**Monte Carlo (MC) simulation**

Motivation

* Star Trek Next Generation – Season 3, Episode 6, 36:20-37:20; try to estimate the expected value of a Bernoulli random variable; Geordi understands simulation variability?
* First year of graduate school

What is MC simulation?

MC simulation is used to approximate an expected value or probability involving a random variable through multiple runs or “simulations” of the same task. A computer is used most often to obtain these simulations.

The “Monte Carlo” name originates from the Monte Carlo Casino in Monaco. Simulations were performed to better understand probabilities associated from gambling.

Statisticians are usually interested in the following setting:

* Let T be a statistic (estimator) of interest with PDF f(t).
* We may be interested in E(T). To estimate it, we somehow obtain a random sample of size R from f(t) to obtain t1, …, tR.
* Use  to estimate E(T).

Comments:

* The PDF f(t) may be unknown. If T is a function of random variables Y1, …, Yn, then we could instead simulate R samples of size n from the PDF of f(y) and calculate tr for each sample. In a simple setting, T could be a sample mean of Y1, …, Yn.
* The PDF f(t) and CDF F(t) can be estimated using t1, …, tR from the MC simulation. For example, a simple histogram could be constructed as a visualization of the PDF. Also, the CDF can be estimated by using the empirical distribution function (EDF):

Let  be proportion of observations in the sample that fall less than or equal to t. Then  is



where  is the indicator function. Sometimes, this sum is simply represented as



* More generally, you can think of  as estimating . If T was actually a discrete random variable, you could replace the integration with a summation or use Stieltjes integration instead.
* The expected values of a function of T, say E(g(T)), can be estimated as well by using . For example, suppose you want to estimate P(T > c) for some constant c. Then g(tr) = I(tr > c) and  estimates  (if T was continuous between -∞ to ∞). Equivalently, you could represent this estimated integral through the EDF: 1 – 

Why does MC simulation work?

From your first statistics course, you learned how to estimate a population mean μ. This involved taking a random sample from a population to obtain y1, …, yn and calculating . As long as the sample was large enough,  provided a decent estimate of μ.

The “decent estimate” part results from the law of large numbers (LNN)! The weak LLN (WLLN) from p. 235 of Casella and Berger (2002) is:

Let Y1, Y2, … be iid random variables with E(Yi) = μ and Var(Yi) = σ2 < ∞. Define . Then, for every ε > 0,



that is,  converges in probability to μ ().

Comments:

* Note that Casella and Berger put the subscript n on  to emphasize the statistic’s dependence on the sample size.
* Of course, n needs to be “large” then for us to obtain a “decent estimate”.
* When an estimator approaches a constant as n → ∞, we say that estimator is *consistent*; see p. 233 of Casella and Berger (2002).
* The WLLN numbers holds for some function g(Yi) as well, so that we can say  as n → ∞; see p. 246 of Casella and Berger (2002).

The last comment above also validates our use of MC simulation because T is a function of Y1, …, Yn. In general, for a function of T itself, we have



as R → ∞. When performing an actual MC simulation, we would then use  to estimate  for a fixed value of R. Often, our hope in a MC simulation is that  is similar in value to a parameter θ, say, that we are trying to estimate with g(T).

The EDF is also a good estimate of the CDF. Continuing the same notation from the WLLN statement, for Y1, Y2, … as n → ∞, we can rely on one of three reasons for why the EDF is a good estimate:

*  for each fixed y
* The strong LLN implies that  for each fixed y (wp1 is the same as almost sure convergence).
* Glivenko-Cantelli Theorem:  (Ferguson 1996, p. 23)

Thus,  is a good estimate of F(t) for a large R.

“Unbiased” estimators

A common purpose of MC simulation in a statistical research setting is to determine if a proposed estimator is unbiased in an approximate sense. In other words, can we show for some sample size that  is close to θ?

You will not be able to determine if an estimator is unbiased through MC simulations (except perhaps when there is a finite population). The goal instead is to show  is fairly close to θ.

A statistical research paper typically will show results from a number of sets of MC simulations, say one set at n = 25, another set at n = 50, …, and calculate  for each set. One of the following scenarios will usually occur:

* If  is close to θ for each set, then there is evidence that g(T) is an unbiased estimator of the parameter. The estimator could be recommended for a sample size as low as what was used for the MC simulations.
* If  is not close to θ, then we have a biased estimator. Typically, one would not want to use this estimator.
* If  is not too close to θ for the smaller sample sizes but close at the larger samples sizes, then there is evidence that g(T) is an unbiased estimator of the parameter for a sufficient sample size. However, one should include in a paper that there is a non-zero bias for the smaller sample sizes.

Example: MC simulation for the variance (MC\_sim\_var.R)

My statistic of interest T is the unbiased version of the sample variance



I will use MC simulation to estimate the population variance and T’s distribution. This distribution will be especially important for later examples when we use MC simulation with confidence intervals and hypothesis tests.

Suppose Yi are iid for i = 1, …, 9 with a distribution N(μ, σ2). Let μ = 2.713333 and σ2 = 2.1955812. Below is the code used for a MC simulation:

> #Settings

> mu<-2.713333

> sigma<-2.195581

> sigma^2

[1] 4.820576

> n<-9

> R<-500

> #Simulate data all at once

> set.seed(9811)

> y.sim<-matrix(data = rnorm(n = n\*R, mean = mu, sd =

sigma), nrow = R, ncol = n)

> y.sim[1,]

[1] 3.5017854 7.0988679 1.7747522 1.5082361 0.6093600

[6] 3.0038499 0.7410184 2.6697463 3.7926440

> var(y.sim[1,])

[1] 3.968372

> #Use apply() to find all of the variances

> t.all<-apply(X = y.sim, MARGIN = 1, FUN = var)

> mean(t.all)

[1] 4.917003

> #Use a for-loop to find all of the variances

> t.all2<-numeric(length = R)

> for(r in 1:R) {

t.all2[r]<-var(y.sim[r,])

}

> mean(t.all2)

[1] 4.917003

Comments:

* Because  is close to σ2, this provides evidence to support that T is an unbiased estimator of σ2. Of course, this result is to be expected here because of the WLLN. Furthermore, as you showed in STAT 882-3, E(T) = σ2 (T is not only an asymptotically unbiased estimator) as well so this particular result is expected.
* Typically, it is best (execution time, re-generating data if needed) to simulate all samples at once rather than one at a time during a for loop. Examples where this may not be the case include when there are very large data sets.
* ALWAYS set a seed number before simulating samples so that you can reproduce the samples later. This is an essential component to the reproducibility of research.
* Both the apply() and for() functions were used to compute the statistic of interest for each simulation. Typically, both functions will take the same amount of time to execute. Note that this was not always the case.
* Is 4.917 really close enough to 4.821 to be o.k. with T as an estimator in this setting? Simply, you can use a confidence interval for a mean to help answer this question!

> mean(t.all) + qt(p = c(0.025, 0.975), df = R – 1) \*

sd(t.all) / sqrt(R)

[1] 4.693526 5.140481

Notice that 4.821 is within the interval! What would happen to this interval if a larger R was taken?

