Lab 7 solutions MAST90125: Bayesian Statistical Learning

Thursday 12 September 2019

Writing Gibbs samplers in linear models.

In this weeks lab, we will discuss how to write code for Gibbs sampling of linear models.

Instructions for assignment

Download `USJudgeRatings.csv` from LMS. Comment the codes below that purports to perform Gibbs sampling for a variety of linear models. See if you can determine what the code is doing. You may find referring back to lectures 12 and 13 useful. Run the code to see if it compares to the results obtained from using `stan` to fit the same models last week (Lab 6 version 2).

Examples of Gibbs samplers

- Linear regression (flat prior for $\boldsymbol{\beta}$, $p(\tau) \propto \tau^{-1}$, where $\tau = (\sigma^2)^{-1}$).

```
#This is a Gibbs sampler where beta is updated as a block. The arguments are
#X:  matrix of predictors dimension n times p. Includes the intercept.
#y:  response vector, length p.
#tau0: initial value for the residual precision.
#iter: number of iterations
#burnin: number of initial iterations to remove
Gibbs.lm1<-function(X,y,tau0,iter,burnin){
p <- dim(X)[2]       #number of predictors
XTX <- crossprod(X)
XTXinv <-solve(XTX)
XTY <- crossprod(X,y)
betahat<-solve(XTX,XTY) #betahat = (t(X)%*%X)^{-1}t(X)%*%y = mean of conditional posterior for beta
tau    <-tau0
library(mvtnorm)

par<-matrix(0,iter,p+1)  #storing iterations, beta (length p) + tau (length 1)
for( i in 1:iter){
  beta <- rmvnorm(1,mean=betahat,sigma=XTXinv/tau) #sample beta
  beta <- as.numeric(beta)
  err  <- y-X%*%beta
  tau  <- rgamma(1,0.5*n,0.5*sum(err^2))          #sample tau.
  par[i,] <-c(beta,tau)                           #store current round of beta, tau in par.
}

par <-par[-c(1:burnin),]                          #removing the first iterations
return(par)
}
```

```r
#Example of unblocked Gibbs sampling. We will update beta element by element. The arguments are

#X:  matrix of predictors dimension n times p. Includes the intercept.
#y:  response vector, length p.
#tau0: initial value for the residual precision.
#iter: number of iterations
#burnin: number of initial iterations to remove
Gibbs.lm2<-function(X,y,tau0,iter,burnin){
p <- dim(X)[2]
diagXTX <-colSums(X^2)     #calculates (t(X)%*%X)_{ii} for all i.
XTY <- crossprod(X,y)
betahat <- XTY/diagXTX     #component of conditional posteriors of beta_i's that is function of y.
tau    <-tau0

beta<-rnorm(p)
par<-matrix(0,iter,p+1)
for( i in 1:iter){
  for(j in 1:p){   #This samples beta element by element
  beta[j]<-0       #If we zero beta_j, then X_jbeta_j = Xbeta.
  Xb     <-X%*%beta
  diff   <-t(X[,j])%*%Xb/diagXTX[j]
  beta[j] <- rnorm(1,mean=betahat[j]-diff,sd=1/sqrt(tau*diagXTX[j]) )
}
  err  <- y-X%*%beta
  tau  <- rgamma(1,0.5*n,0.5*sum(err^2)) #samples tau from conditional posterior.
  par[i,] <-c(beta,tau)
}

par <-par[-c(1:burnin),]
return(par)
}
```

```r
#Example of Gibbs sampling where matrix decompositions are used to diagonalise conditional
#posterior variances.This means beta is still updated as a block, but in a more efficient
#way than in Gibbs.lm1. The arguments are

#X:  matrix of predictors dimension n times p. Includes the intercept.
#y:  response vector, length p.
#tau0: initial value for the residual precision.
#iter: number of iterations
#burnin: number of initial iterations to remove
Gibbs.lm3<-function(X,y,tau0,iter,burnin){
p <- dim(X)[2]  #dimension of p
svdX <-svd(X)   #matrix decomposition to speed up computation.
U    <-svdX$u   #extracting components of decompositions.
Lambda<-svdX$d
V    <-svdX$v
Vbhat <- crossprod(U,y)/Lambda #mean of conditional posterior for transformed parameters.
tau <-tau0

par<-matrix(0,iter,p+1)
for( i in 1:iter){
  sqrttau<-sqrt(tau)
  #posterior variances are diagonal for transformed parameters, so sequence of univariate normal draws
```

```
  vbeta <- rnorm(p,mean=Vbhat,sd=1/(sqrttau*Lambda) )
  beta <-V%*%vbeta   #back transform to original parameter.
  err  <- y-X%*%beta
  tau  <- rgamma(1,0.5*n,0.5*sum(err^2)) #sample tau
  par[i,] <-c(beta,tau)
}

par <-par[-c(1:burnin),]  #remove initial iterations
return(par)
}
```

