**Mini Project 4**

**Name :**

**Maitri Dharmendrakumar Shah (mxs172030)**

**Nayana Thomas(nxt170630)**

**Contribution of team members:**

Maitri:

Wrote equation and calculation in word

Drew a graph about simulation and found the way to distribute it

Learned R Coding

Tried different R codes

Wrote explanation for section 2

Derived conclusions from calculations

Nayana:

Did calculation on paper

Explained the simulation

Learned R coding

Did debugging for R codes

Derived conclusions from graphs

Wrote explanation for section1

1.a

life <- scan(file = "lifetime.txt")

m <- mean(life)

s <- sd(life)

n<-length(life)

m

s

alpha <- (m \* m) / (s\*s)

lambda <- m / (s\*s)

alpha

lambda

neg.loglik.fun <- function(par, dat)

{

result <- sum(dweibull(

dat,

shape = par[1],

scale = par[2],

log = TRUE

))

return(-result)

}

ml.est <-

optim(

par = c(alpha, lambda),

fn = neg.loglik.fun,

method = "L-BFGS-B",

lower = rep(0, 2),

hessian = TRUE,

dat = life

)

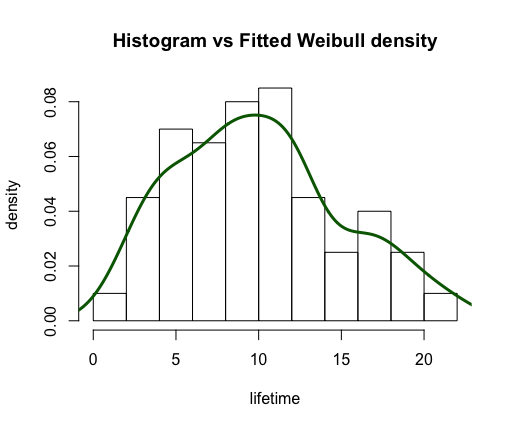
ml.est

This R code will read the lifetime.txt file and read the data from the file. Then find the mean and standard deviation. Then using the equations of the shape and the scale parameters we can find the values of alpha and lambda.

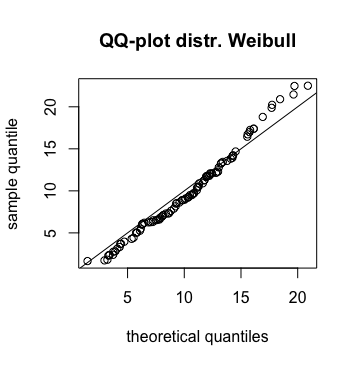
One we get alpha and lambda, we can find the log likelihood function to maximize the data and find the maximum likelihood estimator using the optim function which does the numerical optimization.

Using the above R code we get the value of MLEs of alpha and lambda as 2.124102 and 11.294849 respectively.

1.b.



The above diagram shows the histogram of the data stored in lifetime.txt superimposed with the Weibull density. From the plot we can see that the density curve fitted well with the histogram of the data and hence we can say that Weibull model is a good fit for the lifetime data.



Also, in the above plot, we can see the sample quantiles of the data plotted against the theoretical quantiles of the data and we can see a linear curve. This too shows that Weibull model is a good fit for the data.

R code

#density histogram

hist(life,freq=FALSE,xlab = "lifetime",ylab = "density",main="Histogram vs Fitted Weibull density")

#superimpose the fitted density

weibull.pdf<-function(life,shape=ml.est$par[1],scale=ml.est$par[2])

{

dweibull(

life,

shape = shape,

scale = scale,

log = TRUE

)

}

curve(weibull.pdf,from=0,to=100,add=TRUE)

lines(density(life,adjust=2),col="red",lwd=3)

lines(density(life),col="darkgreen",lwd=3)

1.c. 95% confidence intervals for the parameters α and λ.

Written code is attached.

