Week 9 Lab Solutions – MAST90125: Bayesian Statistical learning

Writing Gibbs samplers for linear models with proper priors for β.

In this lab, we will continue discussing how to write code for Gibbs sampling of linear models with proper priors. We will look at the data in `USJudgeRatings.csv`, which is available on Canvas. We will assume the variable `RTEN` is the response and the other variables as predictors. Meanwhile, how to analyze chains will be included. In addition, we will show another example to see the difference between empirical Bayes and full Bayes.

Download `USJudgeRatings.csv` from Canvas.

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 14 and 15 useful. Try comparing the posterior distributions to see what differences the priors cause.

Comment: Running this code and seeing it does not produce warning messages does not prove anything. You still want to check convergence. Remember in the assignment, you were given the following:

```
Processing chains for calculation of Gelman-Rubin diagnostics. Imagine you have 4 chains of
a multi-parameter problem, and thinning already completed, called par1,par2,par3,par4

Step one: Converting the chains into mcmc lists.
library(coda)
par1<-as.mcmc.list(as.mcmc((par1)))
par2<-as.mcmc.list(as.mcmc((par2)))
par3<-as.mcmc.list(as.mcmc((par3)))
par4<-as.mcmc.list(as.mcmc((par4)))

Step two: Calculating diagnostics

par.all<-c(par1,par2,par3,par4)
gelman.diag(par.all)

Step Three: Calculating effective sample size
effectiveSize(estml)
```

You may find this useful to check the performance of the codes given.

Examples of Gibbs samplers for linear models

First, look at the following two functions. What are they in Lecture 14?

```
Gibbs.lm1<-function(X,y,tau0,iter,burnin){
p <- dim(X)[2]
XTX <- crossprod(X)
XTXinv <-solve(XTX)
XTY <- crossprod(X,y)
```

```
betahat<-solve(XTX,XTY)
tau    <-tau0
library(mvtnorm)

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

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

```
Gibbs.lm2<-function(X,y,tau0,iter,burnin){
p <- dim(X)[2]
svdX <-svd(X)
U    <-svdX$u
Lambda<-svdX$d
V    <-svdX$v
Vbhat <- crossprod(U,y)/Lambda
tau <-tau0

vbeta<-rnorm(p)
par<-matrix(0,iter,p+1)
for( i in 1:iter){
  sqrttau<-sqrt(tau)
  vbeta <- rnorm(p,mean=Vbhat,sd=1/(sqrttau*Lambda) )
  beta <-V%*%vbeta
  err  <- y-X%*%beta
  tau  <- rgamma(1,0.5*n,0.5*sum(err^2))
  par[i,] <-c(beta,tau)
}

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

Solution

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

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

```
## Warning: package 'mvtnorm' was built under R version 4.3.1
```

```
##    user  system elapsed
##   1.251   0.021   1.590
```

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

```
##    user  system elapsed
##   1.169   0.010   1.179
```

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

```
##    user  system elapsed
##   1.178   0.008   1.186
```

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

```
##    user  system elapsed
##   0.050   0.004   0.054
```

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

```
##    user  system elapsed
##   0.046   0.006   0.052
```

