Statistical Simulation – An Introduction

James H. Steiger

Department of Psychology and Human Development
Vanderbilt University

Multilevel Regression Modeling, 2009

Statistical Simulation – An Introduction

1. Introduction
    - When We Don't Need Simulation
    - Why We Often Need Simulation
    - Basic Ways We Employ Simulation
2. Confidence Interval Estimation
    - The Confidence Interval Concept
    - Simple Interval for a Proportion
    - Wilson's Interval for a Proportion

When We Don't Need Simulation

As we have already seen, many situations in statistical inference are easily handled by asymptotic normal theory. The parameters under consideration have estimates that are either unbiased or very close to being so, and formulas for the standard errors allow us to construct confidence intervals around these parameter estimates. If parameter estimate has a distribution that is reasonably close to its asymptotic normality at the sample size we are using, then the confidence interval should perform well in the long run.

Why We Often Need Simulation I

However, many situations, unfortunately, are not so simple. For example:

1. The aymptotic distribution might be known, but convergence to normality might be painfully slow
2. We may be interested in some complex function of the parameters, and we haven't got the statistical expertise to derive even an asymptotic approximation to the distribution of this function.

Why We Often Need Simulation II

In situations like this, we often have a reasonable candidate for the distribution of the basic data generation process, while at the same time we cannot fathom the distribution of the quantity we are interested in, because that quantity is a very complex function of the data. In such cases, we may be able to benefit substantially from the use of statistical simulation.

Simulation in Statistical Inference I I

There are several ways that statistical simulation is commonly employed:

Generation of confidence intervals by bootstrapping. In this approach, the sampling distribution of the parameter estimate $\hat{\theta}$ is simulated by sampling, over and over, from the current data, and (re-)computing parameter estimates $\hat{\theta}^*$ from each "bootstrapped" sample. The variability shown by the many $\hat{\theta}^*$ values gives us a hint about the variability of the one estimate $\hat{\theta}$ we got from our data.

Simulation in Statistical Inference II I

Monte Carlo investigations of the performance of statistical procedures. In this approach, the data generation model and the model parameters are specified, along with a sample size. Data are generated according to the model. The statistical procedure is applied to the data. This process is repeated many times, and records are kept, allowing us to examine how the statistical procedure performs at recovering the (known) true parameter values.

Simulation in Statistical Inference III I

Generation of estimated posterior distributions. In the Bayesian framework, we enter the analysis process with a "prior distribution" of the parameter, and emerge from the analysis process with a "posterior distribution" that reflects our knowledge after viewing the data. When we see a $\hat{\theta}$, we have to remember that it is a point estimate. After seeing it, we would be foolish to assume that $\theta = \hat{\theta}$.

Conventional Confidence Interval Estimation

When we think about confidence interval estimation, it is often in the context of the mechanical procedure we employ when normal theory pertains. That is, we take a parameter estimate and add a fixed distance around it, approximately ± 2 standard errors.

There is a more general way of thinking about confidence interval estimation, and that is, the confidence interval is a range of values of the parameter for which the data cannot reject the parameter.

Conventional Confidence Interval Estimation

For example, consider the traditional confidence interval for the sample mean when σ is known. Suppose we know that $\sigma = 15$ and $N = 25$ and we observe a sample mean of $\overline{X}_\bullet = 105$. Suppose we ask the question, what value of μ is far enough away from 105 in the positive direction so that the current data would barely reject it? We find that this value of μ is the one that barely produces a Z-statistic of -1.96.

We can solve for this value of μ, and it is:

$$-1.96 = \frac{\overline{X}_\bullet - \mu}{\sigma/\sqrt{N}} = \frac{105 - \mu}{3} \tag{1}$$

Rearranging, we get $\mu = 110.88$.

Conventional Confidence Interval Estimation

Of course, we are accustomed to obtaining the 110.88 from a slightly different and more mechanical approach.

The point is, one notion of a confidence interval is that it is a range of points that includes all values of the parameter that would not be rejected by the data. This notion was advanced by E.B. Wilson in the early 1900's.

In many situations, the mechanical approach agrees with the "zone of acceptability" approach, but in some simple situations, the methods disagree.

