Simulation and Bootstrap Methods

Instructor: Sharandeep Singh Pandher*

1 Introduction

The simulation and bootstrapping teachings are widely used to enhance your statistical inference and hypothesis testing with limited data in the context of generalized linear model. Firstly, **Simulation** is a method of creating **artificial** data that mimics the characteristics of the real data. You can use simulation to test different scenarios, assumptions, or hypotheses without relying on the actual data. For example, you can simulate how a new product would perform in the market based on historical data and some assumptions about customer behavior while **Bootstrapping** is a special type of simulation that uses the existing data to generate new data. Bootstrapping involves **resampling** the data **with replacement**, meaning that each observation can be selected more than once. By doing this, you can create **multiple samples** of the same size as the original data, but with different combinations of observations. The benefits and limitations of simulation and bootstrapping are below:

- **Benefits**: These techniques do not require strong assumptions about the data distribution or the test statistic, and they can handle complex models and non-standard situations.They can also provide more accurate and reliable estimates and confidence intervals

- **Limitations**: They can be computationally intensive and time-consuming, sensitive to the choice of parameters and resampling methods, and difficult to interpret and communicate.

In this Chapter, we will study simulation of generalized linear models when response is binary and count. Firstly we need to give attention on three key properties of statistical estimators. Then, we discuss common measures of estimator performance, looking specifically at **coefficients** and **standard errors**

- There are three basic properties of statistical estimators that researchers might want to evaluate using simulation or Monte Carlo simulations: (1) bias, (2) efficiency, and (3) consistency. We define these terms below, but in the simulation examples we primarily focus on bias and efficiency in measuring estimator performance.

- **Bias** is about getting the right answer on average, across many repeated samples. Formally, an estimator is unbiased if its expected value is equal to the true parameter ie an

*Address: Department of Mathematics and Statistics, University of Alberta, Edomoton, AB,T6G 2G1, Canada, e-mail: sharand1@ualberta.ca;

estimator $\hat{\theta}$ for the true parameter θ, is unbiased if $E(\hat{\theta}) = \theta$. does not mean that every single estimate is exactly equal to the truth. Instead, it means that many estimates computed from repeated samples will cluster around the true parameter; the average of the estimates across repeated samples should be (nearly) equal to the truth.

- **Efficiency** refers to the variance around an estimatethe amount of variability in the estimates that an estimator produces from sample to sample. An estimator is efficient if there is little variability across samples and inefficient if there are large fluctuations. Inefficiency alone is sometimes viewed as a less important problem than bias, perhaps because an unbiased but inefficient estimator will still get the right answer on average.

- **consistency** refers to an estimator getting closer and closer to the truth as the sample size increases. This is different from bias, which implies that an estimator averages to the truth even when the sample size is small. A consistent estimator may or may not be biased, but its main distinguishing feature is that its expected value gets closer and closer to the true value as the sample size increases. Consistency is also related to the notion of variance in an estimator as both are driven by sample size. Taken to the extreme, in a sample with infinite observations, a consistent estimator will be unbiased with a variance equal to zero; it will simply be a spike at the true parameter value.

- **Measuring Estimator Performance in r**: In a regression model context, it is common to equate bias with the estimates of the coefficients and efficiency with the estimates of their standard errors

    1. **Coefficients**: We describe two methods for evaluating coefficient estimates here: (a) absolute bias and (b) mean squared error. Both methods are similar, measuring the average distance between an estimate and its true value.
    (a)**Absolute bias**:One way to measure coefficient performance is with absolute bias (AB), which is sometimes called error in estimation. For the Estimator $\hat{\theta}$, the error in estimation can be computed as $\mid \hat{\theta} - \theta \mid$ or the distance between an estimator and its target parameter.
    (b) **Mean Squared Error (MSE):** The MSE of a point estimate is the expectation of its squared deviation from the truth. ie $MSE = E[(\hat{\theta} - \theta)^2]$.
    2. **Standard Errors**: Assessing whether a standard error is correct is a bit more difficult to conceptualize at first glance compared with evaluating coefficients. We have several options for measuring standard error performance using simulations. We describe two of them here: (a) the standard deviation method and (b) coverage probabilities. The main goal of each one is to evaluate how well a given method for computing standard errors produces standard errors that reflect the true variability of the coefficient estimates
    (a)**Standard Deviation**: With the standard errors saved, our first approach for assessing standard error performance is the standard deviation method. Because the simulation process mimics the repeated samples phenomenon, the amount of variability in the simulated coefficient estimates should reflect the true amount of variability from sample to sample.
    (b)**Coverage probability**: A coverage probability is the proportion of simulated

