Then sort each data frame based on descending order in column 2

# Also use the
#        obtain.conf.interval.from.vector.vec.about.value.y.4March2022
# function to get the CIfraction confidence interval (CI) [CIleft, CIright]
# about N from Nvalues.mean and from  Nvalues.median
# This function returns a vector containing the confidence interval,
#  its length, the confidence level that was requested, here CIfraction,
#  the number of times the estimate for N is exactly N, and the value
#  the confidence interval is for (here N).

# return the list containing Nm.df and Nmed.df and the confidence interval
# vector from Nvalues.mean and from  Nvalues.median, and also
# Nvalues.mean and Nvalues.median

##########################################################################

# Do the numMCsamples Monte Carlo sampling runs using basic R commands

for (i in 1:numMCsamples) {
   ksamples <- sample(Nvector, size = k, replace = Lrepl, prob = NULL)
   rNm <- round(2 * mean(ksamples) - 1)
   Nmed <- 2 * median(ksamples) - 1
   NmVector[rNm] <- NmVector[rNm] + 1L
   NmedVector[Nmed] <- NmedVector[Nmed] + 1L

   Nvalues.mean[i] <- rNm
   Nvalues.median[i] <- Nmed
}

Nm.df <- data.frame(j = jVector, Nm.frac = NmVector / numMCsamples)
Nmed.df <-
       data.frame(j = jVector, Nmed.frac = NmedVector / numMCsamples)

# sort the data frames
```

```
sort.indices <- sort.list(NmVector, decreasing = TRUE)
Nm.df <- Nm.df[sort.indices, ]

sort.indices <- sort.list(NmedVector, decreasing = TRUE)
Nmed.df <- Nmed.df[sort.indices, ]

# get the confidence intervals
vec <- Nvalues.mean
Nmean.CIvector <-
   obtain.conf.interval.from.vector.vec.about.value.y.4March2022(vec,
                    y = N, conf.level = CIfraction)

vec <- Nvalues.median
Nmedian.CIvector <-
   obtain.conf.interval.from.vector.vec.about.value.y.4March2022(vec,
                    y = N, conf.level = CIfraction)


list.to.return <- list(Nmean.df = Nm.df, Nmedian.df = Nmed.df,
  Nmean.CIvector = Nmean.CIvector, Nmedian.CIvector = Nmedian.CIvector,
  Nvalues.mean = Nvalues.mean, Nvalues.median = Nvalues.median)

return(list.to.return)

# example run:
# set.seed(123)
# twodf <- mean.or.median.of.k.samples.for.finding.N(N = 100,
#      k = 4, Lrepl = FALSE, numMCsamples = 100000, CIfraction = 0.95)
# plot(twodf[[1]])
# plot(twodf[[2]])

}
```

The function that calculates confidence intervals

Here is the function (used within the code above) that will calculate confidence intervals about y = N from a vector vec of predicted values for N. Some of the ways of handling vector indices and calculating a “best value” (here a “best interval”) in this code come up often in programming.

```
obtain.conf.interval.from.vector.vec.about.value.y.4March2022 <-
                     function(vec, y, conf.level) {
# March 4, 2022   Alan E. Berger
```

```
#
# Given a vector vec of real numbers, and a real number y, and
# a positve confidence level, conf.level, less than 1 (e.g., 0.95),
# return the smallest confidence interval (CI)
#     [CIleft, CIright]
# that contains y AND
# contains at least NCI = conf.level * length(vec) of the entries of vec.
# Actually, use ceiling(NCI)
# which is the smallest integer that is at least as large as NCI.
# Also return the number of entries of vec that equal y.

# Return the vector containing CIleft, CIright and CIlen = CIright - CIleft
# Note this confidence interval is NOT necessarily symmetric about y.

# Method: Let v equal sort(vec) and use windows of
# subsets of v of length NCI, denoted by
#                                         v[j], v[j+1], ... , v[j+NCI-1]
# and find the one that contains y in
# its range AND for which v[j+NCI-1] - v[j] is smallest

# Initialize values, then examine all such intervals to find
# the smallest length confidence interval

veclen <- length(vec)
v <- sort(vec)
NCI <- ceiling(conf.level * veclen)
# check y is in the range of vec
if((y < v[1]) || (y > v[veclen])) stop("y NOT in range of vec")

CIleft <- v[1]
CIright <- v[veclen]
CIlen <- CIright - CIleft

# examine all sub-intervals of length NCI

for(j in 1:(veclen - NCI + 1)) {
   jrt <- NCI - 1 + j
   a <- v[j]
   b <- v[jrt]
   if((y < a) || (y > b)) next   # skip if y not in this interval
   lenab <- b - a
   if(lenab < CIlen) {
      CIleft <- a
      CIright <- b
      CIlen <- lenab
   }
}

# get number of entries in vec that equal y
exact.count <- length(which(vec == y)) # be aware of possible
#  finite precision issues,
# for example, (6./10.) * 7. - (7./10.) * 6. == 0  is FALSE
# When vec and y have integer values this will not be a problem.

