ARTICLE TEMPLATE

Supplement material of paper "Estimation for Weibull parameters with generalized progressive hybrid censored data"

Xuanjia Zuo[a], Liang Wang[a], Huizhong Lin[a], Sanku Dey[b], Li Yan[c]

a. School of Mathematics, Yunnan Normal University, Kunming, P.R. China;
b. Department of Statistics, St. Anthony's College, Shillong, India;
c. Department of Administrative Sciences, University of Quebec in Outaouais, Gatineau, Canada.

ARTICLE HISTORY
Compiled November 17, 2021

Program code of simulation

```
#Required packages: nleqslv,snowfall
library(parallel)
library(snowfall)
sfInit(parallel = TRUE, cpus = floor(detectCores()*0.9)) #detect the number of
#computer's core and initialize them
#preparation of the simulation fucntion
#Variable interpretation
#cs:censoring scheme, al:parameter alpha, be:parameter beta
#times:number of repetitions, other variables have the same meaning as in paper
sim<-function(cs,k,T,al,be,a1,b1,a2,b2,times){
    ga<-0.05;pren<-c(18,25,25,50,50,50);prem<-c(15,15,20,40,40,40) #set the signif-
    #-icance level and the sample size pair (n,m)
    #Set the censoring scheme
    if(cs==1){pr<-c(rep(0,14),3)}
    if(cs==2){pr<-c(rep(0,14),10)}
    if(cs==3){pr<-c(rep(0,19),5)}
    if(cs==4){pr<-c(rep(0,39),10)}
    if(cs==5){pr<-c(10,rep(0,39))}
    if(cs==6){pr<-c(5,rep(0,38),5)}
    n<-pren[cs];m<-prem[cs] #select the specific sample size pair
#generate the progressively Type-II censored samples and stored as a data frame
df<-matrix(nrow=times,ncol=m)
for(it in 1:times){
    u<-runif(m);e<-c()
    for(i in 1:m)
    {e[i]<-1/(i+sum(pr[(m-i+1):m]))}
    z<-u^e;us<-c()
```