R code:

life <- scan(file = "lifetime.txt")

m <- mean(life)

s <- sd(life)

n<-length(life)

m

s

alpha <- (m \* m) / (s\*s)

lambda <- m / (s\*s)

alpha

lambda

neg.loglik.fun <- function(par, dat)

{

result <- sum(dweibull(

dat,

shape = par[1],

scale = par[2],

log = TRUE

))

return(-result)

}

ml.est <-

optim(

par = c(alpha, lambda),

fn = neg.loglik.fun,

method = "L-BFGS-B",

lower = rep(0, 2),

hessian = TRUE,

dat = life

)

ml.est

diffa <- ml.est$par[1] + (c(-1, 1) \* qnorm(0.975) \* 1) / sqrt(ml.est$hessian[1,1])

diffa

difflambda<-ml.est$par[2]+(c(-1,1)\*qnorm(0.975)\*1)/sqrt(ml.est$hessian[2,2])

difflambda

Solution

> diffa <- ml.est$par[1] + (c(-1, 1) \* qnorm(0.975) \* 1) / sqrt(ml.est$hessian[1,1])

>

> diffa

[1] 1.810808 2.437397

> difflambda<-ml.est$par[2]+(c(-1,1)\*qnorm(0.975)\*1)/sqrt(ml.est$hessian[2,2])

> difflambda

[1] 10.25264 12.33705

θˆ±zα/2 ·σ(θˆ) = [θˆ−zα/2 ·σ(θˆ), θˆ+zα/2 ·σ(θˆ)]

is a (1 − α)100% confidence interval for θ.

Thus 95% confidence interval for alpha = (1.810808, 2.437397)

And 95% confidence interval for lambda = (10.25264, 12.33705)

Question 2:

( a )

Section 1:

1 – α = 0.95

ux = 5

uy = 5

σ2x = 1

For

n = 5

(σy /σx)2 = 1

Section 2:

counter1 <- 0

counter2 <- 0

totalwidth1 <- 0

totalwidth2 <- 0

for (i in 1:5000)

{

n <- 5

ux <- 5

uy <- 5

sx <- 1

ratio <- 1

sy <- sqrt(ratio \* sx)

x <- rnorm(n, ux, sx)

y <- rnorm(n, uy, sy)

mx <- mean(x)

my <- mean(y)

sdx <- sd(x)

sdy <- sd(y)

bar <- mx - my

####----pooled sample variance

sigma2 <-

((n - 1) \* sdx \* sdx + (n - 1) \* sdy \* sdy) / (n + n - 2)

var <- sigma2 \* (2 / n)

stdError <- sqrt(var)

width1 <- 2 \* qt(0.975, (2 \* n - 2)) \* stdError

totalwidth1 <- totalwidth1 + width1

coff1 <- bar + c(-1, 1) \* (width1 / 2)

# coff1

####---Satterthwaite's approximation

se <- sqrt((sdx ^ 2 / n) + (sdy ^ 2 / n))

v <-

(((sdx ^ 2 / n) + (sdy ^ 2 / n)) ^ 2) / ((sdx ^ 4 / (n ^ 2 \* (n - 1))) +

(sdy ^ 4 / (n ^ 2 \* (n - 1))))

width2 <- 2 \* qt(0.975, v) \* se

totalwidth2 <- totalwidth2 + width2

coff2 <- bar + c(-1, 1) \* (width2 / 2)

# coff2

coff <- c(coff1, coff2)

if (coff1[2] > 0 & coff1[1] < 0) {

counter1 = counter1 + 1

}

if (coff2[2] > 0 & coff[1] < 0) {

counter2 = counter2 + 1

}

}

prob1 <- counter1 / 5000

prob2 <- counter2 / 5000

prob1

prob2

exwidth1 <- totalwidth1 / 5000

exwidth2 <- totalwidth2 / 5000

exwidth1

exwidth2

Narration of code:

For the specified values, to get Monte Carlo simulation we used rnorm method to find values of x and y and after getting the values we found mean and standard deviation and then put in the equations of those two confidence intervals. As ux-uy is 0, to find coverage probability, we are maintaining counters for that and to find expected width we are adding those values and after repeating this process for 5000 times we will be able to get expected values of widths.

