ST 502 Project 1

N Gawron, K. Settlemyre, Z. Abahana

Introduction

In statistical inference, we are interested in properties of an estimator to make inference. One of the major ways of conducting inference is observing the confidence interval (CI), which is the plausible values for the parameter of interest. In observing CI, we are confident that our parameter of interest is in the interval at given confidence level (say, 95%). CI's vary in performance in differing situations. In this project, we are interested assessing different methods to determine which competing CI to use. We are interested in comparing six different CI methods for $Y = exp(\lambda)$ population, and the parameter of interest for the CI is λ which is estimated by $1/\overline{Y}$. The methods assessed are: exact interval; large-sample normality based using CLT & delta method; raw percentile parametric/non-parametric bootstrap intervals; reflected percentile non-parametric bootstrap interval; and the bootstrap t-interval using non-parametric bootstrap.

Overall, we found that exact and CLT / Approximate intervals perform best when considering an appropriate distribution of any size. For the misspecified population , exact and CLT methods actually did better than our specified coverage rate. Non-parametric methods (that are not t-statistic based) were the only method to create an appropriate coverage rate given a large sample size. As sample size increased we noticed that bootstrap based methods (parametric and non parametric) intervals improved in performance (in terms of coverage rate) and approached the approved. Lengths for intervals with an incorrect population are roughly halved in comparison to their correct population ed counterparts.

Note that you can find all the code for this project at the Github Link Here!

CI Discussion

The project will use Monte Carlo simulation for the data generation process on exponential and related gamma distribution and examines the properties of the CI methods for a set of λ's over increasing sample sizes (n's). The performance aspects of the estimators are compared in terms of coverage rate, proportion of intervals missed, length of the interval and standard error of the average length of the interval.

The most analytic way to approach a confidence interval is to use an exact method, based on a quantity called a pivot. If we consider a random variable $\mathbf{X}$ with parameter λ the pivot $Q(X_1, \ldots, X_n, \lambda)$ is a function based on the sample data and the parameter itself. In essence, we need this pivot to be a function of our estimate and parameter of interest, with no other 'unknowns'. This pivot will also have a probability distribution associated with it. It is important for a pivot to have this distribution not rely on any unknown values.

As we mentioned above, it may be cumbersome to derive a pivot function from a statistic and a parameter of interest. It may be even more difficult to find a distribution for this pivot function that does not depend on our parameter of interest. If this analytic approach to finding a distribution cannot be done - we can use approximate distributions to help us. Briefly, the CLT allows us to approximate the average of an iid

sample of random variables in terms of the Normal for as sample size grows. Since our parameter of interest λ can be estimated by the mean of our sample data - we can somehow take advantage of this approximate distribution to help us create the bounds for our confidence interval.

These methods above are all well and good, but what if we are interested in CI of an estimate that does not involve the mean? For starter, the CLT approach we outlined will not help us.

In this case, the bootstrap is an all-purpose method for constructing approximate CI for any population quantity (without making assumptions about the population distribution). Bootstrapping is method that applies a random sampling with replacement. Bootstrap uses the empirical distribution function to approximate the CDF. The bootstrap CI is made by repeatedly taking random samples of size (n) from the empirical distribution function and computing the mean of each bootstrap sample. These bootstrap sample means approximate the distribution of the sample mean and can be used to find a CI.

The idea behind parametric bootstrapping is to take the initial data and assume our parameter estimates from the data are the true values of the population. What this means is we take the simulated data given, and compute our parameter estimates based on a method like MOM or MLE. We approximate the distribution using these assumed true values, and then take a random sample of size n from this distribution, finding parameter estimates from it. We repeat this process many, many times (let's say B times), taking samples of size n and finding estimates, and using these B samples to form approximations of the sampling distribution for our estimators, that being a bootstrap distribution. We can form confidence intervals from the bootstrapped distribution by taking the $(1 - \frac{\alpha}{2})$ and $\frac{\alpha}{2}$ quantiles. For parametric bootstrapping, we make assumptions that the underlying shape of the data is parametric, that it can be modeled.

Nonparametric bootstrapping is similar to parametric in the sense we are taking repreated resamplings. We take our initial data and take a resample of size n from it, with replacement, and find estimates of our parameters. Note the difference here - we are merely resampling from our provided (or in our case simulated) data to create a bootstrap sample. We did not make any assumptions about the underlying distribution we are working with. To continue, we repeat this process B times, taking resamples of size n with replacement and finding estimates, and then using these B resamples to form an approximate sampling distribution for our estimators. To form a confidence interval, we take the $(1 - \frac{\alpha}{2})$ and $\frac{\alpha}{2}$ quantiles of the bootstrapped distribution. For nonparametric bootstrapping, we make no assumption about the underlying distribution.

Reflected nonparametric bootstrapping percentile uses the nonparametric bootstrap on the difference between the value of interest and the estimator $\delta = \hat{\lambda} - \lambda$ when our interest is in the quantity of distribution of the differences. The distribution of this quantity is not known, therefore, we use nonparametric bootstrap to approximate it. In the case of nonparametric bootstrapping, we generate $B = 1000$ resamples of size n from observed data with replacement. For each resample, we estimate of $\hat{\lambda}$ and approximate the distribution of $\hat{\lambda} - \lambda$ using the bootstrap distirbution from our bootstrap samples of $\hat{\lambda} - \lambda$. We then take the $\alpha/2$ and $1 - \alpha/2$ quantiles of our distribution to form our interval. Why would we use this? Well if it was difficult to

analytically compute the distribution of we may want to use a bootstrap estimate. Otherwise, we would be able to use an exact method since $\hat{\lambda} - \lambda$ is a pivot.

Non-parametric bootstrap t-interval can be used to create a 't-type' statistic to use as a pivot, and approximate the quantile of the sampling distribution of

$$T = \frac{\hat{\lambda} - \lambda}{\hat{SE}(\hat{\lambda})}$$

The steps involve creating a nonparametric bootstrap resample and calculating for each resample as in the quantity

$$\delta^* = \frac{\hat{\lambda}^* - \lambda}{\hat{SE}(\hat{\lambda}^*)}$$

We use bootstrapped (B=100) resamples to approximate the value of T. The number of Bootstrap replications is shortened to 100 to save on computation time.

For each reseample we estimate $\hat{\lambda}$ and approximate the distirbution of $\hat{\lambda}$ by a bootstrap distribution $\hat{\lambda}^* - \lambda$. We then take the $\alpha/2$ and $1 - \alpha/2$ quantiles to gives us our $(1 - \alpha)100\%$ confidence interval.

The big idea about this interval is we actually need to create another estimate. This standard error in this interval is still a random quantity. This is why we now treat our bootstrapped estimates at the true value of our parameter - and conduct a second bootsrapped sample (of a much smaller size to save on computation time) to estiamte this value. The standard devious from this bootstrapped sample will be our estimate for the standard error in this t-test.

For how well our confidence intervals perform, we're interested in the length of the interval, the proportion of intervals that capture our parameter of interest, the proportion of parameters that miss above our parameter, the proportion of intervals that miss below our parameter, and the standard error of the average length of the interval. The way we construct our confidence interval, we need the standard error, which measures variability in the sampling distribution. This value contributes to the margin of error, which is half the length of the interval. The larger the standard error, the more variability, the larger our confidence interval. If we can reduce our standard error, we reduce our margin of error and get a more narrow interval, reducing uncertainty around the true value. Given repeated sampling, we might want to see the proportion of intervals that capture the true value. With a specified level of confidence, $(1 - \alpha)100$, we expect that approximately $(1-\alpha)100$ of those intervals capture the true value. Similarly, there should be approximately $\alpha * 100$ intervals that miss the true value.

We examine coverage rate as a way to determine how accurate our interval is. The greater the coverage rate - the more optimal our interval is for estimating a particular parameter. We note that 1 is the maximum value for coverage rate - this means all interval capture are desired interval.

On the other hand, we look at length in the opposite way. Smaller the better. If an interval on average is smaller this means there is less variability in our estimate of a certain parameter.

In some instances we may be interested in a misspesfied population. In the creation of these confidence intervals, we rely on a sample from a parent population. For example, an exact interval is created based on the assumption that Y_i comes from a sample of a certain parent population, in our case the exponential. We are interested in looking at a wrong population if we want to observe how robust our methods are to different data. This can help us answer a lot of questions. For example, Is the exact interval going to become an awful interval (in terms of coverage rate) if we do not give it data that is from an exponential distribution?

Simulation and Derivations

We will be setting up functions to create a singular confidence interval for one pair of n and λ values.

Our goal to create summary information and compare these intervals is to evaluate this function for the 12 combinations of settings for n and λ, with 1000 replications each. We can then compute our summary statistics:

- proportion of intervals capturing the true parameter, coverage rate
- proportion of intervals that missed the parameter from above
- proportion of intervals that missed the parameter from below
- Average length of the interval
- Standard error of the average length for the interval

Exact Interval

We begin by noting that $\hat{\lambda}_{MOM} = \frac{1}{\bar{Y}}$. The exact interval is derived by starting to create a pivot using our estimator and parameter λ.

We suppose $Y_i \sim exp(\lambda) = Ga(1, \lambda)$.

It follows from MGF arguments that the num of Y_i's yeilds:

$$\sum_{i=1}^{n} Y_i \sim Ga(n, \lambda) \Longrightarrow \bar{Y} \sim Ga(n, n\lambda)$$

We note by the definition of the MOM argument, and multiplying by λ we arrive at:

$$\lambda \overline{Y} = \lambda \frac{1}{\hat{\lambda}} \sim Ga\left(n, n\frac{\lambda}{\lambda}\right) = Ga\left(n, n\right)$$

Our exact pivot is $\frac{\lambda}{\hat{\lambda}}$.

