06-12-2023-2

Hengde Ouyang

2023-06-12

2. Horseshoe Prior Investigation

$$\beta_j|\lambda_j, \tau \sim N(0, \tau^2\lambda_j^2)$$

$$\lambda_j \sim C^+(0,1), \quad j = 1, ..., p$$

In our function, we define:

$$\sigma_j = \tau\lambda_j$$

1. Horseshoe prior is well-known with its shrinkage ability. In our simulation, it indeed shows this ability.

   Function:

```
library(LaplacesDemon)
```

```
##
## Attaching package: 'LaplacesDemon'

## The following objects are masked from 'package:mvtnorm':
##
##     dmvt, rmvt
```

```
MH_horseshoe_Sampling <- function(Y,delta,tau,
                        A,beta0,sigma0,var.prop,
                        m,B,eta,
                        Wmat_option=0){


  accept_beta = 0
  accept_lambda = 0
  beta = beta0
  lambda = lambda0
  sigma = sigma0

  # What we want to record
  BETA = matrix(0,m,dim(A)[2])
  LAMBDA = matrix(0,m,dim(A)[2])
  ThetaRecord <- matrix(0, m, length(Y))
```

```r
C_stat = c()


# For safety m>B
if (B>m){
  B = 0
}


# 0 means we use Harrell C statistics
# 1 means we use Uno C statistics
if (Wmat_option==0){
  Wmat <- HarrellC_Wmat(Y, delta, tau)
}else if (Wmat_option==1){
  Wmat <- UnoC_Wmat(Y, delta, tau)
}else{  # Other Possible C index...
  Wmat <- HarrellC_Wmat(Y, delta, tau)
}


for (i in 1:m){

  # Sample beta from proposal distribution
  beta.p = t(rmvnorm(1,beta,var.prop))


  # Compute theta from current and last iteration
  theta.p = THETA(A,beta.p)
  theta = THETA(A,beta)

  # Record theta from last iteration
  ThetaRecord[i,] <- theta

  # Compute C-statistics from current and last iteration
  HC.p = HarrellC(theta.p, Wmat)
  HC = HarrellC(theta, Wmat)

  # Record C-statistics from last iteration
  C_stat = c(C_stat,HC)



  lrMH = eta*log(HC.p) +
        sum(dnorm(beta.p,beta0,sigma,log=T))-
        eta*log(HC) -
        sum(dnorm(beta,beta0,sigma,log=T))



    if (log(runif(1))<lrMH){
      beta = beta.p
      accept_beta = accept_beta + 1
    }
```

```
      BETA[i,] = beta



  #########################################################################

    lambda.p = exp(t(rnorm(dim(A)[2],log(lambda),rep(1,dim(A)[2]))))
    sigma.p = lambda.p*beta_tau

    lrMH_lambda = sum(dnorm(beta,beta0,sigma.p,log=T))+
                  sum(dhalfcauchy(lambda.p,lambda_scale,log = T))-
                  sum(dnorm(beta,beta0,sigma,log=T))-
                  sum(dhalfcauchy(lambda,lambda_scale,log = T))



    if (log(runif(1))<lrMH_lambda){
        lambda = lambda.p
        sigma = sigma.p
        accept_lambda = accept_lambda + 1
      }
      LAMBDA[i,] = lambda
  }

  #########################################################################


  if (B == 0){
    return(list(BETA=BETA,
                LAMBDA = LAMBDA,
                accept_beta=accept_beta/m,
                accept_lambda=accept_lambda/m,
                THETA = ThetaRecord,
                C_stat = C_stat))
  }else{
    return(list(BETA=BETA[-c(1:B),],
                LAMBDA = LAMBDA[-c(1:B),],
                accept_beta=accept_beta/m,
                accept_lambda=accept_lambda/m,
                THETA = ThetaRecord[-c(1:B),],
                C_stat = C_stat[-c(1:B)]))
  }


}
```

```
system.time({
result_horseshoe = MH_horseshoe_Sampling(Y,delta,tau,
                      A,beta0,sigma0,var.prop,
                      m,B,eta,
                      Wmat_option)
result_no_horseshoe = MH_Sampling(Y,delta,tau,
```

```
                      A,beta0,sigma0,var.prop,
                      m,B,eta,
                      Wmat_option)
})
```