As an example, Wilson described an alternative approach to obtaining a confidence interval on a simple proportion.

A Simple Interval for the Proportion

We can illustrate the traditional approach with a confidence interval for a single binomial sample proportion.

Example (Traditional Confidence Interval for a Population Proportion)

Suppose we obtain a sample proportion of $\hat{p} = 0.65$ based on a sample size of $N = 100$.

The estimated standard error of this proportion is $\sqrt{.65(1 - .65)/100} = 0.0477$.

The standard normal theory 95% confidence interval has endpoints given by $.65 \pm (1.96)(0.0477)$, so our confidence interval ranges from 0.5565 to 0.7435.

A Simple Interval for the Proportion

An R function to compute this interval takes only a couple of lines:

```
> simple.interval <- function(phat,N,conf)
+ {
+ z <- qnorm(1-(1-conf)/2)
+ dist <- z * sqrt(phat*(1-phat)/N)
+ lower = phat - dist
+ upper = phat + dist
+ return(list(lower=lower,upper=upper))
+ }
> simple.interval(.65,100,.95)

$lower
[1] 0.5565157

$upper
[1] 0.7434843
```

Wilson's Interval

The approach in the preceding example ignores the fact that the standard error is estimated from the same data used to estimate the sample proportion. Wilson's approach asks, which values of p are barely far enough away from $\hat{p}$ so that $\hat{p}$ would reject them. These points are the endpoints of the confidence interval.

Wilson's Interval

The Wilson approach requires us to solve the equations.

$$z = \frac{\hat{p} - p}{\sqrt{p(1-p)/N}} \tag{2}$$

and

$$-z = \frac{\hat{p} - p}{\sqrt{p(1-p)/N}} \tag{3}$$

Be careful to note that the denominator has p, *not* $\hat{p}$.

Wilson's Interval

If we square both of the above equations, and simplify by defining $\theta = z^2/N$, we arrive at

$$(\hat{p} - p)^2 = \theta p(1 - p) \tag{4}$$

This can be rearranged into a quadratic equation in p, which we learned how to solve in high school algebra with a (long-forgotten, C.P.?) simple if messy formula. The solution can be expressed as

$$p = \frac{1}{1+\theta}\left(\hat{p} + \theta/2 \pm \sqrt{\hat{p}(1-\hat{p})\theta + \theta^2/4}\right) \tag{5}$$

Wilson's Interval

We can easily write an R function to implement this result.

```
> wilson.interval <- function(phat,N,conf)
+ {
+ z <- qnorm(1 - (1-conf)/2)
+ theta <- z^2 /N
+ mult <- 1/(1+theta)
+ dist <- sqrt(phat*(1-phat)*theta + theta^2 / 4)
+ upper = mult*(phat + theta/2 + dist)
+ lower = mult*(phat + theta/2 - dist)
+ return(list(lower=lower,upper=upper))
+ }
> wilson.interval(.65,100,.95)


$lower
[1] 0.5525444

$upper
[1] 0.7363575
```

Confidence Intervals through Simulation I

The methods discussed above both assume that the sample distribution of the proportion is normal. While the distribution is normal under a wide variety of circumstances, it can depart substantially from normality when N is small or when either p or $1 - p$ approaches 1. An alternative approach to assuming that the distribution of the estimate is normal is to simulate the distribution.

Confidence Intervals through Simulation II

This non-parametric approach involves:

1. Decide on a number of replications
2. For each replication
    1. Take a random sample of size N, with replacement, from the data
    2. Compute the statistic
    3. Save the results
3. When all the replications are complete, compute the .975 and .025 quantiles in the simulated distribution of estimates
4. These values are the endpoints of a 95% confidence interval

Applying the Simulation Approach I

When the data are binary, the simulation procedure discussed above amounts to sampling from the binomial distribution with p set equal to the current sample proportion $\hat{p}$.

(Note: Gelman & Hill sample from the normal distribution in one of their examples, but this is not necessary with R.) This involves much more computational effort than the methods discussed previously.

