Comparison study 1 - 0.4 Table 4

2024-07-24

```
################################################
########## LIBRARIES ###########################
################################################
options(scipen = 999)
#options(digits = 22)
library(sda)
```

```
## Loading required package: entropy
```

```
## Loading required package: corpcor
```

```
## Loading required package: fdrtool
```

```
library(statmod)
library(dlbayes)
library(GIGrvg)
library(fitdistrplus)
```

```
## Loading required package: MASS
```

```
## Loading required package: survival
```

```
library(invgamma)
library(extraDistr)
```

```
##
## Attaching package: 'extraDistr'
```

```
## The following objects are masked from 'package:invgamma':
##
##     dinvchisq, dinvgamma, pinvchisq, pinvgamma, qinvchisq, qinvgamma,
##     rinvchisq, rinvgamma
```

```
library(rootSolve)
library(pracma)
```

```
##
## Attaching package: 'pracma'
```

```
## The following objects are masked from 'package:rootSolve':
##
##     gradient, hessian
```

```
## The following objects are masked from 'package:Matrix':
##
##     expm, lu, tril, triu
```

```
library(truncnorm)
library(fGarch)
```

```
## NOTE: Packages 'fBasics', 'timeDate', and 'timeSeries' are no longer
```

```
## attached to the search() path when 'fGarch' is attached.
##
## If needed attach them yourself in your R script by e.g.,
##         require("timeSeries")
```

```
library(BGLR)
library(spate)
```

```
## Loading required package: mvtnorm
```

```
library(readr)
```

```
##
## Attaching package: 'readr'
```

```
## The following object is masked from 'package:spate':
##
##     cols
```

```
library(Matrix)
library(sna)
```

```
## Loading required package: statnet.common
```

```
##
## Attaching package: 'statnet.common'
```

```
## The following objects are masked from 'package:base':
##
##     attr, order
```

```
## Loading required package: network
```

```
##
## 'network' 1.18.2 (2023-12-04), part of the Statnet Project
## * 'news(package="network")' for changes since last version
## * 'citation("network")' for citation information
## * 'https://statnet.org' for help, support, and other information
```