samples for which the estimated confidence interval includes the true parameter. In this way, computing a coverage probability is akin to assessing whether the method for computing confidence intervals (and, thus, standard errors) is living up to its definition. If whatever method we are using to compute a confidence interval is correct, we should observe a 95 percentage confidence interval derived from that method that includes the true parameter in 95 percentage of the simulated samples. If this number is less than 95 percentage our method of producing standard errors is computing estimates of those standard errors that are too small, on average. If it is larger than 95 percentage, we know that the standard errors our method calculates are too large, on average. Of course, we do not need to limit ourselves to 95 percentage confidence intervals to make use of the concept of a coverage probability. For example, we could select a 50 percentage confidence interval, which we would then expect to return a coverage probability of 0.50. In fact, focusing only on 95 percentage confidence intervals (or the even higher 99 percentage threshold) might make it harder to detect methods that are systematically overestimating them because there is not much room to be wrong between 95percentage and 100 percentage.

1.1 binary model

We start with binary models, where the dependent variable can take on two outcomes, usually coded 1 or 0. here we will discuss logit and probit models.

1.1.1 logit model

For the logit model, we simulate can be written as

$$P(y = 1) = logit^{-1}(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) \quad (1)$$

where $\beta_1, \beta_2, \beta_3$ are regression coefficients and x_1, x_2, x_3 are predictors. To compute the probability of observing 1 (p) with logit, we use the inverse logit function $logit^{-1}$.

$$\frac{exp(p)}{1 + exp(p)} \quad (2)$$

Using (2) in (1), we get

$$P(y = 1) = \frac{exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)}{1 + exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)} \quad (3)$$

- We can create a **inverse logit function** in **r** as

```
rm(list=ls())
library(nlme)
library(lattice)
library(MASS)
library(stats)
set.seed(112102)
```

```
###### create inverse logit function ####

inv.logit=function(p){
  return (exp(p)/(1+exp(p)))
}
```

- **Define true parameters and predictors**

```
################ Define Sample size, true parameters and indepndent variables####
n=400                 #### sample size
beta0=0.2             ### True value
beta1=-0.80           ### True value
beta2=1.1             ### True value
beta3= 0.55           ###True value
X1= runif(n, -1,1)    ### create a sample of n observations for predictor
X2= runif(n, -1,1)   #####create a sample of n observations for predictor
X3= runif(n, -1,1)   #### create a sample of n observations for predictor
```

- **To produce the y (dependent variable), we will use rbinom() and check y takes values either 0 or 1.**

```
y=rbinom(n, 1, inv.logit(beta0+beta1*x1+beta2*x2+beta3*x3))

####### check y takes values 1 or 0####
table(y)
y
  0   1
177 223
```

- **To evaluate the logit model as an estimators of our estimators, we define coverage probability function as below:**

```
######## Create coverage probabilty function ############
coverage= function(b,se,true, level=.95, df=inf){
  qtile=level+(1-level)/2
  lower.bound=b-qt(qtile,df=df)*se
  upper.bound=b+qt(qtile,df=df)*se
  true.in.ci=ifelse(true >=lower.bound & true<=upper.bound,1,0)
  cp=mean(true.in.ci)
  mc.lower.bound=cp-1.96*sqrt((cp*(1-cp))/length(b))
  mc.upper.bound=cp+1.96*sqrt((cp*(1-cp))/length(b))
  return(list(coverage.probability=cp,true.in.ci=true.in.ci,
   ci=cbind(lower.bound,upper.bound),mc.eb=c(mc.lower.bound,mc.upper.bound)))
}
```

- **To evaluate the logit model as an estimator of our parameters using glm()**

```
###############To evaluate the logit model as an estimator
of our parameters using glm()####

reps=1000

par.est.logit= matrix(NA,nrow =reps, ncol = 8)
for (i in 1: reps){
y=rbinom(n, 1, inv.logit(beta0+beta1*x1+beta2*x2+beta3*x3))
model=glm(y~x1+x2+x3, family=binomial(link=logit))
vcv=vcov(model)
par.est.logit[i,1]=model$coef[1]
par.est.logit[i,2]=model$coef[2]
par.est.logit[i,3]=model$coef[3]
par.est.logit[i,4]=model$coef[4]
par.est.logit[i,5]=sqrt(diag(vcv)[1])
par.est.logit[i,6]=sqrt(diag(vcv)[2])
par.est.logit[i,7]=sqrt(diag(vcv)[3])
par.est.logit[i,8]=sqrt(diag(vcv)[4])
}
```