Because Y has a normal distribution, we know that



Equivalently, we know that T has a Gamma distribution with α = (n-1)/2 and β = 2σ2/(n-1) using the Casella and Berger (2002) parameterization. Let’s compare this distribution to a histogram of t1, …, tR and a plot of the EDF:

> #EDF

> win.graph(width = 10)

> par(mfrow = c(1,2))

> plot.ecdf(x = t.all, verticals = TRUE, do.p = FALSE,

lwd = 2, panel.first = grid(), ylab = "Probability",

xlab = "t", col = "red", main = "")

> abline(h = c(0,1))

> curve(expr = pgamma(q = x, shape = (n-1)/2, scale =

2\*sigma^2/(n-1)), add = TRUE, col = "blue", lwd = 2)

> legend(x = 10, y = 0.5, legend = c("EDF", "CDF"), bty =

"n", lty = 1, col = c("red", "blue"))

> #PDF

> hist(x = t.all, freq = FALSE, main = "", col = "red",

ylim = c(0, 0.2))

> curve(expr = dgamma(x = x, shape = (n-1)/2, scale =

2\*sigma^2/(n-1)), add = TRUE, col = "blue", lwd = 2)



> #Probabilities and quantiles

> probs<-c(0.025, 0.1, 0.5, 0.9, 0.975)

> quant.est<-quantile(x = t.all, probs = probs, type = 1)

#Inverse of EDF

> quant.true<-qgamma(p = probs, shape = (n-1)/2, scale =

2\*sigma^2/(n-1))

> data.frame(probs, quant.est, quant.true)

probs quant.est quant.true

2.5% 0.025 1.472609 1.313445

10% 0.100 2.067811 2.102699

50% 0.500 4.430517 4.425362

90% 0.900 7.958433 8.051306

97.5% 0.975 11.045761 10.565826

> F.hat<-ecdf(x = t.all) #Returns a function to find

estimated probabilities

> F.hat(quant.est)

[1] 0.026 0.100 0.500 0.900 0.976

Comments:

* As we would expect,  is a good estimate of F(t) for a large R.
* Examine the differences between the estimated and true quantiles. Notice that as the probabilities become more extreme, the quantile differences generally become greater. Why does this occur?
* Differences between what F.hat(quant.est) produces and the actual probabilities used in quantile() are due to discontinuities in the EDF.

True confidence level (coverage)

A 95% confidence interval for a parameter is NOT truly a 95% confidence interval most of the time. The true confidence level, also known as “coverage”, is the proportion of time that the confidence interval truly contains the parameter of interest. One of the main purposes of MC simulation studies in statistical research is to estimate this true confidence level.

A confidence level is attributed to the hypothetical process of repeating the same sampling and calculations over and over again (from which the name “frequentist” is derived) so that (1 – α)100% of the intervals contain the parameter. Therefore, a MC simulation to estimate the true confidence level then performs the following:

* Simulate R data sets under the same conditions where the parameter of interest (say, θ) is known
* Calculate the confidence interval for each data set
* Determine if the parameter is within each interval
* Estimate the true confidence level as , where  if the confidence interval for the rth data set contains θ and  otherwise

Notice that we are just calculating  in the last step where  is . Thus, this quantity is estimating an expected value: . Our hope is that this expected value is equal to 1 – α.

The expected length of a confidence interval is important too. In this case, , where  is the upper bound and  is the lower bound for the confidence interval calculated with the rth data set.

When you are evaluating a number of confidence intervals, the best confidence interval is the one that

* Has a true confidence level closest to the stated (1 – α)100% confidence level
* Has the shortest expected length

Unfortunately, often one confidence interval will not satisfy both of the bullets above. In those cases, one will typically assign a greater priority to the true confidence level. Among those intervals with true confidence levels close to (1 – α)100%, the best interval has the shortest expected length.

Size and power

An α = 0.05 hypothesis test for a parameter does NOT truly have a type I error rate of 0.05 most of the time. The true size of a test then is the proportion of time that the null hypothesis is rejected when the null hypothesis is true. Again, one of the main purposes of MC simulation studies in statistical research is to estimate the size of a test.

The MC simulation process is:

* Simulate R data sets under the same conditions where the parameter of interest (say, θ) is known and the null hypothesis is true
* Calculate the p-value for each data set
* Determine if each p-value is less than the stated level of α
* Estimate the true size as , where  if the p-value is less than α for the rth data set and  otherwise

When the null hypothesis is not true, the same calculations above will estimate the power of a hypothesis test.

When you are evaluating a number of hypothesis testing procedures, the best procedure is the one that

* Has a true size closest to the stated α level
* Has the largest power

Unfortunately, often one hypothesis testing procedure will not satisfy both of the bullets above. In those cases, one will typically assign a greater priority to the true size. Among those procedures with true sizes close to α, the best procedure has the largest power.

Conservative vs. liberal inferences

A conservative confidence interval is one that has a true confidence level larger than the stated level. For example, a 95% confidence interval with a true confidence level of 99% is VERY conservative.

A liberal confidence interval is one that has a true confidence level smaller than the stated level. For example, a 95% confidence interval with a true confidence level of 80% is VERY liberal.

In most situations, one will prefer a conservative interval over a liberal interval.

Some individuals will mistaken conservativeness and liberalness to mean an interval has a larger or smaller than necessary expected length, respectively. This is not necessarily true.

The liberalness of a confidence interval is often due to it being too short on average. However, an interval which is not located at the right place can be liberal too. For example, suppose a population mean is μ = 2 and most intervals in a set of simulations are around 2.9999 < μ < 3.0001. Depending on the situation, this could be considered a “short” in length interval, but just not located in the correct spot.

Similar statements can be made about the conservativeness of an interval not necessarily being larger than necessary in expected length; however, I see this occur much less than the liberal location problem described above.

Hypothesis testing procedures can be conservative and liberal as well:

* A conservative test does not reject the null hypothesis enough when the null hypothesis is true (rejects < 100α% of the time).
* A liberal test rejects the null hypothesis too often when the null hypothesis is true (rejects > 100α% of the time).

Example: MC simulation for the variance (MC\_sim\_var.R)

Continuing the last example, we are now going to estimate the true confidence level for two intervals:

1. Normal-based

Let Yi ~ N(μ, σ2) for i = 1, …, n and suppose the Yi‘s are independent. Let t be the observed sample variance. The standard confidence interval for σ2 used in this situation is



where  is the 1 – α/2 quantile from a chi-square distribution with n – 1 degrees of freedom.

1. Asymptotic

A commonly used book for an asymptotics class is “A Course in Large Sample Theory” by Thomas Ferguson (1996). Page 46 of the book shows that



where μ4 = . Thus, the asymptotic variance for T is



The estimated asymptotic variance is then



where . Using the normality result, a confidence interval for σ2 is



where  is the 1 – α/2 quantile from a standard normal distribution. An important aspect of this interval is that the distribution of Y1, …, Yn is not stated.

