ORIGINAL RESEARCH ARTICLE

Loss Functions in Restricted Parameter Spaces and Their Bayesian Applications. Supplementary Materials

ARTICLE HISTORY
Compiled January 15, 2019

Examples: Moderate and Large Sample Sizes

While the cases of small sample sizes were used in the main body of the manuscript only ($n = 15$), it is also on interest to investigate how different the corresponding results for moderate and large sample sizes. Below, we revisit four examples considered in the work using moderate ($n = 100$) and large ($n = 1000$) sample sizes. Overall, all stated conclusions about the relative performances of the proposed estimators compared to the standard one stand. While the quantitatively an increased sample sizes leads to smaller differences (due to the improved estimation for all methods), the qualitative patterns are the same.

Estimation of a Probability

Figure 1 presents the MSE, variance and bias for the proposed estimator of a probability, for the Agresti-Coull estimator, and for the squared error loss function estimator using the total sample size $n = 100$.

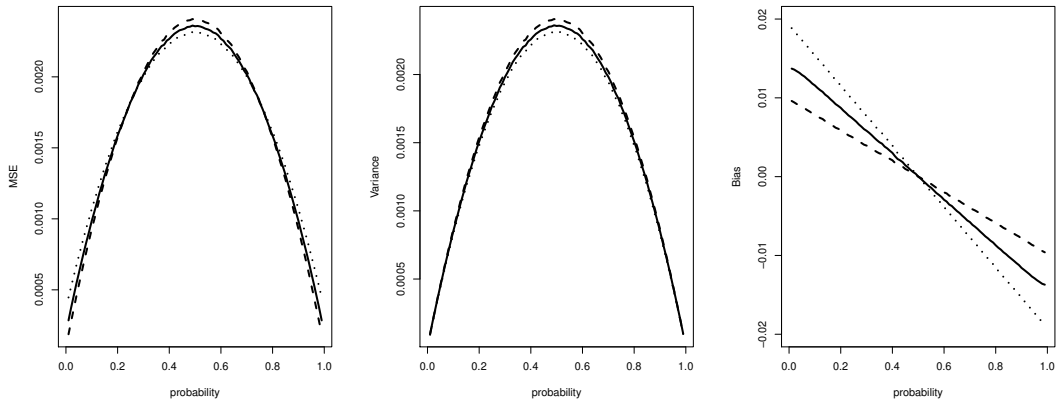

Figure 1. MSE, variance and bias for the restricted symmetric squared error loss function estimator $\hat{p}_{iq}$ (solid), the squared error loss function estimator $\hat{p}_q$ (dashed) and the Agresti-Coull estimator $\hat{p}_{AC}$ (dotted). Results are based on $n = 100$ observations and 10^4 simulations.

As expected, the magnitude of differences is now smaller for all considered characteristics. Specifically, the MSE associating with $\hat{p}_q$ is much closer to those of $\hat{p}_{AC}$ and $\hat{p}_{iq}$. Nevertheless, the proposed estimator $\hat{p}_{iq}$, again, performs better (in terms of

the MSE) than the Bayes estimator obtained using the squared error loss function $\hat{p}_q$ in the interval $\theta \in (0.2, 0.8)$, and than Agresti-Coull estimator $\hat{p}_{AC}$ in the interval $\theta \in (0.8, 1)$.

Figure 2 presents the MSE, variance and bias for the proposed estimator of a probability, for the Agresti-Coull estimator, and for the squared error loss function estimator using the total sample size $n = 1000$.