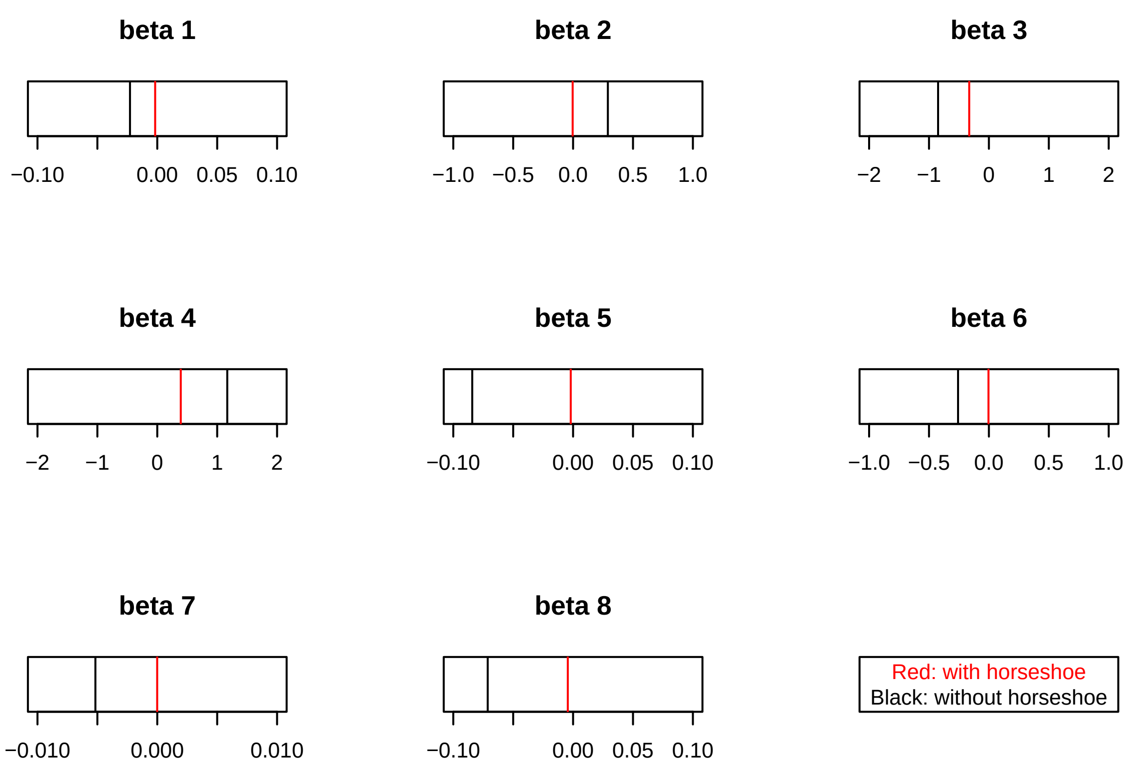
```
##    user  system elapsed
##  220.24   22.66  324.18
```

Comparing the posterior mean for each beta w/wo horseshoe prior:

```
posterior_mean_beta = rbind(colMeans(result_no_horseshoe$BETA),
                            colMeans(result_horseshoe$BETA))
colnames(posterior_mean_beta) =  c("beta1","beta2","beta3","beta4",
                                   "beta5","beta6","beta7","beta8")
rownames(posterior_mean_beta) = c("w/o hs","w hs")
posterior_mean_beta
```

```
##                 beta1        beta2      beta3     beta4        beta5        beta6
## w/o hs -0.022775302  0.290246059 -0.8485934 1.1683106 -0.084179789 -0.257475138
## w hs   -0.001763546 -0.003673065 -0.3286724 0.3924602 -0.001962776 -0.003677703
##                 beta7        beta8
## w/o hs -5.164651e-03 -0.071277654
## w hs   -7.756313e-06 -0.004408845
```

Try to visualize it:



And we can compare the C statistics:

```
Wmat = HarrellC_Wmat(Y,delta,tau)
HarrellC(colMeans(result_no_horseshoe$THETA),Wmat)
```

```
## [1] 0.6426407
```