```
## sna: Tools for Social Network Analysis
## Version 2.7-2 created on 2023-12-05.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
```

```
##
## Attaching package: 'sna'
```

```
## The following object is masked from 'package:pracma':
##
##     cutpoints
```

```
library(e1071)
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:pracma':
##
##     sigmoid
```

```r
library(bayesplot)
```

```
## This is bayesplot version 1.11.1
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##    * Does _not_ affect other ggplot2 plots
##    * See ?bayesplot_theme_set for details on theme setting
```

```r
library(ggplot2)
library(rstanarm)
```

```
## Loading required package: Rcpp
## This is rstanarm version 2.32.1
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!
## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.
## - For execution on a local, multicore CPU with excess RAM we recommend calling
##   options(mc.cores = parallel::detectCores())
##
## Attaching package: 'rstanarm'
## The following object is masked from 'package:pracma':
##
##     logit
```

```r
#####################################################
########## GLOBAL FUNCTIONS ######################
#####################################################
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}


#####################################################
########## DATA SIMULATION ######################
#####################################################
niter = 5000
burn0 = 2000
nsim = 5
p = 100
n = 50
plots_ind = F

#Store results for sim
rsim_SN <- matrix(NA, nsim, p)
rsim_N <- matrix(NA, nsim, p)

skew0 <- 0.4
delta_0 <- sign(skew0)*sqrt(22/7/2*(abs(skew0))^(2/3)/((abs(skew0))^(2/3)+((4-22/7)/2)^(2/3)))
a <- 0.4 #dlhyper(x, y)
```

```r
hyper <- a #dlhyper(x, y)
a0 <- 1
b0 <- 0.5



 corr_mat <- matrix(NA,nrow = p,ncol = p)

 for(i in 1:p){
   for(j in 1:p){

     corr_mat[i,j] <- abs(i-j)

   }
 }

 c_mat <- 0.5^corr_mat
 s_mat <- diag(p)

 x <- {}

 for(i in 1:n){

   x <- rbind(x,mvrnorm(n=1, mu=rep(0,p), Sigma=c_mat))

 }

 x <- as.matrix(x)
colnames(x) <- paste0("X", rep(1:p))