Solution

```
#Formatting data, and running chains.
data<-read.csv('USJudgeRatings.csv')
response<-data$RTEN  #response variable
n<-dim(data)[1]
intercept <-matrix(1,n,1) #Intercept (to be estimated without penalty)
Pred<-data[,2:12]         #Predictor variables.
Pred<-as.matrix(scale(Pred))
X   <-cbind(intercept,Pred)

system.time(chain1<-Gibbs.lm1(X=X,y=response,tau0=1,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    2.79    0.02    2.83
```

```
system.time(chain2<-Gibbs.lm1(X=X,y=response,tau0=5,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    2.77    0.00    2.77
```

```
system.time(chain3<-Gibbs.lm1(X=X,y=response,tau0=0.2,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    2.78    0.00    2.78
```

```
system.time(chain4<-Gibbs.lm2(X=X,y=response,tau0=1,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    1.17    0.01    1.20
```

```
system.time(chain5<-Gibbs.lm2(X=X,y=response,tau0=5,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    1.20    0.00    1.21
```

```
system.time(chain6<-Gibbs.lm2(X=X,y=response,tau0=0.2,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##     1.2     0.0     1.2
```

```
system.time(chain7<-Gibbs.lm3(X=X,y=response,tau0=1,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    0.11    0.00    0.11
```

```
system.time(chain8<-Gibbs.lm3(X=X,y=response,tau0=5,iter=10000,burnin=2000))
```

```
##    user  system elapsed
```

```
##    0.07    0.00    0.07
```

```r
system.time(chain9<-Gibbs.lm3(X=X,y=response,tau0=0.2,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##    0.08    0.00    0.08
```

```r
library(coda)
#Estimating Gelman -Rubin diagnostics.
#Note 8000 iterations were retained, so 50:50 split is iteration 1:4000 and iteration 4001:8000

#However first we must convert the output into mcmc lists for coda to interpret.
ml1<-as.mcmc.list(as.mcmc((chain1[1:4000,])))
ml2<-as.mcmc.list(as.mcmc((chain2[1:4000,])))
ml3<-as.mcmc.list(as.mcmc((chain3[1:4000,])))
ml4<-as.mcmc.list(as.mcmc((chain1[4000+1:4000,])))
ml5<-as.mcmc.list(as.mcmc((chain2[4000+1:4000,])))
ml6<-as.mcmc.list(as.mcmc((chain3[4000+1:4000,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

```
##       Point est. Upper C.I.
##  [1,]  1.0003413  1.0011865
##  [2,]  1.0010677  1.0029840
##  [3,]  1.0001089  1.0005646
##  [4,]  1.0000408  1.0003908
##  [5,]  1.0001809  1.0007105
##  [6,]  1.0007508  1.0015460
##  [7,]  1.0003355  1.0011845
##  [8,]  0.9999888  1.0002935
##  [9,]  1.0000326  1.0002266
## [10,]  0.9998869  1.0000718
## [11,]  0.9998638  0.9999721
## [12,]  0.9999424  1.0002215
## [13,]  1.0007316  1.0021488
```

```r
#effective sample size.
effectiveSize(estml)
```

```
##     var1     var2     var3     var4     var5     var6     var7     var8
## 24000.00 24000.00 24000.00 24000.00 24000.00 24000.00 24000.00 24000.00
##     var9    var10    var11    var12    var13
## 24000.00 24000.00 24000.00 24000.00 14297.98
```