Note that there are extreme situations where  can be estimated to be negative. Obviously, the asymptotic interval cannot be found in those situations.

Below is a function that calculates all three intervals.

> sim.func<-function(y, alpha = 0.05, B = 4999) {

n<-length(y)

t<-var(y)

#normal-based interval

lower1<-(n - 1)\*t / qchisq(p = 1 - alpha/2, df = n – 1)

upper1<-(n - 1)\*t / qchisq(p = alpha/2, df = n - 1)

normal.based<-c(lower1, upper1)

#asymptotic interval

mu.hat4<-1/n\*sum((y - mean(y))^4)

asym<-t + qnorm(p = c(alpha/2, 1-alpha/2)) \*

sqrt((mu.hat4 - t^2)/n)

c(normal.based, asym)

}

> #Test function

> sim.func(y = y.sim[1,])

[1] 1.8105389 14.5646327 0.4607171 7.4760273

For the first sample, the 95% confidence intervals are:

1. Normal-based: (1.81, 14.56)
2. Asymptotic: (0.46, 7.48)

Each interval contains σ2 = 4.82! Next, I can use the apply() function to “apply” the sim.func() to each row of y.sim:

> save.int<-t(apply(X = y.sim, MARGIN = 1, FUN = sim.func))

> head(round(save.int,2))

[1,] 1.81 14.56 0.46 7.48

[2,] 3.42 27.49 3.42 11.56

[3,] 0.48 3.82 0.70 1.38

[4,] 2.73 21.99 -0.29 12.28

[5,] 2.23 17.96 2.07 7.71

[6,] 2.33 18.72 2.31 7.89

My summarize() function below calculates the estimated true confidence levels and estimated expected lengths for all of the intervals.

> summarize<-function(intervals, sigma.sq) {

#True confidence levels

normal.based1<-mean(ifelse(test = sigma.sq >

intervals[,1], yes = ifelse(test = sigma.sq <

intervals[,2], yes = 1, no = 0), no = 0), na.rm =

TRUE)

asym1<-mean(ifelse(test = sigma.sq > intervals[,3],

yes = ifelse(test = sigma.sq < intervals[,4], 1,

0), 0), na.rm = TRUE)

#Expected length

normal.based2<-mean(intervals[,2] - intervals[,1],

na.rm = TRUE)

asym2<-mean(intervals[,4] - intervals[,3], na.rm =

TRUE)

#Count number of simulated data sets excluded

normal.based.na<-sum(is.na(intervals[,1]) |

is.na(intervals[,2])) #Should be 0

asym.na<-sum(is.na(intervals[,3]) |

is.na(intervals[,4]))

interval.names<-c("Normal-based", "Asymptotic")

true.conf<-c(normal.based1, asym1)

exp.length<-c(normal.based2, asym2)

na.data.sets<-c(normal.based.na, asym.na)

cat("True confidence levels and expected lengths:

\n")

save.res<-data.frame(name = interval.names, true.conf

= true.conf, exp.length = round(exp.length,2),

na.data.sets = na.data.sets)

print(save.res)

save.res

}

> est.true.conf<-summarize(intervals = save.int, sigma.sq =

sigma^2)

True confidence levels and expected lengths:

name true.conf exp.length na.data.sets

1 Normal-based 0.960 15.80 0

2 Asymptotic 0.712 5.79 0

Comments:

* Because the asymptotic interval cannot always be calculated for data sets, I use the na.rm = TRUE argument for many of my calculations. For the example here, all data sets were o.k. If I was doing this MC simulation for a paper, I would need to state the proportion of time when the interval could not be calculated. AGAIN, this information would NEED TO BE STATED!
* Which interval is best? Why do you think this occurred?

Standard deviation (or variance) of an estimator

Understanding the standard deviation (or variance) of an estimator is usually very important. For example, a Wald test statistic for H0: θ = θ0 vs. Ha: θ ≠ θ0 takes on the form



If  is too small, then Z will be too large in absolute value. Overall, this would cause Z to reject the null hypothesis too often. Similar statements can be made regarding Wald confidence intervals.

Question: Suppose T is a maximum likelihood estimator of θ. What would you use for ? What are common qualities of this estimator?

To examine whether or not  is a good estimator of  through MC simulation, we would like to see how close  is to . Unfortunately,  is usually not available to us except for in simple problems (e.g., the variance of a sample mean is σ2/n, where Var(Yi) = σ2 for i = 1, …, n).

Fortunately, there is a way around  being unknown. Below is the MC simulation way used most often to examine :

* Simulate R data sets under the same conditions where the parameter of interest is known
* Calculate  for each data set
* Calculate the average estimated variance across all simulations as . What is this quantity estimating?
* Calculate the sample variance for T across all simulations as , where . This sample variance provides an estimate of  that will be good as long as R is large. Why?
* Compare  to . This is usually done using a ratio:

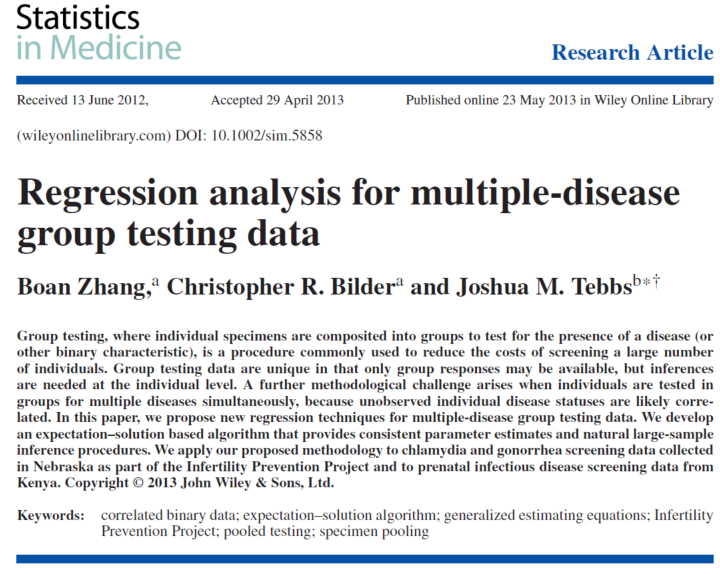


If w is close to 1, this provides evidence that  is a an approximate unbiased estimator for . If w < 1, this says that  is too low. This can then have detrimental effects on inference when using . A similar statement can be made if w > 1, but the consequences are not necessarily as severe (e.g., the result would be a conservative test).

Note that the ratios of standard deviations can also be compared through an expression like w. This can sometimes be a preferred way to perform the comparison because the squared unit scale of variances can somewhat distort differences.

Example: Zhang, Bilder, and Tebbs (*Statistics in Medicine*, 2013, 4954-4966)

Below is the abstract



In summary, this research developed a generalized estimating equation (GEE) approach to estimate a regression model for correlated binary responses. What made this paper unique was the responses are unobservable!