CIvector <- c(CIleft = CIleft, CIright = CIright, CIlen = CIlen,
              conf.level = conf.level,
              exact.count = exact.count, y = y)
return(CIvector)
}
```

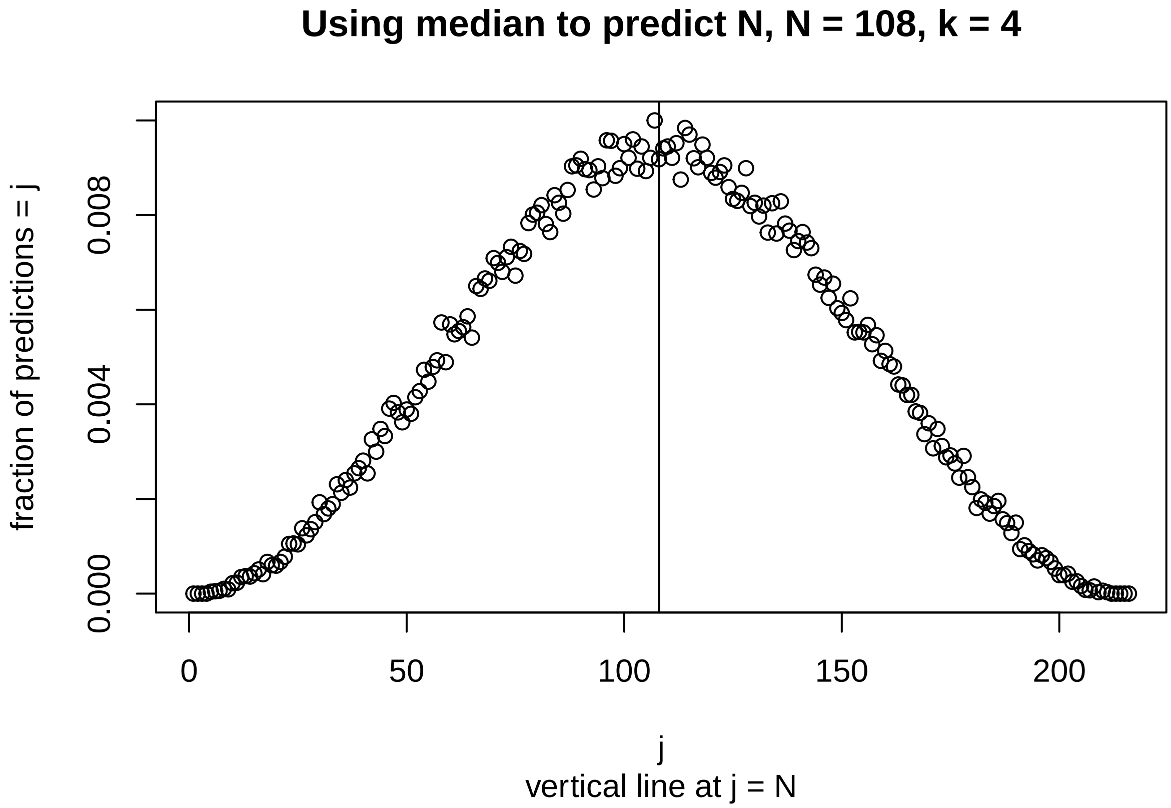
Use the two functions above to investigate how well using the mean and using the median do in predicting N.

We will do this for N = 108 and k = 4 correponding to the **Probability puzzle: How many balls in the bag?** article by Michael Fletcher. For a "real" research study one would do Monte Carlo sampling with multiple values of the relevant parameters, here N, k, numMCsamples and CIfraction.

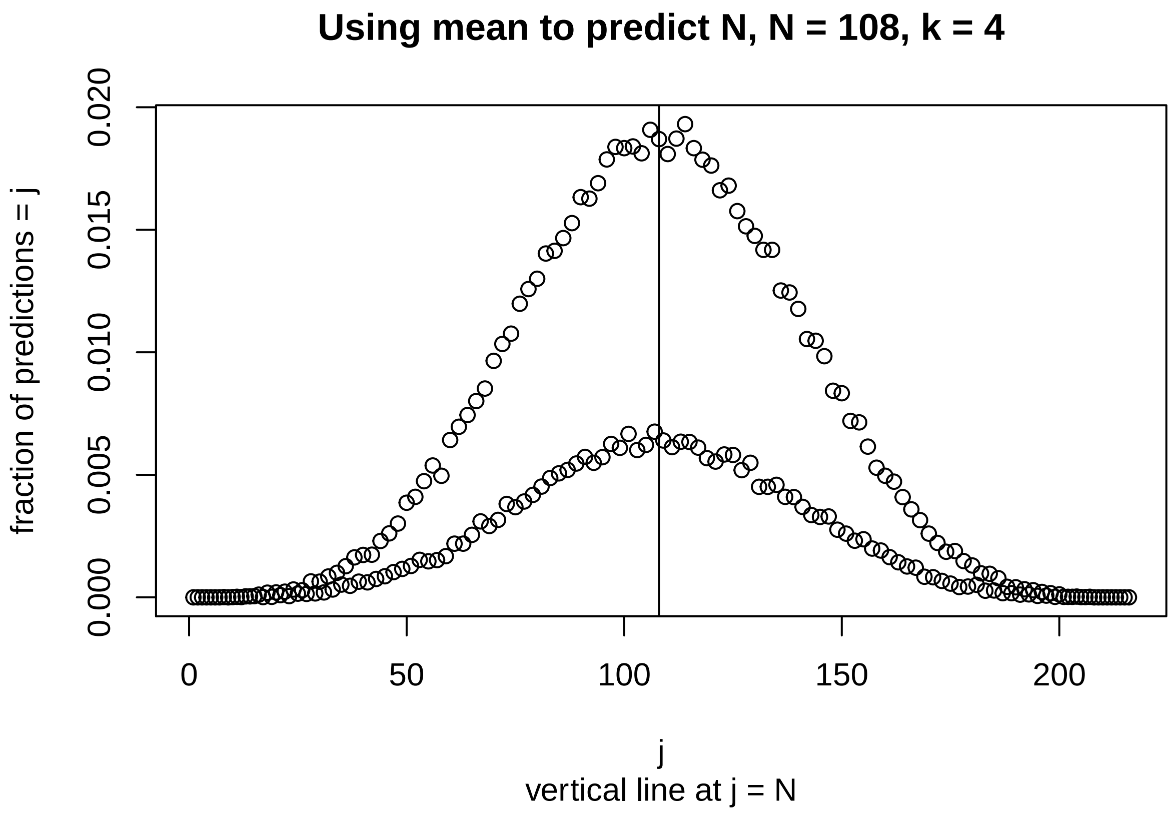
```
# have in effect sourced the two functions above

