# Bonus Project

### Section 1:

Firstly, I was getting bootstrap resample from both parametric and non-parametric methods. For parametric method, the shape and rate are coming from MLE. For non-parametric, I was using the boot package in R, which generate 1000 resamples. Then I calculated CI of three methods(ie, normal, basic and percentile) using both parametric and non-parametric methods.

### Section 2:

Following is the results.

> ci.normal.par

[1] 39.36372 61.63164

> ci.normal.npar

[1] 25.75044 48.01836

> ci.basic.par

[1] 43.49728 63.49728

> ci.basic.npar

[1] 29.5 49.5

> ci.percentile.par

[1] 35.5 55.5

> ci.percentile.npar

[1] 35.5 55.5

From the results, we can see that generally speaking, parametric method gives a higher CI range compared with non-parametric one. However, for percentile CI, there is no distinct difference.

### Section3:

library(boot)

CPUdata = c(70,36,43,69,82,48,34,62,35,15,59,139,46,37,42,30,55,56,36,82,38,89,54,25,35,24,22,9,56,19)

#parametric method

neg.loglik.fun = function(par,dat){

result = sum(dgamma(dat, shape = par[1], rate = par[2], log = TRUE) )

return( - result )

}

ml.est = ml.est = optim(par = c(3,0.1),fn = neg.loglik.fun, method = "L-BFGS-B",lower = rep(0,2), hessian = TRUE, dat = CPUdata)

shape = ml.est$par[1]

rate = ml.est$par[2]

resample = function( n, dat ){

ml.est = ml.est = optim(par = c(3,0.1),fn = neg.loglik.fun, method = "L-BFGS-B",lower = rep(0,2), hessian = TRUE, dat = CPUdata)

shape = ml.est$par[1]

rate = ml.est$par[2]

data = rgamma(n, shape, rate)

result = data[(n+1)\*0.5]

return(result)

}

median.est.par = mean(replicate( 1000, resample( 5, CPUdata) )) #get 1000 estimate median from parametric bootstrap

median = qgamma(0.5, shape, rate )

#nonparametric method

median.npar <- function(x, indices){

result <- median(x[indices])

return(result)

}

median.npar.boot <- boot(CPUdata, median.npar, R = 1000, sim = "ordinary",stype="i")

median.est.npar = median.npar.boot $ t0

#computing CI using different method.

SE = sqrt( var(median.npar.boot$t ))

#Normal approximation CI

B.par = 1/1000 \* sum( median.npar.boot$t) - median

B.npar = median.est.par - median

ci.normal.par = c( median.est.par - B.par - qnorm( 1-0.05/2) \* SE, median.est.par -B.par - qnorm( 0.05/2) \* SE)

ci.normal.npar = c( median.est.npar - B.npar - qnorm( 1-0.05/2) \* SE, median.est.npar -B.npar - qnorm( 0.05/2) \* SE)

#basic bootstrap CI

ci.basic.par = c( 2\*median.est.par - sort(median.npar.boot$t)[(1000+1)\*(1-0.05/2)], 2\*median.est.par - sort(median.npar.boot$t)[(1000+1)\*(0.05/2)])

ci.basic.npar =c( 2\*median.est.npar - sort(median.npar.boot$t)[(1000+1)\*(1-0.05/2)], 2\*median.est.npar - sort(median.npar.boot$t)[(1000+1)\*(0.05/2)])

#percentile

ci.percentile.par = c( sort(median.npar.boot$t)[(1000+1)\*(0.05/2)], sort(median.npar.boot$t)[(1000+1)\*(1-0.05/2)])

ci.percentile.npar = c( sort(median.npar.boot$t)[(1000+1)\*(0.05/2)], sort(median.npar.boot$t)[(1000+1)\*(1-0.05/2)])

ci.normal.par

ci.normal.npar

ci.basic.par

ci.basic.npar

ci.percentile.par

ci.percentile.npar