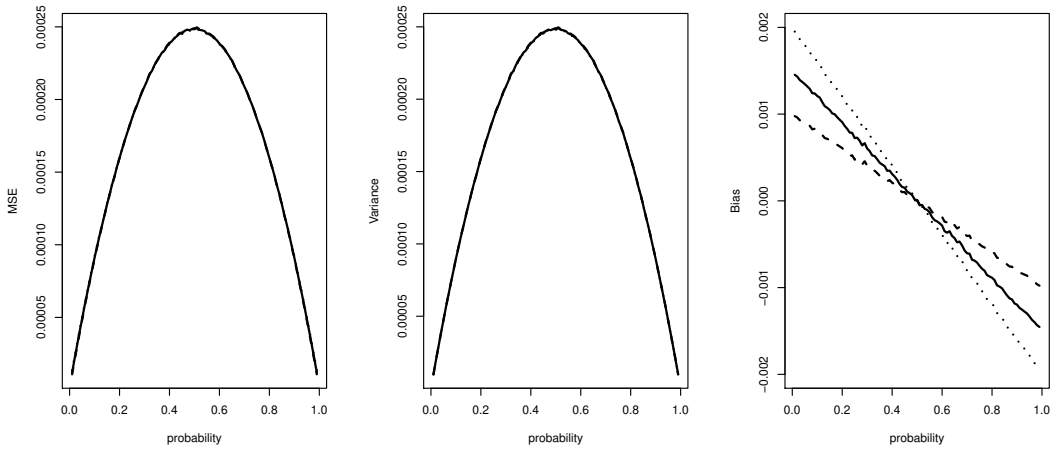

Figure 2. MSE, variance and bias for the restricted symmetric squared error loss function estimator $\hat{p}_{iq}$ (solid), the squared error loss function estimator $\hat{p}_q$ (dashed) and the Agresti-Coull estimator $\hat{p}_{AC}$ (dotted). Results are based on $n = 1000$ observations and 10^4 simulations.

For even large sample sizes, all three estimators have same characteristics and the curves corresponds to the MSE and Variance are not distinguishable. Also, the differences in bias are of smaller magnitude. Overall, the proposed estimator does provide particular benefits in the intervals of θ for small and moderate sample sizes while performing similarly to alternatives for the large sample sizes.

Restricted Estimation of a Normal Distribution Mean

Figure 3 presents the MSE associated with the four considered estimators J, U_1, U_2, U_2', for each value of μ in intervals (-a,a) using the total sample sizes $n = 100$, and $a = 2$ and $a = 4$.

The differences between all estimators are now smaller but the proposed estimators U_2, U_2' still provide benefit over U_1 for wide intervals in both cases. Again, the benefit is larger for a narrow interval of values. Interestingly, estimators U_1 and U_2' now do not have a rising "tails" in the MSE on the bounds as the increased sample size allows to improve the estimation. At the same time, the MSE for estimator U_2 still increases on the bounds. However, in contrast to the performance using $n = 15$, the MSE associated with the estimator U_2 is now never above the MSE of the estimator J.

Figure 4 presents the MSE associated with the four considered estimators J, U_1, U_2, U_2', for each value of μ in intervals (-a,a) using the total sample sizes $n = 1000$, and $a = 2$ and $a = 4$.

For the large sample size, the same qualitative pattern stands while the differences in the MSE is smaller. Importantly, the MSE of the estimator U_2 does not increase on the tails and behaves similarly to the estimator U_1 and U_2'.

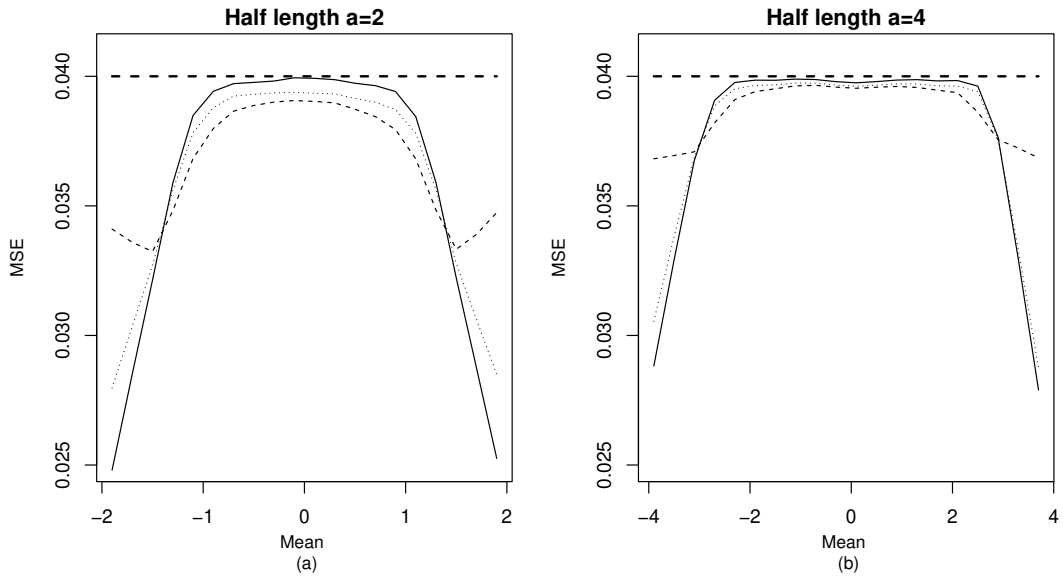

Figure 3. MSE corresponding to different values of the restricted mean parameter μ with (a) $a = 2$ and (b) $a = 4$ and the Bayes estimator U_1 (solid), U_2 (dashed) and U_2' (dotted) and simple mean estimator J (solid dashed). Results are based on $n = 100$ and 10^4 replications.