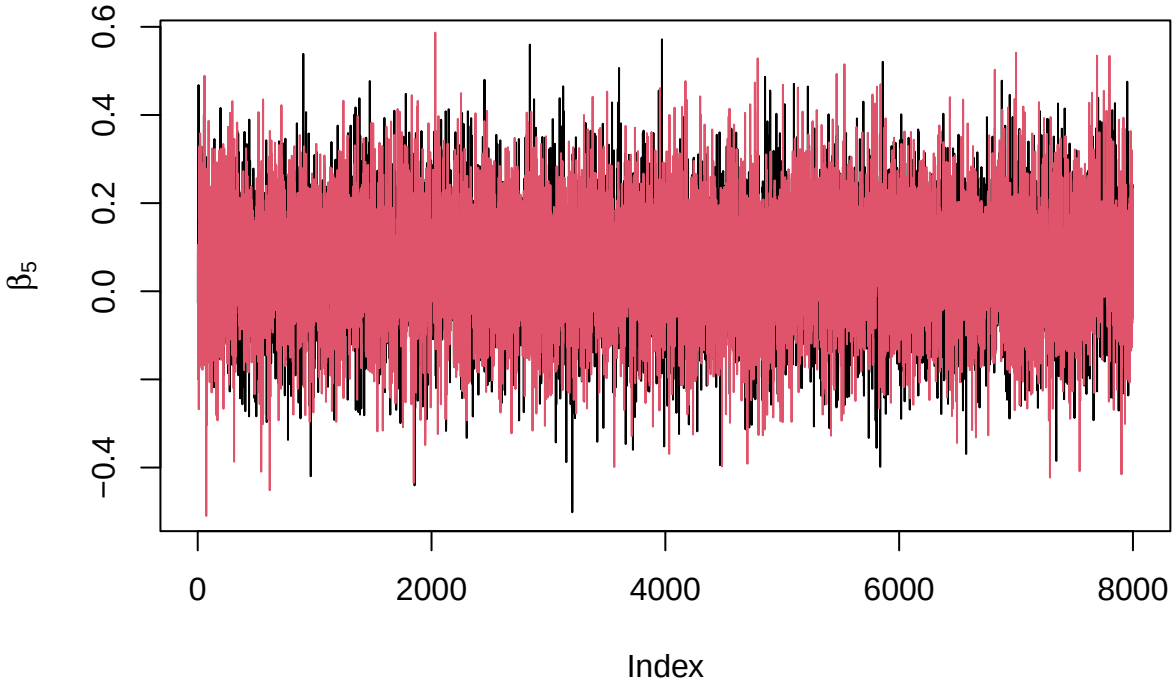
```r
system.time(chain6<-Gibbs.lm2(X=X,y=response,tau0=0.2,iter=10000,burnin=2000))
```

```
##    user  system elapsed
##   0.045   0.002   0.046
```

```r
#Comparing one co-efficient (the 5th)
plot(chain1[,5],type='l',ylab=expression(beta[5]))
lines(chain4[,5],type='l',col=2,ylab=expression(beta[5]))

library(coda)
```

```
## Warning: package 'coda' was built under R version 4.3.1
```




```
#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,]  1.0000546  1.0001983
##  [2,]  1.0000130  1.0001521
##  [3,]  1.0001571  1.0006927
##  [4,]  0.9998717  0.9999236
##  [5,]  0.9999469  1.0002265
##  [6,]  1.0000144  1.0003599
##  [7,]  1.0002340  1.0008845
##  [8,]  0.9999186  1.0001678
##  [9,]  0.9999212  1.0000308
## [10,]  0.9999739  1.0002151
```

```
## [11,]  1.0000273  1.0002939
## [12,]  0.9999667  1.0002181
## [13,]  1.0004611  1.0014586
```

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

```
##     var1     var2     var3     var4     var5     var6     var7     var8
## 24000.00 24000.00 23550.31 23454.82 23439.34 24000.00 24270.82 24095.24
##     var9    var10    var11    var12    var13
## 24000.00 24000.00 24000.00 24000.00 13396.32
```

```
#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,]  1.0001409  1.0002675
##  [2,]  0.9998673  0.9999074
##  [3,]  1.0001804  1.0008016
##  [4,]  1.0002031  1.0007557
##  [5,]  1.0000950  1.0006095
##  [6,]  0.9999659  1.0001967
##  [7,]  1.0000982  1.0004461
##  [8,]  1.0002069  1.0007164
##  [9,]  1.0000727  1.0004012
## [10,]  1.0002360  1.0007758
## [11,]  1.0000479  1.0002582
## [12,]  1.0005202  1.0014558
## [13,]  1.0008467  1.0023129
```

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

```
##     var1     var2     var3     var4     var5     var6     var7     var8
## 24454.83 24000.00 24000.00 24000.00 23139.69 24835.09 24000.00 24000.00
##     var9    var10    var11    var12    var13
## 24000.00 24000.00 23976.29 24000.00 13664.65
```

Then, we focus on things we talked about in Lecture 15.