# set plot size
# from
# https://stackoverflow.com/questions/1279003/specify-width-and-height-of-plot
# see answer by Cybernetic May 20, 2018
# sets the options for the plot size in inches
set_plot_dimensions <- function(width_choice, height_choice) {
       options(repr.plot.width=width_choice, repr.plot.height=height_choice)
       }
set_plot_dimensions(5, 4)

set.seed(123)  # so can reproduce the results

resultsList <- mean.or.median.of.k.samples.for.finding.N(N = 108L,
        k = 4L, Lrepl = FALSE, numMCsamples = 100000L, CIfraction = 0.95)

str(resultsList)  # see what is in resultsList and what the object names are
```

```
## List of 6
##  $ Nmean.df        :'data.frame':    216 obs. of  2 variables:
##   ..$ j      : int [1:216] 114 106 112 108 102 98 100 116 104 110 ...
##   ..$ Nm.frac: num [1:216] 0.0193 0.0191 0.0187 0.0187 0.0184 ...
##  $ Nmedian.df      :'data.frame':    216 obs. of  2 variables:
##   ..$ j        : int [1:216] 107 114 115 102 96 97 112 100 118 104 ...
##   ..$ Nmed.frac: num [1:216] 0.01 0.00984 0.0097 0.0096 0.00958 0.00957 0.00952 0.009
##  $ Nmean.CIvector  : Named num [1:6] 46 164 118 0.95 1870 108
##   ..- attr(*, "names")= chr [1:6] "CIleft" "CIright" "CIlen" "conf.level" ...
##  $ Nmedian.CIvector: Named num [1:6] 32 178 146 0.95 918 108
##   ..- attr(*, "names")= chr [1:6] "CIleft" "CIright" "CIlen" "conf.level" ...
##  $ Nvalues.mean    : num [1:100000] 86 100 114 162 146 72 102 150 79 72 ...
##  $ Nvalues.median  : num [1:100000] 81 92 114 159 184 67 113 156 46 63 ...
```