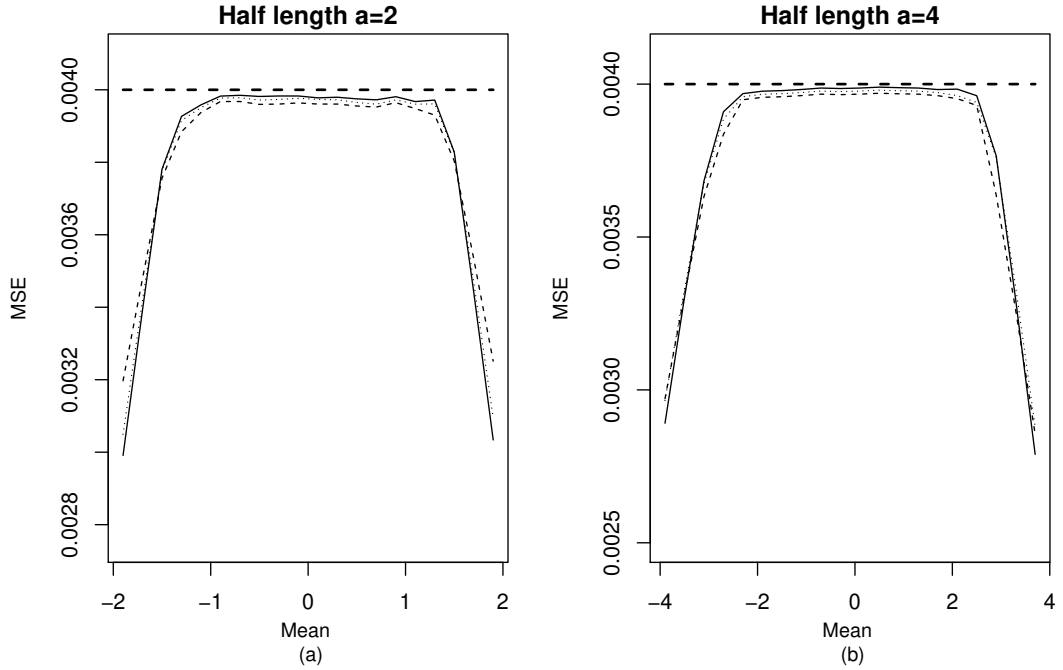

Figure 4. MSE corresponding to different values of the restricted mean parameter μ with (a) $a = 2$ and (b) $a = 4$ and the Bayes estimator U_1 (solid), U_2 (dashed) and U_2' (dotted) and simple mean estimator J (solid dashed). Results are based on $n = 1000$ and 10^4 replications.

Bayesian Estimation of the Parameters of a Gamma Distribution

The differences in the MSEs for both parameters for various values of the true parameters α_1 and α_2 are given in Figure 5 for the sample size $n = 100$ and in Figure 6 for the sample size $n = 1000$.

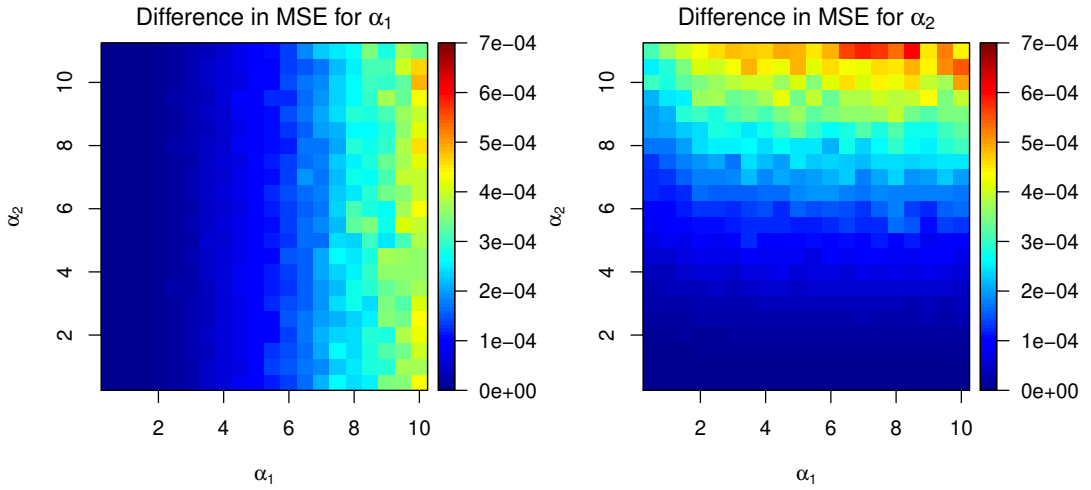

Figure 5. Difference in the MSEs for parameters α_1 and α_2 for their different true values and using Bayes estimator under the squared error loss function and Bayes estimator under $L_{sq}^{(2)}$. Results are based on $n = 100$ and 10^4 replications.