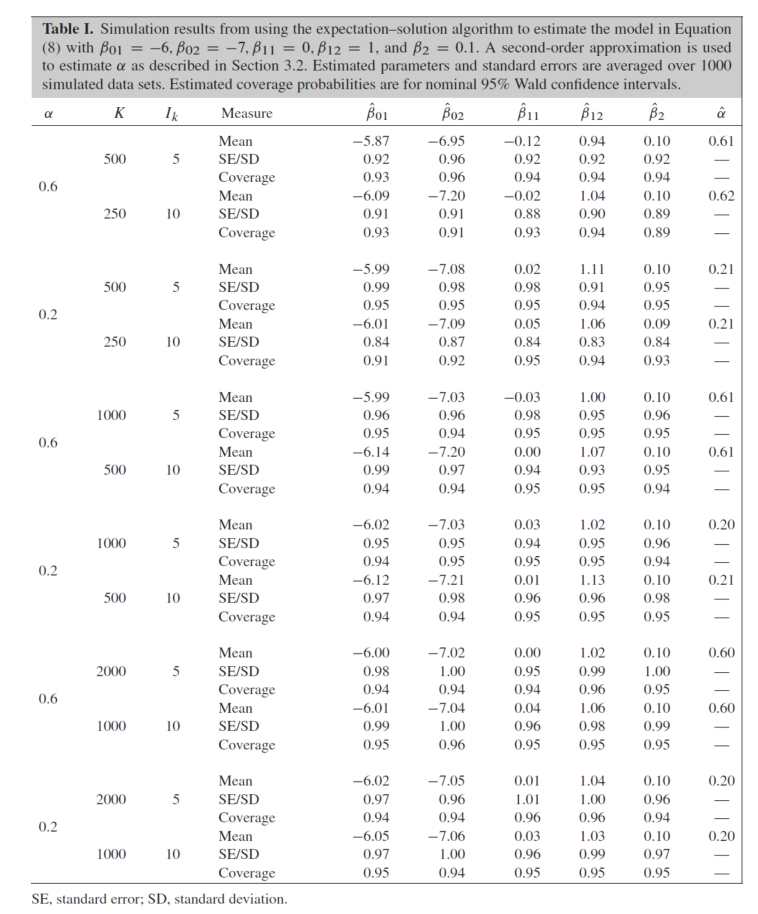
Next is a table summarizing some of our simulation study results. Some of the information displayed includes:

* α – correlation between the binary random variables with estimator 
* K × Ik – overall sample size
* βij – regression parameter with estimator 

Note that Statistics in Medicine could have displayed the information in the table a little better so I have drawn in some lines to help separate items.

What can you conclude about

* Approximate unbiasedness of  and 
* Approximate unbiasedness of 
* True confidence level for confidence intervals for these simulation settings?





Example: MC simulation for the variance (MC\_sim\_var.R)

What should we use for ? When forming the asymptotic-based confidence interval, we saw that



where . Therefore, this seems like a reasonable value to use for .

Because T has a Gamma distribution with α = (n-1)/2 and β = 2σ2/(n-1), we know Var(T) when Y has a normal distribution! This variance is

Var(T) = αβ2 = (n-1)/2 × 4σ4/(n-1)2 = 2σ4/(n-1)

A reasonable estimator for Var(T) then would be 2t2/(n – 1).

Below is my code. Ideally, my sim.func() function from earlier would have contained all of this code. However, because this was a separate topic in the notes, I decided to write a new function.

> var.examine<-function(y) {

n<-length(y)

t<-var(y)

#asymptotic estimate

mu.hat4<-1/n\*sum((y - mean(y))^4)

asym.est<-(mu.hat4 - t^2)/n

#normal-based estimate

norm.est<-2\*t^2/(n-1)

c(t, asym.est, norm.est)

)

}

> #Test function

> var.examine(y = y.sim[1,])

[1] 3.968372 3.202857 3.936995 1.789653 1.984186

> #Calculate variances

> save.var<-t(apply(X = y.sim, MARGIN = 1, FUN =

var.examine))

> mean.var<-colMeans(save.var[,2:3])

> samp.var<-var(save.var[,1])

> mean.var

[1] 3.131160 7.658224

> samp.var

[1] 6.468911

> mean.var/samp.var #w

[1] 0.4840321 1.1838506

> true.var<-2\*sigma^4/(n-1)

> true.var

[1] 5.809488

> mean.var/true.var #w with true variance

[1] 0.5389735 1.3182270

The asymptotic variance estimator is estimated to under estimate Var(T) by about 52%! The normal-based estimator over estimates by about 18%. Because we can calculate the actual value of Var(T) here, I have included



as well. We obtain similar results in terms of under and over estimation using this expression too.

Because the variance is on squared unit scale, differences between the estimator and its true value can be amplified. Below are the same types of calculations, but now using the standard deviations:

> mean.sd<-colMeans(sqrt(save.var[,2:3]))

> sample.sd<-sd(save.var[,1])

> mean.sd

[1] 1.477608 2.458502

> sample.sd

[1] 2.543405

> mean.sd/sample.sd #w with sd

[1] 0.5809565 0.9666181

> true.sd<-sqrt(true.var)

> true.sd

[1] 2.410288

> mean.sd/true.sd #w with true sd

[1] 0.6130421 1.0200033

The asymptotic variance estimator still severely under estimates, but the normal-based estimator does a good job.

Simulation variability

The results from one set of simulations (R simulated data sets) are likely to be at least a little different than from another set of simulations even when all of the simulation “settings” (e.g., distribution for Y, sample size, …) are the same. This variability can be controlled by the choice of R. As R gets large, we would expect the results from separate sets of simulations to get closer to each other (simulation variability decreases) because they are all estimating the same quantity.

In most situations, we would only run one set of simulations for the same simulation settings. Also, we can never choose R = ∞, so there are practical considerations that one needs to take into account when choosing R. Therefore, it would be of interest to know for a fixed R what the simulation variability is.

When we are estimating true confidence levels, test sizes, and power levels, we are simply estimating a proportion. Therefore, we can use STAT 875 methods to calculate a confidence interval for a proportion to understand the simulation variability!

Let  be an estimated proportion (like the estimated confidence level or size) and π be the population proportion (like the true confidence level or size). The (1 – γ)100% Wald confidence interval for π is



This is the interval typically taught in most introductory statistics courses, but unfortunately it does not perform as well (estimated true confidence level can be much lower than the true confidence level) as other intervals.

A better interval is the Wilson (score) interval:



where



and xr = 1 if the rth interval contains θ (or test rejects) and 0 otherwise.

The lower and upper endpoints of the interval give you a range for where you would expect the true value (e.g., true size) to be. Note that the Wald and Wilson intervals will be similar for a large R.

When using these intervals, you can examine if the stated level for 1 – α or α (depending on what you are calculating) is within the interval. If it is not, then there is evidence that the confidence interval (or hypothesis test) method is not working at the stated confidence level (or size level).