##################################################
########## LATENT STRUCTURE TO SIM Y #############
##################################################


Su_0<- sqrt(1-((2/pi)*(delta_0^2)))
Eu_0 <- sqrt(2/pi)*delta_0
tau_0 <- stats::rgamma(n = 1, shape = p * hyper, rate = 1/2) #can test varying rate
phi_0 <- c(LaplacesDemon::rdirichlet(n = 1, alpha = rep(hyper,p)))
psi_0 <- stats::rexp(p, rate = 1/2) #can vary the parameter
s2_0 <- 1
#s2b_0 <- mean(phi_0^2 * psi_0 * tau_0^2)

#choosing uniform prior for sigma2 and delta

beta_0 <- c(runif(10,min = 0,max = 0),
            runif(5,min = 4,max = 5),
            runif(20,min = 0,max = 0),
            runif(5,min = 2,max = 3),
```

```
            runif(p-40,min = 0,max = 0))
#beta_0 <- c(rep(0,10),runif(5,min = 0.5,max = 1),rep(0,25),runif(5,min = 0.5,max = 1),rep(0,5))

for (isim in 1:nsim){
print(paste0("Simulation ", isim))
t_0 <- truncnorm::rtruncnorm(n,a = 0,b=Inf,mean = 0,sd=1)

mu_y0 <- as.matrix(x)%*%beta_0 + (sqrt(s2_0)/Su_0)*(delta_0*t_0 - Eu_0)
sd_y0 <- (s2_0*sqrt((1-delta_0)^2))/Su_0
y_0 <- rnorm(n, mean = mu_y0, sd = sd_y0)
y <- y_0
plot(density(y_0))
skewness(y)
plot(density(y - as.matrix(x)%*%beta_0))
skewness(y - as.matrix(x)%*%beta_0)


#write.csv(as.data.frame(cbind(x,y)),"~/Documents/Arno_sim1/simdat.csv", row.names = FALSE)


#simdat <- read_csv("~/Documents/Arno_sim1/simdat.csv",col_names = FALSE)

simdat = cbind(x,y)

#y <- as.matrix(simdat[,ncol(simdat)])
#x <- as.matrix(simdat[,1:(ncol(simdat)-1)])

########################################################
########## INITIAL VALUES AND PREAMBLE #############
########################################################


### SENSITIVITY ANALYSIS ###

#init val
psi_init = rep(1,p)
phi_init = rep(1,p)/p
beta_init <- rep(2,p)
t_init = rep(mean(truncnorm::rtruncnorm(100*n,a = 0,b=Inf,mean = 0,sd=1)),n)
tau_init <- 0.5*(p*a - 1)
delta_init <- 0.5
#s2e_init <- 5


psi <- psi_init
phi <- phi_init
tau <- tau_init
beta <- beta_init
t <- t_init
delta <- delta_init
s2e <- 1

beta_mat <- matrix(NA,nrow = niter, ncol = p)
```

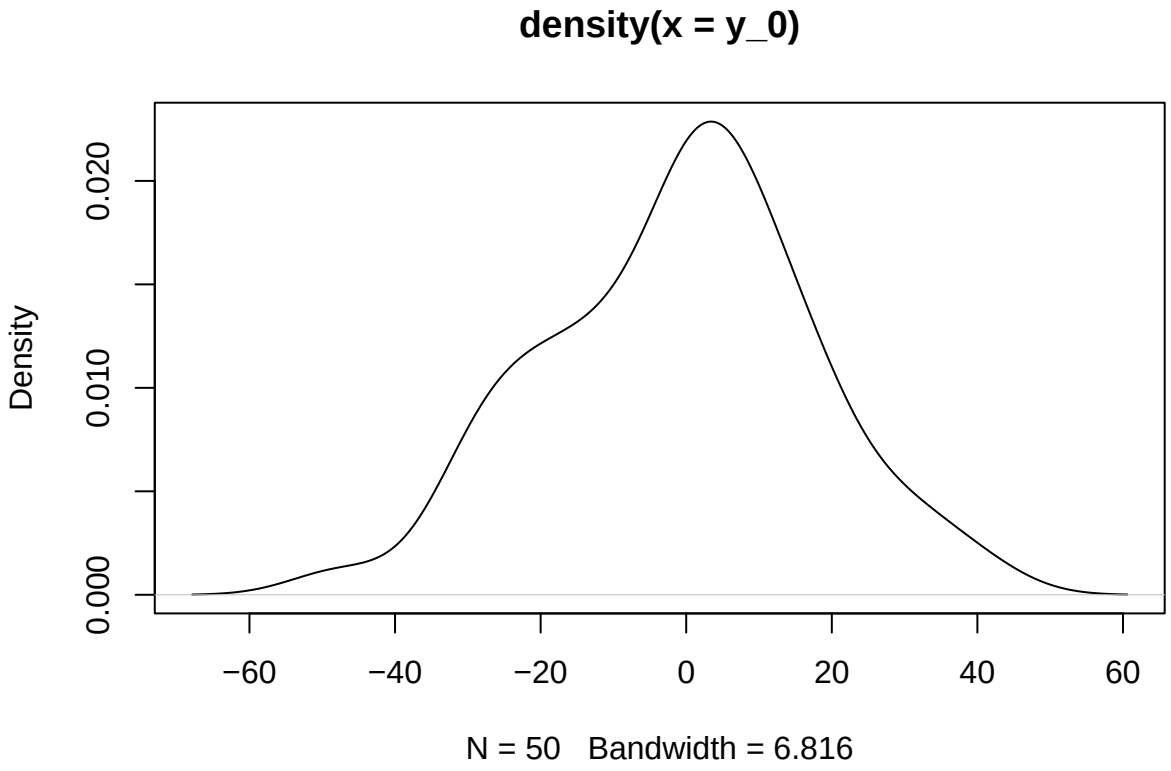
```
tm <- {}
s2m <- {}