- **Absolute Bias (AB)**

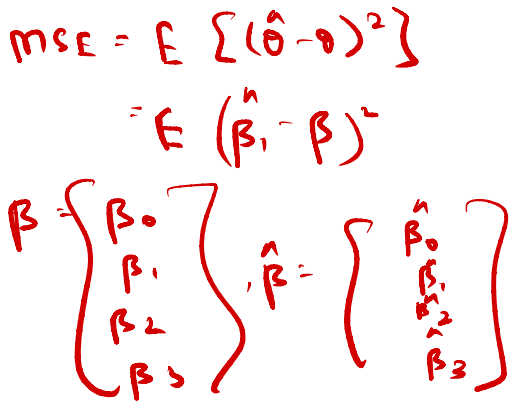
```
> ############## using Absolute Bias (AB) #############
> ab.beta0= mean(abs(par.est.logit[,1]-beta0))
> ab.beta0
[1] 0.09206365
> ab.beta1= mean(abs(par.est.logit[,2]-beta1))
> ab.beta1
[1] 0.1531469
> ab.beta2= mean(abs(par.est.logit[,3]-beta2))
> ab.beta2
[1] 0.1580717
> ab.beta3= mean(abs(par.est.logit[,4]-beta3))
> ab.beta3
[1] 0.1514403
```

Note:Our results show that the mean AB for our estimates of $\beta_0, \beta_1, \beta_2, \beta_3$ were about 0.0921, 0.1531, 0.1581 and 0.1514. These numbers do not have much meaning on their ownit is hard to say whether we have evidence of large or small AB in this case. Instead, measures like these take on more meaning when compared with another method of estimating these parameters. At that point, we could then conclude which method resulted in a lower AB in the parameter estimates. We will explore this idea further below:

- **MSE**: Again, the results are not extremely informative in isolation, but rather are more useful when comparing two or more methods of estimating the parameters. When making those comparisons, we would generally define the estimator that minimized AB or

MSE as performing better. In terms of which measure to use, the differences between them are relatively minor and both will nearly always give you the same pattern of results. MSE is more commonly used because it offers a balance between bias and efficiency. For now, we move to assessing the estimates of standard errors in simulated data.

```
> ################## Mean square error(MSE) for each parameter####
>
> mse.beta0= mean((par.est.logit[,1]-beta0)^2)
> mse.beta0
[1] 0.01298885
> mse.beta1= mean((par.est.logit[,2]-beta1)^2)
> mse.beta1
[1] 0.03664825
> mse.beta2= mean((par.est.logit[,3]-beta2)^2)
> mse.beta2
[1] 0.03957116
> mse.beta3= mean((par.est.logit[,4]-beta3)^2)
> mse.beta3
[1] 0.03579846
```



- **Evaluating the estimates of the standard errors using standard deviation method**

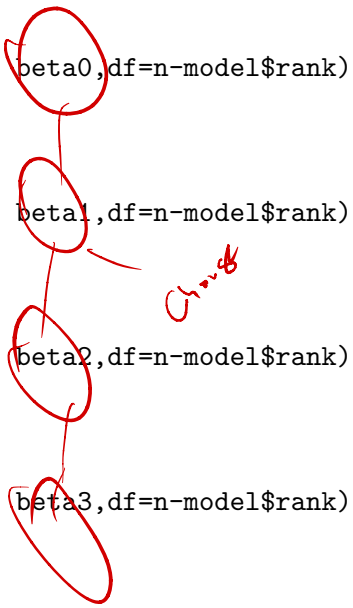
```
> ###### evaluating the estimates of the standard errors using
        standard  deviation method ####
>
> sd.beta0=sd(par.est.logit[,1])
> sd.beta0
[1] 0.1139955
>
> mean.se.beta0=sd(par.est.logit[,5])
>
> mean.se.beta0
[1] 0.002251542
>
> sd.beta1=sd(par.est.logit[,2])
>
> sd.beta1
[1] 0.1914085
>
> mean.se.beta1=sd(par.est.logit[,6])
>
> mean.se.beta1
[1] 0.005209691
>
> sd.beta2=sd(par.est.logit[,3])
>
```

```
> sd.beta2
[1] 0.1977693
>
> mean.se.beta2=sd(par.est.logit[,7])
>
> mean.se.beta2
[1] 0.006228417
>
> sd.beta3=sd(par.est.logit[,4])
>
> sd.beta3
[1] 0.189298
>
> mean.se.beta3=sd(par.est.logit[,8])
>
> mean.se.beta3
[1] 0.004890284
```