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

```
#Inputs: iter: no of iterations.
#Z: covariate matrix for parameters with normal prior.
#X: covariate matrix for parameters with flat prior.
#y: response vector.
#burnin: no of initial iterations to throw out.
#taue_0, tauu_0, initial values for tau, \tau_\beta
#a.e, b.e, a.u, b.u, hyper-parameters for priors for \tau, \tau_\beta.
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
  beta0<-rnorm(p) #initial value for \beta_0 (parameters with flat prior 'fixed effects')
  u0   <-rnorm(q,0,sd=1/sqrt(tauu))
     #intial value for u_0 (parameters with normal prior ,'random effects')

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

  #storing results.
  par <-matrix(0,iter,p+q+2) #matrix for storing iterations, p fixed effects,
   # q random effects, 2 because two inverse variance components.

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

  #Calculate WTy
  WTy<-crossprod(W,y)

  for(i in 1:iter){
    #Conditional posteriors.
    tauu <-rgamma(1,a.u+0.5*q,b.u+0.5*sum(u0^2)) #sampling tau_u from conditional posterior.
    #Updating component of normal posterior for beta,u
    Prec <-WTW + tauu*I0/taue
    P.mean <- solve(Prec)%*%WTy
    P.var  <-solve(Prec)/taue
    betau <-rmvnorm(1,mean=P.mean,sigma=P.var) #sample beta, u from joint full conditional posterior.
    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 from conditional posterior.
    #storing iterations.
    par[i,]<-c(betau,1/sqrt(tauu),1/sqrt(taue))
         #Note we are storing standard deviation, not precisions.
    beta0  <-betau[1:p]
```

```
    u0     <-betau[p+1:q]
  }

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

Solution

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



system.time(chain10<-normalmm.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,
                            taue_0=1,tauu_0=1,a.u=0.001,b.u=0.001,a.e=0.001,b.e=0.001))
```

```
##    user  system elapsed
##   1.536   0.009   1.546
```

```
system.time(chain11<-normalmm.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,
                           taue_0=0.2,tauu_0=5,a.u=0.001,b.u=0.001,a.e=0.001,b.e=0.001))
```

```
##    user  system elapsed
##   1.529   0.009   1.538
```

```
system.time(chain12<-normalmm.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,burnin=2000,
                           taue_0=5,tauu_0=0.2,a.u=0.001,b.u=0.001,a.e=0.001,b.e=0.001))
```

```
##    user  system elapsed
##   1.499   0.009   1.507
```

```
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)
```

```r
#Gelman-Rubin diagnostic. All the diagnostic point estimates are very close to 1,
# indicating convergence has been reached.
gelman.diag(estml)[[1]]
```

```
##         Point est. Upper C.I.
## beta1    1.0002109   1.000893
## u1       1.0002007   1.000837
## u2       0.9999599   1.000182
## u3       1.0000264   1.000088
## u4       0.9999911   1.000355
## u5       1.0002621   1.000940
## u6       1.0000302   1.000410
## u7       1.0001155   1.000463
## u8       1.0001515   1.000589
## u9       0.9999328   1.000042
## u10      0.9999980   1.000252
## u11      1.0000949   1.000394
## sigma_b  1.0005242   1.001379
## sigma_e  1.0000859   1.000484
```