```r
#However first we must convert the output into mcmc lists for coda to interpret.
ml1<-as.mcmc.list(as.mcmc((chain4[1:4000,])))
ml2<-as.mcmc.list(as.mcmc((chain5[1:4000,])))
ml3<-as.mcmc.list(as.mcmc((chain6[1:4000,])))
ml4<-as.mcmc.list(as.mcmc((chain4[4000+1:4000,])))
ml5<-as.mcmc.list(as.mcmc((chain5[4000+1:4000,])))
ml6<-as.mcmc.list(as.mcmc((chain6[4000+1:4000,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

```
##       Point est. Upper C.I.
##  [1,]   1.000128   1.000475
##  [2,]   1.007395   1.017415
##  [3,]   1.023773   1.055979
##  [4,]   1.039445   1.092409
##  [5,]   1.033504   1.076018
##  [6,]   1.018484   1.038918
##  [7,]   1.019510   1.048409
##  [8,]   1.064335   1.146444
##  [9,]   1.146979   1.340675
## [10,]   1.043523   1.090600
## [11,]   1.120122   1.277644
## [12,]   1.011997   1.027093
## [13,]   1.011637   1.029686
```

```r
#effective sample size.
effectiveSize(estml)
```

```
##        var1        var2        var3        var4        var5        var6
## 23485.67566   610.57400   368.25729   325.25018   218.61119   300.50895
##        var7        var8        var9       var10       var11       var12
##   365.78797    77.65203    78.53642    51.70065    54.56167   253.53053
##       var13
##  1894.86162
```

```r
#However first we must convert the output into mcmc lists for coda to interpret.
ml1<-as.mcmc.list(as.mcmc((chain7[1:4000,])))
ml2<-as.mcmc.list(as.mcmc((chain8[1:4000,])))
ml3<-as.mcmc.list(as.mcmc((chain9[1:4000,])))
ml4<-as.mcmc.list(as.mcmc((chain7[4000+1:4000,])))
ml5<-as.mcmc.list(as.mcmc((chain8[4000+1:4000,])))
ml6<-as.mcmc.list(as.mcmc((chain9[4000+1:4000,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

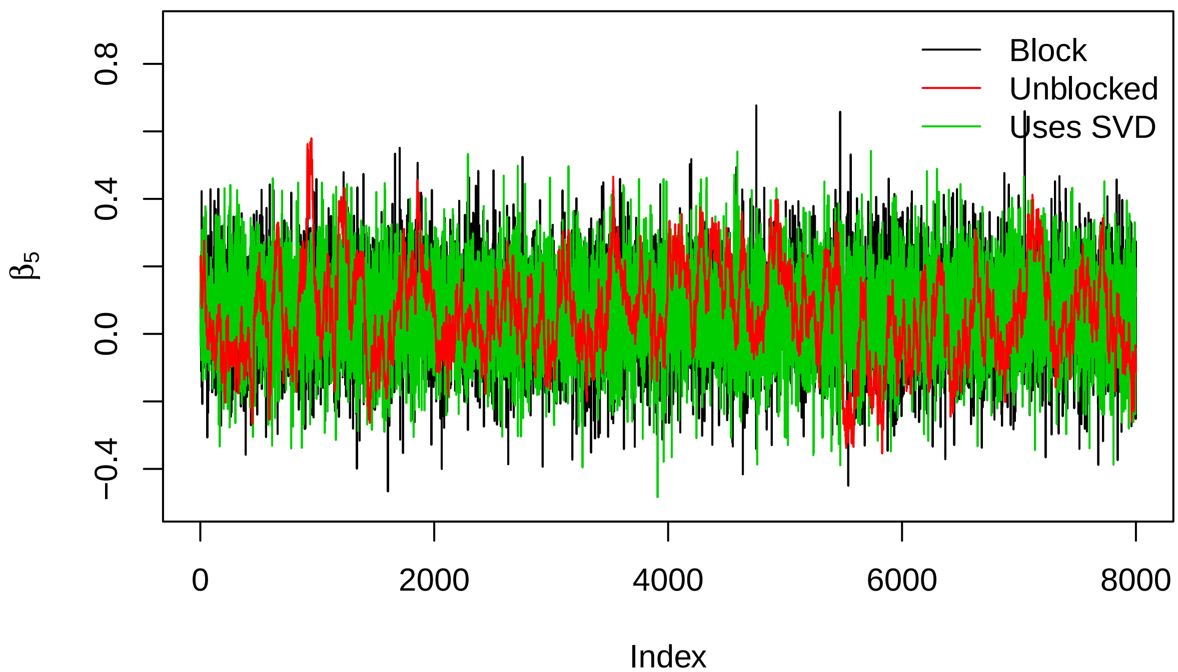
```
##       Point est. Upper C.I.
##  [1,]  0.9998276  0.9998839
##  [2,]  0.9999642  1.0001546
##  [3,]  0.9999795  1.0001247
##  [4,]  0.9999889  1.0003297
##  [5,]  1.0000917  1.0002862
##  [6,]  1.0001071  1.0005994
##  [7,]  1.0000764  1.0003444
##  [8,]  0.9998229  0.9998997
##  [9,]  0.9999119  1.0000383
## [10,]  1.0000571  1.0003483
## [11,]  0.9999042  1.0001241
## [12,]  0.9998289  0.9999014
## [13,]  1.0002738  1.0009841
```