( b )

Section 1:

1: Pooled sample variance

2: Satterthwaite’s approximation

Summary:

n: 5 varx: 1 vary: 1.048809 Prob1: 0.9482 exp1: 2.902411 Prob2: 0.9512 exp2: 3.01147

n: 5 varx: 1 vary: 1.048809 Prob1: 0.9488 exp1: 2.895573 Prob2: 0.9534 exp2: 3.00248

n: 5 varx: 1 vary: 1.048809 Prob1: 0.9496 exp1: 2.923579 Prob2: 0.9524 exp2: 3.034108

n: 5 varx: 1 vary: 1.048809 Prob1: 0.9524 exp1: 2.875351 Prob2: 0.954 exp2: 2.983418

n: 10 varx: 1 vary: 1.048809 Prob1: 0.9448 exp1: 1.900893 Prob2: 0.9456 exp2: 1.914757

n: 10 varx: 1 vary: 1.048809 Prob1: 0.953 exp1: 1.900092 Prob2: 0.954 exp2: 1.914463

n: 10 varx: 1 vary: 1.048809 Prob1: 0.949 exp1: 1.899016 Prob2: 0.9496 exp2: 1.913482

n: 10 varx: 1 vary: 1.048809 Prob1: 0.9488 exp1: 1.896758 Prob2: 0.9498 exp2: 1.910751

n: 30 varx: 1 vary: 1.048809 Prob1: 0.958 exp1: 1.056782 Prob2: 0.958 exp2: 1.057583

n: 30 varx: 1 vary: 1.048809 Prob1: 0.9502 exp1: 1.055857 Prob2: 0.9504 exp2: 1.056641

n: 30 varx: 1 vary: 1.048809 Prob1: 0.9458 exp1: 1.051775 Prob2: 0.9458 exp2: 1.052556

n: 30 varx: 1 vary: 1.048809 Prob1: 0.9548 exp1: 1.053645 Prob2: 0.9548 exp2: 1.054446

n: 100 varx: 1 vary: 1.048809 Prob1: 0.9466 exp1: 0.5710195 Prob2: 0.9466 exp2: 0.5710638

n: 100 varx: 1 vary: 1.048809 Prob1: 0.9488 exp1: 0.5715763 Prob2: 0.9488 exp2: 0.5716201

n: 100 varx: 1 vary: 1.048809 Prob1: 0.954 exp1: 0.571278 Prob2: 0.954 exp2: 0.5713223

n: 100 varx: 1 vary: 1.048809 Prob1: 0.9508 exp1: 0.5708347 Prob2: 0.9508 exp2: 0.5708792

Section 2:

nvec <- c(5, 10, 30, 100)

ratiovec <- c(1, 1.1, 1.5, 2)

for (i in 1:length(nvec)) {

n <- nvec[i]

for (j in 1:length(ratiovec)) {

counter1 <- 0

counter2 <- 0

totalwidth1 <- 0

totalwidth2 <- 0

ratio <- ratiovec[j]

for (i in 1:5000)

{

ux <- 5

uy <- 5

sx <- 1

ratio <- 1.1

sy <- sqrt(ratio \* sx)

x <- rnorm(n, ux, sx)

y <- rnorm(n, uy, sy)

mx <- mean(x)

my <- mean(y)

sdx <- sd(x)

sdy <- sd(y)

bar <- mx - my

####----pooled sample variance

sigma2 <-

((n - 1) \* sdx \* sdx + (n - 1) \* sdy \* sdy) / (n + n - 2)

var <- sigma2 \* (2 / n)

stdError <- sqrt(var)

width1 <- 2 \* qt(0.975, (2 \* n - 2)) \* stdError

totalwidth1 <- totalwidth1 + width1

coff1 <- bar + c(-1, 1) \* (width1 / 2)