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

```
##    beta1       u1       u2       u3       u4       u5       u6       u7
## 24000.00 25310.29 22676.24 22165.17 24000.00 19415.22 20673.86 24000.00
##       u8       u9      u10      u11  sigma_b  sigma_e
## 22860.33 19166.54 24000.00 24000.00 13443.39 15329.54
```

```r
#Reporting posterior means and credible intervals.
#Means
colMeans(rbind(chain10,chain11,chain12))
```

```
##       beta1          u1          u2          u3          u4          u5
##  7.60249568  0.01397495  0.25828893  0.20374729  0.04415980 -0.06617360
##          u6          u7          u8          u9         u10         u11
##  0.15004635  0.01110977 -0.02183022  0.24715125  0.05870011  0.27222132
##     sigma_b     sigma_e
##  0.20383247  0.11831747
```

```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%  7.566566 -0.02997486 0.1006920 0.04528655 -0.1444325 -0.2739716
## 97.5% 7.638759  0.05817590 0.4179443 0.35598653  0.2307468  0.1346438
##                u6         u7         u8          u9        u10       u11
## 2.5%  -0.03108493 -0.2568735 -0.2988203 -0.04344572 -0.2475964 0.1801804
## 97.5%  0.33705909  0.2752924  0.2349650  0.58120731  0.3508765 0.3642080
##          sigma_b    sigma_e
## 2.5%  0.1223467 0.09350106
## 97.5% 0.3490019 0.15130892
```

Example: LASSO with either γ fixed or estimated from the data.

In Week 8 Lab, you were given a function to fit a Bayesian LASSO, assuming the penalty, γ, was fixed. You would have noted that unlike frequentist LASSO, coefficient estimates in a Bayesian LASSO were never exactly zero.

In order to estimate γ, we return to the conditional posteriors we outlined in lecture 15 for Bayesian LASSO:

$$\begin{aligned} p(\boldsymbol{\beta}|\mathbf{y},\mathbf{X},\sigma_1^2,\dots\sigma_p^2,\tau_e) &= \mathcal{N}(\tau_e(\tau_e\mathbf{X}'\mathbf{X}+\mathbf{K}^{-1})^{-1}\mathbf{X}'\mathbf{y},(\tau_e\mathbf{X}'\mathbf{X}+\mathbf{K}^{-1})^{-1}), \text{where } \mathbf{K}_{jj}=\sigma_j^2 \\ p(\tau_e|\mathbf{y},\boldsymbol{\beta},\mathbf{X}) &= \text{Ga}(\alpha_e+n/2,\gamma_e+(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})'(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})/2), \\ p((\sigma_j^2)^{-1}|\gamma,\boldsymbol{\beta}) &= \text{InvGaussian}(\gamma/|\boldsymbol{\beta}_j|,\gamma^2). \end{aligned}$$

However we want to estimate γ as well. We know that the only place γ appeared in the joint distribution $p(\mathbf{y},\boldsymbol{\beta},\tau_e,\sigma_1^2,\dots\sigma_p^2,\gamma)$ is in the expansion of the Laplace (or double exponential) prior for $\boldsymbol{\beta}$,

$$\prod_{j=1}^{p}\frac{1}{\sqrt{2\pi\sigma_j^2}}e^{-\frac{\beta_j^2}{2\sigma_j^2}}\frac{\gamma^2}{2}e^{-\frac{\gamma^2\sigma_j^2}{2}} \propto (\gamma^2)^p e^{-\frac{\gamma^2\sum_{j=1}^{p}\sigma_j^2}{2}}.$$

Looking at this kernel, we see that γ^2 (note not γ) corresponds to a kernel of a Gamma distribution. We also know that a Gamma prior and Gamma likelihood leads to a Gamma posterior. Therefore if we assume $\gamma^2 \sim \text{Ga}(\alpha,\delta)$ *a priori*, then the conditional posterior of γ^2 is,

$$p(\gamma^2|\sigma_1^2,\dots\sigma_p^2) = \text{Ga}(\alpha+p,\delta+0.5\sum_{j=1}^{p}\sigma_j^2)$$

As $\gamma \in (0,\infty)$, squaring γ is a one to one transformation. Therefore, there is no problem sampling γ^2 from the conditional posterior, taking the square root to obtain a draw for γ and then cycling through the remaining conditional posteriors.