Note: To use the standard deviation approach, we need to check the standard errors calculated each time we estimated our glm model should be very close to the standard deviation of the 1,000 simulated coefficient estimates. Specifically, we can compare the mean of our 1,000 estimated standard errors with the observed standard deviation of our 1,000 coefficient estimates.
Notice that for parameters $\beta_0, \beta_1, \beta_2, \beta_3$ the numbers are not close. for instance, β_3, standard deviation is 0.189298 while mean standard error is 0.004890. This means that on average the standard error is not accurately reflecting the true coefficient variability.

- **Coverage Probabilty**

```
> cp.beta0=coverage(par.est.logit[,1],par.est.logit[,5], beta0,df=n-model$rank)
> cp.beta0$coverage.probability
[1] 0.941
>
> cp.beta1=coverage(par.est.logit[,2],par.est.logit[,6], beta1,df=n-model$rank)
> cp.beta1$coverage.probability
[1] 0.954
>
> cp.beta2=coverage(par.est.logit[,3],par.est.logit[,7], beta2,df=n-model$rank)
> cp.beta2$coverage.probability
[1] 0.953
>
> cp.beta3=coverage(par.est.logit[,4],par.est.logit[,8], beta3,df=n-model$rank)
> cp.beta3$coverage.probability
[1] 0.948
```



Note: The result is four coverage probabilities very near 95 percentage (0.941 for β_0, 0.954 for β_1,0.953 for β_2, and 0.948 β_3. These are unlikely to be exactly 0.95 because we only have a finite number of estimates, but they will converge toward 0.95 as the number of repetitions increases.

1.2 probit model

In the probit model, the probability of observing a 1 is assumed to come from the normal CDF instead of the inverse logit. To simulate a probit model, we simply replace the **inv.logit()** function with **pnorm()** function (recall that this computes the normal CDF). Then, **we must set the link function to probit in the glm() function** to estimate the probit model rather than the logit model.

```
> y=rbinom(n, 1, pnorm(beta0+beta1*x1+beta2*x2+beta3*x3))
>
> ####### check y takes values 1 or 0####
> table(y)
y
  0   1
164 236
> ################To evaluate the probit model as an estimator
                                          of our parameters using glm()####
>
> reps=1000
>
> par.est.probit= matrix(NA,nrow =reps, ncol = 8)
> for (i in 1: reps){
    y=rbinom(n, 1, pnorm(beta0+beta1*x1+beta2*x2+beta3*x3))
    model=glm(y~x1+x2+x3, family=binomial(link=probit))
    vcv=vcov(model)
    par.est.probit[i,1]=model$coef[1]
    par.est.probit[i,2]=model$coef[2]
    par.est.probit[i,3]=model$coef[3]
    par.est.probit[i,4]=model$coef[4]
    par.est.probit[i,5]=sqrt(diag(vcv)[1])
    par.est.probit[i,6]=sqrt(diag(vcv)[2])
    par.est.probit[i,7]=sqrt(diag(vcv)[3])
    par.est.probit[i,8]=sqrt(diag(vcv)[4])
  }
> ###### evaluating the estimates of the standard errors using
                                    standard  deviation method ####
>
> mean(par.est.probit[,1])
[1] 0.2029778
>
>
> mean(par.est.probit[,2])
[1] -0.807873
>
>
> mean(par.est.probit[,3])
[1] 1.118731
>
>
```

```
> mean(par.est.probit[,4])
[1] 0.5539505
>
> ############## Coverage Probabilty 95% CI##########
>
> cp.beta0=coverage(par.est.probit[,1],par.est.probit[,5], beta0,df=n-model$rank)
> cp.beta0$coverage.probability
[1] 0.943
>
> cp.beta1=coverage(par.est.probit[,2],par.est.probit[,6], beta1,df=n-model$rank)
> cp.beta1$coverage.probability
[1] 0.962
>
> cp.beta2=coverage(par.est.probit[,3],par.est.probit[,7], beta2,df=n-model$rank)
> cp.beta2$coverage.probability
[1] 0.962
>
> cp.beta3=coverage(par.est.probit[,4],par.est.probit[,8], beta3,df=n-model$rank)
> cp.beta3$coverage.probability
[1] 0.958
```

Note: we see that the means of the estimates are very close to the true values, and the coverage probabilities are near 0.95.