# coff1

####---Satterthwaite's approximation

se <- sqrt((sdx ^ 2 / n) + (sdy ^ 2 / n))

v <-

(((sdx ^ 2 / n) + (sdy ^ 2 / n)) ^ 2) / ((sdx ^ 4 / (n ^ 2 \* (n - 1))) +

(sdy ^ 4 / (n ^ 2 \* (n - 1))))

width2 <- 2 \* qt(0.975, v) \* se

totalwidth2 <- totalwidth2 + width2

coff2 <- bar + c(-1, 1) \* (width2 / 2)

# coff2

coff <- c(coff1, coff2)

if (coff1[2] > 0 & coff1[1] < 0) {

counter1 = counter1 + 1

}

if (coff2[2] > 0 & coff[1] < 0) {

counter2 = counter2 + 1

}

}

prob1 <- counter1 / 5000

prob2 <- counter2 / 5000

# prob1

# prob2

exwidth1 <- totalwidth1 / 5000

exwidth2 <- totalwidth2 / 5000

# exwidth1

#exwidth2

cat(

'n:',

n,

'varx:',

sx,

'vary:',

sy,

'Prob1:',

prob1,

'exp1:',

exwidth1,

'Prob2:',

prob2,

'exp2:',

exwidth2,

"\n"

)

}

}

Narration of code:

For different values, same code as 2a is run. By assigning values to different vectors and running them through for loop.

( c )

* Interpretation of b:

As n is increasing, both intervals are giving same values for coverage probabilities. When n is small, Satterthwaite’s approximation is giving better approximation, but it is giving large expected width. On the other hand, when n is large, pooled sample variance is giving bad approximation for coverage probability, but it is giving small expected width.

* When n is large, there is no use to compare expected width, as we will approximately same values. If n is less than 30, then we are getting different values for widths, so when n is not large, it is ok to compare expected width.
* Comment:

When n is large, Pooled sample variance is giving approximately same value for coverage probability, but for n is not large it is not giving perfect outputs, it is underestimating.

In any case, Satterthwaite’s approximation is giving approximately same value for coverage probability, it is not overestimating in any case.

* Yes, it is a good idea.
* In any case, we can use Satterthwaite’s approximation. If n is not large, then use of Pooled Sample Variance will be disadvantageous.
* The answer depends on n. From the summary of 2 b, we can get idea that as n is increasing, approximation is getting better values.

There is no difference for different values of (σY /σX).

( d )

Section 1:

If we change values like

ux = 15

uy = 10

σ2x = 2

Answer:

n: 5 varx: 2 vary: 1.48324 Prob1: 0.948 exp1: 4.982561 Prob2: 0.9518 exp2: 5.217964

n: 5 varx: 2 vary: 1.48324 Prob1: 0.9482 exp1: 4.955454 Prob2: 0.9504 exp2: 5.187771

n: 5 varx: 2 vary: 1.48324 Prob1: 0.9484 exp1: 4.976773 Prob2: 0.9524 exp2: 5.204381

n: 5 varx: 2 vary: 1.48324 Prob1: 0.9492 exp1: 4.974123 Prob2: 0.9524 exp2: 5.203763

n: 10 varx: 2 vary: 1.48324 Prob1: 0.9468 exp1: 3.259756 Prob2: 0.9484 exp2: 3.29935

n: 10 varx: 2 vary: 1.48324 Prob1: 0.947 exp1: 3.256223 Prob2: 0.9482 exp2: 3.295343

n: 10 varx: 2 vary: 1.48324 Prob1: 0.9518 exp1: 3.275317 Prob2: 0.9526 exp2: 3.314507

n: 10 varx: 2 vary: 1.48324 Prob1: 0.9482 exp1: 3.268608 Prob2: 0.9506 exp2: 3.307882