Code for implementing LASSO with either γ fixed or estimated Note: To run this, you need to install R package *LaplacesDemon*

```
#LASSO with fixed gamma
#Arguments are
#iter: no of iterations
#Z: Predictor matrix for effects with LASSO penalty
#X: Predictor matrix for effects without LASSO penalty (typically only intercept)
#y: response vector
#burnin: number of initial iterations to discard.
#taue_0: initial guess for residual precision.
#gamma: prior parameter for Laplace (double exponential) prior for u
#a.e, b.e: hyper-parameters of gamma prior for taue

normallassofixed.Gibbs<-function(iter,Z,X,y,burnin,taue_0,gamma,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
```

```
  tauu <-rinvgaussian(q,gamma/abs(rnorm(q)),gamma^2)

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

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

  #Calculating log predictive densities
  lppd<-matrix(0,iter,n)

  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

    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)
    betau <-as.numeric(betau)
    err   <- y-W%*%betau
    taue  <-rgamma(1,a.e+0.5*n,b.e+0.5*sum(err^2))
    tauu <-rinvgaussian(q,gamma/abs(betau[-c(1:p)]),gamma^2)

    #storing iterations.
    par[i,]<-c(betau,1/sqrt(taue))
    #Storing log=predictive density
    lppd[i,]= dnorm(y,mean=as.numeric(W%*%betau),sd=1/sqrt(taue))
  }

  lppd      = lppd[-c(1:burnin),]
  lppdest   = sum(log(colMeans(lppd)))        #Estimating lppd for whole dataset.
  pwaic2    = sum(apply(log(lppd),2,FUN=var))
     #Estimating effective number of parameters.
  par <-par[-c(1:burnin),]
  colnames(par)<-c(paste('beta',1:p,sep=''),paste('u',1:q,sep=''),'sigma_e')
  mresult<-list(par,lppdest,pwaic2)
  names(mresult)<-c('par','lppd','pwaic')
  return(mresult)
}

#############################################################################################
#Now the function where gamma is updated in the Gibbs sampler.
#Arguments are
#iter: no of iterations
#Z: Predictor matrix for effects with LASSO penalty
#X: Predictor matrix for effects without LASSO penalty (typically only intercept)
```

```r
#y: response vector
#burnin: number of initial iterations to discard.
#taue_0: initial guess for residual precision.
#a.l, b.l: hyper-parameters of gamma prior for (gamma^2),
#where gamma is parameter of Laplace for u.
#a.e, b.e: hyper-parameters of gamma prior for taue

normallassoestimated.Gibbs<-function(iter,Z,X,y,burnin,taue_0,a.l,b.l,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
  gamma2<-rgamma(1,a.l,b.l)
  gamma <-sqrt(gamma2)
  tauu <-rinvgaussian(q,gamma/abs(rnorm(q)),gamma^2)

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

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

  #Calculating log predictive densities
  lppd<-matrix(0,iter,n)

  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

    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)
    betau <-as.numeric(betau)
    err   <- y-W%*%betau
    taue  <-rgamma(1,a.e+0.5*n,b.e+0.5*sum(err^2))
    tauu <-rinvgaussian(q,gamma/abs(betau[-c(1:p)]),gamma^2)
    gamma2 <-rgamma(1,a.l+q,b.l+0.5*sum(1/tauu))
    gamma  <-sqrt(gamma2)
    #storing iterations.
    par[i,]<-c(betau,1/sqrt(taue),gamma)
    lppd[i,]= dnorm(y,mean=as.numeric(W%*%betau),sd=1/sqrt(taue))
  }

  lppd      = lppd[-c(1:burnin),]
  lppdest   = sum(log(colMeans(lppd)))        #Estimating lppd for whole dataset.
```

```
  pwaic2    = sum(apply(log(lppd),2,FUN=var))
       #Estimating effective number of parameters.
  par <-par[-c(1:burnin),]
  colnames(par)<-c(paste('beta',1:p,sep=''),paste('u',1:q,sep=''),'sigma_e','gamma')
  mresult<-list(par,lppdest,pwaic2)
  names(mresult)<-c('par','lppd','pwaic')
  return(mresult)
}
```

```
data<-read.csv('./USJudgeRatings.csv')
# data<-read.csv(file.choose())
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))
check1<-normallassoestimated.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,
                        burnin=2000,taue_0=1,a.l=0.1,b.l=0.1,a.e=0.01,b.e=0.01)
```