Corresponding author: Liang Wang. School of Mathematics, Yunnan Normal University, No. 768, Juxian Rd, Chenggong District, Kunming 650500, P.R. China. Email: liang610112@163.com

```
    for(i in 1:m){us[i]<-1-prod(z[(m-i+1):m])}
    xi<-(-log(1-us)/be)^(1/al)
    df[it,]<-xi}
sfLibrary(nleqslv ) #load the package to each core
sfExport('cs','k','T','al','be','a1','b1','a2','b2','ga')
sfExport('pr','n','m') #write needed variables to each core
#Operation function
odd<-sfApply(df,1,function(x){
    xi<-x
    #judge the case
    if(T<xi[k]&xi[k]<xi[m]){case=1}
    if(xi[k]<T&T<xi[m]){case=2}
    if(xi[k]<xi[m]&xi[m]<T){case=3}
    #set parameters based on the type of case
    if(case==1){
        r<-pr[1:k];r[k]<-(m-k)+sum(pr[k:m]);ds<-k
        wa<-function(x){sum((1+r[1:ds])*xi[1:ds]^x)}
        wa1<-function(x){sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds]))}
        wa2<-function(x){sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds])^2)}
        mef<-function(x){
            w<-sum((1+r[1:ds])*xi[1:ds]^x)
            w1<-sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds]))
            1/x+1/ds*sum(log(xi[1:ds]))-w1/w}}
    if(case==2){
        d<-max(which(xi<T));r<-pr[1:d];r[d+1]<-(m-d)+sum(pr[(d+1):m]);ds<-d
        wa<-function(x){sum((1+r[1:ds])*xi[1:ds]^x)+r[d+1]*T^x}
        wa1<-function(x){sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds]))+r[d+1]*T^x*log(T)}
        wa2<-function(x){sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds])^2)+r[d+1]*T^x*log(T)^2}
        mef<-function(x){
            w<-sum((1+r[1:ds])*xi[1:ds]^x)+r[d+1]*T^x
            w1<-sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds]))+r[d+1]*T^x*log(T)
            1/x+1/ds*sum(log(xi[1:ds]))-w1/w}}
    if(case==3){
        r<-pr;ds<-m
        wa<-function(x){w<-sum((1+r[1:ds])*xi[1:ds]^x)}
        wa1<-function(x){sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds]))}
        wa2<-function(x){sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds])^2)}
        mef<-function(x){
            w<-sum((1+r[1:ds])*xi[1:ds]^x)
            w1<-sum((1+r[1:ds])*xi[1:ds]^x*log(xi[1:ds]))
            1/x+1/ds*sum(log(xi[1:ds]))-w1/w}}
    #MLE
x0<-1 #the initial value
repeat{pea<-nleqslv(x0,mef)
if(pea$x>0|x0==100){break};x0<-x0+1}
mea<-pea$x;meb<-ds/wa(mea) #MLE of alpha and beta
#ACI
I<-matrix(nrow=2,ncol=2)
I[1,1]<--ds/mea^2-meb*wa2(mea);I[2,2]<--ds/meb^2
I[1,2]<--wa1(mea);I[2,1]<-I[1,2]
```

```
v<-solve(-I) #approximate variance-covariance matrix
aal<-mea+qnorm(ga/2)*sqrt(v[1,1]);aau<-mea+qnorm(1-ga/2)*sqrt(v[1,1])
laa<-aau-aal #interval length for ACI of alpha
if(al<=aau&al>=aal){t.aa<-1}else{t.aa<-0} #conter of CP
abl<-meb+qnorm(ga/2)*sqrt(v[2,2]);abu<-meb+qnorm(1-ga/2)*sqrt(v[2,2])
lab<-abu-abl #interval length for ACI of alpha
if(be<=abu&be>=abl){t.ab<-1}else{t.ab<-0} #conter of CP
#MCMC algorithm
target1<-function(x){
    a1=a2=b1=b2=0
    x^(a1+ds-1)/(b2+wa(x))^(a2+ds)*exp(-x*(b1-sum(log(xi[1:ds]))))}
target2<-function(x){
    x^(a1+ds-1)/(b2+wa(x))^(a2+ds)*exp(-x*(b1-sum(log(xi[1:ds]))))}
rw=function(nn,x0,target){
    n=1;x_old=x0
X = matrix();X[1]=x_old
    while(n<nn){
        x_new=rgamma(1,shape=x_old)
        p_old=target(x_old)
        p_new=target(x_new)
        a = min(1,(p_new*dgamma(x_old,x_new))/(p_old*dgamma(x_new,x_old)))
        b=rbinom(n=1,size=1,prob=a)
        if(b==1){x_old=x_new}
        X[n+1]=x_old;n=n+1}
    result = list('X'=X);return(result)}
N=10000;burn<-5000;M=N-burn
coa<-c();cob<-c()
coa<-rw((N+100),al,target=target2)$X[101:(N+100)]