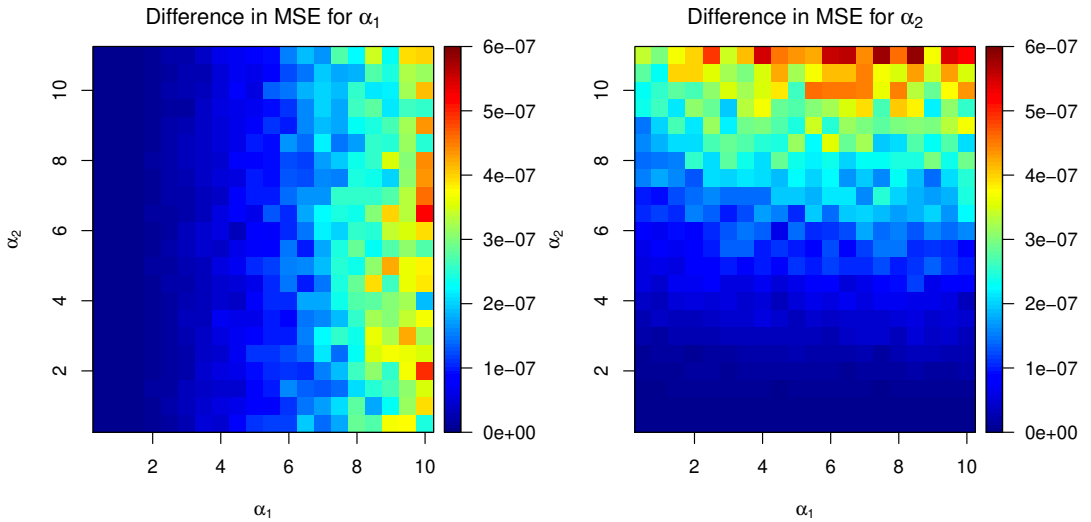

Figure 6. Difference in the MSEs for parameters α_1 and α_2 for their different true values and using Bayes estimator under the squared error loss function and Bayes estimator under $L_{sq}^{(2)}$. Results are based on $n = 1000$ and 10^4 replications.

While the differences in the MSE is smaller as expected, the proposed estimator corresponds to lower values of the MSE for all considered values of the true parameter α_1 and α_2. This means that the proposed estimator can provide benefits in terms of the MSE for moderate and large sample sizes as well.

Bayesian Estimation of the Parameters of a Weibull Distribution

The difference between the MSE of estimators for positive parameters λ, ν are given in Table 1 for the sample size $n = 100$, and in Table 2 for the sample size $n = 1000$. In both cases, the MSE for ν are scaled by $\frac{1}{\nu^\lambda}$ and by n to obtain the results on a similar scale for various parameters.

Table 1. Difference in the MSEs for parameters λ (upper lines) and ν (lower lines) for their different true values and using Bayes estimator under the squared error loss function and Bayes estimator under $L_1^{(m)}$. The MSE for ν are scaled by $1/\nu^\lambda$ and $n = 100$ to obtain the results on a similar scale for various parameters. Results are based on 10^4 replications.

	$\nu = 1$	$\nu = 2$	$\nu = 5$	$\nu = 10$	$\nu = 15$
$\lambda = 1$	0.0105	0.0096	0.0062	0.0122	0.0103
	0.0133	0.0048	0.0208	0.0132	0.0188
$\lambda = 2$	0.0533	0.0417	0.0365	0.0430	0.0479
	0.0144	0.0049	0.0276	0.0285	0.0243
$\lambda = 3$	0.0943	0.1072	0.1138	0.1026	0.1095
	0.0143	0.0091	0.0237	0.0172	0.0101
$\lambda = 4$	0.1703	0.1830	0.1730	0.1852	0.1771
	0.0081	0.0166	0.0196	0.0065	0.0030
$\lambda = 5$	0.3456	0.2944	0.2603	0.2554	0.2445
	0.0094	0.0254	0.0110	0.0021	0.0008

Table 2. Difference in the MSEs for parameters λ (upper lines) and ν (lower lines) for their different true values and using Bayes estimator under the squared error loss function and Bayes estimator under $L_1^{(m)}$. The MSE for ν are scaled by $1/\nu^\lambda$ and $n = 1000$ to obtain the results on a similar scale for various parameters. Results are based on 10^4 replications.