Fitting the two LASSO Gibbs samplers to the US Judge Ratings data

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

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

```
check2<-normallassoestimated.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,
                          burnin=2000,taue_0=5,a.l=0.1,b.l=0.1,a.e=0.01,b.e=0.01)
check3<-normallassoestimated.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,
                        burnin=2000,taue_0=0.2,a.l=0.1,b.l=0.1,a.e=0.01,b.e=0.01)


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((check1$par[1:4000,])))
ml2<-as.mcmc.list(as.mcmc((check2$par[1:4000,])))
ml3<-as.mcmc.list(as.mcmc((check3$par[1:4000,])))
ml4<-as.mcmc.list(as.mcmc((check1$par[4000+1:4000,])))
ml5<-as.mcmc.list(as.mcmc((check2$par[4000+1:4000,])))
ml6<-as.mcmc.list(as.mcmc((check3$par[4000+1:4000,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

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

```
##          Point est. Upper C.I.
## beta1     1.0001772   1.000629
## u1        1.0002857   1.001059
## u2        1.0005268   1.001453
## u3        1.0006153   1.001828
## u4        1.0000602   1.000345
## u5        1.0000233   1.000357
## u6        1.0000884   1.000595
## u7        1.0001400   1.000598
## u8        1.0003283   1.000894
## u9        0.9999441   1.000162
## u10       1.0003473   1.000918
## u11       1.0000078   1.000210
## sigma_e   1.0001205   1.000519
## gamma     1.0013064   1.003250
```

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

```
##     beta1        u1        u2        u3        u4        u5        u6        u7
## 24000.000 22236.254 14521.407 13715.201 21947.017 16114.325 14737.364 22582.652
##        u8        u9       u10       u11   sigma_e     gamma
## 17424.210 11040.251 19174.049 18393.241 14387.837  4327.511
```

```
#For Empirical Bayes, we will fix gamma to the posterior mean of
#gamma from the chains fitted above. Note draws of gamma were stored in column 14.
#as we have intercept (column 1), 11 predictors (columns 2:12) and one standard deviation (column 13).
gamm.est<-mean(c(check1$par[,14],check2$par[,14],check3$par[,14]));gamm.est
```

```
## [1] 4.224977
```

```
check4<-normallassofixed.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,
                    burnin=2000,taue_0=1,gamma=gamm.est,a.e=0.01,b.e=0.01)
check5<-normallassofixed.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,
                    burnin=2000,taue_0=5,gamma=gamm.est,a.e=0.01,b.e=0.01)
check6<-normallassofixed.Gibbs(iter=10000,Z=Pred,X=intercept,y=response,
                    burnin=2000,taue_0=0.2,gamma=gamm.est,a.e=0.01,b.e=0.01)

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.
fml1<-as.mcmc.list(as.mcmc((check4$par[1:4000,])))
fml2<-as.mcmc.list(as.mcmc((check5$par[1:4000,])))
fml3<-as.mcmc.list(as.mcmc((check6$par[1:4000,])))
fml4<-as.mcmc.list(as.mcmc((check4$par[4000+1:4000,])))
fml5<-as.mcmc.list(as.mcmc((check5$par[4000+1:4000,])))
fml6<-as.mcmc.list(as.mcmc((check6$par[4000+1:4000,])))
fixml<-c(fml1,fml2,fml3,fml4,fml5,fml6)
```

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

```
##         Point est. Upper C.I.
## beta1    1.0002101   1.000703
## u1       1.0001303   1.000279
## u2       1.0000715   1.000430
## u3       0.9999319   1.000096
## u4       0.9999460   1.000069
## u5       1.0006862   1.001978
## u6       1.0011216   1.003007
## u7       1.0001180   1.000444
## u8       1.0000112   1.000198
## u9       1.0002030   1.000881
## u10      1.0003300   1.000877
## u11      1.0000837   1.000557
## sigma_e  0.9999162   1.000137
```

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