```r
#effective sample size.
effectiveSize(estml)
```

```
##     var1     var2     var3     var4     var5     var6     var7     var8
## 24000.00 24724.44 24000.00 24000.00 24000.00 24000.00 25330.83 24000.00
```

```
##      var9     var10     var11     var12     var13
## 24000.00 24000.00 25121.25 24000.00 13821.75
```

```
#Comparing one co-efficient (the 5th)
plot(chain1[,5],type='l',ylim=c(-0.5,0.9),ylab=expression(beta[5]))
lines(chain7[,5],type='l',col=3,ylab=expression(beta[5]))
lines(chain4[,5],type='l',col=2,ylab=expression(beta[5]))
legend('topright',legend=c('Block','Unblocked','Uses SVD'),col=1:3,lty=1,bty='n')
```




```
#Reporting posterior means and credible intervals.
#Means
colMeans(rbind(chain1,chain2,chain3)) #Blocked
```

```
##  [1]  7.602290608  0.011873407  0.280548379  0.143845680  0.059164994
##  [6] -0.166363055  0.223724159 -0.002440861 -0.127163529  0.551011374
## [11] -0.064513015  0.252635156 72.335146917
```

```
colMeans(rbind(chain4,chain5,chain6)) #unblocked
```

```
##  [1]  7.60239336  0.01379158  0.27763385  0.15338201  0.06821053
##  [6] -0.16497641  0.22047548 -0.02897290 -0.09242135  0.53466006
## [11] -0.07101967  0.25478662 73.22236799
```

```
colMeans(rbind(chain7,chain8,chain9)) #Blocked but using svd to speed up computation.
```

```
##  [1]  7.6022853183  0.0119740385  0.2809603957  0.1426368807  0.0595562056
##  [6] -0.1681504228  0.2239776601 -0.0005679958 -0.1301031712  0.5567026741
## [11] -0.0669247771  0.2520019094 72.5556248923
```

```r
#Credible interval
apply(rbind(chain1,chain2,chain3) ,2, FUN =function(x) quantile(x,c(0.025,0.975) )) #Blocked
```

```
##            [,1]        [,2]       [,3]        [,4]       [,5]        [,6]
## 2.5%  7.565643 -0.03743149 0.07402446 -0.06492396 -0.2027753 -0.42575267
## 97.5% 7.638450  0.06095730 0.48186976  0.35529014  0.3231502  0.09399502
##                [,7]       [,8]       [,9]       [,10]      [,11]     [,12]
## 2.5%  -0.004423966 -0.4720929 -0.6451002 -0.02576225 -0.6923035 0.1329368
## 97.5%  0.451810389  0.4617845  0.3931162  1.12753074  0.5588945 0.3716479
##           [,13]
## 2.5%   41.36176
## 97.5% 111.81489
```

```r
apply(rbind(chain4,chain5,chain6) ,2, FUN =function(x) quantile(x,c(0.025,0.975) )) #Unblocked
```

```
##            [,1]        [,2]       [,3]        [,4]       [,5]        [,6]
## 2.5%  7.566024 -0.03604019 0.06853033 -0.05799356 -0.1905866 -0.40733663
## 97.5% 7.638102  0.06287613 0.47676851  0.35472300  0.3433845  0.08847645
##               [,7]       [,8]       [,9]       [,10]      [,11]     [,12]
## 2.5%  0.006880288 -0.4894330 -0.5403710 -0.04866047 -0.6211989 0.1353311
## 97.5% 0.434404271  0.4290811  0.3460951  1.08043579  0.4642468 0.3740263
##           [,13]
## 2.5%   41.82221
## 97.5% 112.87112
```

```r
apply(rbind(chain7,chain8,chain9) ,2, FUN =function(x) quantile(x,c(0.025,0.975) )) #Blocked with svd.
```