We then create a probability statement:

$$1-\alpha = P\left(ga_{\alpha/2} < \frac{\lambda}{\hat{\lambda}} < ga_{1-\alpha/2}\right)$$

We isolate for λ by multiplying both bounds by $\hat{\lambda}$ to arrive at the exact $(1-\alpha)100\%$ CI for λ:

$$(\hat{\lambda}ga_{\alpha/2}\,,\ \hat{\lambda}ga_{1-\alpha/2})$$

We create a function to do this calculation below:

```
# a function for an exact distribution
exact_dist_ci<- function(B,n,lambda,alpha){
  #create an interval using derived form

  # creates a sample of size n with given lambda
  #computes point estimate for lambda
  lambda_hat <- 1/mean(rexp(n,lambda))

  #computes exact interval for bounds
  ub <- lambda_hat*qgamma(1-alpha/2, n, n)
  lb <-  lambda_hat*qgamma(alpha/2, n, n)

  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }
```

Now we let the gamma distribution be the true population. This is the idea of misspesification where we derive our sample data.

```
# a function for a exact dist with gamma dist
exact_dist_ci_gamma<- function(B,n,lambda,alpha){
  #create an interval using derived form
```

```
# creates a sample of size n with given lambda
# computes point estimate for lambda
# gamma(2,lambda)
lambda_hat <- 1/mean(rgamma(n,2,lambda))

#computes exact interval for bounds
ub <- lambda_hat*qgamma(1-alpha/2, n, n)
lb <-  lambda_hat*qgamma(alpha/2, n, n)

#returns the vector of lower and upperbound
bounds<-data.frame(low_bound=lb,up_bound=ub)

return(bounds)

}
```

Large Sample Normality Based

The idea behind this interval is to use the CLT to get an approximate distribution to model our parameter.

We let $Y_i \sim exp(\lambda)$ be an iid sample. If we assume $Var(Y_i) = \frac{1}{\lambda^2} < \infty$ then by the CLT:

$$\overline{Y} \overset{\circ}{\sim} N\left(\frac{1}{\lambda}, \frac{1/\lambda^2}{n}\right)$$

We now employ the Delta Method Normality. We let $g(\overline{Y}) = \frac{1}{\overline{Y}}$ we arrive at:

$$\hat{\lambda}_{MOM} = \frac{1}{\overline{Y}} \overset{\circ}{\sim} N\left(g\left(\frac{1}{\lambda}\right), g'\left(\frac{1}{\lambda}\right)^2 Var(\overline{Y})\right)$$

Note that the derivative $g'(1/\lambda)$ is: λ^2 and $Var(\overline{Y}) = 1/\lambda^2 n$. When substitute the values from above in the delta normality formula.

$$\hat{\lambda}_{MOM} = \frac{1}{\overline{Y}} \overset{\circ}{\sim} N\left(g\left(\frac{1}{\lambda}\right), g'\left(\frac{1}{\lambda}\right)^2 Var(\overline{Y})\right)$$

We arrive at the Approximate Distribution for $\hat{\lambda}$ as:

$$\hat{\lambda} = 1/\overline{Y} \overset{\circ}{\sim} N(\lambda, \lambda^2/n)$$

We can normalize to get an approximate distribution that converges to the standard normal.

$$\frac{\hat{\lambda}-\lambda}{\lambda/\sqrt{n}} \xrightarrow{\text{d}} N(0,1)$$

By Slutskys Thm the fact that $\hat{\lambda}$ is a consistent estimator.

$$\frac{\hat{\lambda}-\lambda}{\hat{\lambda}/\sqrt{n}} \xrightarrow{\text{d}} N(0,1)$$

We can now setup the probability distribution:

$$1-\alpha \approx P\left(-z_{\alpha/2} < \frac{\hat{\lambda}-\lambda}{\hat{\lambda}/\sqrt{n}} < z_{\alpha/2}\right)$$

After solving for λ in the probability statement. We arrive at:

$$\hat{\lambda} \pm z_{\alpha/2}\frac{\hat{\lambda}}{\sqrt{n}}$$

```r
# a function CLT int
CLT_int<- function(B,n,lambda,alpha){
  #create a sample of size n with lambda
  MainSample<-rexp(n,lambda)
  # creates a parametric boot using an estimate for lambda
  lambda_hat<-1/mean(MainSample)



  #creates the bounds of the interval
  ub<- lambda_hat+qnorm(alpha/2,lower.tail = F)*lambda_hat/sqrt(n)
  lb<- lambda_hat-qnorm(alpha/2,lower.tail = F)*lambda_hat/sqrt(n)




  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }
```

Here is a function used to create a singular interval based on the CLT delta normality method. The difference here is we create a sample size of size n for an interval

```
# a function fCLt int with gamma
CLT_int_gamma<- function(B,n,lambda,alpha){
  #create a sample of size n with lambda
  MainSample<-rgamma(n,2,lambda)
  # creates a parametric boot using an estimate for lambda
  lambda_hat<-1/mean(MainSample)



  #creates the bounds of the interval
  ub<- lambda_hat+qnorm(alpha/2,lower.tail = F)*lambda_hat/sqrt(n)
  lb<- lambda_hat-qnorm(alpha/2,lower.tail = F)*lambda_hat/sqrt(n)




  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }
```

Simulation

Here is a basic function. We note that the estimate of λ, $\hat{\lambda} = \frac{1}{\bar{Y}}$.

Parametric Bootstrap

```
# a function for a  parametric bootstrap for lambda hat
param_boots<- function(B,n,lambda,alpha){
  #create a sample of size n with lambda
  MainSample<-rexp(n,rate =lambda)
  # creates a parametric boot using an estimate for lambda
```

```
   # create B samples from a dist hence param.
  Bootstrap<- replicate(B,rexp(n,rate=1/mean(MainSample)))

  #compute statisitc for each of the B samples
  lambda_hat_j = 1 / colMeans(Bootstrap)

  #using these value as an approx dist
  #creat CIs

  ub<- unname(quantile(lambda_hat_j,1-alpha/2))
  lb<- unname(quantile(lambda_hat_j,alpha/2))

  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }



# a function for a  parametric bootstrap for lambda hat from gamma
param_boots_gamma<- function(B,n,lambda,alpha){
  #create a sample of size n with lambda
  MainSample<-rgamma(n,2,rate =lambda)
  # creates a parametric boot using an estimate for lambda

   # create B samples from a dist hence param.
  Bootstrap<- replicate(B,rexp(n,rate=1/mean(MainSample)))

  #compute statisitc for each of the B samples
  lambda_hat_j = 1 / colMeans(Bootstrap)

  #using these value as an approx dist
  #creat CIs
```

```
  ub<- unname(quantile(lambda_hat_j,1-alpha/2))
  lb<- unname(quantile(lambda_hat_j,alpha/2))

  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }
```

Non Parametric Bootstrap

```
# a function for a non parametric bootstrap for lambda hat
np_boots<- function(B,n,lambda,alpha){
  #create a sample of size n with lambda
  MainSample<-rexp(n,lambda)
  # create B samples
  Bootstrap<- replicate(B,sample(MainSample,replace = TRUE))

  #compute statisitc for each of the B samples
  lambda_hat_j = 1 / colMeans(Bootstrap)

  #using these value as an approx dist
  #creat CIs

  ub<- unname(quantile(lambda_hat_j,1-alpha/2))
  lb<- unname(quantile(lambda_hat_j,alpha/2))

  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }
```

```r
# for the misspesficed gamma(2,lambda)
# a function for a non parametric bootstrap for lambda hat
np_boots_gamma<- function(B,n,lambda,alpha){

  #create a sample of size n with lambda from gamma with alpha =2
  MainSample<-rgamma(n,2,lambda)

  # create B samples
  Bootstrap<- replicate(B,sample(MainSample,replace = TRUE))

  #compute statisitc for each of the B samples for lambda
  # note estimating
  lambda_hat_j = (1 / colMeans(Bootstrap))

  #using these value as an approx dist
  #creat CIs

  ub<- unname(quantile(lambda_hat_j,1-alpha/2))
  lb<- unname(quantile(lambda_hat_j,alpha/2))

  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

  }
```

Nested Bootstrap $s.e.(\hat{\lambda})$

```r
# t test stat via nested boot
np_boot_se_t<-function(B,n,lambda,alpha){

  MainSample<-rexp(n,lambda)
```

```
lambdahat<-1/mean(MainSample)


# create B samples
Bootstrap<- replicate(B,sample(MainSample,replace = TRUE))

#compute statisitc for each of the B samples
lambda_hat_j = 1 / colMeans(Bootstrap)



SE_lambda_j_hat<-sqrt((1/B)*sum((lambda_hat_j-mean(lambda_hat_j))^2))



#treating Boostrap as the truth
#gets us SE(\theta)
#looks at columns from the Bootstrap data
bootSE<-apply(X=Bootstrap,MARGIN=2,M=50,n,FUN=function(x,M,n){
  #treating Bootsrap data as our real data we resample here
  tempData<-replicate(M,sample(x=x,size=n,replace=TRUE))
  #find mean and lambda estimate for each data set
  tempMeans<-colMeans(tempData)
  lambda_hat_est<- 1 / tempMeans
  sd(lambda_hat_est)
})



#create t-stats
tStats<-(lambda_hat_j-lambdahat)/bootSE

#have to create std error for boot
SE_lambda_hat_j <-sd(lambda_hat_j)

#now create interval
#intervals
lb=lambdahat-unname(quantile(tStats,1-alpha/2))*SE_lambda_hat_j
ub=lambdahat-unname(quantile(tStats,alpha/2))*SE_lambda_hat_j
```

```
  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

}
```

```
# nested boot strap for t-stat
np_boot_se_t_gamma<-function(B,n,lambda,alpha){

  MainSample<-rgamma(n,2,lambda)

  lambdahat<-1/mean(MainSample)

  # create B samples
  Bootstrap<- replicate(B,sample(MainSample,replace = TRUE))

  #compute statisitc for each of the B samples
  lambda_hat_j = 1 / colMeans(Bootstrap)


  SE_lambda_j_hat<-sqrt((1/B)*sum((lambda_hat_j-mean(lambda_hat_j))^2))


  #treating Boostrap as the truth
  #gets us SE(\theta)
  #looks at columns from the Bootstrap data
  bootSE<-apply(X=Bootstrap,MARGIN=2,M=50,n,FUN=function(x,M,n){
    #treating Bootsrap data as our real data we resample here
    tempData<-replicate(M,sample(x=x,size=n,replace=TRUE))
    #find mean and lambda estimate for each data set
    tempMeans<-colMeans(tempData)
    lambda_hat_est<- 1 / tempMeans
    sd(lambda_hat_est)
  })
```

```
  #create t-stats
  tStats<-(lambda_hat_j-lambdahat)/bootSE

  #have to create std error for boot
  SE_lambda_hat_j <-sd(lambda_hat_j)

  #now create interval
  #intervals
  lb=lambdahat-unname(quantile(tStats,1-alpha/2))*SE_lambda_hat_j
  ub=lambdahat-unname(quantile(tStats,alpha/2))*SE_lambda_hat_j

  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

}
```