	$\nu = 1$	$\nu = 2$	$\nu = 5$	$\nu = 10$	$\nu = 15$
$\lambda = 1$	0.0002	0.0002	0.0023	0.0005	0.0022
	0.0022	0.0009	0.0022	0.0022	0.0010
$\lambda = 2$	0.0083	0.0061	0.0026	0.0069	0.0077
	0.0004	0.0003	0.0026	0.0017	0.0017
$\lambda = 3$	0.0126	0.0119	0.0062	0.0014	0.0178
	0.0006	0.0003	0.0026	0.0017	0.0006
$\lambda = 4$	0.0071	0.0105	0.0150	0.0054	0.0152
	0.0020	0.0018	0.0020	0.0006	0.0002
$\lambda = 5$	0.0162	0.0220	0.0117	0.0154	0.0013
	0.0013	0.0034	0.0010	0.0001	0.0000

While the differences in the MSEs are smaller for both parameters, they are still positive for all considered true values of λ and ν. It follows that the proposed Bayes estimator leads to a smaller MSE than the estimator corresponding to the squared error loss function for the moderate and large sample sizes.

Codes

Example 1: Estimation of a Probability

```
probability<-seq(0.01,0.99,0.01) # defining the vector of true probability values
nsims<-1000000000 # defining a number of simulations
N<-15 # defining the total sample size

# Creating matrices to store output
variance<-variance.iq<-variance.AC<-res<-res.iq<-res.AC
<-mat.or.vec(length(probability),1)
result<-result.iq<-result.AC<-mat.or.vec(nsims,1)
P<-P.iq<-P.AC<-mat.or.vec(nsims,length(probability))

# Running simulations
for (k in 1:length(probability)){
for (z in 1:nsims){
    x<-sum(rbinom(N, 1, probability[k])) # Generating responses

    # Computing estimators
    P[z,k]<-(x+1)/(N+2)  # Bayesian using the squared error loss function
    P.AC[z,k]<-(x+2)/(N+4)  # Agresti-Coull estimator
    P.iq[z,k]<-1/(1+sqrt(((N-x+1)*(N-x+2))/((x+1)*(x+2)))) # Proposed estimator

    result[z]<-(P[z,k]-probability[k])^2
    result.AC[z]<-(P.AC[z,k]-probability[k])^2
    result.iq[z]<-(P.iq[z,k]-probability[k])^2
    }

  # Computing MSE
  res[k]<-mean(result)
  res.iq[k]<-mean(result.iq)
  res.AC[k]<-mean(result.AC)}

  # Computing Variance
for (i in 1:length(probability)){
variance[i]<-var(P[,i])
variance.iq[i]<-var(P.iq[,i])
variance.AC[i]<-var(P.AC[,i])}

  # Computing Bias
bias<-colMeans(P)-probability
bias.iq<-colMeans(P.iq)-probability
bias.AC<-colMeans(P.AC)-probability

# Computing coverage probabilities

# Normal Approximation
# Creating matrices to store output
variance.each<-variance.each.iq<-variance.each.AC<-coverage<-coverage.iq
```

```
<-coverage.AC<-mat.or.vec(nsims,length(probability))
kappa<-1.96

for (j in 1:length(probability)){
for (z in 1:nsims){

# Computing the standard error for the confidence interval

    se<-variance.each[z,j]<-sqrt(P[z,j]*(1-P[z,j])/N)
    se.iq<-variance.each.iq[z,j]<-sqrt(P.iq[z,j]*(1-P.iq[z,j])/N)
    se.AC<-variance.each.AC[z,j]<-sqrt(P.AC[z,j]*(1-P.AC[z,j])/N)

if (P[z,j]-kappa*se< probability[j] & P[z,j]+kappa*se > probability[j]){
coverage[z,j]<-1}
if (P.iq[z,j]-kappa*se.iq< probability[j] & P.iq[z,j]+kappa*se.iq > probability[j]{
coverage.iq[z,j]<-1}
if (P.AC[z,j]-kappa*se.AC < probability[j] & P.AC[z,j]+kappa*se.AC > probability[j]{
coverage.AC[z,j]<-1}}
    }

# Wilson and Delta Approximation

#Defining a function to compute Wilson and Delta method approaches
wilson<-function(p,N,kappa){
  x<-p*(N+4)-2
  y1<-sqrt(N)/(N+kappa^2)
  hat<-x/N
  y2<-sqrt(hat*(1-hat)+kappa^2/(4*N))
  y<-y1*y2
  return(y)}
  brain<-function(p,n){y1<-n*(n+3)^2*(1-p)*p*(4-2*n^2*p^2+n*(3+2*n*p))^2
  y2<-1+sqrt(((n-n*p+1)*(n-n*p+2))/((n*p+1)*(n*p+2)))
  y3<-4*(1+n-n*p)*(2+n-n*p)*(1+n*p)^3*(2+n*p)^3*y2^4
  y<-y1/y3
  return(y)}

  # Creating matrices to store output
variance.each<-variance.each.iq<-coverage<-coverage.iq<-coverage.AC<-
variance.each.AC<-mat.or.vec(nsims,length(probability))

for (j in 1:length(probability)){
  for (z in 1:nsims){
se.iq<-variance.each.iq[z,j]<-sqrt(brain(P.iq[z,j],n=N))
se.AC<-variance.each.AC[z,j]<-wilson(P.AC[z,j],N=N,kappa=2)
if (P.iq[z,j]-kappa*se.iq< probability[j] & P.iq[z,j]+kappa*se.iq > probability[j]{
    coverage.iq[z,j]<-1}
if (P.AC[z,j]-kappa*se.AC < probability[j] & P.AC[z,j]+kappa*se.AC > probability[j{
    coverage.AC[z,j]<-1}}}
```