```r
# do a scatterplot of the fraction of the numMCsamples predictions for N
# from using 2 * median - 1 that equal each integer j between 1 and 2*N

plot(resultsList$Nmedian.df, ylab = "fraction of predictions = j",
     main = "Using median to predict N, N = 108, k = 4",
     sub = "vertical line at j = N", ylim = c(0., 0.01))
abline(v = 108)  # plot vertical line at j = true value of N
```



```
# now do the corresponding scatter plot for the fraction of the numMCsamples
# predictions for N from using 2 * mean - 1 that equal each
# integer j between 1 and 2*N

plot(resultsList$Nmean.df, ylab = "fraction of predictions = j",
     main = "Using mean to predict N, N = 108, k = 4",
     sub = "vertical line at j = N")
abline(v = 108)
```



The importance of knowing in detail what exactly is going on when one does a calculation

Wow!! Using the mean certainly seems to give results that are more "closely packed" around the true value of N, but what gives with there being TWO bell shaped "curves"? Recall the round function, when applied to a value that is (an integer K + 1/2) will return the **EVEN** integer closest to K, so round(2.5) equals 2 while round(3.5) equals 4. Now here the mean m is equal the mean of k = 4 integers (the sum of the 4 sampled integers divided by 4) so there will be a fair number of cases where m equals an integer plus 1/4 or an integer + 3/4, so in those two cases 2*m will be an integer + 1/2 and so rNm = round(2*m - 1) will be "enriched" in even integers, explaining the "curious" appearance of the plot.

The confidence intervals from using the mean and using the median.

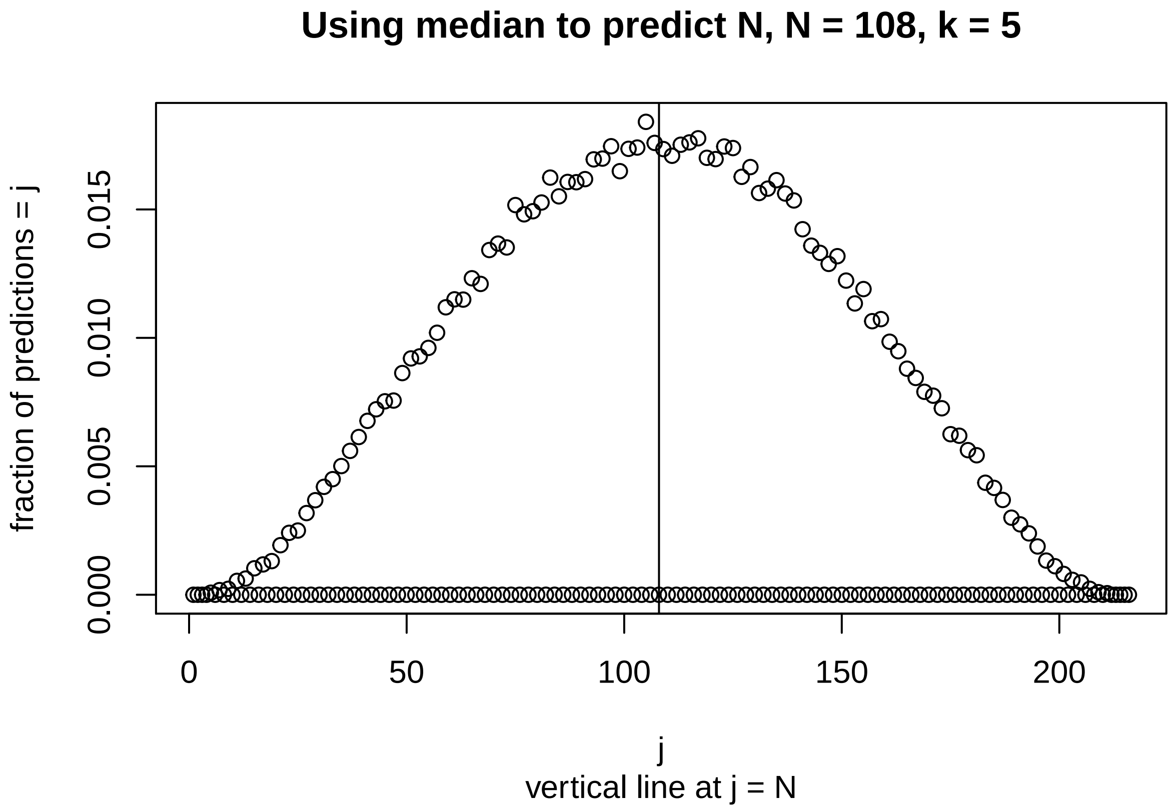
```
# print out the confidence interval from using the median to predict N
resultsList$Nmedian.CIvector
```