Reflected Interval

Here are functions to create the reflected interval under each population setting.

```
# reflected non param interval
reflected_int<-function(B,n,lambda,alpha){

  MainSample<-rexp(n,lambda)

  #observed lambda from sample
  lambda_obs<- 1/mean(MainSample)

  # create B samples
  Bootstrap<- replicate(B,sample(MainSample,replace = TRUE))

  #compute statisitc for each of the B samples
```

```
  lambda_hat_j = 1 / colMeans(Bootstrap)

  #gets bootsrap dist of theta_hat - theta
  lambda_hat_m_lambda<- lambda_hat_j - lambda_obs

  #now create interval
  #intervals
  lb=lambda_obs-unname(quantile(lambda_hat_m_lambda,1-alpha/2))
  ub=lambda_obs-unname(quantile(lambda_hat_m_lambda,alpha/2))
  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

}
```

```
# misspesficy gamma

reflected_int_gamma<-function(B,n,lambda,alpha){

  MainSample<-rgamma(n,2,lambda)

  #observed lambda from sample
  lambda_obs<- 1/mean(MainSample)

  # create B samples
  Bootstrap<- replicate(B,sample(MainSample,replace = TRUE))

  #compute statisitc for each of the B samples
  lambda_hat_j = 1 / colMeans(Bootstrap)

  #gets bootsrap dist of theta_hat - theta
  lambda_hat_m_lambda<- lambda_hat_j - lambda_obs

  #now create interval
  #intervals
```

```
  lb=lambda_obs-unname(quantile(lambda_hat_m_lambda,1-alpha/2))
  ub=lambda_obs-unname(quantile(lambda_hat_m_lambda,alpha/2))
  #returns the vector of lower and upperbound
  bounds<-data.frame(low_bound=lb,up_bound=ub)

  return(bounds)

}
```

Summary Information Functions

We create a huge ,possibly inefficient, function to create summary results after replication of 1000 intervals under each of the 12 settings.

We compute the summary statistics mentioned above in the previous section.

Note how we can easily compute the standard error of the average length.

Consider X_i to be an iid sample of n lengths.

It follows from the CLT that $\overline{X} \overset{\circ}{\sim} N(\overline{X}, \frac{Var(X)}{n})$

Recall the sample standard deviation can be used to approximate the square root of the variance.

So in summary, we simply take the sample standard deviation of the interval lengths per setting and divide by the square root of n.

```
Summarizer4Intervals<- function(n,all_lambdas,B_num,num_replications,interval_funct){

#Goal: creates an info data frame on the
# for summay information about an interval creation procedure

  # n - sample size vector
  # lambda - vector of possible population lambdas to test
  # B_num-  the number of bootstrap samples to take
  # num_replications - the number of intervals within each setting combination
                       # lambda and n to create
  # interval_funct - a string is the name of the function used to create a single confidence interval
```

```r
#This function creates an interval repeated (rep) number of times
#with sampel size n and lambda value lambdav
# given the method used to compute bounds
create_an_int_for_params<-function(n,lambdav,reps){
# creates a dataframe of reps number of intervals
# under the sample size n and lambda parameter  = lambda


  #initalized df for the number of repetitions
  int_npb_reps<-data.frame()


   #progress bar for for loop on repititions
    pb_1 = txtProgressBar(min = 1, max = length(1:reps), initial = 1)

  #for the repitions reps
  for(rep in 1:reps){

    #map applies a function that takes in lambda and sample size
    # to compute the interval estimates
    intervals<-(t(mapply(n,FUN=interval_funct ,B =B_num ,lambda = lambdav, alpha=0.05)))

    #joins the n vector with the interval estimates
    int_npb<-data.frame(matrix(do.call(rbind, intervals),ncol=2))%>%mutate(n=n)

    # updates colnames
    colnames(int_npb) = c("lb","ub","n")

    # binds the intervals from previous replications to one df
    int_npb_reps<-rbind(int_npb,int_npb_reps)
```

```
    # updates progress bar
     setTxtProgressBar(pb_1,rep)

  }

  #returns a df
  return(int_npb_reps)
}









#intializes dataframe
interval_df<-data.frame()



#progress bar for for loop on lambdas
pb = txtProgressBar(min = 0, max = length(all_lambdas), initial = 0)


#sweeping across lambdas
for (lambdav in all_lambdas){

  #creates many many intervals for spesfic lambda
  # and ranges across all sample sizes n
  # THIS TAKES A WHILE
  many_dfs_of_ints<-create_an_int_for_params(n,lambdav,num_replications)

  # creates some summary info like lengths and lambda v
  # creates a length of the interval variable by substracting bounds
  # creates an
  interval_df_temp<-many_dfs_of_ints%>%
```

```
      mutate(lambda=lambdav)%>%
      mutate(Length = ub-lb)%>%
      mutate(isLin=sign(ub-lambdav)!=sign(lb-lambdav))%>%
      mutate(missAbv=lambdav<lb)%>%
      mutate(missBel = lambdav>ub)


    # interval concatenated
    interval_df<-rbind(interval_df,interval_df_temp)
    #progress bar
    setTxtProgressBar(pb,lambdav)

  }




#gives us the summary length
# and the propprotion of captured lengths
# computes the standard error of the length
Means_df<-interval_df%>%
  group_by(lambda,n)%>%
  summarise(mean_Len=mean(Length),prop_captured = mean(isLin),sd_Len=sd(Length))%>%
  mutate(SeAvgLen = sd_Len/sqrt(num_replications))%>%
  select(-sd_Len)

#interval_df%>%filter(isLin==FALSE)%>%group_by(n,lambda)%>%summarise(mean(missAbv))
All_info_df<-interval_df%>%
  filter(isLin==FALSE)%>%
  group_by(n,lambda)%>%
  summarise(prop_above=mean(missAbv),prop_below=mean(missBel))%>%
  merge(Means_df,by=c("n","lambda"))
#interval_df%>%group_by(n,lambda)%>%summarise_at(vars(-group_cols()), mean)
All_info_df<-All_info_df%>%mutate(n=as.factor(n),lambda=as.factor(lambda))
```

```
    return(All_info_df)


}
```

We create a similar function for the misspecification of our population distribution. This function will take in an interval create scheme that uses $Ga(2, \lambda)$. The differences in the function are we look for the interval to capture $\frac{\lambda}{2}$.

```
#####################################################

# this function is for the gamma misspesficiation

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



Summarizer4Intervals_gamma<-function(n,all_lambdas,B_num,num_replications,interval_funct){

  # takes in a vector of sample sizes
  #



  #initalize empty df
  interval_df<-data.frame()

  #This function creates an interval rep number of times
  #with sampel size n and lambda value lambdav
  # given the method used to compute bounds
  create_an_int_for_params<-function(n,lambdav,reps){

    #initalized df for the number of repetitions
    int_npb_reps<-data.frame()

    #progress bar for for loop on lambdas
```

```
  pb_1 = txtProgressBar(min = 1, max = length(1:reps), initial = 1)
  #for the repitions reps
  for(rep in 1:reps){

    #map applies a function that takes in lambda and sample size
    # to compute the interval estimates
    intervals<-(t(mapply(n,FUN=interval_funct ,B =B_num ,lambda = lambdav, alpha=0.05)))

    #joins the n vector with the interval estimates
    int_npb<-data.frame(matrix(do.call(rbind, intervals),ncol=2))%>%mutate(n=n)

    # updates colnames
    colnames(int_npb) = c("lb","ub","n")

    # binds the intervals from previous replications to one df
    int_npb_reps<-rbind(int_npb,int_npb_reps)
    setTxtProgressBar(pb_1,rep)

    }

  #returns a df
  return(int_npb_reps)
}




#intializes dataframe
interval_df<-data.frame()



#progress bar for for loop on lambdas
```

```
pb = txtProgressBar(min = 0, max = length(all_lambdas), initial = 0)



#sweeping across lambdas
for (lambdav in all_lambdas){

  #creates many many intervals for spesfic lambda
  # and ranges across all sample sizes n
  # THIS TAKES A WHILE
  many_dfs_of_ints<-create_an_int_for_params(n,lambdav,num_replications)

  # creates some summary info like lengths and lambda v
  # we consider the parameter lambda/2
  # if it is in the interval and we use this param
  # to determine the miss abova and below percentages
  interval_df_temp<-many_dfs_of_ints%>%
    mutate(lambda=lambdav)%>%
    mutate(Length = ub-lb)%>%
    mutate(isLin=(ub>lambdav/2)==(lb<lambdav/2))%>%
    mutate(missAbv=lambdav/2<lb)%>%
    mutate(missBel = lambdav/2>ub)



  # interval concatenated
  interval_df<-rbind(interval_df,interval_df_temp)
  #progress bar
  setTxtProgressBar(pb,lambdav)

}




#gives us the summary length
Means_df<-interval_df%>%
  group_by(lambda,n)%>%
```

```
    summarise(mean_Len=mean(Length),prop_captured = mean(isLin),sd_Len=sd(Length))%>%
    mutate(SeAvgLen = sd_Len/sqrt(num_replications))%>%
    select(-sd_Len)

  # create prop of intervals missing above and below statisitc
  All_info_df<-interval_df%>%
    filter(isLin==FALSE)%>%
    group_by(n,lambda)%>%
    summarise(prop_above=mean(missAbv),prop_below=mean(missBel))%>%
    merge(Means_df,by=c("n","lambda"))
  #makes n and lambda a factor variable
  All_info_df<-All_info_df%>%mutate(n=as.factor(n),lambda=as.factor(lambda))




  return(All_info_df)


}
```

Results & Discussion

Here we create the summary information for each of our 6 methods, given the correct parametrazation.

```
# here we define the sample sizes we wish to examine
n_to_range<-c(10,30,100,500)

# here are the population lambdas we want to obseve
all_lambdas_vals = c(.5,1,5)

# here are the number of  Bootstraps we do in an outer loop
# given a method requires a bootstrap
B_num_for_boot=1000
```

```
# this number is the number of replications we want for each interval within
# each combination of lambda and n
num_replications_for_int<-1000
```