Example 2: Restricted Estimation of a Normal Distribution Mean

```
# Definting a function to compute the proposed estimator for a choice of (a,b)
estimator<-function(x,a,b){x1<-mean(x)
x2<-mean(x^2)
y1<-2*a*b-2*x2+sqrt((2*x2-2*a*b)^2-4*(a+b-2*x1)*(2*a*b*x1-(a+b)*x2))
y2<-2*(a+b-2*x1)
y<-y1/y2
return(y)}

# Defining the model
model<-function(){for (i in 1:N){X[i] ~ dnorm(mu, 1/4)}
                  mu ~ dunif(-bound, bound)}
production.itr<-burnin.itr<-5000
model.file<-model
path.model<-file.path(tempdir(), "model.file.txt")
R2WinBUGS::write.model(model.file,path.model)

# Defining parameters
sigma<-2 # Defining sigma
N<-15    # Defining sample size
bound<-2 # Defining bound
MU<-seq(-bound,bound,0.10) # Defining a sequence of true parameters
nsims<-1000000 # Defining number of parameters (Warning: long computational time)

# Creating matrices to store output
outcomes<-outcomes.my<-outcomes.trun<-mat.or.vec(nsims,length(MU))
result.final<-mat.or.vec(3,length(MU))
difference.classic<-difference.new<-difference.new2<-mat.or.vec(nsims,1)

# Running simulations
for (i in 1:length(MU)){mu<-MU[i]
for (z in 1:nsims){X<-rnorm(N,mu,sigma) # Generating the response
# Computing posterior samples
jagsobj<-rjags::jags.model(path.model,data= list('X' = X,'N' =N,'bound'=bound),
n.chains=2,quiet=TRUE)
update(jagsobj,n.iter=burnin.itr,progress.bar="none")
tt<-rjags::jags.samples(jagsobj, "mu", n.iter=production.itr/2,progress.bar="none")
t<- c(tt$mu)
x.classic<-mean(t) # Estimator under Sq. Error Loss and Uniform Prior
x.new<-estimator(t,-bound,bound) # Proposed estimator
x.new2<-estimator(t,-bound*1.25,bound*1.25) # Proposed estimator for wider interval
difference.classic[z]<-(x.classic-mu)^2
difference.new[z]<-(x.new-mu)^2
difference.new2[z]<-(x.new2-mu)^2}
# Computing MSE
result.final[1,i]<-mean(difference.classic)
result.final[2,i]<-mean(difference.new)
result.final[3,i]<-mean(difference.new2)}
```