dm <- {}
phm <- {}
pm <- {}
tam <- {}


#########################
##### GIBBS SAMPLER #####
#########################

for(iter in 1:niter) {

  ##### BETA #####

  Su <- Su <- sqrt(1-((2/pi)*(delta^2)))
  Eu <- sqrt(2/pi)*delta

  for(j in 1:p){

    c1 <- sum(x[,j]^2) + ((1-delta^2)*s2e)/((Su^2) *(psi[j]*(phi[j]^2)*(tau^2)))
    c2 <- sum(x[,j]*y) - sum(x[,j]*x[,-j]%*%beta[-j]) - sum(x[,j]*delta*t*sqrt(s2e)/(Su)) + sum(x[,j]*Eu
    mu_beta <- c2/c1
    sig2_beta <- ((1-delta^2)*s2e)/((Su^2) * c1)

    beta[j] <- stats::rnorm(1,mean = mu_beta, sd = sqrt(sig2_beta))

  }

  beta[which(abs(beta) <= (0.0000000000000000000000000000000000000000001))] <- (0.0000000000000000000000


  ##### T #####

  Su <- sqrt(1-((2/pi)*(delta^2)))
  Eu <- sqrt(2/pi)*delta

  mu_t <- delta*( (Su/sqrt(s2e))*(y - x%*%beta) + Eu )
  sig2_t <- 1-delta^2

  t <- truncnorm::rtruncnorm(n,a = 0,b=Inf,mean = mu_t,sd=sqrt(sig2_t))

  tm <- rbind(tm,t)


  ##############
  ### Delta ###
  ##############

  Su <- sqrt(1-(2/pi)*delta^2)
  Eu <- sqrt(2/pi)*delta
```

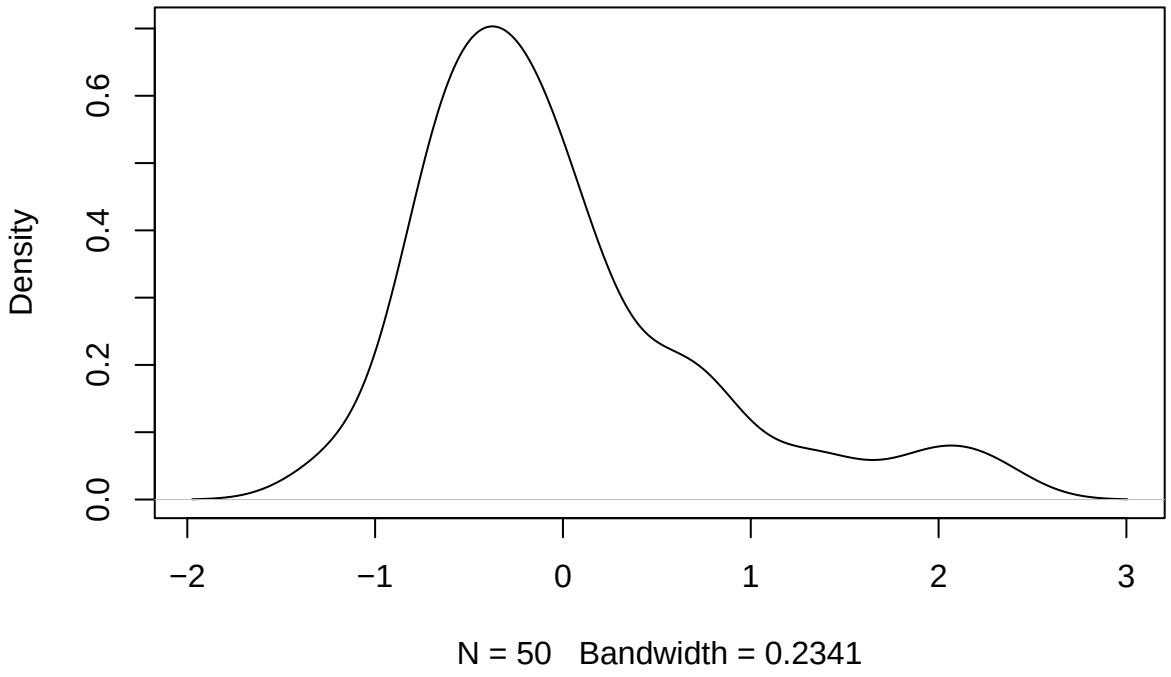
```r
nu <- 1

mode_delta <- delta_init
if(iter==1) t2 = mode_delta else t2 = delta

mu_prop <- t2 # current xi
v <- mu_prop + rnorm(1,0,nu^2) # proposal xi
u <- runif(1,0,1)

v1 <- tanh(v)  #proposal delta
v2 <- tanh(mu_prop) #current delta

p1 <- (Su^n)*(1-v1^2)^(-n/2)
p2 <- (Su^2)/(2*(1-v1^2)*s2e)
p3 <- sum( (y - x%*%beta - v1*t*(sqrt(s2e)/Su) + Eu*(sqrt(s2e)/Su))^2 )
p4 <- (0.5 - 0.5*v1)^(a0-1) * (0.5 + 0.5*v1)^(b0-1)


r1 <- log(p1) + (-0.5*p2*p3) + log(p4) + log(sech(v)^2)

pp1 <- (Su^n)*(1-v2^2)^(-n/2)
pp2 <- (Su^2)/(2*(1-v2^2)*s2e)
pp3 <- sum( (y - x%*%beta - v2*t*(sqrt(s2e)/Su) + Eu*(sqrt(s2e)/Su))^2 )
pp4 <- (0.5 - 0.5*v2)^(a0-1) * (0.5 + 0.5*v2)^(b0-1)


r2 <- log(pp1) + (-0.5*pp2*pp3) + log(pp4) + log(sech(mu_prop)^2)

ratio <- r1 - r2


if(log(u)<ratio) delta = v1 else delta = v2

dm <- rbind(dm,delta)

Su <- sqrt(1-((2/pi)*(delta^2)))
Eu <- sqrt(2/pi)*delta

#PSI
co <- 1
sqco <- sqrt(co)
mu_psi = sqco*(phi*tau)/abs(beta)                                  #mu_j for psi
psi1 = statmod::rinvgauss(n = length(mu_psi),mean = mu_psi,shape = sqco)     #calculating psi