```
##      CIleft     CIright       CIlen  conf.level exact.count           y
##       32.00      178.00      146.00        0.95      918.00      108.00
```

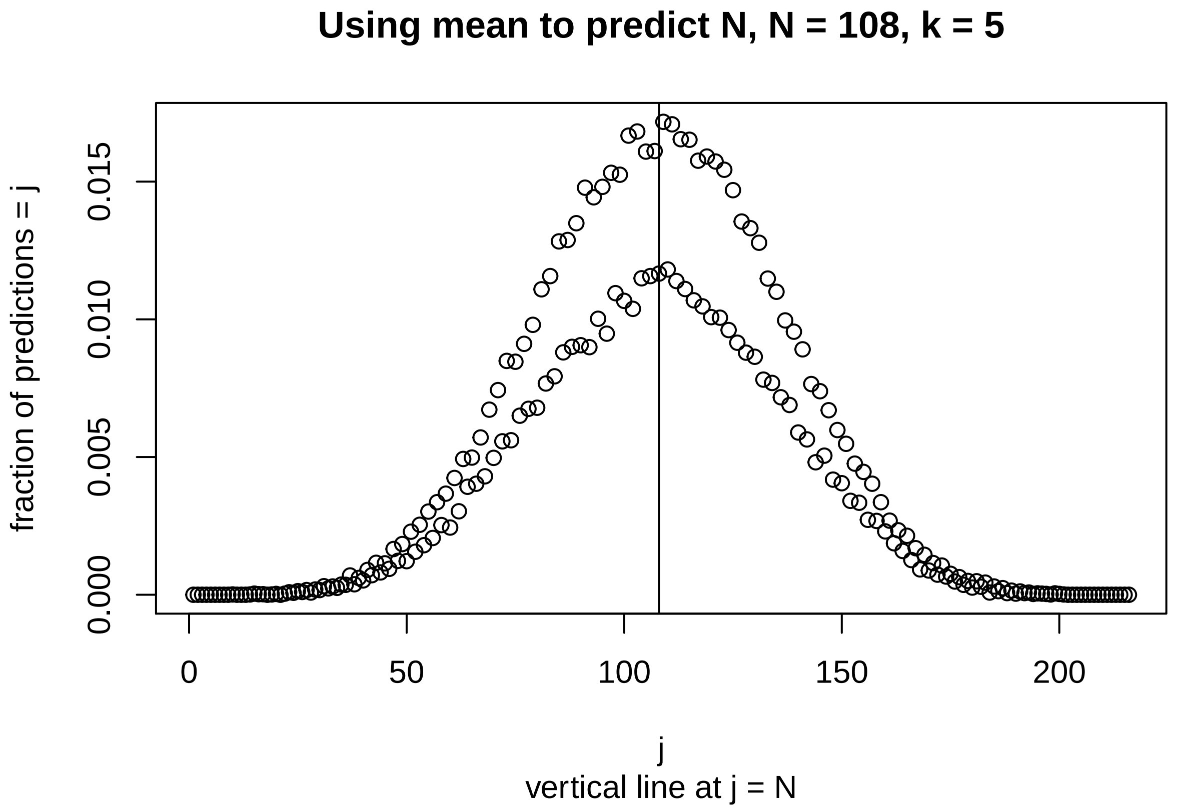
```
# and now print out the CI from using the mean to predict N
resultsList$Nmean.CIvector
```

```
##      CIleft     CIright       CIlen  conf.level exact.count           y
##       46.00      164.00      118.00        0.95     1870.00      108.00
```

```
# the CI for the mean is narrower, and more predicted values
# are exactly N from using the mean - though that is no doubt "helped" by
# the "enrichment" in even integer predictions since N is 108 (even)

# in the theme of knowing details of a calculation, let's look at plots
# from having k = 5

resultsList <- mean.or.median.of.k.samples.for.finding.N(N = 108L,
        k = 5L, Lrepl = FALSE, numMCsamples = 100000L, CIfraction = 0.95)