Example 3: Bayesian Estimation of the Parameters of a Gamma Distribution

```
# Defining functions to compute the estimator using Lindley (1980) approach [18]
# First parameter of gamma distribution
alpha.lind<-function(alpha,lambda,n,a,b,c,d){l30<-(-n)*psigamma(alpha, deriv = 2)
  l03<-2*n*alpha/(lambda^3)
  l21<-0
  l12<-(-n)/(lambda^2)
  p1<-(c-1)/alpha-d
  p2<-(a-1)/lambda-b
  t11<-alpha/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t12<-t21<-lambda/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t22<-((lambda^2)*psigamma(alpha, deriv = 1))/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  A<-0
  B12<-t11^2
  B21<-t21*t22
  C12<-3*t11*t12
  C21<-t22*t11+2*t21^2
  A12<-t11
  A21<-t12
  y<-alpha+(1/2)*(A+l30*B12+l03*B21+l21*C12+l12*C21)+p1*A12+p2*A21
  return(y)}

# Second parameter of gamma distribution
lambda.lind<-function(alpha,lambda,n,a,b,c,d){
l30<-(-n)*psigamma(alpha, deriv = 2)
  l03<-2*n*alpha/(lambda^3)
  l21<-0
  l12<-(-n)/(lambda^2)
  p1<-(c-1)/alpha-d
  p2<-(a-1)/lambda-b
  t11<-alpha/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t12<-t21<-lambda/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t22<-((lambda^2)*psigamma(alpha, deriv = 1))/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  A<-0
  B12<-t12*t11
  B21<-t22^2
  C12<-t11*t22+2*t12^2
  C21<-3*t22*t21
  A12<-t21
  A21<-t22
  y<-lambda+(1/2)*(A+l30*B12+l03*B21+l21*C12+l12*C21)+p1*A12+p2*A21
  return(y)}

 # Squared first parameter of gamma distribution
 alpha.lind.square<-function(alpha,lambda,n,a,b,c,d){
 l30<-(-n)*psigamma(alpha, deriv = 2)
  l03<-2*n*alpha/(lambda^3)
  l21<-0
```

```
  l12<-(-n)/(lambda^2)
  p1<-(c-1)/alpha-d
  p2<-(a-1)/lambda-b
  t11<-alpha/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t12<-t21<-lambda/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t22<-((lambda^2)*psigamma(alpha, deriv = 1))/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  A<-0
  B12<-2*alpha*t11^2
  B21<-2*alpha*t21*t22
  C12<-2*alpha*3*t11*t12
  C21<-2*alpha*(t22*t11+2*t21^2)
  A12<-2*alpha*t11
  A21<-2*alpha*t12
  y<-alpha^2+(1/2)*(A+l30*B12+l03*B21+l21*C12+l12*C21)+p1*A12+p2*A21
  return(y)}

 # Squared second parameter of gamma distribution
lambda.lind.square<-function(alpha,lambda,n,a,b,c,d){
l30<-(-n)*psigamma(alpha, deriv = 2)
  l03<-2*n*alpha/(lambda^3)
  l21<-0
  l12<-(-n)/(lambda^2)
  p1<-(c-1)/alpha-d
  p2<-(a-1)/lambda-b
  t11<-alpha/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t12<-t21<-lambda/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  t22<-((lambda^2)*psigamma(alpha, deriv = 1))/(n*(alpha*psigamma(alpha, deriv = 1)-1))
  A<-0
  B12<-2*lambda*t12*t11
  B21<-2*lambda*t22^2
  C12<-2*lambda*(t11*t22+2*t12^2)
  C21<-2*lambda*3*t22*t21
  A12<-2*lambda*t21
  A21<-2*lambda*t22
  y<-lambda^2+(1/2)*(A+l30*B12+l03*B21+l21*C12+l12*C21)+p1*A12+p2*A21
  return(y)}
require(MASS)
# Defining the grid of parameters and prior distribution
A.grid<-seq(0.5,10,0.5) # First parameter
B.grid<-seq(0.5,11,0.5) # Second parameter
prior<-0.0001

N<-15 # Defining the sample size
nsims<-1000000 # Defining the number of simulations
# Creating matrices to store output
result.onemore.A<-result.new.A<-result.standard.A<-result.standard.B
result.new.B<-result.onemore.B<-mat.or.vec(length(A.grid),length(B.grid))
MSE.standard.A<-MSE.new.A<-MSE.standard.B<-MSE.new.B<-mat.or.vec(nsims,1)

for(u in 1:length(A.grid)){
```

```
A<-A.grid[u]
for (v in 1:length(B.grid)){
B<-B.grid[v]
for (z in 1:nsims){X<-rgamma(N, shape=A, rate = B) # Generating responses
MLE<-fitdistr(X, 'gamma')  # Obtaining MLE estimates required by Lindley (1980)
A.mle<-MLE$estimate[1]
B.mle<-MLE$estimate[2]

# Computing standard estimators under Squared Error Loss
A.standard<-alpha.lind(A.mle,B.mle,n=N,a=prior,b=prior,c=prior,d=prior)
B.standard<-lambda.lind(A.mle,B.mle,n=N,a=prior,b=prior,c=prior,d=prior)
MSE.standard.A[z]<-(A.standard-A)^2
MSE.standard.B[z]<-(B.standard-B)^2

# Computing estimators under proposed loss function
A.new<-sqrt(alpha.lind.square(A.mle,B.mle,n=N,a=prior,b=prior,c=prior,d=prior))
B.new<-sqrt(lambda.lind.square(A.mle,B.mle,n=N,a=prior,b=prior,c=prior,d=prior))
MSE.new.A[z]<-(A.new-A)^2
MSE.new.B[z]<-(B.new-B)^2}

# Computing MSE
    result.standard.A[u,v]<-mean(MSE.standard.A)
    result.new.A[u,v]<-mean(MSE.new.A)
    result.standard.B[u,v]<-mean(MSE.standard.B)
    result.new.B[u,v]<-mean(MSE.new.B)}}
```