psi = 1/psi1


pm <- rbind(pm,psi)

#TAU
phi[which(abs(phi) <= (0.00000000000000000000000000000000000001))] <- (0.0000000000000000000000000000
mix <- abs(beta)/phi
tau <- GIGrvg::rgig(n=1, lambda=p*a - p, psi=1, chi=2*sum(mix)) #calculating tau post
```

```
  tam <- rbind(tam,tau)

  #PHI
  t_phi <-  mapply( "rgig",  n=rep(1,length(beta)),
                    lambda = rep(a-1,length(beta)),
                    psi = rep(1,length(beta)),
                    chi =c(2*abs(beta)))


  phi  <- t_phi/sum(t_phi)
  phi[which(abs(phi) <= (0.00000000000000000000000000000000000000001))] <- (0.0000000000000000000000000000

  phm <- rbind(phm,phi)


  beta_mat[iter,] <- t(beta)

  if(iter %% 100==0) {
    # Print on the screen some message
    cat(paste0("iteration: ", iter, "\n"))
  }
}

bm <- beta_mat[(burn0+1):niter,]

getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

postmodeSN <- rep(NA, p)
for (i in 1:p){
  postmodeSN[i] <- getmode(bm[,i])
}

rsim_SN[isim, ] <- postmodeSN

if (plots_ind){

post1 <- as.data.frame(beta_mat[,1:p])
colnames(post1) <- c(paste0("beta",seq(1:p)))
color_scheme_set("blue")

trace.plot(t(post1), BurnIn = burn0)

}

dlresult <- dlbayes::dl(x=x, y=y, burn = burn0, nmc = 5000, hyper = a)

rsim_N[isim, ] <- dlbayes::dlvs(dlresult)

print("Real beta's")
print(which(abs(beta_0)>0))
```

```r
print("Assuming asymmetric error")
print(which(abs(postmodeSN)>1))
print("Assuming symmetric error")
print(which(abs(dlbayes::dlvs(dlresult))>0))

}
```

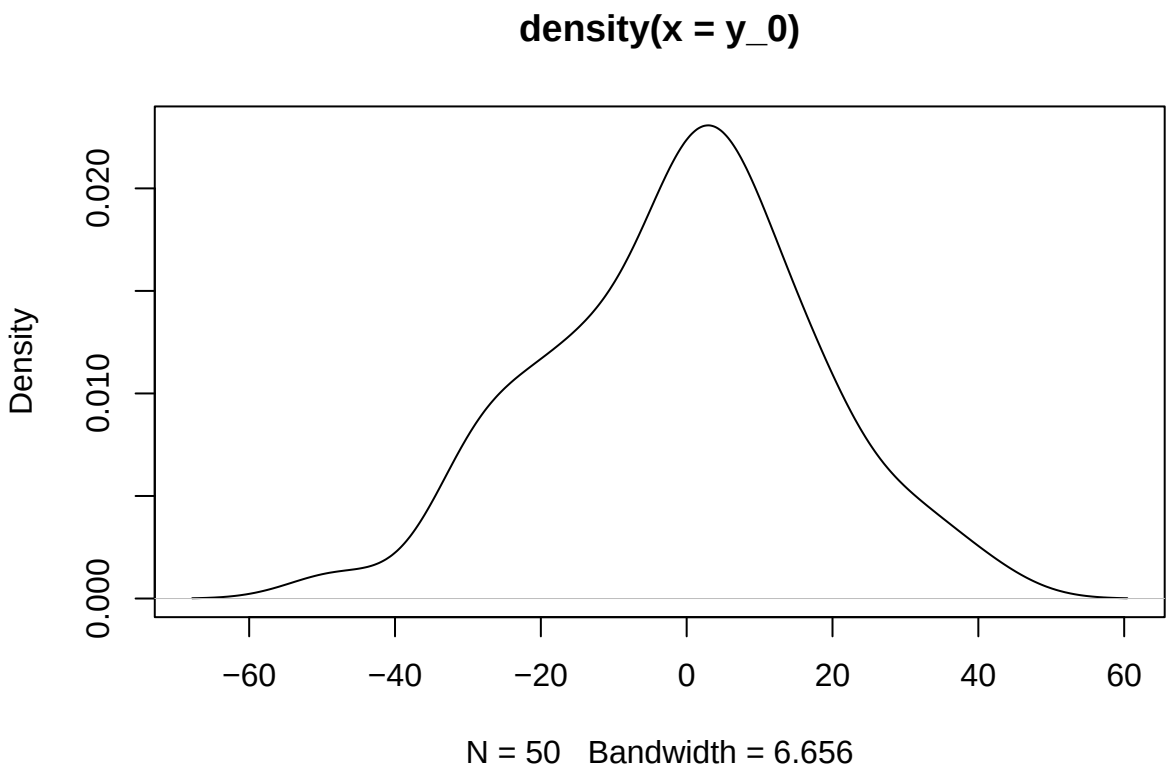
```
## [1] "Simulation 1"
```

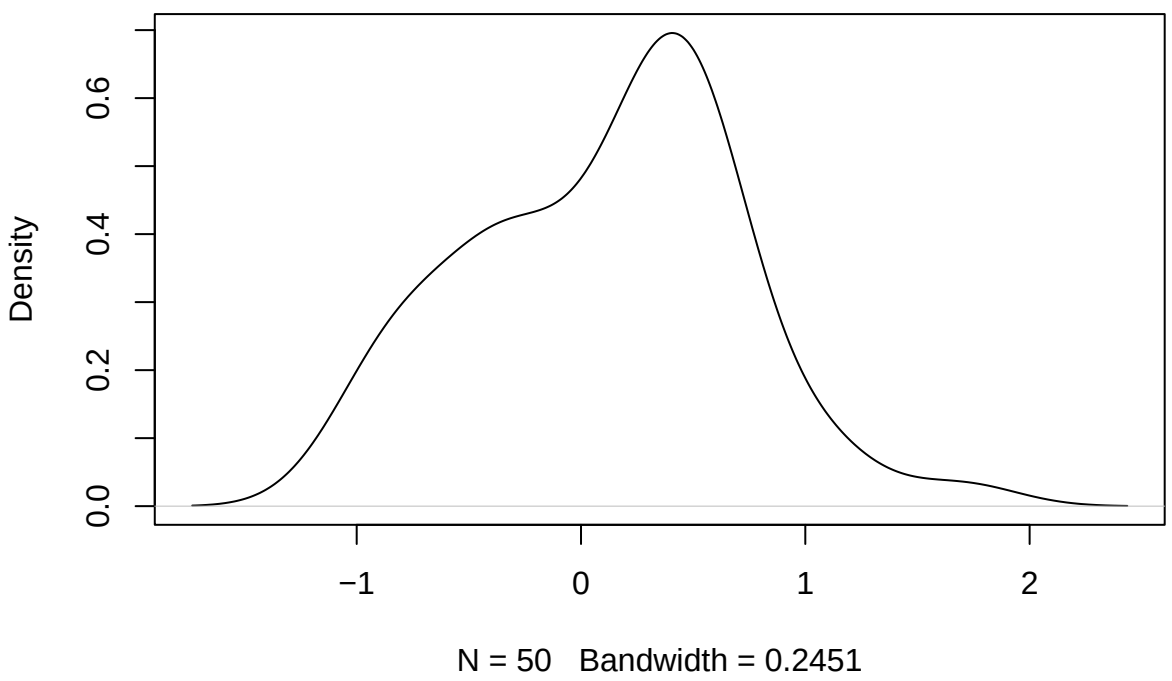





```
## iteration: 100
## iteration: 200
## iteration: 300
## iteration: 400
## iteration: 500
## iteration: 600
## iteration: 700
## iteration: 800
## iteration: 900
## iteration: 1000
## iteration: 1100
## iteration: 1200
## iteration: 1300
## iteration: 1400
## iteration: 1500
## iteration: 1600
## iteration: 1700
## iteration: 1800
## iteration: 1900
## iteration: 2000
## iteration: 2100
## iteration: 2200
## iteration: 2300
## iteration: 2400
## iteration: 2500
## iteration: 2600
## iteration: 2700
## iteration: 2800
## iteration: 2900
```