# plot results from using the median

plot(resultsList$Nmedian.df, ylab = "fraction of predictions = j",
     main = "Using median to predict N, N = 108, k = 5",
     sub = "vertical line at j = N")
abline(v = 108)  # plot vertical line at j = true value of N
```



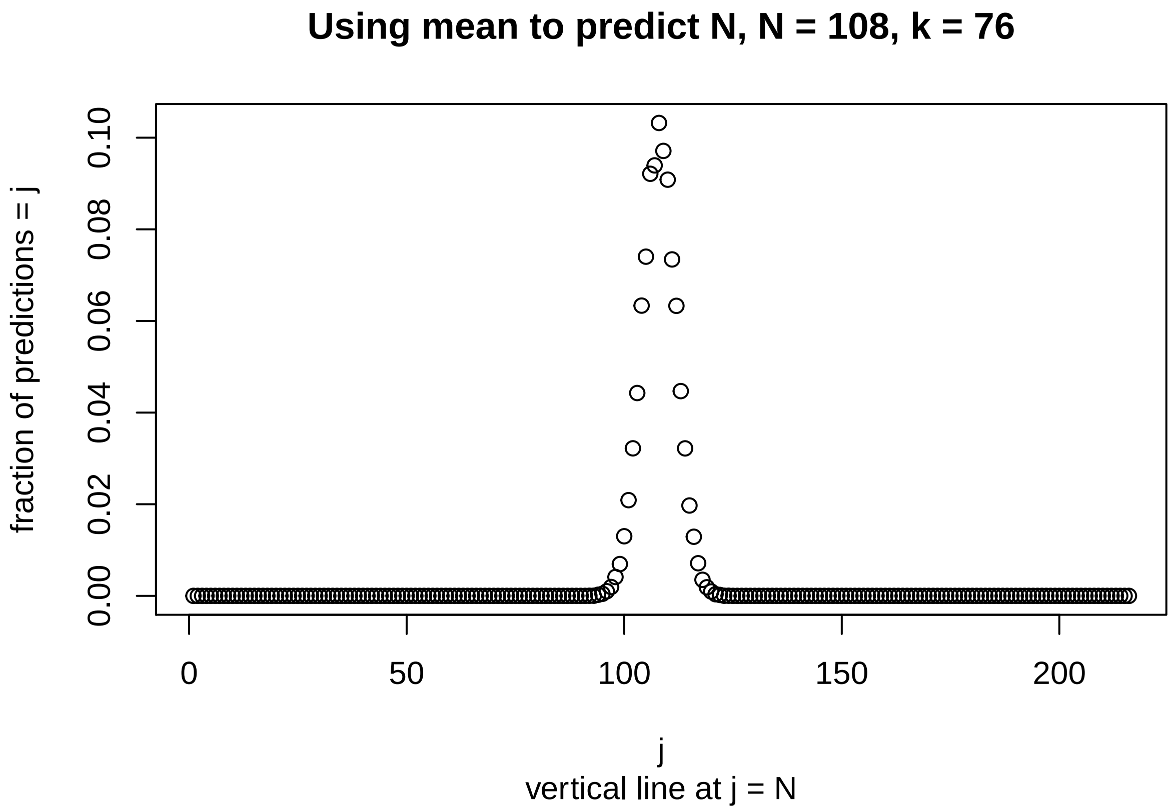
```r
# Now what !!!!!
# Well recall that when k is odd, the median of the k samples will be the
# middle value (of the vector V of sorted sample values),
# and the estimate for N from the median is 2*median - 1
# which is always an ODD integer

# next plot the results from using the mean

plot(resultsList$Nmean.df, ylab = "fraction of predictions = j",
     main = "Using mean to predict N, N = 108, k = 5",
     sub = "vertical line at j = N")
abline(v = 108)  # plot vertical line at j = true value of N
```



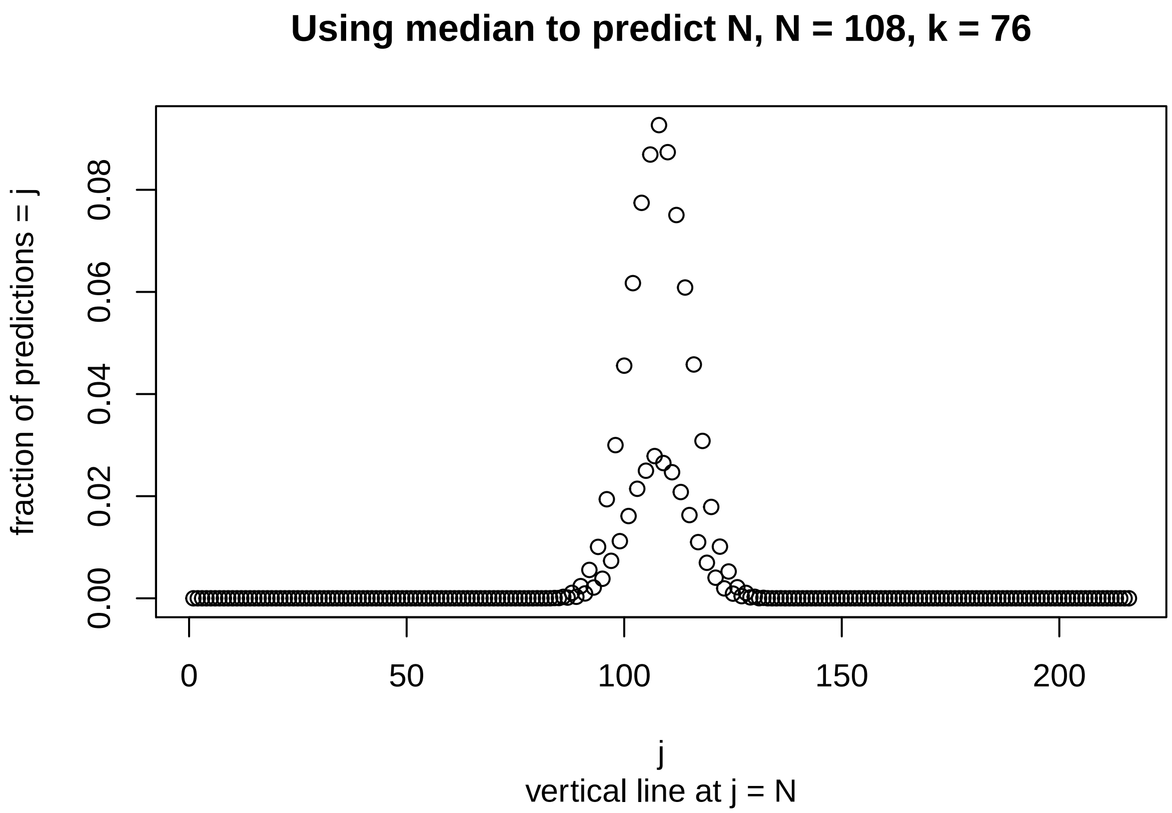
```
# OK, now what is going on