To save time on knitting of this document and overall runtime. The following functions were run once and saved as *.Rda* files in the Github Repository for this project.

```
# all of these dataframes are created previously and stored via save functions

All_info_paramboot<-Summarizer4Intervals(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "param_boots",B_num=1000)



All_info_exact_ci<-Summarizer4Intervals(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "exact_dist_ci",B_num=1000)




All_info_paramboot<-Summarizer4Intervals(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "np_boots",B_num=1000)



All_info_paramboot<-Summarizer4Intervals(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "CLT_int",B_num=1000)


#shortened the Bootstrap  replications to 100 to save on computation time
```

```
All_info_np_boot_se_t<-Summarizer4Intervals(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "np_boot_se_t",B_num=100)



All_info_reflect<-Summarizer4Intervals(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "reflected_int",B_num=1000)



#exact for gamma

All_info_exact_gamma<-Summarizer4Intervals_gamma(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "exact_dist_ci_gamma",B_num=1000)

#CLT for gamma
All_info_CLT_gamma<-Summarizer4Intervals_gamma(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "CLT_int_gamma",B_num=1000)

All_info_t_int_gamma<-Summarizer4Intervals_gamma(n=n_to_range,
all_lambdas = all_lambdas_vals,
num_replications = 1000,
interval_funct = "np_boot_se_t_gamma",B_num=80)
```

```
# this loads data frames created from running the code in the chunck above
# this allows us shorten  knitting time
load("Exact_CI_info.Rda")
load("ParamBoot.Rda")
load("Sum_np_Boots.Rda")
load("Sum_info_CLT.Rda")
```

```
load("ReflectedInt.Rda")
load("T_test_info.Rda")



# this loads the infromation from the gamma misspesfication summary dfs
load("NPBoot_gamma.Rda")
load("Reflected_gamma_info.Rda")
load("info_gamma_param.Rda")
load("All_info_CLT_gamma.Rda")
load("All_info_exact_gamma.Rda")
load("All_info_t_int_gamma.Rda")
```

Tables of Summary Infromation for $exp(\lambda)$

```
kable(All_info_exact_ci,caption = "Exact Confidence Interval Summary Infromation for sample exp(n,lambda
```

Table 1: Exact Confidence Interval Summary Infromation for sample exp(n,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.4705882	0.5294118	0.6734571	0.949	0.0074858
10	1	0.5116279	0.4883721	1.3600736	0.957	0.0146538
10	5	0.5409836	0.4590164	6.7800268	0.939	0.0827558
100	0.5	0.4250000	0.5750000	0.1974916	0.960	0.0005931
100	1	0.5500000	0.4500000	0.3976292	0.960	0.0012598
100	5	0.6511628	0.3488372	1.9778227	0.957	0.0060530
30	0.5	0.6046512	0.3953488	0.3693671	0.957	0.0022681
30	1	0.5400000	0.4600000	0.7398550	0.950	0.0044303
30	5	0.3409091	0.6590909	3.6654456	0.956	0.0213645
500	0.5	0.4166667	0.5833333	0.0875370	0.940	0.0001240
500	1	0.4042553	0.5957447	0.1754399	0.953	0.0002463
500	5	0.6041667	0.3958333	0.8793380	0.952	0.0012799

```
kable(All_info_CLT,caption = "CLT & Delta Method Summary Infromation for sample exp(n,lambda)")
```

Table 2: CLT & Delta Method Summary Infromation for sample exp(n,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen	Method
10	0.5	0.0350877	0.9649123	0.6697513	0.943	0.0074470	CLT
10	1	0.0816327	0.9183673	1.3708356	0.951	0.0153278	CLT
10	5	0.1315789	0.8684211	6.8442346	0.962	0.0749766	CLT
100	0.5	0.4313725	0.5686275	0.1979626	0.949	0.0006489	CLT
100	1	0.3225806	0.6774194	0.3957966	0.938	0.0013123	CLT
100	5	0.4509804	0.5490196	1.9856139	0.949	0.0062513	CLT
30	0.5	0.4222222	0.5777778	0.3770340	0.955	0.0023056	CLT
30	1	0.3061224	0.6938776	0.7384772	0.951	0.0045311	CLT
30	5	0.3250000	0.6750000	3.7131726	0.960	0.0213978	CLT
500	0.5	0.5172414	0.4827586	0.0881478	0.942	0.0001268	CLT
500	1	0.3863636	0.6136364	0.1758046	0.956	0.0002421	CLT
500	5	0.5111111	0.4888889	0.8778004	0.955	0.0012546	CLT

```
kable(Info_npBoots,caption = "Nonparametric Bootstrap Summary Infromation for sample exp(n,lambda)")
```

Table 3: Nonparametric Bootstrap Summary Infromation for sample exp(n,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.9160839	0.0839161	0.7622461	0.857	0.0124593
10	1	0.9220779	0.0779221	1.5485020	0.846	0.0258012
10	5	0.8175182	0.1824818	7.6821184	0.863	0.1333830
100	0.5	0.6792453	0.3207547	0.1969779	0.947	0.0008632
100	1	0.7323944	0.2676056	0.3943759	0.929	0.0017796
100	5	0.6935484	0.3064516	1.9695273	0.938	0.0090834
30	0.5	0.7244898	0.2755102	0.3736826	0.902	0.0031416
30	1	0.8021978	0.1978022	0.7591511	0.909	0.0064200
30	5	0.7961165	0.2038835	3.7984630	0.897	0.0324788
500	0.5	0.6666667	0.3333333	0.0875419	0.949	0.0001893
500	1	0.4347826	0.5652174	0.1740318	0.954	0.0003760
500	5	0.5918367	0.4081633	0.8753918	0.951	0.0019614

```
kable(All_info_paramboot,caption = "Parametric Bootstrap Summary Infromation for sample exp(n,lambda)")
```

Table 4: Parametric Bootstrap Summary Infromation for sample exp(n,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.9310345	0.0689655	0.8372409	0.913	0.0097024
10	1	0.9687500	0.0312500	1.6393562	0.936	0.0172243
10	5	0.8987342	0.1012658	8.1804520	0.921	0.0910582
100	0.5	0.7045455	0.2954545	0.2010611	0.956	0.0006707
100	1	0.6363636	0.3636364	0.3997508	0.945	0.0013274
100	5	0.7037037	0.2962963	2.0133173	0.946	0.0065780
30	0.5	0.7258065	0.2741935	0.3914506	0.938	0.0022707
30	1	0.8983051	0.1016949	0.7853176	0.941	0.0046877
30	5	0.8196721	0.1803279	3.9218946	0.939	0.0239227
500	0.5	0.5652174	0.4347826	0.0876591	0.954	0.0001510
500	1	0.6250000	0.3750000	0.1752678	0.960	0.0002890
500	5	0.4285714	0.5714286	0.8755037	0.958	0.0014330

```
kable(All_info_reflect,caption="Reflected Non-Parametric Interval for sample exp(n,lambda)")
```

Table 5: Reflected Non-Parametric Interval for sample exp(n,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.0583942	0.9416058	0.7850495	0.863	0.0133142
10	1	0.0479452	0.9520548	1.5396573	0.854	0.0262147
10	5	0.0168067	0.9831933	7.6346384	0.881	0.1235694
100	0.5	0.1428571	0.8571429	0.1996005	0.930	0.0009104
100	1	0.1363636	0.8636364	0.3941156	0.934	0.0017914
100	5	0.1060606	0.8939394	1.9737659	0.934	0.0088951
30	0.5	0.0384615	0.9615385	0.3752934	0.896	0.0030885
30	1	0.0645161	0.9354839	0.7515311	0.907	0.0060049
30	5	0.0786517	0.9213483	3.7690405	0.911	0.0317157
500	0.5	0.2407407	0.7592593	0.0875281	0.946	0.0001920
500	1	0.3207547	0.6792453	0.1748576	0.947	0.0004021
500	5	0.3888889	0.6111111	0.8760210	0.946	0.0019173

```
kable(All_info_np_boot_se_t,caption = "Non-Parametric Bootstrap to Create T-Test from sample exp(n,lambd
```

Table 6: Non-Parametric Bootstrap to Create T-Test from sample exp(n,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.6455696	0.3544304	0.9268728	0.921	0.0223399
10	1	0.5362319	0.4637681	1.7727933	0.931	0.0360784
10	5	0.6860465	0.3139535	9.2178461	0.914	0.2312907
100	0.5	0.5211268	0.4788732	0.1964929	0.929	0.0014347
100	1	0.4931507	0.5068493	0.3913252	0.927	0.0028354
100	5	0.6065574	0.3934426	1.9452995	0.939	0.0141778
30	0.5	0.5308642	0.4691358	0.3871179	0.919	0.0044666
30	1	0.5833333	0.4166667	0.7745117	0.916	0.0087055
30	5	0.5151515	0.4848485	3.8838053	0.934	0.0431870
500	0.5	0.5729167	0.4270833	0.0852312	0.904	0.0005007
500	1	0.5180723	0.4819277	0.1694697	0.917	0.0010541
500	5	0.5151515	0.4848485	0.8567805	0.934	0.0052350

Tables of Summary Infromation for $Ga(2, \lambda)$

```
kable(All_info_exact_gamma,caption = "Exact Confidence Interval Summary Infromation for sample Ga(2,lamb
```

Table 7: Exact Confidence Interval Summary Infromation for sample Ga(2,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.3333333	0.6666667	0.3251915	0.997	0.0023835
10	1	0.6666667	0.3333333	0.6419440	0.997	0.0047040
10	5	0.5000000	0.5000000	3.2704014	0.992	0.0254562
100	0.5	0.3750000	0.6250000	0.0986638	0.992	0.0002243
100	1	0.2500000	0.7500000	0.1963932	0.992	0.0004531
100	5	0.2500000	0.7500000	0.9850157	0.992	0.0021972
30	0.5	0.7500000	0.2500000	0.1819441	0.996	0.0007619
30	1	0.5000000	0.5000000	0.3629698	0.994	0.0015329
30	5	0.8333333	0.1666667	1.8170030	0.994	0.0074919

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
500	0.5	0.6666667	0.3333333	0.0438156	0.994	0.0000432
500	1	0.5454545	0.4545455	0.0877563	0.989	0.0000896
500	5	0.5000000	0.5000000	0.4387361	0.996	0.0004375

```
kable(All_info_CLT_gamma,caption = "CLT & Delta Method Summary Infromation for sample Ga(2,lambda)")
```