```
## iteration: 3000
## iteration: 3100
## iteration: 3200
## iteration: 3300
## iteration: 3400
## iteration: 3500
## iteration: 3600
## iteration: 3700
## iteration: 3800
## iteration: 3900
## iteration: 4000
## iteration: 4100
## iteration: 4200
## iteration: 4300
## iteration: 4400
## iteration: 4500
## iteration: 4600
## iteration: 4700
## iteration: 4800
## iteration: 4900
## iteration: 5000
## [1] 1000
## [1] 2000
## [1] 3000
## [1] 4000
## [1] 5000
## [1] 6000
## [1] 7000
## [1] "Real beta's"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming asymmetric error"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming symmetric error"
## [1] 11 12 13 14 15 37 38 40
## [1] "Simulation 2"
```

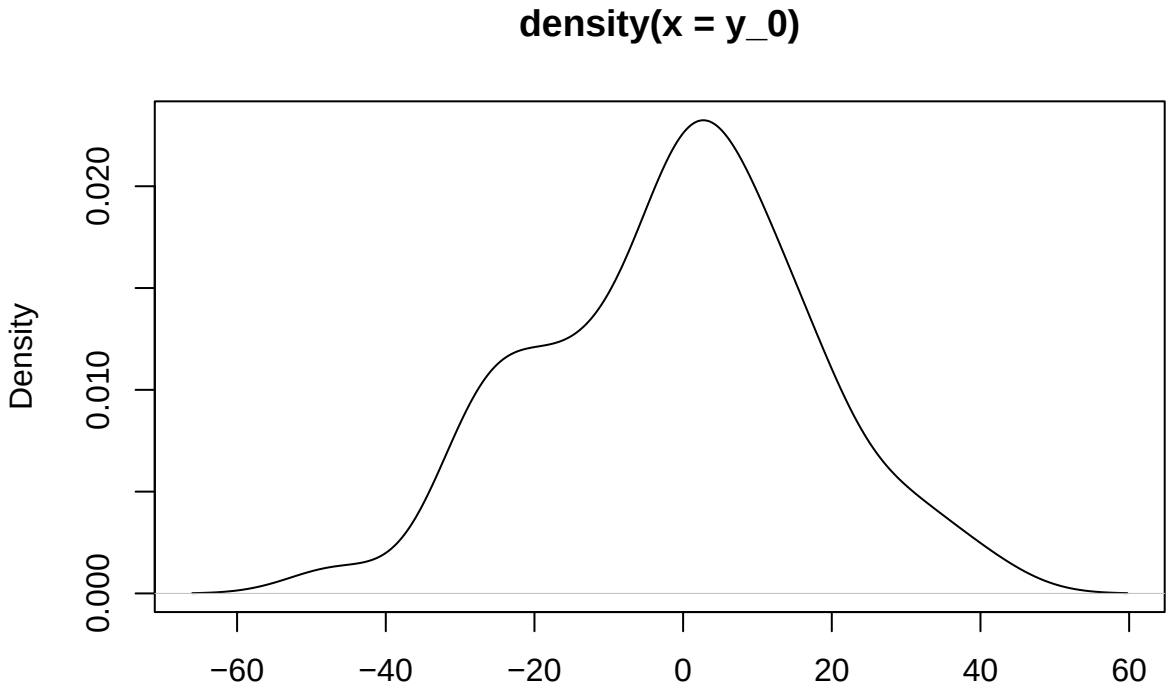


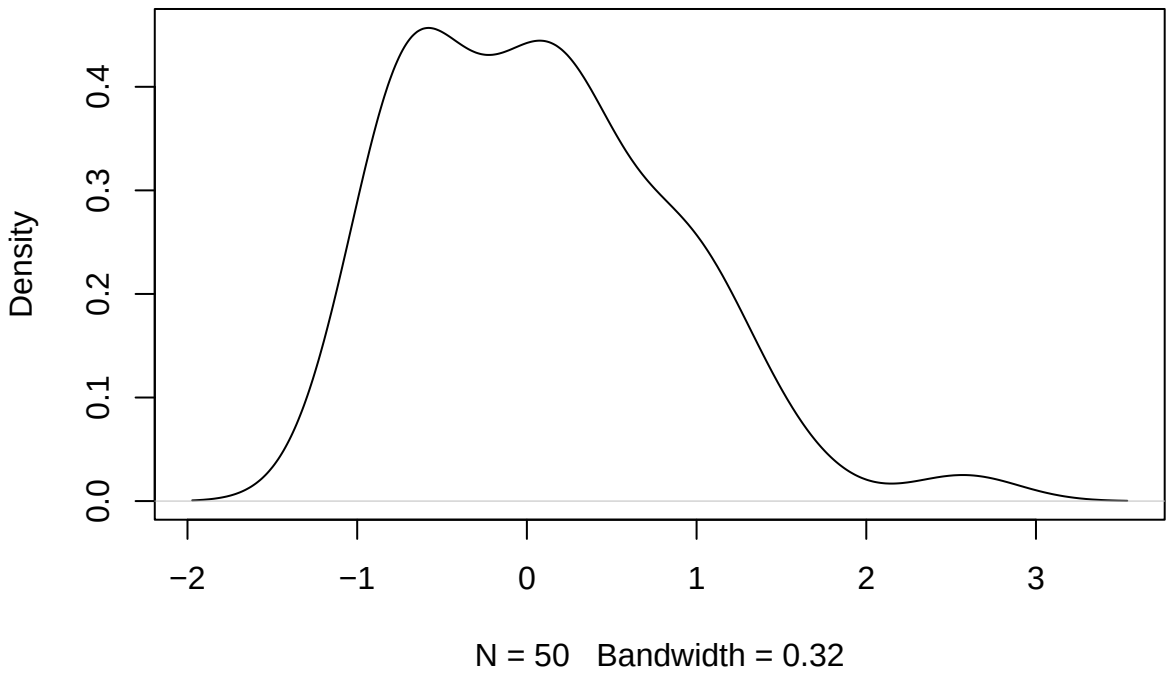



```
## iteration: 100
## iteration: 200
## iteration: 300
## iteration: 400
## iteration: 500
```

```
## iteration: 600
## iteration: 700
## iteration: 800
## iteration: 900
## iteration: 1000
## iteration: 1100
## iteration: 1200
## iteration: 1300
## iteration: 1400
## iteration: 1500
## iteration: 1600
## iteration: 1700
## iteration: 1800
## iteration: 1900
## iteration: 2000
## iteration: 2100
## iteration: 2200
## iteration: 2300
## iteration: 2400
## iteration: 2500
## iteration: 2600
## iteration: 2700
## iteration: 2800
## iteration: 2900
## iteration: 3000
## iteration: 3100
## iteration: 3200
## iteration: 3300
## iteration: 3400
## iteration: 3500
## iteration: 3600
## iteration: 3700
## iteration: 3800
## iteration: 3900
## iteration: 4000
## iteration: 4100
## iteration: 4200
## iteration: 4300
## iteration: 4400
## iteration: 4500
## iteration: 4600
## iteration: 4700
## iteration: 4800
## iteration: 4900
## iteration: 5000
## [1] 1000
## [1] 2000
## [1] 3000
## [1] 4000
## [1] 5000
## [1] 6000
## [1] 7000
## [1] "Real beta's"
##  [1] 11 12 13 14 15 36 37 38 39 40
```

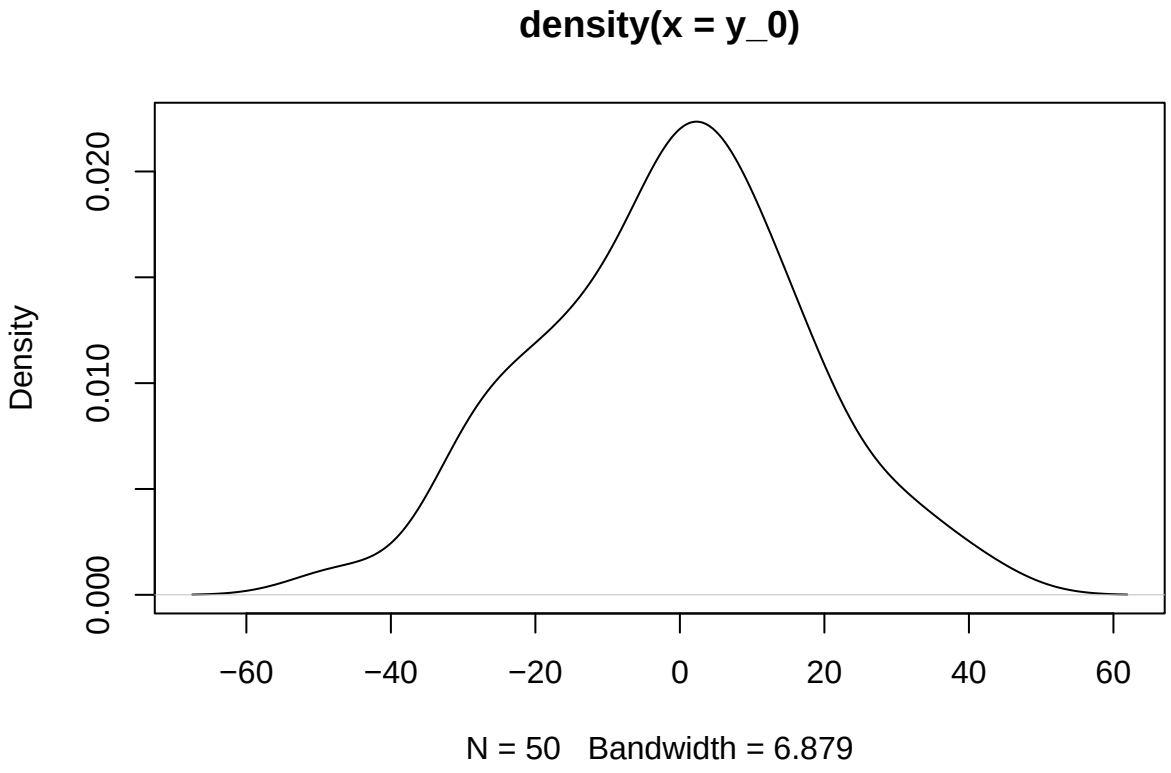
```
## [1] "Assuming asymmetric error"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming symmetric error"
## [1] 11 12 13 14 15 37 38 39 40
## [1] "Simulation 3"
```