Applying the Simulation Approach II

```
> bootstrap.interval ← function(phat,N,conf,reps)
+ {
+   lower.p ← (1-conf)/2
+   upper.p ← 1 - lower.p
+   lower ← rep(NA,length(phat))
+   upper ← rep(NA,length(phat))
+   for(i in 1:length(phat))
+   {
+   x ← rbinom(reps,N,phat[i])
+   lower[i] ← quantile(x,lower.p,names=F)/N
+   upper[i] ← quantile(x,upper.p,names=F)/N
+   }
+   return(list(lower=lower,upper=upper))
+ }
> bootstrap.interval(.95,30,.95,1000)

$lower
[1] 0.8666667

$upper
[1] 1
```

The approach just illustrated is called "bootstrapping by the percentile method." Note that it will produce different results when starting from a different seed, since random draws are involved.

Comparing the Intervals I

In many situations, the intervals will yield results very close to each other.

However, suppose $\hat{p} = .95$ and $N = 30$. Then

```
> simple.interval(.95,30,.95)

$lower
[1] 0.8720108

$upper
[1] 1.027989

> bootstrap.interval(.95,30,.95,1000)

$lower
[1] 0.8666667

$upper
[1] 1
```

Comparing the Intervals II

On the other hand,

```
> wilson.interval(.95,30,.95)

$lower
[1] 0.8094698

$upper
[1] 0.9883682
```

Now we see that there is a substantial difference between the results. The question is, which confidence interval actually performs better?

Comparing the Intervals – Exact Calculation

There are a number of ways of characterizing the performance of confidence intervals. For example, we can examine how close the actual coverage probability is to the nominal value. In this case, we can, rather easily, compute the exact coverage probabilities for each interval, because R allows us to compute exact probabilities from the binomial distribution, and N is small. Therefore, we can

1. Compute every possible value of $\hat{p}$
2. Determine the confidence interval for that value
3. See whether the confidence interval contains the true value of p
4. Add up the probabilities for intervals that do cover p

An R Function for Exact Calculation

In the R function below, we compute these coverage probabilities for a given N,p, and confidence level. (We ignore the fact that the bootstrapped interval can vary according to the number of replications and the random seed value.)

```
> actual.coverage.probability ← function(N,p,conf)
+ {
+ x ← 0:N
+ phat ← x/N
+ probs ← dbinom(x,N,p)
+ wilson ← wilson.interval(phat,N,conf)
+ simple ← simple.interval(phat,N,conf)
+ bootstrap ← bootstrap.interval(phat,N,conf,1000)
+ s←0
+ w←0
+ b←0
+ results ← new.env()
+ for(i in 1:N+1) if((simple$lower[i] < p)&(simple$upper[i] >p)) s←s+probs[i]
+ for(i in 1:N+1) if((wilson$lower[i] < p)&(wilson$upper[i] >p)) w←w+probs[i]
+ for(i in 1:N+1) if((bootstrap$lower[i] < p)&(bootstrap$upper[i] >p)) b←b+probs[i]
+ return(list(simple.coverage=s,wilson.coverage=w,bootstrap.coverage=b))
+ }
> actual.coverage.probability(30,.95,.95)
```

```
$simple.coverage
[1] 0.7820788

$wilson.coverage
[1] 0.9392284

$bootstrap.coverage
[1] 0.7820788
```

Note that the Wilson interval is close to the nominal coverage level, while the traditional and bootstrap intervals perform rather poorly.

Comparing the Intervals – Monte Carlo Approach

Suppose that we had not realized that the exact probabilities were available to us. We could still get an excellent approximation of the exact probabilities by Monte Carlo simulation.