for(i in 1:N){cob[i]<-rgamma(1,(a2+ds),(b2+wa(coa[i])))}
pia<-sort(coa[5001:N]);pib<-sort(cob[5001:N])
iea<-mean(pia) #IP bayes for alpha
ieb<-mean(pib) #IP bayes for beta
coa<-rw((N+100),al,target=target1)$X[101:(N+100)]
for(i in 1:N){
a2=b2=0;cob[i]<-rgamma(1,(a2+ds),(b2+wa(coa[i])))}
pna<-sort(coa[5001:N]);pnb<-sort(cob[5001:N])
nea<-mean(pna) #NIP bayes for alpha
neb<-mean(pnb) #NIP bayes for beta
wna<-c();wnb<-c();wia<-c();wib<-c()
for(sk in 1:(M*ga)){
    wna[sk]<-pna[sk+M-(M*ga+1)]-pna[sk]
    wnb[sk]<-pnb[sk+M-(M*ga+1)]-pnb[sk]
    wia[sk]<-pia[sk+M-(M*ga+1)]-pia[sk]
    wib[sk]<-pib[sk+M-(M*ga+1)]-pib[sk]}
kna<-which.min(wna);knb<-which.min(wnb)
kia<-which.min(wia);kib<-which.min(wib)
nal<-pna[kna];nau<-pna[kna+M-(M*ga+1)]
lna<-nau-nal #length for HPD of alpha under NIP
if(al<=nau&al>=nal){t.na<-1}else{t.na<-0} #counter
nbl<-pnb[knb];nbu<-pnb[knb+M-(M*ga+1)]
```

```
lnb<-nbu-nbl #length for HPD of beta under NIP
if(be<=nbu&be>=nbl){t.nb<-1}else{t.nb<-0} #counter
ial<-pia[kia];iau<-pia[kia+M-(M*ga+1)]
lia<-iau-ial #length for HPD of alpha under IP
if(al<=iau&al>=ial){t.ia<-1}else{t.ia<-0}
ibl<-pib[kib];ibu<-pib[kib+M-(M*ga+1)]
lib<-ibu-ibl #length for HPD of beta under IP
if(be<=ibu&be>=ibl){t.ib<-1}else{t.ib<-0}
#return the multithreaded calculation results
return(c(mea,meb,nea,neb,iea,ieb,
laa,lab,lna,lnb,lia,lib,
t.aa,t.ab,t.na,t.nb,t.ia,t.ib))
}) #end of parallel computing
####sorting of simulation results
abma<-mean(abs(odd[1,]-al));abmb<-mean(abs(odd[2,]-be))
abna<-mean(abs(odd[3,]-al));abnb<-mean(abs(odd[4,]-be))
abia<-mean(abs(odd[5,]-al));abib<-mean(abs(odd[6,]-be))
reab<-matrix(nrow=2,ncol=3)
reab[1,1]<-abma;reab[1,2]<-abna;reab[1,3]<-abia
reab[2,1]<-abmb;reab[2,2]<-abnb;reab[2,3]<-abib
rownames(reab)<-c("alpah","beta")
colnames(reab)<-c('MLE','NIP','IP')
msma<-mean((odd[1,]-al)^2);msmb<-mean((odd[2,]-be)^2)
msna<-mean((odd[3,]-al)^2);msnb<-mean((odd[4,]-be)^2)
msia<-mean((odd[5,]-al)^2);msib<-mean((odd[6,]-be)^2)
rems<-matrix(nrow=2,ncol=3)
rems[1,1]<-msma;rems[1,2]<-msna;rems[1,3]<-msia
rems[2,1]<-msmb;rems[2,2]<-msnb;rems[2,3]<-msib
rownames(rems)<-c("alpah","beta")
colnames(rems)<-c('MLE','NIP','IP')
awma<-mean(odd[7,]);awmb<-mean(odd[8,])
awna<-mean(odd[9,]);awnb<-mean(odd[10,])
awia<-mean(odd[11,]);awib<-mean(odd[12,])
reaw<-matrix(nrow=2,ncol=3)
reaw[1,1]<-awma;reaw[1,2]<-awna;reaw[1,3]<-awia
reaw[2,1]<-awmb;reaw[2,2]<-awnb;reaw[2,3]<-awib
rownames(reaw)<-c("alpah","beta")
colnames(reaw)<-c('MLE','NIP','IP')
cpma<-mean(odd[13,]);cpmb<-mean(odd[14,])
cpna<-mean(odd[15,]);cpnb<-mean(odd[16,])
cpia<-mean(odd[17,]);cpib<-mean(odd[18,])
recp<-matrix(nrow=2,ncol=3)
recp[1,1]<-cpma;recp[1,2]<-cpna;recp[1,3]<-cpia
recp[2,1]<-cpmb;recp[2,2]<-cpnb;recp[2,3]<-cpib
rownames(recp)<-c("alpah","beta")
colnames(recp)<-c('MLE','NIP','IP')
####sorting of simulation results end
####output simulation results
print(list('n, m, CS, T, k'=round(c(n,m,cs,T,k),0),'ABs'=round(reab,4),
'MSEs'=round(rems,4),'CPs'=round(recp,4),'AWs'=round(reaw,4)))}
```

```
###example
> sim(cs=1,k=10,T=1.2,al=1,be=0.6,a1=1,b1=1,a2=3,b2=5,times=10000)
$`n, m, CS, T, k`
[1] 18 15  1  1 10

$ABs
        MLE    NIP     IP
alpah 0.3190 0.3167 0.2595
beta  0.2039 0.2029 0.1340

$MSEs
        MLE    NIP     IP
alpah 0.2155 0.2124 0.1306
beta  0.0843 0.0853 0.0297

$CPs
        MLE    NIP     IP
alpah 0.9618 0.9464 0.9604
beta  0.9388 0.9370 0.9669

$AWs
        MLE    NIP     IP
alpah 1.3691 1.3184 1.1948
beta  0.8905 0.8539 0.6888


sfStop() #release the core after completing all the work
```