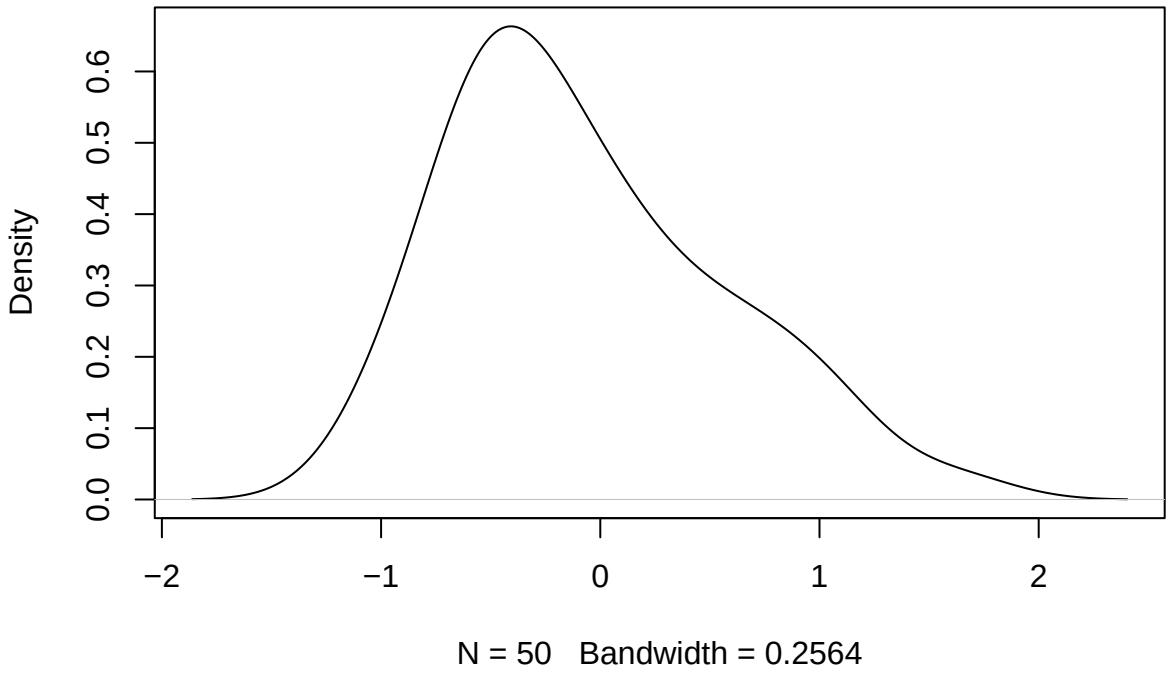




```
## iteration: 100
## iteration: 200
## iteration: 300
## iteration: 400
## iteration: 500
## iteration: 600
## iteration: 700
## iteration: 800
## iteration: 900
## iteration: 1000
## iteration: 1100
## iteration: 1200
## iteration: 1300
## iteration: 1400
## iteration: 1500
## iteration: 1600
## iteration: 1700
## iteration: 1800
## iteration: 1900
## iteration: 2000
## iteration: 2100
## iteration: 2200
## iteration: 2300
## iteration: 2400
## iteration: 2500
## iteration: 2600
## iteration: 2700
## iteration: 2800
## iteration: 2900
## iteration: 3000
## iteration: 3100
## iteration: 3200
## iteration: 3300
## iteration: 3400
## iteration: 3500
## iteration: 3600
## iteration: 3700
## iteration: 3800
## iteration: 3900
## iteration: 4000
## iteration: 4100
## iteration: 4200
## iteration: 4300
## iteration: 4400
## iteration: 4500
## iteration: 4600
## iteration: 4700
## iteration: 4800
## iteration: 4900
## iteration: 5000
## [1] 1000
## [1] 2000
## [1] 3000
## [1] 4000
```

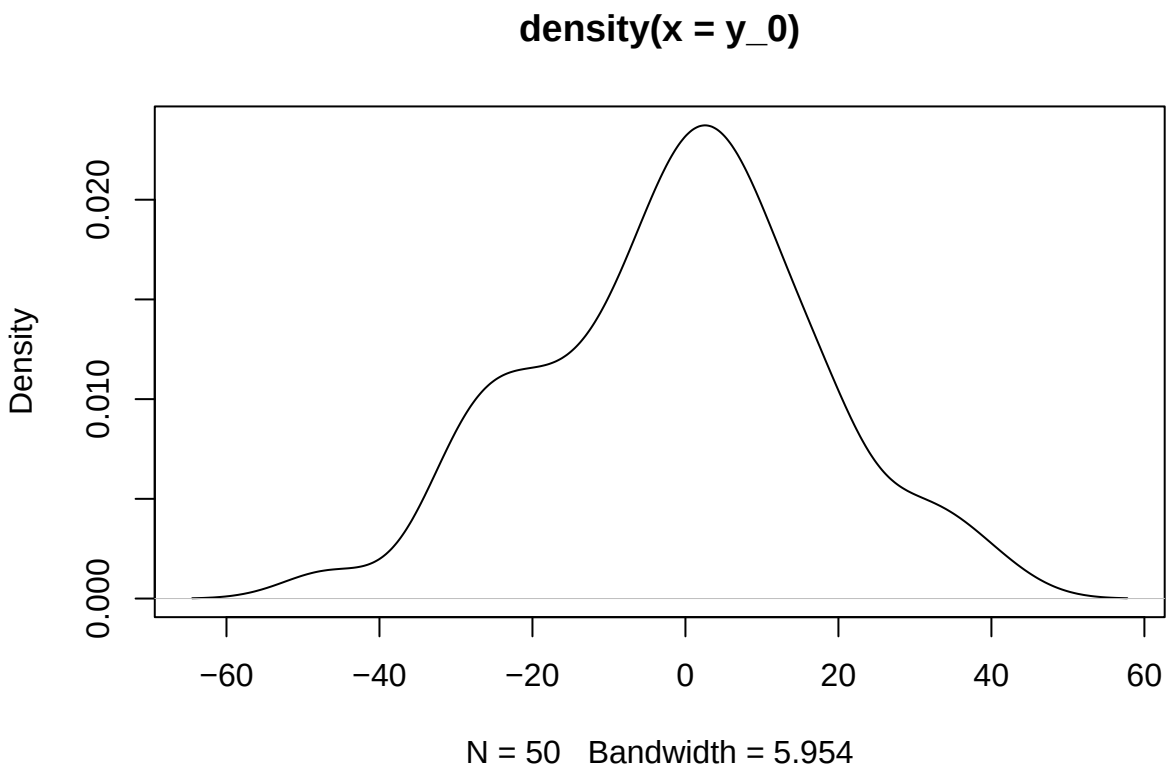
```
## [1] 5000
## [1] 6000
## [1] 7000
## [1] "Real beta's"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming asymmetric error"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming symmetric error"
## [1] 11 12 13 14 15 37 38 39 40
## [1] "Simulation 4"
```

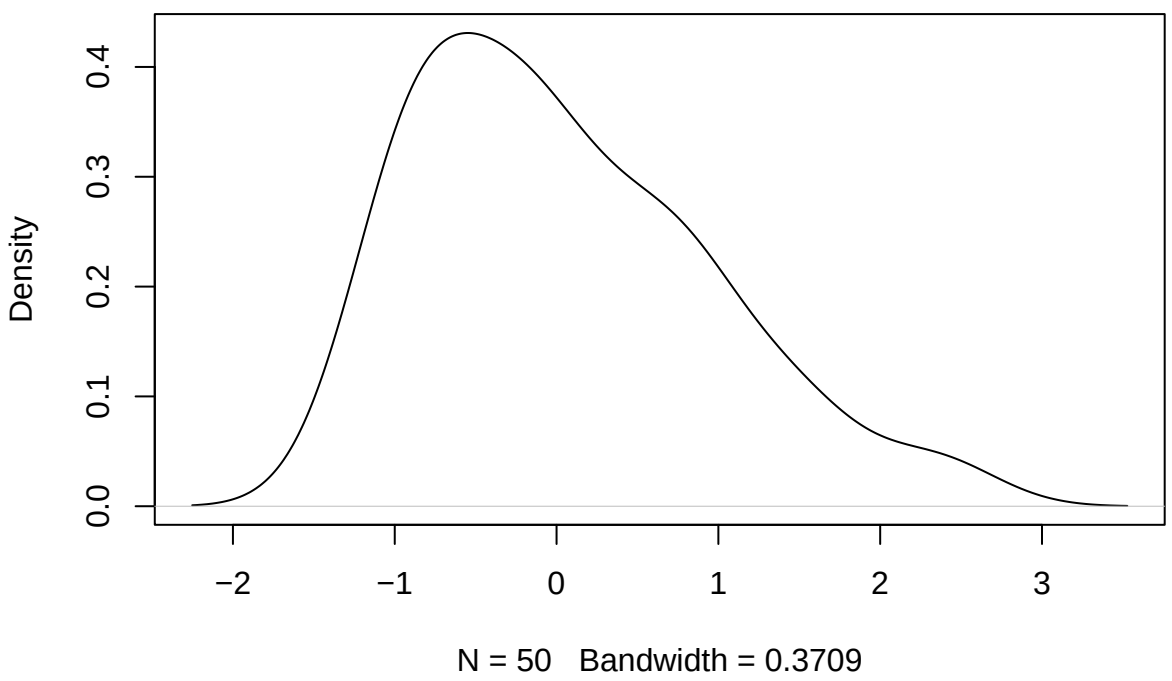





```
## iteration: 100
## iteration: 200
## iteration: 300
## iteration: 400
## iteration: 500
## iteration: 600
## iteration: 700
## iteration: 800
## iteration: 900
## iteration: 1000
## iteration: 1100
## iteration: 1200
## iteration: 1300
## iteration: 1400
## iteration: 1500
## iteration: 1600
## iteration: 1700
## iteration: 1800
## iteration: 1900
## iteration: 2000
## iteration: 2100
## iteration: 2200
## iteration: 2300
## iteration: 2400
## iteration: 2500
## iteration: 2600
## iteration: 2700
## iteration: 2800
## iteration: 2900
```

```