Monte Carlo simulation works as follows:

1. Choose your parameters
2. Choose a number of replications
3. For each replication:
    1. Generate data according to the model and parameters
    2. Calculate the test statistic or confidence interval
    3. Keep track of performance, e.g., whether the test statistic rejects, or whether the confidence interval includes the true parameter
4. Display the results

Monte Carlo Simulation – An Example Function

In the function below, we simulate 10,000 Monte Carlo replications

```
> estimate.coverage.probability←function(N,p,conf,reps,seed.value=12345)
+ {
+ ## Set seed, create empty matrices to hold results
+ set.seed(seed.value)
+ results ← new.env()
+ coverage.wilson←0
+ coverage.simple←0
+ coverage.bootstrap←0
+ ## Loop through the Monte Carlo replications
+ for(i in 1:reps)
+   {
+ ## create the simulated proportion
+    phat ← rbinom(1,N,p)/N
+ ## calculate the intervals
+    wilson ← wilson.interval(phat,N,conf)
+    simple ← simple.interval(phat,N,conf)
+    bootstrap ← bootstrap.interval(phat,N,conf,1000)
+ ## test for coverage, and update the count if successful
+    if((simple$lower < p)&(simple$upper >p))
+       coverage.simple ← coverage.simple + 1
+    if((wilson$lower < p)&(wilson$upper >p))
+       coverage.wilson ← coverage.wilson + 1
+    if((bootstrap$lower < p)&(bootstrap$upper >p))
+       coverage.bootstrap ← coverage.bootstrap + 1
+
+   }
+ ## convert results from count to probability
+ results$simple ← coverage.simple/reps
+ results$wilson ← coverage.wilson/reps
+ results$bootstrap ←coverage.bootstrap/reps
+ ## return as a named list
+ return(as.list(results))
+ }
```

Some Output

```
> estimate.coverage.probability(30,.95,.95,10000)

$bootstrap
[1] 0.7853

$wilson
[1] 0.9381

$simple
[1] 0.788
```

Monte Carlo Simulation across Parameter Values

To get a better idea of the overall performance of the two interval estimation methods when $N = 30$, we might examine coverage rates as a function of p. With our functions written, we are all set to go. We simply set up a vector of p values, and store the results as we go.

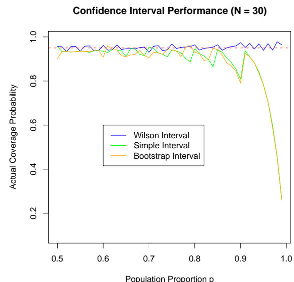
Here is some code:

```
> ## set up empty vectors to hold 50 cases
> p ← matrix(NA,50,1)
> wilson ← matrix(NA,50,1)
> simple ← matrix(NA,50,1)
> bootstrap ← matrix(NA,50,1)
> ## step from .50 to .99, saving results as we go
> for (i in 1:50)
+ {
+  p[i]←(49+i)/100
+  res ← actual.coverage.probability(30,p[i],.95)
+  wilson[i] ← res$wilson.coverage
+  simple[i] ← res$simple.coverage
+  bootstrap[i] ← res$bootstrap.coverage
+ }
```

Monte Carlo Simulation – An Example

Below, we graph the results, presenting coverage probability as a function of p. The performance advantage of the Wilson interval is obvious.

```
> plot(p,wilson,type="l",col="blue",
+   ylim=c(.1,.99),xlab="Population Proportion p",
+   ylab="Actual Coverage Probability",main="Confidence Interval Performance (N = 30)")
> lines(p,simple,col="green")
> lines(p,bootstrap,col="orange")
> abline(.95,0,lty=2,col="red")
> legend(.6,.6,c("Wilson Interval","Simple Interval","Bootstrap Interval"),
+     col=c("blue","green","orange"),lty=c(1,1,1))
```



Why Do We Need Simulation?

The preceding examples demonstrate how we can use simulation in a very simple situation, for two essentially different purposes:

1. To help construct confidence intervals after having observed data
2. To examine the performance of a test statistic or interval estimation procedure in situations where the parameters are "known"

Gelman & Hill refer to the first situation as *predictive simulation*, and the second as *fake data simulation*.

The situations we examined, we didn't actually need simulation – better procedures were available.

Why Do We Need Simulation?

Simulation is widely used because, in many situations, we don't have a quality procedure like the Wilson interval. Even when procedures might exist somewhere in the statistical literature, we might not be aware of them, or be able to make the appropriate connection. In such situations, simulation can save huge amounts of time while still providing very accurate answers to our questions.

Simulating Replicated Data

Gelman & Hill present a library function, `sim`, for simulating, quickly and efficiently, a posterior distribution of parameter values from a `lm` or `glm` fit object obtained from predicting y from k predictors in X. The steps in their procedure are described on page 143.