head(resultsList$Nmean.df)
```

```
##       j Nm.frac
## 109 109 0.01717
## 111 111 0.01708
## 103 103 0.01682
## 101 101 0.01667
## 113 113 0.01654
## 115 115 0.01652
```

```
# We see enrichment in ODD integer predictions for N

# Here is why:
# The mean m of 5 integers will have the form (with K an integer)
# m = K, K + 0.2, K + 0.4, K + 0.6, or K + 0.8
# and 2*m - 1 will then have the form
# 2K - 1, 2K - 0.6, 2K - 0.2, 2K + 0.2, 2K + 0.6
# and round applied to these will give
# 2K - 1, 2K - 1, 2K, 2K, 2K + 1
# so enriched in ODD integers !!

# Finally, lets look at a case where k is fairly close to N

resultsList <- mean.or.median.of.k.samples.for.finding.N(N = 108L,
        k = 76L, Lrepl = FALSE, numMCsamples = 100000L, CIfraction = 0.95)

# plot the results from using the mean to predict N

plot(resultsList$Nmean.df, ylab = "fraction of predictions = j",
     main = "Using mean to predict N, N = 108, k = 76",
     sub = "vertical line at j = N")
```



```
# The mean plot behaves nicely, since the fraction of the time the
# mean of a large number of integer values will be an integer + 1/2
# is rather small

# Now plot the results from using the median to predict N

plot(resultsList$Nmedian.df, ylab = "fraction of predictions = j",
     main = "Using median to predict N, N = 108, k = 76",
     sub = "vertical line at j = N")
```