Example 4: Bayesian Estimation of the Parameters of a Weibull Distribution

```
# Defining the function to compute estimator under proposed loss function
estimator<-function(x,k){x1<-mean(x^k)
x2<-mean(x^(-k))
y<-(x1/x2)^(1/(2*k))
return(y)}

# Defining the model
my<-function(){for (i in 1:N){X[i] ~dweib(parameter[1],parameter[2])
    }
  parameter[1] ~ dgamma(0.0001, 0.0001)
  parameter[2] ~ dgamma(0.0001, 0.0001)
  }
burnin.itr<-production.itr<-5000
model.file<-my
path.model<-file.path(tempdir(), "model.file.txt")
R2WinBUGS::write.model(model.file,path.model)

N<-15 # Defining the sample size
nsims<-10000 # Specifying number of simulations (Warning: long computational time)

# Defining grid of parameters
SHAPE.ALL<-c(1,2,3,4,5)
SCALE.ALL<-c(0.5,1,5,10,15)

# Creating matrices to store output
result1.classic<-result1.new<-result1.new2<-result1.new3<-result1.new4<-
result2.classic<-result2.new<-result2.new2<-result2.new3<-result2.new4<-
mat.or.vec(length(SHAPE.ALL),length(SCALE.ALL))
difference1.classic<-difference1.new<-difference2.classic<-difference2.new<-mat.or.vec(n

for (u in 1:length(SHAPE.ALL)){SHAPE<-SHAPE.ALL[u]
for (v in 1:length(SCALE.ALL)){SCALE<-SCALE.ALL[v]
for (z in 1:nsims){

X<-rweibull(N, shape=SHAPE, scale = SCALE) # Generating responses

# Computing samples of posterior distribution
jagsobj<-rjags::jags.model(path.model,data= list('X' = X,'N' = N),
n.chains=2,quiet=TRUE)
update(jagsobj,n.iter=burnin.itr,progress.bar="none")
tt<-rjags::jags.samples(jagsobj, "parameter",
n.iter=production.itr,progress.bar="none")
t1<- c(tt$parameter[1,,])
t2<- c(tt$parameter[2,,])

x1.classic<-mean(t1) # Standard Estimators under Squared Error Loss
x2.classic<-mean(t2)
```

```
x1.new<-estimator(t1,k=1) # Proposed Estimators
x2.new<-estimator(t2,k=1)

difference1.classic[z]<-(x1.classic-SHAPE)^2
difference1.new[z]<-(x1.new-SHAPE)^2
difference2.classic[z]<-(x2.classic-(1/SCALE)^SHAPE)^2
difference2.new[z]<-(x2.new-(1/SCALE)^SHAPE)^2}

# Computing MSE
result1.classic[u,v]<-mean(difference1.classic)
result1.new[u,v]<-mean(difference1.new)
result2.classic[u,v]<-mean(difference2.classic)
result2.new[u,v]<-mean(difference2.new)
}
}
```

Application to the paediatric clinical

```
# Defining the function for the proposed estimator on (a,b) interval
estimator<-function(x,n,a,b){
x1<-(x+1)/(n+2)
x2<-((x+1)*(x+2))/((n+2)*(n+3))
y1<-2*a*b-2*x2+sqrt((2*x2-2*a*b)^2-4*(a+b-2*x1)*(2*a*b*x1-(a+b)*x2))
y2<-2*(a+b-2*x1)
y<-y1/y2
return(y)}

# Computing the proposed estimator
proposed<-estimator(x=19,n=97,a=0.10,b=1)

# Computing the conventional estimator
standard<-19/97

# Finding the sample sizes required to achieve 90% power under both estimators
power.prop.test(p1 = .50, p2 = standard, power = .90,
alternative="one.sided", sig.level=.05)

power.prop.test(p1 = .50, p2 = proposed, power = .90,
alternative="one.sided", sig.level=.05)
```