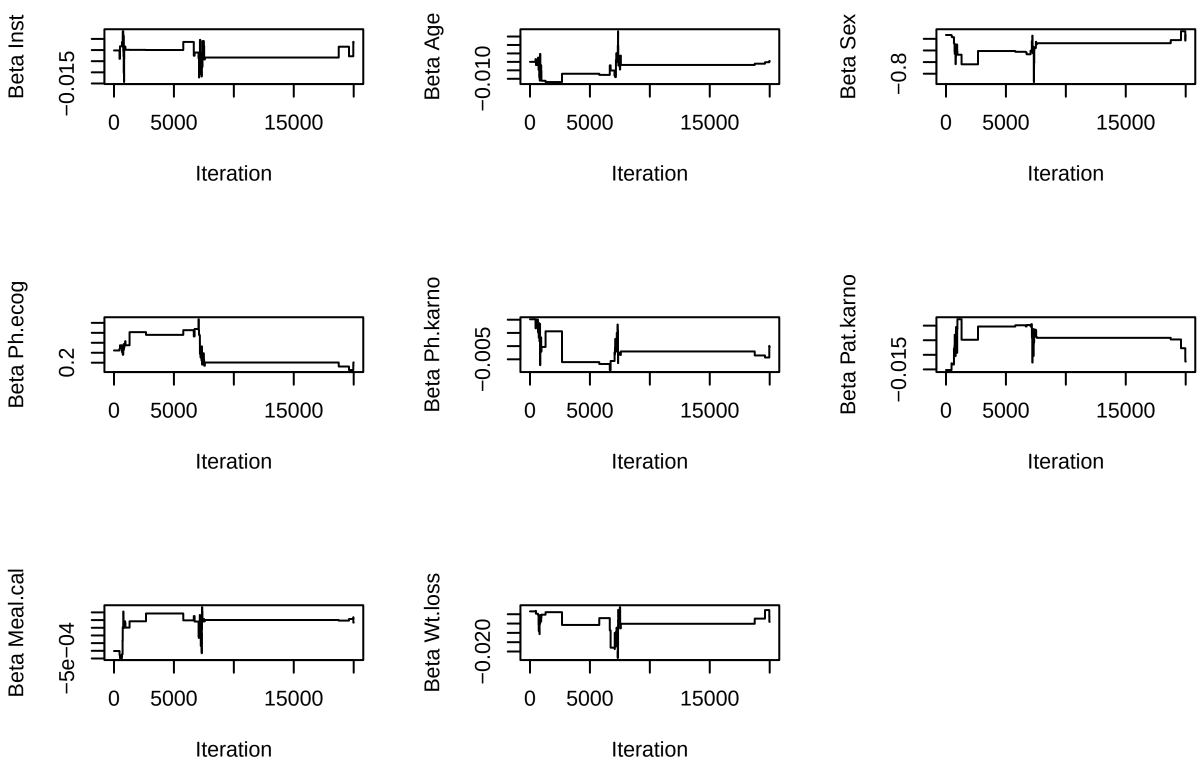
```
HarrellC(colMeans(result_horseshoe$THETA),Wmat)
```

```
## [1] 0.6596893
```

2. However, the issue of a low acceptance rate still remains..

No matter how I tune the parameters, it's still difficult to raise the acceptance rate.

```
beta_iter(result_horseshoe,m,B)
```



Numerical Reason:
The algorithm seems to prefer sampling a low value of lambda. A low value of lambda leads to a small prior variance of beta, which leads to a small acceptance rate. (Based on the simulation result before, smaller prior variance will have smaller acceptance rate)
What's more, when you want to increase beta_tau, the sampling value of lambda will decrease. (No change

in prior variance!)
I also personally searched some papers, trying to find out the reason.