2 Count Models

We next move to a few examples of simulations with count models. Count models are used for dependent variables that take on positive integer values. Examples include the number of births at a hospital in a day or the number of patents awarded during some period. Count models allow the analyst to model such variables as a function of independent variables. A typical starting point for estimating the parameters of a count model is Poisson regression.

2.1 The Poisson Model

The poisson distribution has one parameter λ, which is both the mean and variance of the distribution. poisson regression links the systematic portion of the model to a poisson distribution through λ. It assumes the natural log of the dependent variable's expected value can be modeled by the sum of the independent variables multiplied by their coefficients. We can simulate such a poisson model using the **rpois()** function. To produce the dependent variable, we setλ to the exponentiated systematic component of the model (because the dependent variable cannot be negative), as in rpois($n, \lambda = exp(\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3)$). Everything else is exactly as we have seen before in binary model.

```
> reps=1000
>
> par.est.pois= matrix(NA,nrow =reps, ncol = 8)
```

```
> for (i in 1: reps){
    y=rpois(n,exp(beta0+beta1*x1+beta2*x2+beta3*x3))
    model=glm(y~x1+x2+x3, family="poisson")
    vcv=vcov(model)
    par.est.pois[i,1]=model$coef[1]
    par.est.pois[i,2]=model$coef[2]
    par.est.pois[i,3]=model$coef[3]
    par.est.pois[i,4]=model$coef[4]
    par.est.pois[i,5]=sqrt(diag(vcv)[1])
    par.est.pois[i,6]=sqrt(diag(vcv)[2])
    par.est.pois[i,7]=sqrt(diag(vcv)[3])
    par.est.pois[i,8]=sqrt(diag(vcv)[4])
  }
> ###### mean of the coefficients  ####
>
> mean(par.est.pois[,1])
[1] 0.1968005
> mean(par.est.pois[,2])
[1] -0.8012398
> mean(par.est.pois[,3])
[1] 1.099325
> mean(par.est.pois[,4])
[1] 0.5517635
>
>
> ############## Coverage Probabilty 95% CI##########
>
> cp.beta0=coverage(par.est.pois[,1],par.est.pois[,5], beta0,df=n-model$rank)
> cp.beta0$coverage.probability
[1] 0.948
>
> cp.beta1=coverage(par.est.pois[,2],par.est.pois[,6], beta1,df=n-model$rank)
> cp.beta1$coverage.probability
[1] 0.949
>
> cp.beta2=coverage(par.est.pois[,3],par.est.pois[,7], beta2,df=n-model$rank)
> cp.beta2$coverage.probability
[1] 0.958
>
> cp.beta3=coverage(par.est.pois[,4],par.est.pois[,8], beta3,df=n-model$rank)
> cp.beta3$coverage.probability
[1] 0.954
```

Note: The equivalence of the mean and variance in the Poisson distribution is an important **limitation** of the Poisson regression model because count processes in the social world often **unfold** such that the **variance is larger than the mean**, a phenomenon called **over-dispersion**. Several options exist to handle this problem, including the **negative binomial model**.

2.2 Comparing Poisson and Negative Binomial Models

The negative binomial (NB) is a more flexible approach to modeling count data because it can accommodate overdispersion through the estimation of a second parameter that allows the variance to be different from the mean. The main point to keep in mind is that because there is a separate parameter for the variance, the NB is not adversely affected by over-dispersion. To illustrate the impact of overdispersion, we simulate overdispersed data, estimate Poisson and NB models, then compare the results. To simulate overdispersed count data we use the **rnbinom() function**, which takes the arguments size,dispersion parameter, and mu(which is the mean). We set the dispersion parameter to 0.50. In the following, code we setup only β_0, β_1 and predictor x_1 for comparison.

```
reps=1000

par.est.pnb= matrix(NA,nrow =reps, ncol = 4)
for (i in 1: reps){
  y=rnbinom(n, size=0.5, mu= exp(beta0+beta1*X1))
  model.p=glm(y~x1, family="poisson")
  model.nb=glm.nb(y~x1)
  vcv.p=vcov(model.p)
  vcv.nb=vcov(model.nb)
  par.est.pnb[i,1]=model.p$coef[2]
  par.est.pnb[i,2]=model.nb$coef[2]
  par.est.pnb[i,3]=sqrt(diag(vcv.p)[2])
  par.est.pnb[i,4]=sqrt(diag(vcv.nb)[2])
  cat("completed", i, "of",reps,"\n")
}
> ###### mean of the coefficients  ####
>
> mean(par.est.pnb[,1]) #### Possion Estimates
[1] -0.7990593
> mean(par.est.pnb[,2])  #### NB Estimates
[1] -0.8023741
>
> ################### MSE####
> mean((par.est.pnb[,1])^2) #### Possion Estimates
[1] 0.6634644
> mean((par.est.pnb[,2])^2)  #### NB Estimates
[1] 0.667425
> ############## Coverage Probabilty 95% CI##########
>
> coverage(par.est.pnb[,1],par.est.pnb[,3], beta1,df=n-model.p$rank)$coverage.probability
                                                          #### Possion SE ####
[1] 0.706
>
> coverage(par.est.pnb[,2],par.est.pnb[,4], beta1,df=n-model.nb$rank)$coverage.probability
                                                                #### NB SE ###
[1] 0.932
```