Table 8: CLT & Delta Method Summary Infromation for sample Ga(2,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.1666667	0.8333333	0.3286817	0.994	0.0025359
10	1	0.0000000	1.0000000	0.6450779	0.994	0.0046393
10	5	0.0000000	1.0000000	3.2949320	0.990	0.0249318
100	0.5	0.1250000	0.8750000	0.0983338	0.992	0.0002241
100	1	0.5000000	0.5000000	0.1973092	0.998	0.0004294
100	5	0.0000000	1.0000000	0.9792002	0.998	0.0021727
30	0.5	0.1250000	0.8750000	0.1823622	0.992	0.0007721
30	1	0.0000000	1.0000000	0.3648671	0.994	0.0015095
30	5	0.0000000	1.0000000	1.8170573	0.998	0.0075033
500	0.5	0.0000000	1.0000000	0.0438085	0.996	0.0000426
500	1	0.5000000	0.5000000	0.0876096	0.996	0.0000862
500	5	0.4444444	0.5555556	0.4390043	0.991	0.0004413

```
kable(np_boots_gamma_info,caption = "Nonparametric Bootstrap Summary Infromation for sample Ga(2,lambda
```

Table 9: Nonparametric Bootstrap Summary Infromation for sample Ga(2,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.25	0.7563025	0.2436975	0.2290245	0.881	0.0028590
10	0.5	0.7927928	0.2072072	0.4683739	0.889	0.0055588
10	2.5	0.7818182	0.2181818	2.3654231	0.890	0.0290461
100	0.25	0.6037736	0.3962264	0.0693214	0.947	0.0002594
100	0.5	0.6307692	0.3692308	0.1383382	0.935	0.0005317
100	2.5	0.6875000	0.3125000	0.6926333	0.936	0.0025342
30	0.25	0.6986301	0.3013699	0.1276306	0.927	0.0008257

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
30	0.5	0.8157895	0.1842105	0.2566405	0.924	0.0016896
30	2.5	0.6944444	0.3055556	1.2486038	0.928	0.0079098
500	0.25	0.4905660	0.5094340	0.0308407	0.947	0.0000546
500	0.5	0.6153846	0.3846154	0.0617724	0.948	0.0001135
500	2.5	0.6122449	0.3877551	0.3090537	0.951	0.0005705

```
kable(Info_param_gamma,caption = "Parametric Bootstrap Summary Infromation for sample Ga(2,lambda)")
```

Table 10: Parametric Bootstrap Summary Infromation for sample Ga(2,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	1.0000000	0.0000000	0.3959139	0.977	0.0031994
10	1	1.0000000	0.0000000	0.7956803	0.981	0.0060923
10	5	1.0000000	0.0000000	3.9711190	0.978	0.0303089
100	0.5	0.8333333	0.1666667	0.1002067	0.994	0.0002468
100	1	0.8333333	0.1666667	0.1996704	0.994	0.0004704
100	5	0.5555556	0.4444444	0.9956606	0.991	0.0024830
30	0.5	1.0000000	0.0000000	0.1921714	0.991	0.0008356
30	1	0.8888889	0.1111111	0.3862538	0.991	0.0016532
30	5	0.9000000	0.1000000	1.9354034	0.990	0.0085740
500	0.5	0.6666667	0.3333333	0.0437904	0.997	0.0000610
500	1	0.7272727	0.2727273	0.0878518	0.989	0.0001249
500	5	0.4000000	0.6000000	0.4387199	0.995	0.0006029

```
kable(All_info_reflect_gamma,caption="Reflected Non-Parametric Interval for sample Ga(2,lambda)")
```

Table 11: Reflected Non-Parametric Interval for sample Ga(2,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.0370370	0.9629630	0.2307838	0.892	0.0027868
10	1	0.0813008	0.9186992	0.4610273	0.877	0.0054491
10	5	0.0892857	0.9107143	2.3424675	0.888	0.0285011
100	0.5	0.1578947	0.8421053	0.0691376	0.943	0.0002490
100	1	0.2000000	0.8000000	0.1376055	0.940	0.0005265

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
100	5	0.3333333	0.6666667	0.6928806	0.934	0.0025927
30	0.5	0.0833333	0.9166667	0.1276776	0.928	0.0008179
30	1	0.1326531	0.8673469	0.2546977	0.902	0.0017110
30	5	0.2000000	0.8000000	1.2734773	0.920	0.0084081
500	0.5	0.2884615	0.7115385	0.0308556	0.948	0.0000562
500	1	0.3777778	0.6222222	0.0618891	0.955	0.0001152
500	5	0.4318182	0.5681818	0.3083972	0.956	0.0005722

```
kable(All_info_t_int_gamma,caption = "Non-Parametric Bootstrap to Create T-Test from sample Ga(2,lambda
```

Table 12: Non-Parametric Bootstrap to Create T-Test from sample Ga(2,lambda)

n	lambda	prop_above	prop_below	mean_Len	prop_captured	SeAvgLen
10	0.5	0.7333333	0.2666667	0.2934235	0.925	0.0046312
10	1	0.6478873	0.3521127	0.5578126	0.929	0.0085682
10	5	0.7065217	0.2934783	2.7712442	0.908	0.0436043
100	0.5	0.4337349	0.5662651	0.0684485	0.917	0.0004648
100	1	0.5000000	0.5000000	0.1370011	0.916	0.0009336
100	5	0.5384615	0.4615385	0.6906724	0.922	0.0045056
30	0.5	0.6235294	0.3764706	0.1318614	0.915	0.0011933
30	1	0.6470588	0.3529412	0.2629447	0.932	0.0023774
30	5	0.5466667	0.4533333	1.3331542	0.925	0.0124455
500	0.5	0.5217391	0.4782609	0.0302420	0.931	0.0001776
500	1	0.4307692	0.5692308	0.0603530	0.935	0.0003402
500	5	0.5441176	0.4558824	0.3002069	0.932	0.0018048

Visual Summaries

Below we Add a tag to each summary table so we have a correspondence with each method.

```
#creates a label for each dataframe about what method was used to create sum info
All_info_exact_ci$Method = "Exact"
Info_npBoots$Method = "NPBoot"
All_info_paramboot$Method = "PBoot"
```

```
All_info_CLT$Method ="CLT"
All_info_reflect$Method = "Reflected"
All_info_np_boot_se_t$Method = "T_Test"



All_info_reflect_gamma$Method = "Reflected"
np_boots_gamma_info$Method = "NPBoot"
Info_param_gamma$Method = "PBoot"
All_info_CLT_gamma$Method = "CLT"
All_info_exact_gamma$Method = "Exact"
All_info_t_int_gamma$Method = "T_Test"
```

Here we vertically concatenate the infromation together for each parameterization: correct and incorrect. Note that the methods are still in the same order.

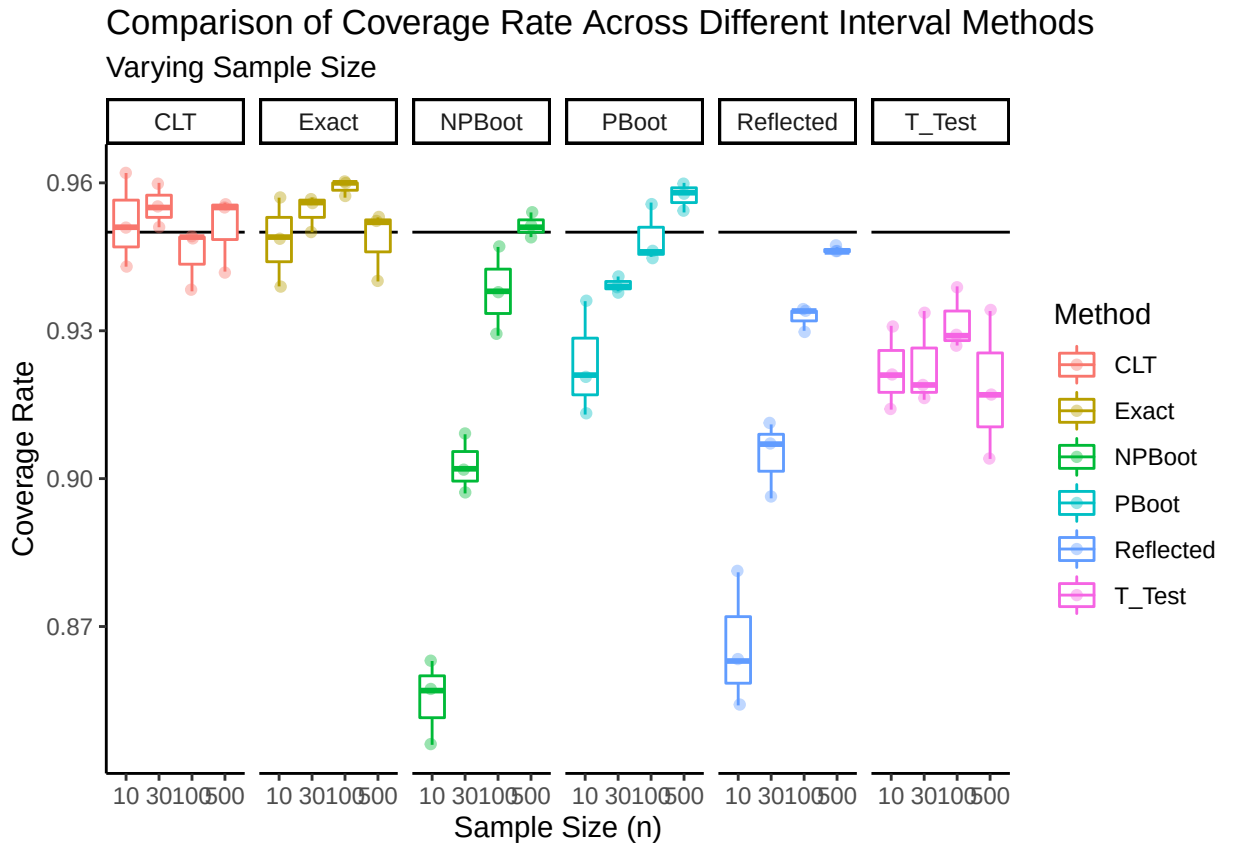
```
# going to vertically stack all dfs from 6 methods
merged_all_dfs<-rbind(All_info_paramboot,
                      Info_npBoots,
                      All_info_exact_ci,
                      All_info_CLT,
                      All_info_reflect,
                      All_info_np_boot_se_t)


# creates a larger dataframe of summary infoamtion for the 6 interval methods
# when constructed usin the mispesficed population interval
merged_all_dfs_gamma<-merged_all_dfs_gamma<-rbind(Info_param_gamma,
                                                  np_boots_gamma_info,
                                                  All_info_exact_gamma,
                                                  All_info_CLT_gamma,
                                                  All_info_reflect_gamma,
                                                  All_info_t_int_gamma)
```