It seems that for some algorithm, this issue will occur when using horseshoe prior: "become stuck after a few thousand iterations"
https://dl.acm.org/doi/pdf/10.5555/3455716.3455789 (Page 10)
3. I also personally try another approach.
(It might not be valid, but I just want to explore)
Metropolis-Gibbs Algorithm?

```
MH_horseshoe_Sampling2 <- function(Y,delta,tau,
                        A,beta0,sigma0,var.prop,
                        m,B,eta,
                        Wmat_option=0){


  accept_beta = 0
  accept_lambda = 0
  beta = beta0
  lambda = lambda0
  sigma = sigma0

  # What we want to record
  BETA = matrix(0,m,dim(A)[2])
  LAMBDA = matrix(0,m,dim(A)[2])
  V = matrix(0,m,dim(A)[2])
  ThetaRecord <- matrix(0, m, length(Y))
  C_stat = c()


  # For safety m>B
  if (B>m){
    B = 0
  }


  # 0 means we use Harrell C statistics
  # 1 means we use Uno C statistics
  if (Wmat_option==0){
    Wmat <- HarrellC_Wmat(Y, delta, tau)
  }else if (Wmat_option==1){
    Wmat <- UnoC_Wmat(Y, delta, tau)
  }else{  # Other Possible C index...
    Wmat <- HarrellC_Wmat(Y, delta, tau)
  }


  for (i in 1:m){

    # Sample beta from proposal distribution
    beta.p = t(rmvnorm(1,beta,var.prop))
```

```r
  # Compute theta from current and last iteration
  theta.p = THETA(A,beta.p)
  theta = THETA(A,beta)

  # Record theta from last iteration
  ThetaRecord[i,] <- theta

  # Compute C-statistics from current and last iteration
  HC.p = HarrellC(theta.p, Wmat)
  HC = HarrellC(theta, Wmat)

  # Record C-statistics from last iteration
  C_stat = c(C_stat,HC)



  lrMH = eta*log(HC.p) +
        sum(dnorm(beta.p,beta0,sigma,log=T))-
        eta*log(HC) -
        sum(dnorm(beta,beta0,sigma,log=T))



    if (log(runif(1))<lrMH){
      beta = beta.p
      accept_beta = accept_beta + 1
    }
    BETA[i,] = beta


##############################################################################
########## For sampling lambda, I try Gibbs sampling this time ###############
  lambda = sqrt(1/rgamma(dim(A)[2],shape=1,rate=(1/v)+(beta^2/(2*beta_tau^2))))
  v = 1/rgamma(dim(A)[2],shape=1,rate=1+1/lambda^2)
  sigma = lambda*beta_tau


    LAMBDA[i,] = lambda
    V[i,] = v
}
 #####################################################################



if (B == 0){
  return(list(BETA=BETA,
              LAMBDA = LAMBDA,
              V = V,
              accept_beta=accept_beta/m,
              THETA = ThetaRecord,
              C_stat = C_stat))
}else{
  return(list(BETA=BETA[-c(1:B),],
```

```
                LAMBDA = LAMBDA[-c(1:B),],
                V = V[-c(1:B)],
                accept_beta=accept_beta/m,
                THETA = ThetaRecord[-c(1:B),],
                C_stat = C_stat[-c(1:B)]))
  }


}
```

This time, I can successfully tune the parameter beta_tau:

```
system.time({
  m = 22000
  B = 2000
  v = rep(1,dim(A)[2])
  beta_tau = 10
  result_horseshoe21 = MH_horseshoe_Sampling2(Y,delta,tau,
                      A,beta0,sigma0,var.prop,
                      m,B,eta,
                      Wmat_option)
  beta_tau = 1
  result_horseshoe22 = MH_horseshoe_Sampling2(Y,delta,tau,
                      A,beta0,sigma0,var.prop,
                      m,B,eta,
                      Wmat_option)
  beta_tau = 0.1
  result_horseshoe23 = MH_horseshoe_Sampling2(Y,delta,tau,
                      A,beta0,sigma0,var.prop,
                      m,B,eta,
                      Wmat_option)
    beta_tau = 0.01
   result_horseshoe24 = MH_horseshoe_Sampling2(Y,delta,tau,
                      A,beta0,sigma0,var.prop,
                      m,B,eta,
                      Wmat_option)

})
```

```
##    user  system elapsed
##  445.91   38.99  683.00
```

```
hs = data.frame(Beta_Tau = c(10,1,0.1,0.01),
                AcceptanceRate=c(result_horseshoe21$accept_beta,
                                 result_horseshoe22$accept_beta,
                                 result_horseshoe23$accept_beta,
                                 result_horseshoe24$accept_beta),
                WOhorseshoe_Beta1 = c(rep(colMeans(result_horseshoe$BETA)[1],4)),
                Whorseshoe_Beta1 = c(colMeans(result_horseshoe21$BETA)[1],
                               colMeans(result_horseshoe22$BETA)[1],
                               colMeans(result_horseshoe23$BETA)[1],
                               colMeans(result_horseshoe24$BETA)[1]))
hs
```

```
##   Beta_Tau AcceptanceRate WOhorseshoe_Beta1 Whorseshoe_Beta1
## 1    10.00     0.86586364      -0.001763546     -0.089500857
## 2     1.00     0.77836364      -0.001763546     -0.070091333
## 3     0.10     0.38918182      -0.001763546     -0.003590347
## 4     0.01     0.02136364      -0.001763546     -0.001424735
```

We use beta 1 as an example. It seems that acceptance rate is related to the shrinkage effect. (With a lower acceptance rate, the beta will shrink more close to 0?)