```
##            [,1]        [,2]      [,3]        [,4]       [,5]       [,6]
## 2.5%  7.566051 -0.03803262 0.0769504 -0.06711342 -0.2014690 -0.4260054
## 97.5% 7.638689  0.06198022 0.4846847  0.35117813  0.3229448  0.0883075
##                [,7]       [,8]       [,9]       [,10]      [,11]     [,12]
## 2.5%  -0.003135034 -0.4685738 -0.6462462 -0.01599512 -0.6851662 0.1355050
## 97.5%  0.448216706  0.4671695  0.3816365  1.11751273  0.5523074 0.3693659
##           [,13]
## 2.5%   41.25019
## 97.5% 113.12326
```

- Linear mixed model/ ridge regression (flat prior for $\boldsymbol{\beta}_0$, $p(\tau) = \text{Ga}(\alpha_e, \gamma_e)$, where $\tau = (\sigma^2)^{-1}$), $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \sigma_\beta^2 \mathbf{I})$, $(\sigma_\beta^2)^{-1} = \tau_\beta \sim \text{Ga}(\alpha_\beta, \gamma_\beta)$.

```r
#Arguments are
#X:  matrix of predictors dimension n times p with flat prior. Includes the intercept.
#Z:  matrix of predictors dimension n times q with normal prior for u.
#y:  response vector, length p.
#taue_0,tauu_0: initial value for the residual and random effect precision.
#a.u,b.u. Hyper-parameter for gamma prior for tau_u
#a.e,b.e. Hyper-parameter for gamma prior for tau_e
#iter: number of iterations
#burnin: number of initial iterations to remove
normalmm.Gibbs<-function(iter,Z,X,y,burnin,taue_0,tauu_0,a.u,b.u,a.e,b.e){
  n   <-length(y) #no. observations
  p   <-dim(X)[2] #no of fixed effect predictors.
  q   <-dim(Z)[2] #no of random effect levels
  tauu<-tauu_0
  taue<-taue_0
  #starting value for u.
  u0   <-rnorm(q,0,sd=1/sqrt(tauu))

  #Building combined predictor matrix.
  W<-cbind(X,Z)            #for the joint conditional posterior for b,u
  WTW <-crossprod(W)
  library(mvtnorm)

  #storing results.
  par <-matrix(0,iter,p+q+2)  #p beta coefficient, q u coefficients and 2 precision coefficient.

  #Create modified identity matrix for joint posterior.
  I0  <-diag(p+q)
  diag(I0)[1:p]<-0

  for(i in 1:iter){
    #Conditional posteriors.
    tauu <-rgamma(1,a.u+0.5*q,b.u+0.5*sum(u0^2)) #sample tau_u
    #Updating component of normal posterior for beta,u
    Prec <-WTW + tauu*I0/taue
    P.mean <- solve(Prec)%*%crossprod(W,y)
    P.var  <-solve(Prec)/taue
    betau <-rmvnorm(1,mean=P.mean,sigma=P.var) #sample beta, u
    betau <-as.numeric(betau)
    err   <- y-W%*%betau
    taue  <-rgamma(1,a.e+0.5*n,b.e+0.5*sum(err^2)) #sample tau_e
    #storing iterations for beta, u, and standard deviation of e, u.
    par[i,]<-c(betau,1/sqrt(tauu),1/sqrt(taue))
    u0     <-betau[p+1:q]  #extracting u so we can update tau_u.
  }

par <-par[-c(1:burnin),] #removing initial iterations
colnames(par)<-c(paste('beta',1:p,sep=''),paste('u',1:q,sep=''),'sigma_b','sigma_e')
 return(par)
}
```

Solution

```r
system.time(chain10<-normalmm.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,taue_0=1,tauu_(
```

```
##    user  system elapsed
##    3.67    0.00    3.67
```

```r
system.time(chain11<-normalmm.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,taue_0=0.2,tau
```

```
##    user  system elapsed
##    3.56    0.00    3.59
```

```r
system.time(chain12<-normalmm.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,taue_0=5,tauu_(
```

```
##    user  system elapsed
##    3.63    0.00    3.64
```

```r
library(coda)
#Estimating Gelman -Rubin diagnostics.
#Note 8000 iterations were retained, so 50:50 split is iteration 1:4000 and iteration 4001:8000