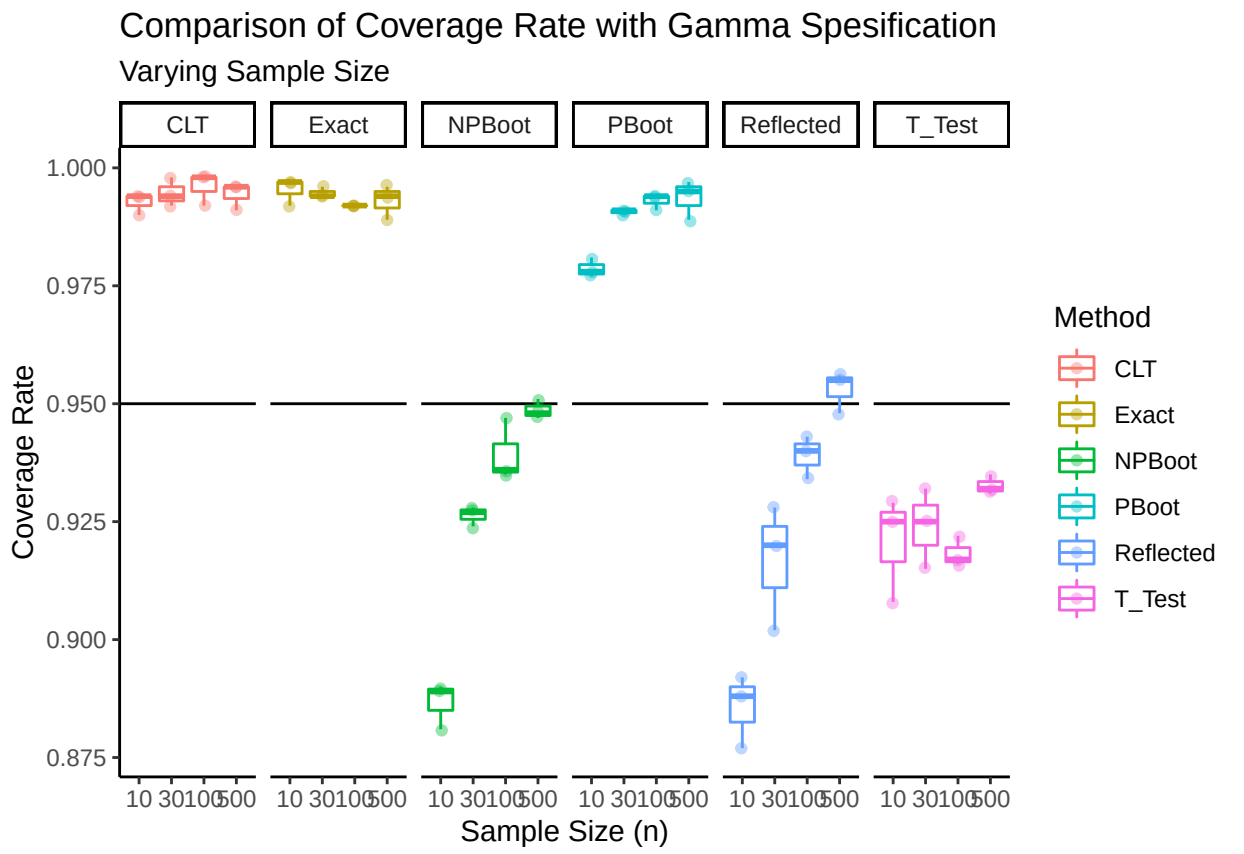
We now are going to examine how to coverage rate changes when we look at changing sample size.

The box plot below is a crude representation of the coverage rate. This box plot is created by 3 points, each representing one of three population λ's used. These individual points represent the coverage rate for a setting with the n on the x-axis, and a particular λ. These points are jittered on top of the boxplots.

```
# creates
ggplot(merged_all_dfs,aes(x=n,y=prop_captured))+theme_classic()+
  geom_hline(yintercept = .95)+geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,widt
  facet_grid(~Method)+
  labs(title="Comparison of Coverage Rate Across Different Interval Methods",subtitle="Varying Sample S
```




```
ggplot(merged_all_dfs_gamma,aes(x=n,y=prop_captured))+theme_classic()+
  geom_hline(yintercept = .95)+geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,widt
  facet_grid(~Method)+
  labs(title="Comparison of Coverage Rate with Gamma Spesification",subtitle="Varying Sample Size",y="C
```

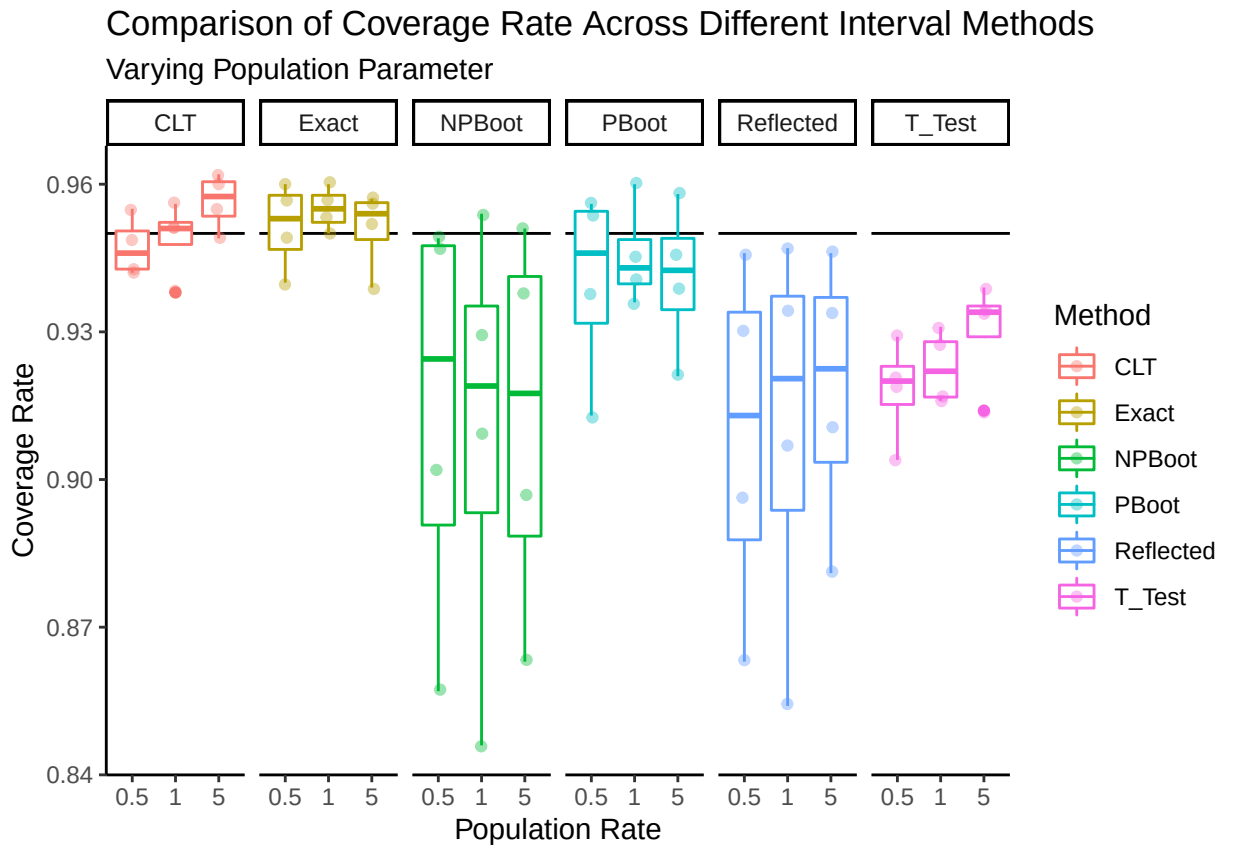


We also note that the horizontal line going across each figure represents the coverage rate we prescribed throughout the setting: $1 - \alpha$

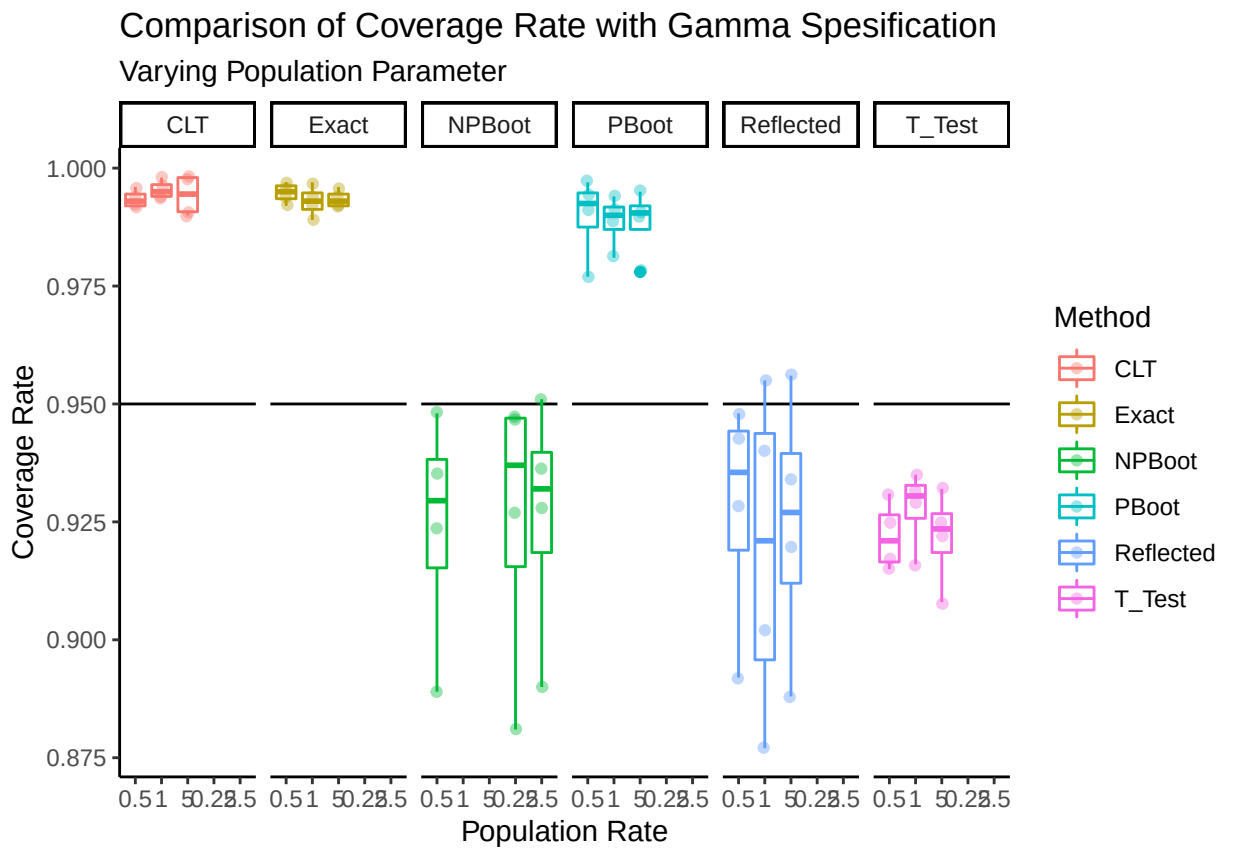
Examing Population Parameter's Coverage Impact

```
# created a plot from the correct spesification
# looking at lambda

ggplot(merged_all_dfs,aes(x=lambda,y=prop_captured))+theme_classic()+
  geom_hline(yintercept = .95)+geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,width
  facet_grid(~Method)+
  labs(title="Comparison of Coverage Rate Across Different Interval Methods",subtitle="Varying Populati
```