More often in statistical papers, a slightly different perspective is used to examine simulation variability. Let’s consider the case of a true size level first. If the stated size level is α, one would expect all of the estimated size levels to fall within



IF the testing procedure is working as stated. This formula results from using a normal approximation to the binomial. For the true confidence level, you can use



instead. If the estimated true confidence level or size falls outside of its corresponding interval, we have evidence that the statistical method is not working at the stated level.

The  in the last paragraph may even be replaced with simply 2 or 3 as well. As I tell my students when I teach introductory statistics courses, all “data” should fall within ±2 or ±3 standard deviations from its mean. Here, the data is one estimate of the true confidence level or size. If the estimated true confidence level or size falls outside of this interval, we have evidence that the statistical method is not working at the stated level.

Example: MC simulation for the variance (MC\_sim\_var.R)

Continuing the last example, below are the Wald and Wilson intervals.

> gamma1<-0.05

> pi.hat<-est.true.conf$true.conf

> R.used<-R - est.true.conf$na.data.sets #In case not all

data sets used

> sum.x<-R.used\*pi.hat

> pi.tilde<-(sum.x + qnorm(p = 1-gamma1/2)^2 / 2) / (R.used

+ qnorm(p = 1-gamma1/2)^2)

> #Wald

> lower1<-pi.hat + qnorm(p = gamma1/2)\*sqrt(pi.hat\*(1-

pi.hat)/R.used)

> upper1<-pi.hat + qnorm(p = 1-gamma1/2)\*sqrt(pi.hat\*(1-

pi.hat)/R.used)

> data.frame(name = est.true.conf$name, lower =

round(lower1, 3), upper = round(upper1, 3))

name lower upper

1 Normal-based 0.943 0.977

2 Asymptotic 0.672 0.752

> #Wilson

> lower2<-pi.tilde + qnorm(p = gamma1/2) \* sqrt(R.used) /

(R.used + qnorm(p = 1-gamma1/2)^2) \* sqrt(pi.hat \* (1-

pi.hat) + qnorm(1-gamma1/2)^2/(4\*R.used))

> upper2<-pi.tilde + qnorm(p = 1-gamma1/2) \* sqrt(R.used) /

(R.used + qnorm(p = 1-gamma1/2)^2) \* sqrt(pi.hat \* (1-

pi.hat) + qnorm(1-gamma1/2)^2/(4\*R.used))

> data.frame(name = est.true.conf$name, lower =

round(lower2, 3), upper = round(upper2, 3))

name lower upper

1 Normal-based 0.939 0.974

2 Asymptotic 0.671 0.750

The asymptotic interval does not have the stated confidence levels. The normal-based interval contains 0.95.

The expected range using γ = 0.01 is

> #Expected range

> (1-alpha) + qnorm(c(0.005,0.995)) \* sqrt((1-

alpha)\*alpha/R)

[1] 0.9248939 0.9751061

The same conclusions are reached using this prospective.

What should you chose for R?

If the time it takes to complete a set of simulations is not too much of a concern, what should you choose for R? Let’s examine the expected range again for different values of R and for a true confidence level of 0.95 (code is in MC\_sim\_var.R):

> R.vec<-c(100, 500, 1000, 5000, 10000, 100000)

> R.mat<-matrix(data = R.vec, nrow = length(R.vec), ncol =

1)

> calc.range<-function(R, alpha = 0.05) {

(1-alpha) + qnorm(c(0.005,0.995)) \* sqrt((1-

alpha)\*alpha/R)

}

> save.range<-apply(X = R.mat, MARGIN = 1, FUN =

calc.range)

> #Scientific notation is used for printing R.vec so I

turned it off

> data.frame(R = format(R.vec, scientific = FALSE), lower =

round(save.range[1,], 4), upper = round(save.range[2,],

4))

R lower upper

1 100 0.8939 1.0061

2 500 0.9249 0.9751

3 1000 0.9322 0.9678

4 5000 0.9421 0.9579

5 10000 0.9444 0.9556

6 100000 0.9482 0.9518

What values of R are not adequate? Definitely R = 100 (although I have seen this used when time was a concern).

What values are over doing it? R = 100,000 and probably R = 10,000 unless you wanted to be extremely precise.

A value of R = 500 can be justified if there is “some” time concern. I have used this value in MANY papers. Only once was I questioned by a referee about using this low of a number. R = 1000 would be fine for most situations.

One of my main points here is that you do not need to take an extremely large R. There are a diminishing amount of benefits from choosing larger values of R. For example:

* Compare R = 5000 to 10000 – there is little difference between the ranges
* Compare R = 1000 to 5000 – you need to ask yourself if it is really worth it to have the 0.01 gain on either end of the range when the time it takes to complete a set of simulations is increased by 5 times

Also, in a typical simulation study, you need to understand that:

* More than one set of simulations are performed. Thus, if a simulation study is projected to take 1 week of computer time, using a larger number of simulations could really increase the time it takes to complete it.
* Programming errors and/or the addition of new statistical methods to the study often occur. This can cause re-running an entire simulation study multiple times ☹.

What if α = 0.10 or 0.01?

> #alpha = 0.01

> save.range2<-apply(X = R.mat, MARGIN = 1, FUN =

calc.range, alpha = 0.01)

> data.frame(R = format(R.vec, scientific = FALSE), lower =

round(save.range2[1,], 4), upper =

round(save.range2[2,], 4))

R lower upper

1 100 0.9644 1.0156

2 500 0.9785 1.0015

3 1000 0.9819 0.9981

4 5000 0.9864 0.9936

5 10000 0.9874 0.9926

6 100000 0.9892 0.9908

> #alpha = 0.10

> save.range3<-apply(X = R.mat, MARGIN = 1, FUN =

calc.range, alpha = 0.10)

> data.frame(R = format(R.vec, scientific = FALSE), lower =

round(save.range3[1,], 4), upper =

round(save.range3[2,], 4))

R lower upper

1 100 0.8227 0.9773

2 500 0.8654 0.9346

3 1000 0.8756 0.9244

4 5000 0.8891 0.9109

5 10000 0.8923 0.9077

6 100000 0.8976 0.9024

Note that



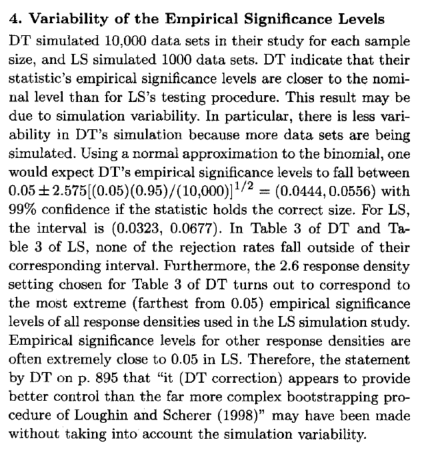
deceases as α decreases for a fixed R. Thus, the ranges become smaller for a fixed R as α decreases. However, smaller values of α with smaller values of R can lead to an upper range above 1. This can result in one to choose a value of R such that there is some “room” above the upper limit.