Note: (Faraway, 2006) introduced that the Poisson model's coefficient estimates are consistent if overdispersion is the only problem, but that the standard errors are too small. We can check this by computing the means of the simulated estimates, MSE, and coverage probabilities. The results show that both estimators produced coefficient estimates with means near the true value of 0.80. Additionally, MSE is very similar for both. However, the coverage probabilities show a big difference. The Poisson standard errors produce a coverage probability of 0.706 while the NB coverage probability is 0.932. Thus, when the data are overdispersed the Poisson model's assumptions produce estimates of coefficient variability (standard errors) that are too small.

3 Overview of the Bootstrap Methods

Bootstrap methods are widely used general approaches for statistical inference. They are computer-intensive resampling methods, and are very useful for some difficult problems, such as variance estimations for intractable estimators and statistical inference when parametric assumptions are in doubt or when parametric inference is highly complicated. For example, it is usually difficult to compute the variances of a sample median or a sample percentile or a sample correlation coefficient. In these cases, it is straightforward to use the bootstrap method to compute estimates of standard errors and confidence intervals of these estimators. Bootstrap methods are often easy to implement, though may be computationally intensive, and can be applied to a wide variety of problems. Therefore, bootstrap has become a very popular statistical tool in modern statistics.

The idea of a bootstrap method is usually to approximate a distribution by the empirical distribution of the observed data, implemented by repeatedly resampling from the observed dataset with replacement (with the same sample size as the observed dataset). For example, suppose that $(x_1, x_2, , x_n)$ is an observed dataset, and suppose that one wishes to estimate the variance of the sample median. A simple bootstrap method proceeds as follows. We can sample from this observed dataset with replacement. The resulting sample, denoted by$(x_1^*, x_2^*, ..., x_n^*)$, is called a bootstrap sample. Then, we compute the sample median of this bootstrap sample. Repeating this process B times (B is often large, say $B = 1000$), we obtain B median estimates from the B bootstrap samples. We then compute the sample variance of these B median estimates and obtain a bootstrap estimate of the variance of the sample median from the original dataset. The sampling distribution of these B estimates is an approximation to the true distribution of the sample median from the original dataset.

As another example, we know that the MLE of a parameter is asymptotically normally distributed. In practice, this asymptotic distribution is often used to construct approximate confidence intervals and hypothesis testing where the sample size is in fact finite. Since the sample size is finite in practice, we may want to know how close the distribution of the MLE is to normality, so that we can judge how reliable the approximate confidence intervals and testing results are. We can use a bootstrap method to check this, as illustrated as follow. Suppose that we fit a mixed effects model, such as an NLME model, to a longitudinal dataset (with sample size n) using the likelihood method, and we wish to check if the resulting MLEs of the parameters are approximately normal. A simple bootstrap method can be performed as follows:

- sample from the original dataset with replacement and obtain a bootstrap sample.

- fit the mixed effects model to the bootstrap sample using the likelihood method and obtain MLEs of the parameters.
- Repeating the procedure B times, one obtains B sets of parameter estimates (MLEs)

The sampling distribution of the B estimates of a parameter is an approximation to the true sampling distribution of the MLE of this parameter based on the original dataset. One can then, for example, obtain an approximate confidence interval from the bootstrap samples by taking the α and 1α (say, $\alpha = 0.05$) quantiles of the B estimates. A bootstrap estimate of the standard error of the parameter estimate is the sample standard error of the B estimates.
For a parametric bootstrap method, one would fit a parametric model and obtain bootstrap samples from the fitted parametric model. The estimates are again computed from the bootstrap samples. For more detailed discussions of Bootstrap methods, see Efron and Tibshirani (1993) and Davison and Hinkley (2006).