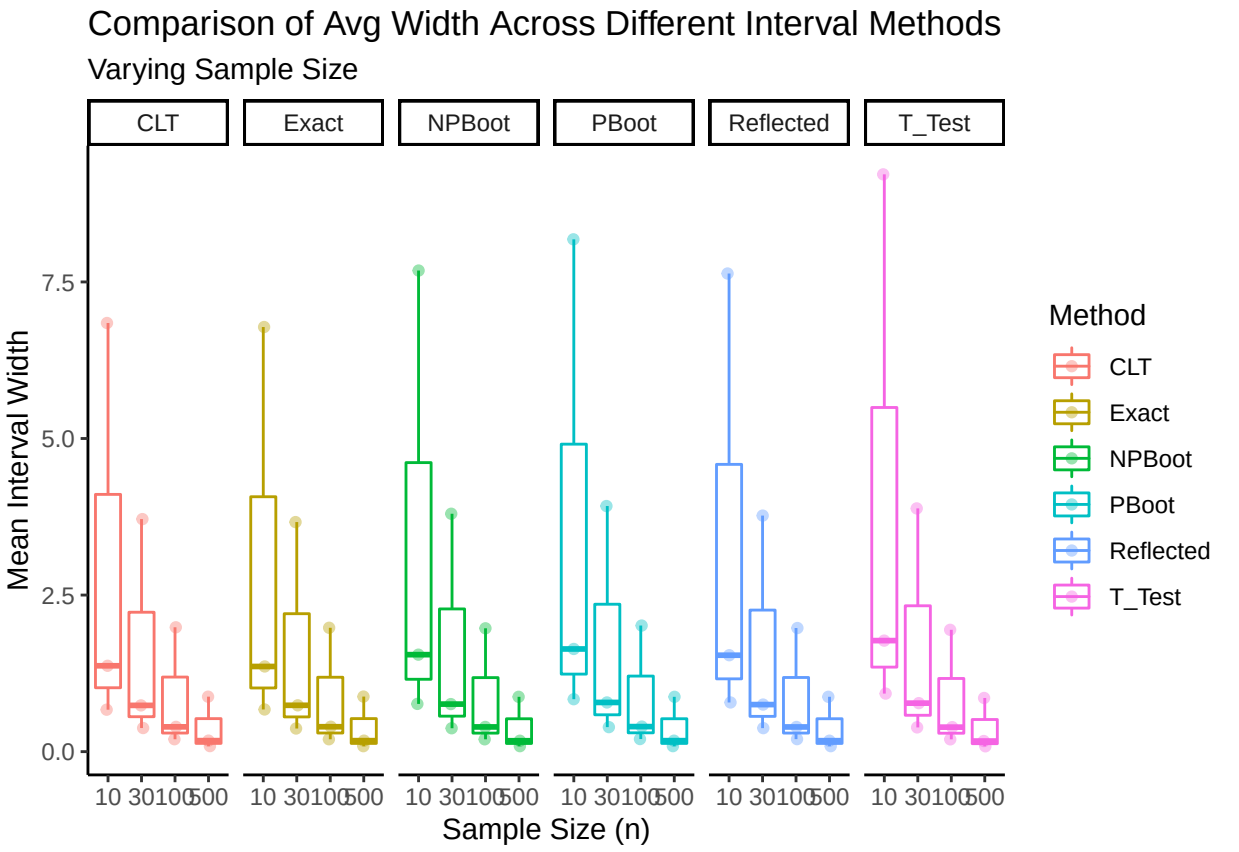

```
# plot for misspesficiation
# looking at lambda values
ggplot(merged_all_dfs_gamma,aes(x=lambda ,y=prop_captured))+theme_classic()+
  geom_hline(yintercept = .95)+geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,width
  facet_grid(~Method)+
  labs(title="Comparison of Coverage Rate with Gamma Spesification",subtitle="Varying Population Parame
```



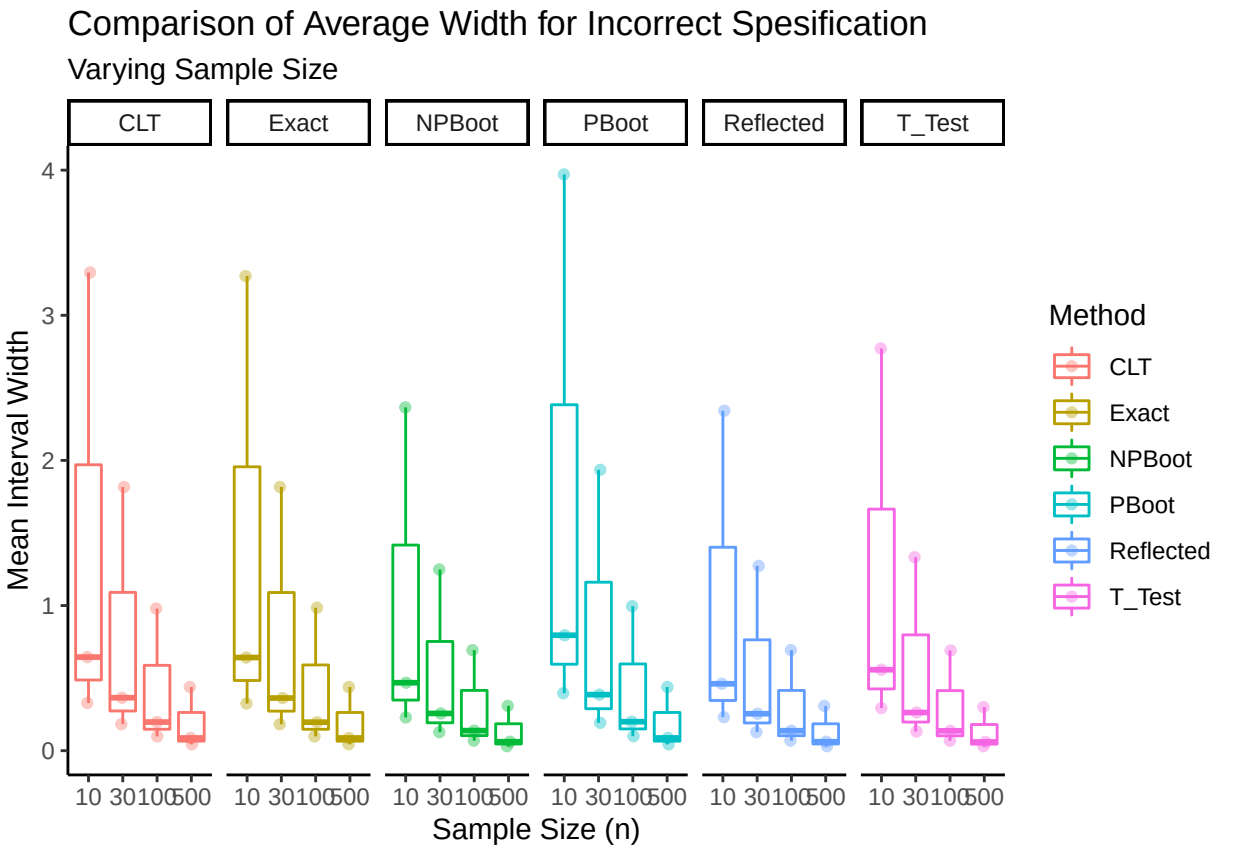
We see that changing the lambda population parameter does not really change coverage rate for an interval method.