Example: Bilder and Loughin (*Biometrics*, 2001, p. 1253-1255)

This is a paper where I pointed out that the authors of another paper (and the corresponding referees and editors) perhaps did not really understand simulation variability!

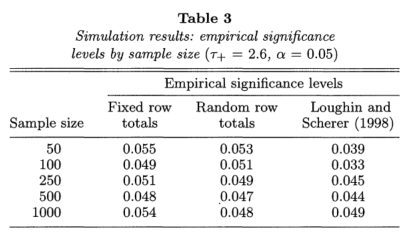
Decady and Thomas (“DT”, *Biometrics*, 2000, p. 893-896) proposed a new way to perform a hypothesis test for a particular data problem. They said their method was better than what was proposed in Loughin and Scherer (“LS”, *Biometrics*, 1998, p. 630-637) because their method generally had an estimated size closer to the state level.

Below is part of Bilder and Loughin (2001):



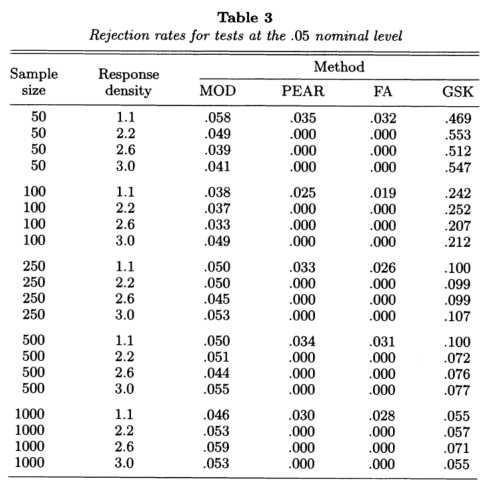
“Empirical significance levels” is another name for “estimated size”.

Below is the corresponding table from Decady and Thomas (2000):





Below is the corresponding table from Loughin and Scherer (1998):





Note that the n = 1000 result for Loughin and Scherer (1998) was incorrectly entered into Decady and Thomas (2000).

How long will a set of simulations take?

Before running a set of simulations, it is usually very useful to estimate how long it will take to complete by timing a much smaller set. For example, time how long 10 simulations will take, and multiply the result by R/10 to estimate how long all R will take. This can help you determine if you can simply wait for the set of simulations to complete on your own computer, run the set of simulations at nighttime, or run the set of simulations on other computers.

There are a number of tools available for this time estimation. I most often use proc.time() before and after my MC simulation code to record the time it takes for 10 simulations.

Example: MC simulation for the variance (MC\_sim\_var.R)

The MC simulations for this example do not take long to complete, so this may not be the best example to show how useful estimating time to completion can be. Below is how I can estimate the time to completion using 100 simulation runs (this is much larger than I would normally use).

> start.time<-proc.time()

> set.seed(8712)

> save.int<-t(apply(X = y.sim[1:100,], MARGIN = 1, FUN =

sim.func))

> est.true.conf<-summarize(intervals = save.int, sigma.sq =

sigma^2)

True confidence levels and expected lengths:

name true.conf exp.length na.data.sets

1 Normal-based 0.96 15.76 0

2 Asymptotic 0.74 6.09 0

> end.time<-proc.time()

> save.time<-end.time-start.time

> cat("\n Number of minutes running:", save.time[3]/60, "\n

\n")

Number of minutes running: 0.0001666667

> save.time[3]/60\*R/100 #Estimate total number of minutes

elapsed

0.0008333333

> #Illustrate information available

> end.time

user system elapsed

74.40 1.04 3447.32

> names(end.time) #print() method gives only the first

three elements in it

[1] "user.self" "sys.self" "elapsed" "user.child"

[5] "sys.child"

> end.time[3]

elapsed

3642.82

The third element returned by proc.time() is the important one because it gives the time elapsed since R was started in seconds. Based on using this code, the estimated time my set of simulations will take is just 0.00083 minutes. Below is what happens when I run the full set of simulations:

> start.time<-proc.time()

> set.seed(8712)

> save.int<-t(apply(X = y.sim, MARGIN = 1, FUN = sim.func))

> est.true.conf<-summarize(intervals = save.int, sigma.sq =

sigma^2)

True confidence levels and expected lengths:

name true.conf exp.length na.data.sets

1 Normal-based 0.960 15.80 0

2 Asymptotic 0.712 5.79 0

> end.time<-proc.time()

> save.time<-end.time-start.time

> cat("\n Number of minutes running:", save.time[3]/60, "\n

\n")

Number of minutes running: 0.0005

My estimate was longer than what actually occurred, but this is likely due to the very short period of time for the simulations to begin with.

Alternatively, system.time() essentially uses proc.time() in the same way as I did and then performs the same subtraction between start and end time. However, I find it is a little more awkward to use. Below are a few examples:

> save.time<-system.time(expr = temp<-t(apply(X =

y.sim[1:100,], MARGIN = 1, FUN = sim.func)))

> save.time[3]

elapsed

0

> save.time[3]/60\*R/100 #Estimate total number of minutes

elapsed

0

> head(temp, n = 2)

[1,] 1.810539 14.56463 0.4607171 7.476027

[2,] 3.417547 27.49199 3.4218801 11.559406

> save.time<-system.time(expr = {

set.seed(8712)

save.int<-t(apply(X = y.sim[1:100,], MARGIN = 1, FUN =

sim.func))

est.true.conf<-summarize(intervals = save.int, sigma.sq

= sigma^2)

}

True confidence levels and expected lengths:

name true.conf exp.length na.data.sets

1 Normal-based 0.9 15.19 0

2 Asymptotic 0.7 5.70 0

3 Bootstrap percentile 0.8 6.66 0

> save.time

user system elapsed

0 0 0

The 0 values are simply due to these simulation runs being very quick.

What is taking so long in the code?

There are occasions when you would like to investigate which portions of the code are taking a lot of time to execute. Once identified, this may lead you to alternative ways to program a set of simulations in order to shorten the run time.

One could manually use functions like proc.time() to check parts of the code. Alternatively, one could use functions like Rprof() to obtain a profile of the time it takes for functions to execute.

Example: MC simulation for the variance (MC\_sim\_var.R)

I am going to see R = 50,000 here so that we can obtain a better view of what Rprof() produces.