## iteration: 3000
## iteration: 3100
## iteration: 3200
## iteration: 3300
## iteration: 3400
## iteration: 3500
## iteration: 3600
## iteration: 3700
## iteration: 3800
## iteration: 3900
## iteration: 4000
## iteration: 4100
## iteration: 4200
## iteration: 4300
## iteration: 4400
## iteration: 4500
## iteration: 4600
## iteration: 4700
## iteration: 4800
## iteration: 4900
## iteration: 5000
## [1] 1000
## [1] 2000
## [1] 3000
## [1] 4000
## [1] 5000
## [1] 6000
## [1] 7000
## [1] "Real beta's"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming asymmetric error"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming symmetric error"
## [1] 11 12 13 14 15 37 38 39 40
## [1] "Simulation 5"
```






```
## iteration: 100
## iteration: 200
## iteration: 300
## iteration: 400
## iteration: 500
```

```
## iteration: 600
## iteration: 700
## iteration: 800
## iteration: 900
## iteration: 1000
## iteration: 1100
## iteration: 1200
## iteration: 1300
## iteration: 1400
## iteration: 1500
## iteration: 1600
## iteration: 1700
## iteration: 1800
## iteration: 1900
## iteration: 2000
## iteration: 2100
## iteration: 2200
## iteration: 2300
## iteration: 2400
## iteration: 2500
## iteration: 2600
## iteration: 2700
## iteration: 2800
## iteration: 2900
## iteration: 3000
## iteration: 3100
## iteration: 3200
## iteration: 3300
## iteration: 3400
## iteration: 3500
## iteration: 3600
## iteration: 3700
## iteration: 3800
## iteration: 3900
## iteration: 4000
## iteration: 4100
## iteration: 4200
## iteration: 4300
## iteration: 4400
## iteration: 4500
## iteration: 4600
## iteration: 4700
## iteration: 4800
## iteration: 4900
## iteration: 5000
## [1] 1000
## [1] 2000
## [1] 3000
## [1] 4000
## [1] 5000
## [1] 6000
## [1] 7000
## [1] "Real beta's"
##  [1] 11 12 13 14 15 36 37 38 39 40
```

```
## [1] "Assuming asymmetric error"
##  [1] 11 12 13 14 15 36 37 38 39 40
## [1] "Assuming symmetric error"
## [1] 11 12 13 14 15 37 38 39
```

```
VS_SN <- colSums(abs(rsim_SN)>1)
VS_SN
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5
##  [38] 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
VS_N <- colSums(abs(rsim_N)>0)
VS_N
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
##  [38] 5 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```