#However first we must convert the output into mcmc lists for coda to interpret.
ml1<-as.mcmc.list(as.mcmc((chain10[1:4000,])))
ml2<-as.mcmc.list(as.mcmc((chain11[1:4000,])))
ml3<-as.mcmc.list(as.mcmc((chain12[1:4000,])))
ml4<-as.mcmc.list(as.mcmc((chain10[4000+1:4000,])))
ml5<-as.mcmc.list(as.mcmc((chain11[4000+1:4000,])))
ml6<-as.mcmc.list(as.mcmc((chain12[4000+1:4000,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

```
##         Point est. Upper C.I.
## beta1    1.0005905  1.0017822
## u1       1.0002126  1.0008362
## u2       1.0002637  1.0009665
## u3       1.0003958  1.0012218
## u4       1.0000295  1.0001686
## u5       1.0001599  1.0005810
## u6       0.9998912  0.9999608
## u7       1.0000812  1.0003106
## u8       1.0001532  1.0006321
## u9       1.0004075  1.0010858
## u10      1.0000476  1.0004752
## u11      1.0000844  1.0005442
## sigma_b  1.0005441  1.0015565
## sigma_e  1.0000453  1.0004379
```

```r
#effective sample size.
effectiveSize(estml)
```

```
##    beta1       u1       u2       u3       u4       u5       u6       u7
## 23250.48 24000.00 22996.89 22942.12 24000.00 19920.87 21743.77 24000.00
##       u8       u9      u10      u11  sigma_b  sigma_e
## 22309.72 19109.04 24000.00 24000.00 12994.46 15588.48
```

```r
#Reporting posterior means and credible intervals.
#Means
```

```r
colMeans(rbind(chain10,chain11,chain12))
```

```
##       beta1          u1          u2          u3          u4          u5
##  7.60242403  0.01390440  0.25834971  0.20378433  0.04400074 -0.06571223
##          u6          u7          u8          u9         u10         u11
##  0.15023617  0.01105592 -0.02307964  0.25012577  0.05703116  0.27183842
##     sigma_b     sigma_e
##  0.20527654  0.11821907
```

```r
#95 % central Credible interval
apply(rbind(chain10,chain11,chain12) ,2, FUN =function(x) quantile(x,c(0.025,0.975) ))
```

```
##          beta1          u1         u2        u3         u4         u5
## 2.5%  7.566171 -0.02986222 0.09991092 0.0441440 -0.1425814 -0.2745050
## 97.5% 7.638656  0.05788413 0.41911658 0.3592429  0.2365529  0.1331029
##               u6         u7         u8          u9        u10       u11
## 2.5%  -0.02977526 -0.2622409 -0.2999797 -0.04305112 -0.2493746 0.1784655
## 97.5%  0.33509468  0.2772295  0.2352198  0.58498483  0.3516782 0.3646120
##         sigma_b    sigma_e
## 2.5%  0.1230549 0.09333689
## 97.5% 0.3508011 0.15192766
```

- LASSO.

```r
#Arguments
#X:  matrix of predictors dimension n times p with flat prior. Includes the intercept.
#Z:  matrix of predictors dimension n times q with normal prior for u.
#y:  response vector, length p.
#taue_0: initial value for the residual precision.
#lambda: LASSO penalty
#a.e,b.e. Hyper-parameter for gamma prior for tau_e
#iter: number of iterations
#burnin: number of initial iterations to remove
normallasso.Gibbs<-function(iter,Z,X,y,burnin,taue_0,lambda,a.e,b.e){
  library(LaplacesDemon)
  n   <-length(y) #no. observations
  p   <-dim(X)[2] #no of fixed effect predictors.
  q   <-dim(Z)[2] #no of random effect levels
  taue<-taue_0
  #Note Laplace distribution is compound of normal and exponential.
  #This results with us working with a vector tau_u = dimension of predictor with LASSO penalty.
  tauu <-rinvgaussian(q,lambda/abs(rnorm(q)),lambda^2) #Note LASSO can b

  #Building combined predictor matrix.
  W<-cbind(X,Z)
  WTW <-crossprod(W)
  library(mvtnorm)

  #storing results.
  par <-matrix(0,iter,p+q+1)

  for(i in 1:iter){
    #Conditional posteriors.

    #Updating component of normal posterior for beta,u
    Kinv  <-diag(p+q)
    diag(Kinv)[1:p]<-0
    diag(Kinv)[p+1:q]<-tauu #Adding precision of predictors with LASSO penalty.

    Prec <-taue*WTW + Kinv
    P.var  <-solve(Prec)
    P.mean <- taue*P.var%*%crossprod(W,y)
    betau <-rmvnorm(1,mean=P.mean,sigma=P.var) #sampling beta,u
    betau <-as.numeric(betau)
    err   <- y-W%*%betau
    taue  <-rgamma(1,a.e+0.5*n,b.e+0.5*sum(err^2))      #sampling tau_e, residual precision.
    #sampling tau_u, the augmented vector of precisions for predictor with LASSO penalty.
    tauu  <-rinvgaussian(q,lambda/abs(betau[-c(1:p)]),lambda^2)

    #storing iterations.
    par[i,]<-c(betau,1/sqrt(taue))
  }

par <-par[-c(1:burnin),] #removing early iterations
colnames(par)<-c(paste('beta',1:p,sep=''),paste('u',1:q,sep=''),'sigma_e')
 return(par)
```

```
}
```