> R<-50000

> #Simulate data all at once

> set.seed(9811)

> y.sim<-matrix(data = rnorm(n = n\*R, mean = mu, sd =

sigma), nrow = R, ncol = n)

> setwd(dir = "C:\\chris\\")

> #Specify the specific file to save the profiling

information into

> Rprof(filename = "profile.txt") #Start profiling code

> save.int<-t(apply(X = y.sim, MARGIN = 1, FUN =

sim.func))

> est.true.conf<-summarize(intervals = save.int, sigma.sq

= sigma^2)

True confidence levels and expected lengths:

name true.conf exp.length na.data.sets

1 Normal-based 0.9491200 15.43 0

2 Asymptotic 0.6968979 5.64 162

> Rprof() #Turn off

> summaryRprof(filename = "profile.txt")

$by.self

self.time self.pct total.time total.pct

"FUN" 0.40 18.18 2.02 91.82

"mean" 0.32 14.55 0.44 20.00

"var" 0.18 8.18 0.74 33.64

".External" 0.14 6.36 0.14 6.36

"pmatch" 0.14 6.36 0.14 6.36

"stopifnot" 0.12 5.45 0.36 16.36

"qchisq" 0.10 4.55 0.20 9.09

"match.call" 0.10 4.55 0.16 7.27

"apply" 0.08 3.64 2.12 96.36

"qnorm" 0.08 3.64 0.12 5.45

"mean.default" 0.08 3.64 0.08 3.64

"sys.call" 0.06 2.73 0.06 2.73

"-" 0.04 1.82 0.04 1.82

"\*" 0.04 1.82 0.04 1.82

"/" 0.04 1.82 0.04 1.82

"^" 0.04 1.82 0.04 1.82

"ifelse" 0.04 1.82 0.04 1.82

"is.data.frame" 0.04 1.82 0.04 1.82

"sum" 0.04 1.82 0.04 1.82

".Call" 0.02 0.91 0.02 0.91

":" 0.02 0.91 0.02 0.91

"any" 0.02 0.91 0.02 0.91

"is.atomic" 0.02 0.91 0.02 0.91

"lapply" 0.02 0.91 0.02 0.91

"length" 0.02 0.91 0.02 0.91

$by.total

total.time total.pct self.time self.pct

"eval" 2.20 100.00 0.00 0.00

"source" 2.20 100.00 0.00 0.00

"withVisible" 2.20 100.00 0.00 0.00

"apply" 2.12 96.36 0.08 3.64

"t" 2.12 96.36 0.00 0.00

"FUN" 2.02 91.82 0.40 18.18

"var" 0.74 33.64 0.18 8.18

"mean" 0.44 20.00 0.32 14.55

"stopifnot" 0.36 16.36 0.12 5.45

"qchisq" 0.20 9.09 0.10 4.55

"match.call" 0.16 7.27 0.10 4.55

".External" 0.14 6.36 0.14 6.36

"pmatch" 0.14 6.36 0.14 6.36

"qnorm" 0.12 5.45 0.08 3.64

"mean.default" 0.08 3.64 0.08 3.64

"sys.call" 0.06 2.73 0.06 2.73

"summarize" 0.06 2.73 0.00 0.00

"-" 0.04 1.82 0.04 1.82

"\*" 0.04 1.82 0.04 1.82

"/" 0.04 1.82 0.04 1.82

"^" 0.04 1.82 0.04 1.82

"ifelse" 0.04 1.82 0.04 1.82

"is.data.frame" 0.04 1.82 0.04 1.82

"sum" 0.04 1.82 0.04 1.82

".Call" 0.02 0.91 0.02 0.91

":" 0.02 0.91 0.02 0.91

"any" 0.02 0.91 0.02 0.91

"is.atomic" 0.02 0.91 0.02 0.91

"lapply" 0.02 0.91 0.02 0.91

"length" 0.02 0.91 0.02 0.91

"Rprof" 0.02 0.91 0.00 0.00

"unlist" 0.02 0.91 0.00 0.00

$sample.interval

[1] 0.02

$sampling.time

[1] 2.2

The function keeps track of what function is currently running every 0.02 seconds (default value) for a period of time (2.2 seconds for this example). This function information is written to an external file:



Each line of the file shows the function calls within other function calls. The summaryRprof() function then summarizes which functions are running. Note that the functions given are not just apply() or summarize(), but specific functions that may be called within them.

Comments:

* The $by.self and $by.total are meant to summarize the same information through sorting by different columns in the data frame display. The $by.self part gives information about when a specific function is running only. The $by.total part includes function hierarchy in its timing. For example, we see that the apply() function was executing for 2.12 seconds in total (96.36% of the time), which makes sense given the structure of the code. However, you do not see the function listed in the $by.self part because the execution time of functions that apply() called was not included.
* Notice that the t(apply()) part of the code takes 2.12 seconds and the summarize() part of the code takes 0.06 seconds. Then 2.12 + 0.06 = 2.18 ≈ 2.2 which is equal to the sampling time.

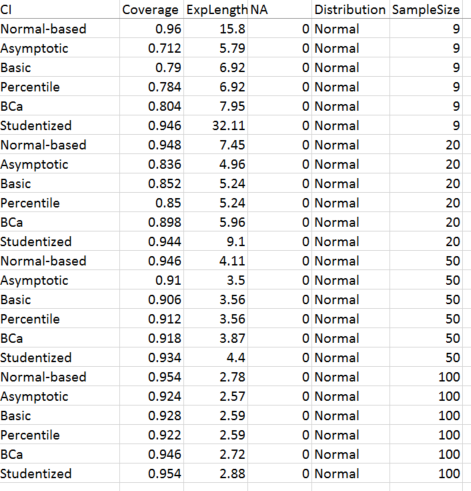
Summarizing results from a simulation study

Statistical research papers typically summarize the numerical results from simulation studies in a table format. The problem with this approach is that there are often a lot numbers to examine! This can make it difficult to detect trends across the simulation settings. Surprisingly, statisticians are not following what we preach to others – use graphs rather than tables. In fact, here’s a paper from a few years ago on this topic:

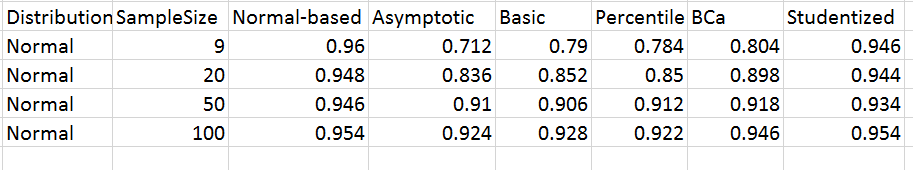
Gelman, Pasarica, and Dodhia (2002). Let’s practice what we preach: Turning tables into graphs. *American Statistician* 56, 121-130.

The purpose of this section is to show a few good ways to summarize results from a simulation study.

A simulation study essentially produces multivariate data. For example, below is a screen capture from an Excel file showing the normal distribution part of a larger set of simulations corresponding to our variance example from earlier:



The sample size of 9 simulations were those performed before, but now with four bootstrap confidence intervals (basic, percentile, BCa, studentized) are included. This “data” is multivariate because each “observation” (row) contains information on six “variables”. In other situations, one may also see part of the same data summarized as



for the true confidence levels. In the end, it is still multivariate data.

In STAT 873, you learn about many different ways to graph multivariate data. Those same types of graphical methods can be used here! In particular, we will focus on trellis plots. Trellis plots are a great way to summarize a MC simulation study because of the qualitative variables involved in the simulation settings. If you are not familiar with these types of plots, please see the graphics section of my STAT 873 lecture notes ([www.chrisbilder.com/multivariate/sections.html](http://www.chrisbilder.com/multivariate/sections.html)).