n: 30 varx: 2 vary: 1.48324 Prob1: 0.9512 exp1: 1.811527 Prob2: 0.9518 exp2: 1.815774

n: 30 varx: 2 vary: 1.48324 Prob1: 0.9504 exp1: 1.811812 Prob2: 0.9506 exp2: 1.816103

n: 30 varx: 2 vary: 1.48324 Prob1: 0.9532 exp1: 1.81717 Prob2: 0.9534 exp2: 1.821465

n: 30 varx: 2 vary: 1.48324 Prob1: 0.9538 exp1: 1.807567 Prob2: 0.9542 exp2: 1.811825

n: 100 varx: 2 vary: 1.48324 Prob1: 0.9552 exp1: 0.9815144 Prob2: 0.9552 exp2: 0.9820691

n: 100 varx: 2 vary: 1.48324 Prob1: 0.9514 exp1: 0.9808808 Prob2: 0.9514 exp2: 0.9814356

n: 100 varx: 2 vary: 1.48324 Prob1: 0.9514 exp1: 0.9803285 Prob2: 0.9514 exp2: 0.9808828

n: 100 varx: 2 vary: 1.48324 Prob1: 0.9524 exp1: 0.980706 Prob2: 0.9524 exp2: 0.9812579

As, we can see that for different values, we getting the same situation only. So, conclusions in c does not depend on values of ux, uy and σ2x.

Section 2:

nvec <- c(5, 10, 30, 100)

ratiovec <- c(1, 1.1, 1.5, 2)

for (i in 1:length(nvec)) {

n <- nvec[i]

for (j in 1:length(ratiovec)) {

counter1 <- 0

counter2 <- 0

totalwidth1 <- 0

totalwidth2 <- 0

ratio <- ratiovec[j]

for (i in 1:5000)

{

ux <- 15

uy <- 10

sx <- 2

ratio <- 1.1

sy <- sqrt(ratio \* sx)

x <- rnorm(n, ux, sx)

y <- rnorm(n, uy, sy)

mx <- mean(x)

my <- mean(y)

sdx <- sd(x)

sdy <- sd(y)

bar <- mx - my

####----pooled sample variance

sigma2 <-

((n - 1) \* sdx \* sdx + (n - 1) \* sdy \* sdy) / (n + n - 2)

var <- sigma2 \* (2 / n)

stdError <- sqrt(var)

width1 <- 2 \* qt(0.975, (2 \* n - 2)) \* stdError

totalwidth1 <- totalwidth1 + width1

coff1 <- bar + c(-1, 1) \* (width1 / 2)

# coff1

####---Satterthwaite's approximation

se <- sqrt((sdx ^ 2 / n) + (sdy ^ 2 / n))

v <-

(((sdx ^ 2 / n) + (sdy ^ 2 / n)) ^ 2) / ((sdx ^ 4 / (n ^ 2 \* (n - 1))) +

(sdy ^ 4 / (n ^ 2 \* (n - 1))))

width2 <- 2 \* qt(0.975, v) \* se

totalwidth2 <- totalwidth2 + width2

coff2 <- bar + c(-1, 1) \* (width2 / 2)

# coff2

coff <- c(coff1, coff2)

if (coff1[2] > 5 & coff1[1] < 5) {

counter1 = counter1 + 1

}

if (coff2[2] > 5 & coff[1] < 5) {

counter2 = counter2 + 1

}

}

prob1 <- counter1 / 5000

prob2 <- counter2 / 5000

exwidth1 <- totalwidth1 / 5000

exwidth2 <- totalwidth2 / 5000

cat(

'n:',

n,

'varx:',

sx,

'vary:',

sy,

'Prob1:',

prob1,

'exp1:',

exwidth1,

'Prob2:',

prob2,

'exp2:',

exwidth2,

"\n"

)

}

}

Narration of code:

This is the same code as 2b, just values of ux, uy and σ2x has been changed and in the comparison part, it is compared to 5 = (15-10) = (ux-uy).