Examining Interval Widths

```
# examin interval width and how to changes with sample size
# exact spesficiation
ggplot(merged_all_dfs,aes(x=n,y=mean_Len ))+theme_classic()+
 geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,width=.05)+
  facet_grid(~Method)+
  labs(title="Comparison of Avg Width Across Different Interval Methods",subtitle="Varying Sample Size"
```




```
# gamma spesficiation
# examin interval width and how to changes with sample size
ggplot(merged_all_dfs_gamma,aes(x=n,y=mean_Len ))+theme_classic()+
 geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,width=.05)+
  facet_grid(~Method)+
  labs(title="Comparison of Average Width for Incorrect Spesification",subtitle="Varying Sample Size",y
```

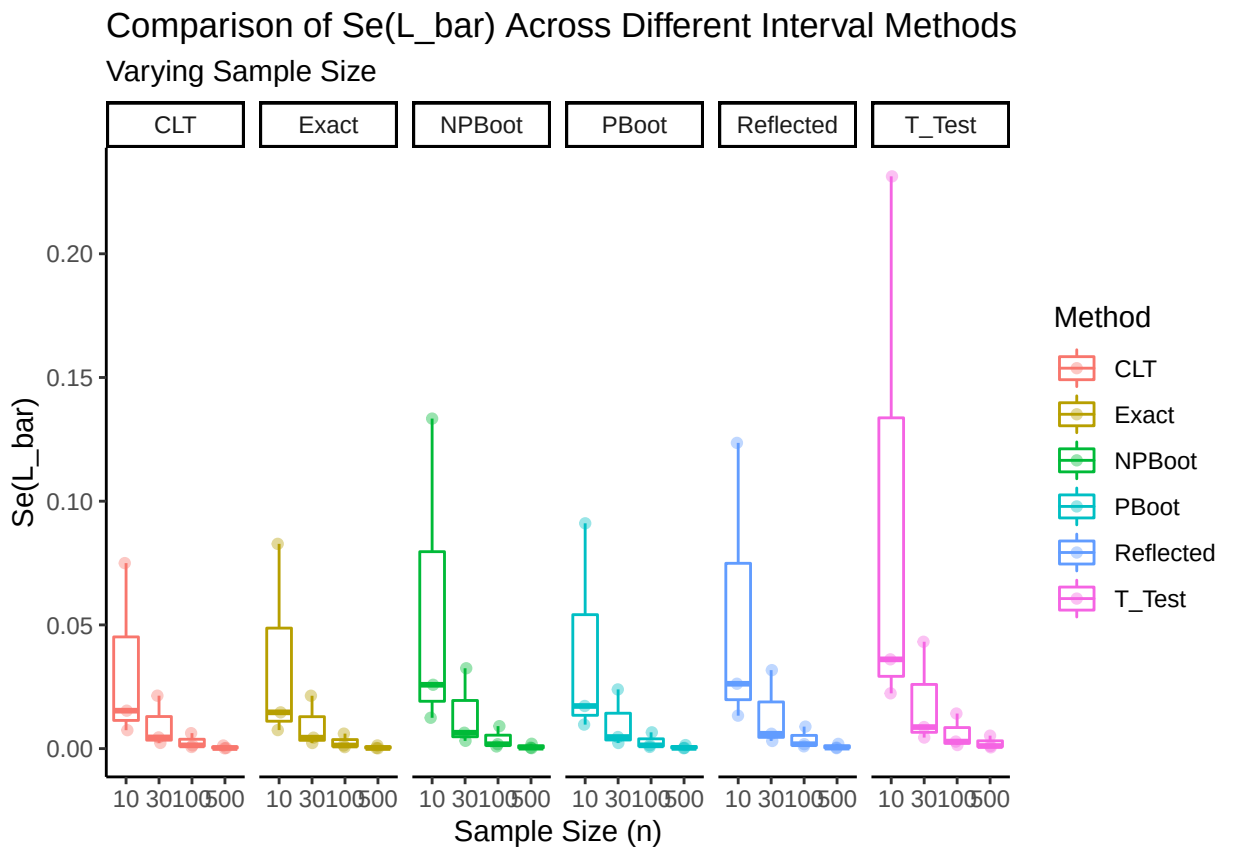


We see that as sample size increases the mean interval width decreases across all methods. We further note that the widths of all the intervals using the mispesficed population distribution $Ga(2, \lambda)$.

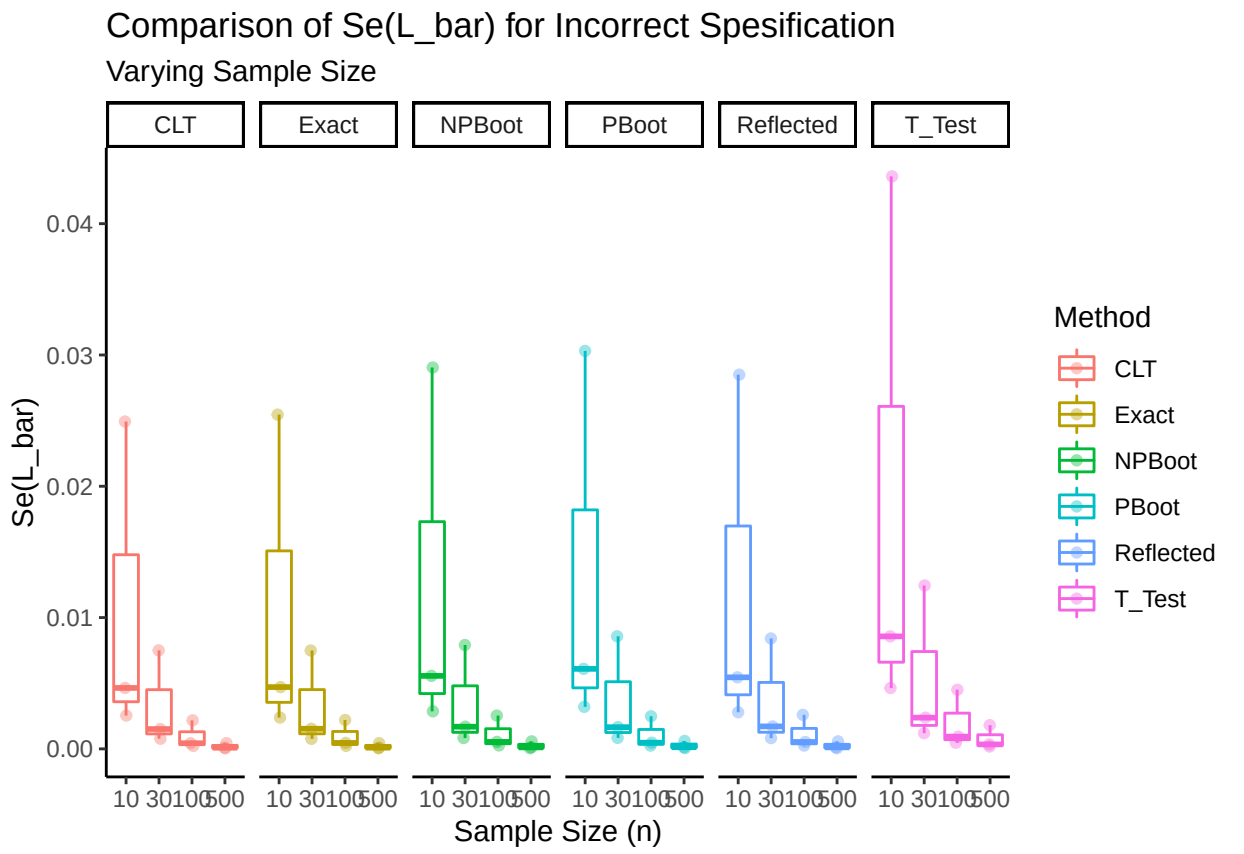
Examining Standard Error

Observe the main trend here. As sample size increases, the standard error of the length decreases as sample size increases across all methods. We note that the average std error of the length is less overall for the incorrect spesification.

```
# examin interval width and how to changes with sample size
# exact spesficiation
ggplot(merged_all_dfs,aes(x=n,y=SeAvgLen ))+theme_classic()+
 geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,width=.05)+
  facet_grid(~Method)+
  labs(title="Comparison of Se(L_bar) Across Different Interval Methods",subtitle="Varying Sample Size"
```




```
# gamma spesficiation
# examin interval width and how to changes with sample size
ggplot(merged_all_dfs_gamma,aes(x=n,y=SeAvgLen ))+theme_classic()+
 geom_boxplot(aes(col=Method))+geom_jitter(aes(col=Method),alpha=0.4,width=.05)+
  facet_grid(~Method)+
  labs(title="Comparison of Se(L_bar) for Incorrect Spesification",subtitle="Varying Sample Size",y="Se
```



As sample size increases the standard error of the average length of intervals decreases.

Conclusion

For the correct exp(λ) distribution, the exact and delta intervals were consistently hovering around 0.95 across sample size when we looked at coverage rates. In contrast, the nonparametric, non-parametric reflected and the parametric bootstrap intervals were not consistent in coverage rate but improved and approached 0.95 as sample size increased. Finally the bootstrap t-interval under performed but consistently with a coverage rate hovering around .93.

Overall, we have found that exact and approximate CLT based methods performed the best in terms of coverage rate across all methods - when considering the correct population. This is if we consider the best coverage rate being higher / closer to one. We may however - want a coverage rate that is closer to our spesfied coverage rate from simulation (.95).

We hypothesis a reason that the t-test bootstrap preforms so poorly in terms of coverage rate is that two quantites needed to be estimated by a bootstrap. Having two estimated quantities increases the variability in our sampling distribution. Having this increases variabliity leads to more intervals missing or not capturing

the true parameter.

For the incorrect distribution, all coverage rates in the exact and CLT methods actually improved beyond .95. The coverage rates seemed to hover around 99% across all settings for these two methods when the $gamma(2, \lambda)$ was used. We note that since our coverage rate is not hovering around our specified .95 coverage rate - this indicates our underlying population is incorrect - which it is. In contrast, as sample size increases our nonparametric methods (raw and reflected) approach our correctly specified coverage rate of .95. We note while coverage rate is greater for the incorrect specification, we may desire have an interval method that gives us an average coverage rate that is what we specified.

One concept that was quite surprising was the fact that the parametric bootstrap coverage rates bounded from above the coverage rates for the nonparametric bootstrap when considering the gamma population. We thought that a wrong distribution would have the opposite effect and actually cause the coverage rates for the parametric model to be worse than a nonparametric interval.

That being said - we can give a overarching higher level argument for why something like the exact interval would decrease in length when looking at the incorrect interval. Observe the interval for estimating λ from an exponential.

$$\hat{\lambda} \pm ga_{\alpha/2} \frac{\hat{\lambda}}{\sqrt{n}}$$

Now lets say we use the same CLT argument, from the $exp(\lambda)$ case, but estimate $\lambda/2$ from our incorrect distribution.

$$\frac{\hat{\lambda}}{2} \pm ga_{\alpha/2} \frac{\hat{\lambda}}{2\sqrt{n}}$$

This interval's width is half the length of the interval when we use the exponential population for our sample estimate.

When we examine the standard error of the average length of all intervals across all settings- we see that lengths are roughly smaller by a factor of 2 for the incorrect population in comparison to the correct.

Further, we note that our standard error of the average length decreases with sample size. As sample size increase the variability in the average variability (or length) of our interval decreases rapidly.

Our recommendation, for a small sample size, is to use an exact based or CLT Approximate interval, when considering the correct $exp(\lambda)$ distribution. If an appropriate large sample size is given (above 500), and one does not truly know anything about the underlying distribution or may have an incorrect sample from an improper population distribution - it may be appropriate to use a non-parametric method such as raw percentile. This is especially true in the coverage rate for these methods approaches the specified coverage rate 0.95.