Example: MC simulation for the variance (MC\_sim\_var.R, sim\_results.csv)

The simulation study variables 1) Type of confidence interval, 2) Distribution, and 3) Sample size, are treated as qualitative in the settings (sample size is ordinal). This allows us to plot the true confidence level (coverage) vs. a variable (confidence interval type) while CONDITIONING on other variables (distribution and/or sample size). It is important to arrange panels in the plots so that you can see trend (if present).

Below is my code used to read in the simulation study data:

> set1<-read.csv(file ="C:\\chris\\sim\_results.csv")

> head(set1)

CI Coverage ExpLength NA. Distribution

1 Normal-based 0.7940000 16.33 0 Gamma

2 Asymptotic 0.6036217 7.98 3 Gamma

3 Basic 0.6440000 8.28 0 Gamma

4 Percentile 0.6300000 8.28 0 Gamma

5 BCa 0.6740000 9.47 0 Gamma

6 Studentized 0.9000000 128.81 0 Gamma

SampleSize

1 9

2 9

3 9

4 9

5 9

6 9

> levels(set1$CI)

[1] "Asymptotic" "Basic" "BCa"

[4] "Normal-based" "Percentile" "Studentized"

> levels(set1$Distribution)

[1] "Exponential" "Gamma" "Logistic" "Normal"

[5] "Uniform"

> table(set1$SampleSize)

9 20 50 100

30 30 30 30

Note that parameters for each distribution were selected so that Var(Y) = 2.1955812.

The lattice and ggplot2 packages are the two main packages available for trellis plots. I will give a brief demonstration of how to use the lattice package only. The ultimate goal is to obtain the following plot:



What can you conclude about the confidence interval procedures?

To obtain the above plot, let’s look at a simpler version first.

> library(lattice)

> dotplot(x = CI ~ Coverage | Distribution, data = set1,

groups = SampleSize, auto.key = TRUE, xlab = "Estimated

true confidence level", layout = c(1,5), ylab =

"Confidence interval method")



This shows a lot of the default behavior for the dotplot() function. A key component is the “|” in the x argument for the function. The vertical line separates the main variables to be plotted in each panel and those that are conditioned upon in the plot.

Below is the code for the final plot constructed:

> #This is one way to obtain all of the sample sizes & put

into a vector where the elements are characters. A more

simple (but less general) way is to just manually enter

the sample size levels as plot.levels<-c("9", "20",

"50", "100")

> plot.levels<-levels(factor(set1$SampleSize))

> dotplot(x = CI ~ Coverage | Distribution, data = set1,

groups = SampleSize, main = "Confidence level

simulation results",

key = list(space = "right", points = list(pch

= 1:4, col = c("black", "red", "blue", "darkgreen")),

text = list(lab = plot.levels)),

panel = function(x, y) {

panel.grid(h = -1, v = 0, lty = "dotted", lwd = 1,

col="lightgray") # h = -1 aligns grid lines with

axis labels

panel.abline(v = 0.95, lty = "solid", lwd = 0.5)

panel.abline(v = c(0.925, 0.975), lty = "dotted", lwd

= 0.5)

panel.xyplot(x = x, y = y, col = c(rep("black", times

= 6), rep("red", times = 6), rep("blue", times =

6), rep("darkgreen", times = 6)), pch = c(rep(1,6),

rep(2,6), rep(3, 6), rep(4, 6)))

},

xlab = "Estimated true confidence level", layout =

c(1,5), ylab = "Confidence interval method")

The same type of code could be done for the estimated expected lengths as well. Below is the plot:



Below is the same plot, but with the x-axis restricted due to the very large in length studentized intervals:



What conclusions can you reach about the expected length of the intervals?

Which confidence interval overall is the best?

Additional notes

Below are some important additional items that can be helpful to know for MC simulation:

* When using a for loop, it is sometimes helpful to print the simulation number (1, 2, …, R) while in the loop. This can be done by simply adding a print(r) statement within the { }.
* The fitting of a model for each simulation is often done during MC simulations. Depending on the model and the numerical iterative procedure used to fit the model, the parameter estimates may not converge. This can cause an error message to be returned and force a premature end to a for loop. In order to not exit the for loop early, the try() function can be used when performing the model fit. For example, code such as

save.fit[r]<-try(model.fit.func(data))

could be used. The try() function will “try” to use model.fit.func(). If an error message is generated by mod.fit.func(), try() catches it and the for loop can move on to the next iteration (depending on subsequent code that would use save.fit).

* The *relative efficiency* of a statistical procedure can be investigated in a MC simulation as well. This measure provides the ratio of two variances for two estimators of interest:



for statistics T(1) and T(2). Notice that when the relative efficiency is calculated over R simulations, we obtain:



Of course, the 1/R part cancels, but I left it in there as a reminder in case the same number of variances are not available for each statistic (e.g., due to non-convergence of a model). When the estimator is biased, one should use mean square errors rather than variances in the relative efficiency equation.

* I strongly recommend saving important results from each simulation to a file outside of R. This will allow you to have a permanent record of the results – perhaps a test statistic or a confidence interval – for each r = 1, …, R outside of the R software package. The reasons for doing this include:
  + If R crashes before you can retrieve summaries for the simulations, you have a way to still obtain the desired summaries.
  + It allows you to view results simulation by simulation at any time to help explain “unusual” results that may occur.
  + You may decide later to summarize the set of simulations in a different way than you originally had intended.

Using functions like write.table() or write.csv() will allow you to write the results out to a file.

* When performing a large simulation study, do not tie up your own main computer with the simulations. You should run them on multiple other computers. For example, UNL students and faculty have practically unlimited computer resources available to them via the Holland Computing Center. At the very least, you could take advantage of the student computers in our department or stat-sim.unl.edu.
* When I have done very large simulation studies in the past, I have used R (or SAS in the more distance past) to send me an e-mail or tweet when a set of simulations are complete. This helps when you are using multiple computers so that you do not need to log on to these computers to check yourself.
* A large simulation study is essentially a factorial experiment. For example, probability distribution of the data, sample size, testing procedure, …, could all be factors with individual treatment levels (e.g., n = 100, 500, 1000 for sample size). One potential way to deal with running time issues is to use appropriate methods learned in a design of experiments class! I have even seen the use of response surface methodology to examine simulation results. In this case, there may be multiple sets of simulations at the same factor levels to obtain “replicates” as you would in a normal design of experiment setting.
* My PhD major professor gave me the following advice about running MC simulations:

If you obtain unexpected results, there is a very good chance that it was a programming error. Go back and verify the code works correctly.

One way to check if your code is correct is to run MC simulations using a sample size “near” infinity. Very often, due to your own or other people’s research, you know the asymptotic outcomes. Thus, if you use a “very large” sample size for your set of simulations, you have a way to check if your program matches the mathematical derivations.

* There may be times when you would like to compare two or more functions with respect to how long they would take. The rbenchmark package provides a convenient way to make this comparison through its benchmark() function. While not really of interest for the MC simulation example involving the variance, I have included a short contrived example of its use in my program corresponding to this example.