1. Compute $\hat{\beta}$, $V_\beta = (X'X)^{-1}$, and the estimated residual variance $\hat{\sigma}^2$ using standard regression approaches.
2. Create `n.sims` random simulations of the coefficient vector β and residual standard deviation σ based on normal theory. That is, for each simulation, create
    1. $\sigma^2 = \hat{\sigma}^2/(\chi^2_{N-k}/(N-k))$
    2. Given the random draw of σ^2, simulate β from a multivariate normal distribution with mean $\hat{\beta}$ and covariance matrix $\sigma^2 V_\beta$
3. These distributions represent posterior distributions for the parameters, representing our uncertainty about them. The assumption is that the prior distribution is uninformative, i.e., you have essentially no knowledge of the parameters prior to gathering data.

Well-Switching

In Gelman & Hill Chapter 5, pages 86–88, an example was introduced involving well-switching behavior in Bangladesh. The first model predicted the binary well-switching variable from a single predictor, distance from the nearest well. Figures are potentially confusing, as one involves coefficients obtained from fitting distance in 100 meter units, the other portrays the fit as a function of distance "in meters." We begin by attaching the wells data.

```
> wells ← read.table("wells.dat", header = TRUE )
> attach(wells)
> dist100 ← dist
```

Next, we fit a logistic regression, using only the distance in meters to the nearest known safe well. We expect, of course, that the probability of switching will be inversely related to the distance.

```
> fit.1 ← glm(switch ~ dist , family = binomial (link = "logit"))
> display (fit.1 , digits = 4)
```

```
glm(formula = switch ~ dist, family = binomial(link = "logit"))
            coef.est coef.se
(Intercept)  0.6060   0.0603
dist        -0.0062   0.0010
---
  n = 3020, k = 2
  residual deviance = 4076.2, null deviance = 4118.1 (difference = 41.9)
```

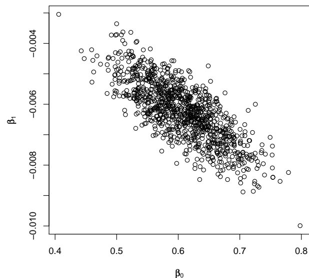
Well Switching II

Next, we simulate the posterior distribution of β_0 and β_1:

```
> sim.1 ← sim(fit.1,n.sims=1000)
```

We can plot the posterior bivariate distribution of the coefficients:

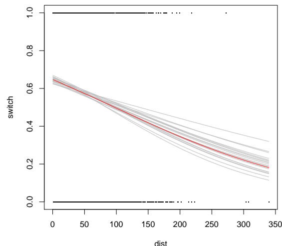
```
> plot(sim.1$coef[,1],sim.1$coef[,2],xlab=expression(beta[0]),
+   ylab=expression(beta[1]))
```



Picturing Uncertainty

Figure 7.6b demonstrates how you can plot uncertainty in the prediction equation, by plotting curves corresponding to values from the simulated posterior distribution. Each pair of values corresponds to a plot line. Gelman & Hill plot 20 lines. I've put the line from the original data in red.

```
> plot(dist,switch,pch=".")
> for(s in 1:20)
+ {
+  curve(invlogit(sim.1$coef[s,1] + sim.1$coef[s,2]*x),
+     col="gray",add=TRUE)
+ }
> curve(invlogit(fit.1$coef[1] + fit.1$coef[2]*x),col="red",add=TRUE)
```



Simulating Predictive Uncertainty

On page 149, Gelman & Hill discuss simulating the uncertainty that occurs when predicting new outcomes.

In this example, they start with the supposition that there is a $\tilde{n} \times 2$ matrix $\tilde{X}$ representing the values of $\tilde{n}$ new households on the predictor variable `dist`. This is what they do:

1. For each simulation, they predict the probability of switching using the predictor values in $\tilde{X}$ and the β values from the simulation
2. Then, for each simulation, they sample a binary (0,1) random variable with probability equal to the probability of switching from step (1).
3. So, after 1000 simulations, each new household has 1000 (0,1) results, each based on one value from the (simulated) posterior distribution of β values
4. I am assuming that the proportion of 1's in the resulting columns is taken as an estimate of the switching probability that reflects our posterior uncertainty in the actual slope and intercept values from the original data
5. This final matrix also reflects the kinds of (very different) actual result patterns that might emerge!