Solution

```
system.time(chain13<-normallasso.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,taue_0=1,lar
```

```
## Warning: package 'LaplacesDemon' was built under R version 3.6.1

##
## Attaching package: 'LaplacesDemon'

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

##    user  system elapsed
##    3.85    0.01    3.87
```

```
system.time(chain14<-normallasso.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,taue_0=0.2,l
```

```
##    user  system elapsed
##    3.81    0.00    3.83
```

```
system.time(chain15<-normallasso.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,taue_0=5,lar
```

```
##    user  system elapsed
##    3.83    0.00    3.85
```

```
library(coda)
#Estimating Gelman -Rubin diagnostics.
#Note 8000 iterations were retained, so 50:50 split is iteration 1:4000 and iteration 4001:8000

#However first we must convert the output into mcmc lists for coda to interpret.
ml1<-as.mcmc.list(as.mcmc((chain13[1:4000,])))
ml2<-as.mcmc.list(as.mcmc((chain14[1:4000,])))
ml3<-as.mcmc.list(as.mcmc((chain15[1:4000,])))
ml4<-as.mcmc.list(as.mcmc((chain13[4000+1:4000,])))
ml5<-as.mcmc.list(as.mcmc((chain14[4000+1:4000,])))
ml6<-as.mcmc.list(as.mcmc((chain15[4000+1:4000,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

```
##          Point est. Upper C.I.
## beta1     1.0002566   1.000848
## u1        0.9999005   1.000086
## u2        0.9999033   1.000077
## u3        0.9999584   1.000242
## u4        0.9999808   1.000308
## u5        1.0000840   1.000529
## u6        0.9999940   1.000225
## u7        1.0001983   1.000805
## u8        1.0004376   1.001369
## u9        0.9999510   1.000200
## u10       1.0005808   1.001583
## u11       1.0000450   1.000421
## sigma_e   1.0001940   1.000551
```

```r
#effective sample size.
effectiveSize(estml)
```

```
##    beta1       u1       u2       u3       u4       u5       u6       u7
## 24000.00 25209.60 24000.00 23195.45 24000.00 23025.30 22804.53 24000.00
##       u8       u9      u10      u11  sigma_e
## 23349.11 21831.36 22794.48 22557.86 13864.81
```

```r
#Reporting posterior means and credible intervals.
#Means
colMeans(rbind(chain13,chain14,chain15))
```

```
##        beta1           u1           u2           u3           u4
##  7.602330480  0.011885803  0.280320153  0.147112797  0.057709753
##           u5           u6           u7           u8           u9
## -0.155574985  0.215273921 -0.003039792 -0.119749787  0.526449930
##          u10          u11      sigma_e
## -0.052009170  0.254331718  0.119817781
```

```r
#Credible interval
apply(rbind(chain13,chain14,chain15) ,2, FUN =function(x) quantile(x,c(0.025,0.975) ))
```

```
##          beta1          u1         u2          u3         u4          u5
## 2.5%  7.566110 -0.03706777 0.07752921 -0.05359614 -0.1918535 -0.41229813
## 97.5% 7.638336  0.06088788 0.47873881  0.35473214  0.3093565  0.09508631
##              u6         u7         u8          u9        u10       u11
## 2.5%  -0.0053994 -0.4412188 -0.6051434 -0.02632271 -0.6259150 0.1362780
## 97.5%  0.4359041  0.4359297  0.3602631  1.08511898  0.5237146 0.3721081
##          sigma_e
## 2.5%  0.09420474
## 97.5% 0.15491646
```