```
# Curioser and Curioser

head(resultsList$Nmedian.df)
```

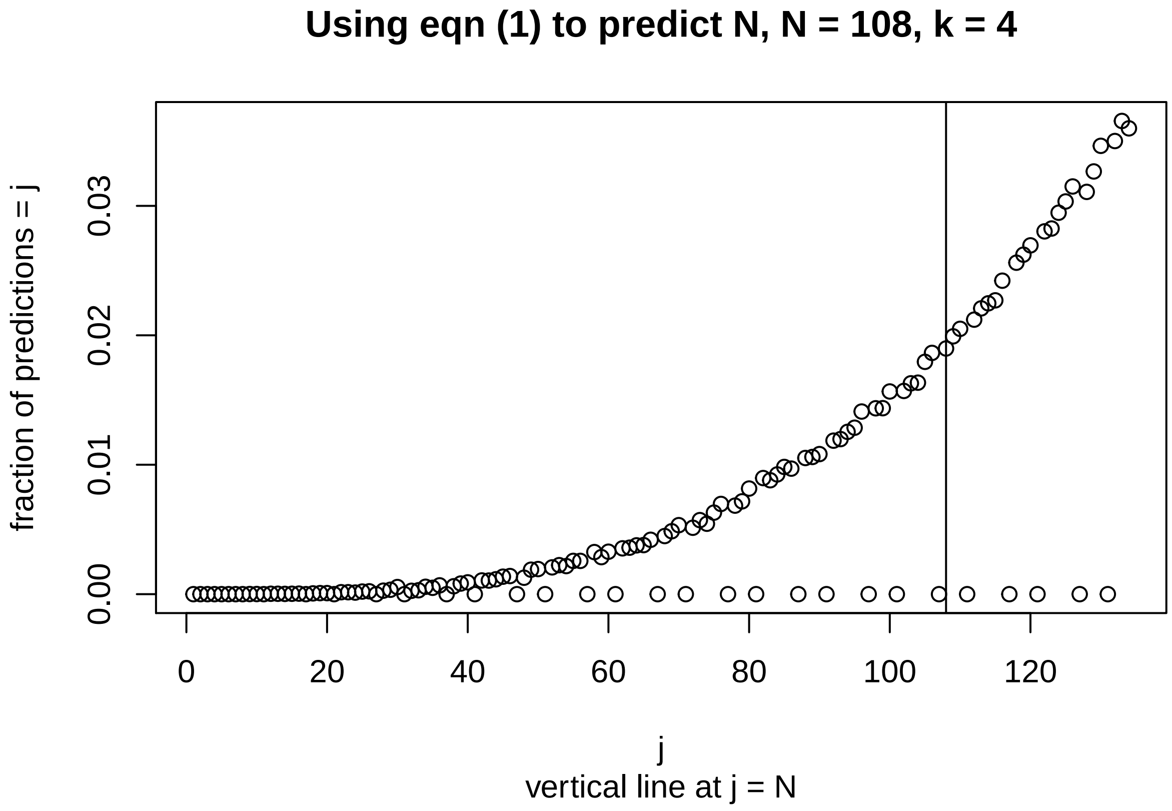
```
##       j Nmed.frac
## 108 108   0.09267
## 110 110   0.08737
## 106 106   0.08691
## 104 104   0.07745
## 112 112   0.07506
## 102 102   0.06172
```

```
# Well, I have an explanation for this too
# With this many samples (and k even), it is fairly likely the median
# (which will be the average of the two middle values of the
# samples) will be the average of 2 CONSECUTIVE integers,
# so 2*median - 1 will have the form (j + j + 1) - 1 which is EVEN
```

Finally, let’s see how well the “good estimate” in equation (1) does

```r
# See how well equation (1) does for predicting N

set.seed(123)

N <- 108L
k <- 4L
numMCsamples <- 100000L
CIfraction <- 0.95
Lrepl <- FALSE


# Initialize required vectors
Nvector <- 1:N
jVector <- 1:134  #  for k = 4, N = 108, 134 is the max from eqn (1)
eqn1Vector <- vector(mode = "integer", length = 134)

Neqn1.values <- numeric(numMCsamples)

# Do the numMCsamples Monte Carlo sampling runs using basic R commands

######### try finishing this code to see how well equation (1) does
######### in predicting N before reading on

for (i in 1:numMCsamples) {
   ksamples <- sample(Nvector, size = k, replace = Lrepl, prob = NULL)
   Neqn1 <- max(ksamples) * (k + 1) / k   -  1
   Neqn1 <- round(Neqn1)
   eqn1Vector[Neqn1] <- eqn1Vector[Neqn1] + 1L

   Neqn1.values[i] <- Neqn1
}

Neqn1.df <- data.frame(j = jVector, Neqn1.frac = eqn1Vector / numMCsamples)

# sort the data frame

sort.indices <- sort.list(eqn1Vector, decreasing = TRUE)
Neqn1.df <- Neqn1.df[sort.indices, ]

# get the confidence intervals
vec <- Neqn1.values
Neqn1.CIvector <-
```

```
   obtain.conf.interval.from.vector.vec.about.value.y.4March2022(vec,
                     y = N, conf.level = CIfraction)

# list.to.return <- list(Neqn1.df = Neqn1.df,
#  Neqn1.CIvector = Neqn1.CIvector,
#  Neqn1.values)

# plot how well equation (1) does in estimating N

 plot(Neqn1.df, ylab = "fraction of predictions = j",
      main = "Using eqn (1) to predict N, N = 108, k = 4",
      sub = "vertical line at j = N")
abline(v = 108)  # plot vertical line at j = true value of N
```



```
# Note round((1:108) * (5/4) - 1) will have "gaps", that is several
# integers whose values will not be attained

# display the confidence interval
Neqn1.CIvector
```

```
##      CIleft     CIright       CIlen  conf.level exact.count            y
##        64.00       134.00         70.00          0.95     1898.00        108.00
```

```
# We see that the estimates for N from equation (1) are quite a bit
# better than the estimates from the mean or median
```

Hope this discussion has been helpful.

= =

Note the reader should not infer any endorsement or recommendation or approval for the material in this article from any of the sources or persons cited above or any other entities mentioned in this article.