Try as I might, I cannot find Figure 7.7. Can anyone help me?

Simulating Predictive Uncertainty – An Example

Here is some code:

```
> n.sims <- 1000
> X.tilde <- matrix(c(1,1,1,1,1,1,1,1,1,1,120,45,109,
> n.tilde <- nrow(X.tilde)
> y.tilde <- array(NA,c(n.sims,n.tilde))
> for (s in 1:n.sims){
+   p.tilde <- invlogit(X.tilde %*% sim.1$coef[s,])
+   y.tilde[s,] <- rbinom(n.tilde,1,p.tilde)
+   }
```

Simulating Predictive Uncertainty – Sample Output

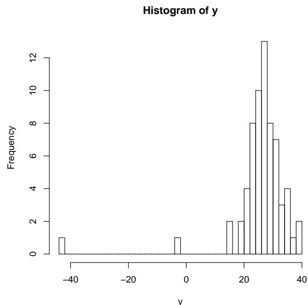
```
> y.tilde[1:20,]

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    1    1    0    1    0    0    1    0     0
 [2,]    1    1    1    0    1    0    1    1    0     1
 [3,]    0    0    0    1    1    0    1    0    0     1
 [4,]    1    1    1    0    1    0    1    0    0     1
 [5,]    1    1    0    1    0    0    1    1    1     0
 [6,]    1    1    0    1    1    0    0    0    1     0
 [7,]    0    1    1    0    1    0    1    0    1     1
 [8,]    1    1    0    0    1    0    0    1    0     0
 [9,]    0    1    0    1    0    0    1    1    0     1
[10,]    1    0    1    0    1    0    1    1    1     0
[11,]    1    0    0    1    1    1    1    1    1     1
[12,]    1    1    0    0    1    0    1    1    0     1
[13,]    0    1    0    1    0    1    1    0    1     1
[14,]    0    0    0    0    1    0    0    0    1     1
[15,]    0    1    1    0    1    0    1    1    1     1
[16,]    0    0    0    1    0    0    1    0    0     1
[17,]    0    1    0    1    0    0    1    1    0     0
[18,]    0    0    0    0    0    0    1    0    0     0
[19,]    0    0    1    1    1    0    0    0    1     1
[20,]    0    0    1    0    1    0    0    0    0     1
```

The Newcombe Light Data

As Gelman & Hill point out on page 159, a most fundamental way to check fit of all aspects of a model is to compare replicated data sets to the actual data. This example involves Newcombe's replicated measurements of estimated speed of light.

```
> y ← scan ("lightspeed.dat", skip=4)
> # plot the data
> hist (y,breaks=40)
```



The Newcombe Light Data – Simple Normal Fit

```
> # fit the normal model
> #(i.e., regression with no predictors)
> lm.light ← lm (y ~ 1)
> display (lm.light)

lm(formula = y ~ 1)
            coef.est coef.se
(Intercept) 26.21    1.32
---
n = 66, k = 1
residual sd = 10.75, R-Squared = 0.00

> n ← length (y)
```

The Newcombe Light Data – Replications

```
> n.sims ← 1000
> sim.light ← sim (lm.light, n.sims)
> y.rep ← array (NA, c(n.sims, n))
> for (s in 1:n.sims){
+   y.rep[s,] ← rnorm (1, sim.light$coef[s], sim.ligh
+ }
> # gather the minimum values from each sample
>
> test ← function (y){
+   min (y)
+ }
> test.rep ← rep (NA, n.sims)
> for (s in 1:n.sims){
+   test.rep[s] ← test (y.rep[s,])
+ }
```

The Newcombe Light Data – Replications

```
> # plot the histogram of test statistics of replications and of actual data
>
> hist (test.rep, xlim=range (test(y), test.rep))
> lines (rep (test(y), 2), c(0,n),col="red")
```