```
##    beta1       u1       u2       u3       u4       u5       u6       u7
## 24000.00 22235.31 15967.09 15567.40 20366.03 19298.91 17410.95 22405.95
##       u8       u9      u10      u11  sigma_e
## 19537.22 12855.45 20290.90 17889.81 14899.74
```

```r
#Combining all chains from each model
check.all1<-rbind(check1$par,check2$par,check3$par)
   #all chains where gamma was estimated.
check.all2<-rbind(check4$par,check5$par,check6$par)
   #all chains where gamma was fixed.

#Plots of results.
par(mfrow=c(5,3))
#Intercept
plot(density(check.all1[,1]),col=1,lty=1,xlab=expression(beta[0]),
          main='Comparison of posteriors for intercepts',cex.lab=1.5)
lines(density(check.all2[,1]),col=2,lty=2)
legend('topright',legend=c(expression(paste(gamma,' estimated')),
              expression(paste(gamma,' fixed'))),col=1:2,lty=1,bty='n',cex=1.5)

#co-officients
for(i in 1:11){
  plot(density(check.all1[,i+1]),col=1,lty=1,xlab=paste('u',i,sep=''),
            main='Comparison of posteriors for coefficents',cex.lab=1.5)
  lines(density(check.all2[,i+1]),col=2,lty=2)
  curve(0.5*gamm.est*exp(-gamm.est*abs(x)),col=3,lty=1,add=TRUE)
  legend('topright',legend=c(expression(paste(gamma,' estimated')),
      expression(paste(gamma,' fixed')),'prior'),col=1:3,lty=1,bty='n',cex=1.5)
}

#Standard deviation
```

```
plot(density(check.all1[,13]),col=1,lty=1,xlab=expression(sigma),
         main='Comparison of posteriors for std deviation',cex.lab=1.5)
lines(density(check.all2[,13]),col=2,lty=2)
legend('topright',legend=c(expression(paste(gamma,' estimated')),
          expression(paste(gamma,' fixed'))),col=1:2,lty=1,bty='n',cex=1.5)

#Comparing variances for Empirical Bayes versus fully Bayesian.

#Empirical Bayes
apply(check.all2,2,FUN=var) #Empirical Bayes
```

```
##        beta1           u1           u2           u3           u4           u5
## 0.0003423721 0.0005084763 0.0089881101 0.0084196376 0.0093404642 0.0110686054
##           u6           u7           u8           u9          u10          u11
## 0.0100481965 0.0199005102 0.0213223992 0.0479428664 0.0299520326 0.0029079075
##      sigma_e
## 0.0002319389
```

```
apply(check.all1,2,FUN=var) #Fully Bayesian
```

```
##        beta1           u1           u2           u3           u4           u5
## 0.0003471600 0.0005122914 0.0091192497 0.0085831008 0.0094318606 0.0114708166
##           u6           u7           u8           u9          u10          u11
## 0.0102029864 0.0207913968 0.0226252298 0.0502097548 0.0315929347 0.0029227155
##      sigma_e        gamma
## 0.0002315826 1.2930483895
```

```
#Fully Bayesian lppd estimate
check1$lppd
```

```
## [1] 34.49642
```

```
check2$lppd
```

```
## [1] 34.59388
```

```
check3$lppd
```

```
## [1] 34.58872
```

```
#Empirical Bayes lppd estimate
check4$lppd
```

```
## [1] 34.53274
```

```
check5$lppd
```

```
## [1] 34.56822
```

```
check6$lppd
```

```
## [1] 34.55171
```

```
#Fully Bayesian effective number of parameters
check1$pwaic
```

```
## [1] 9.152544
```

```
check2$pwaic
```

```
## [1] 9.198427
```

```
check3$pwaic
```

```
## [1] 9.106294
```

```
#Empirical Bayes effective number of parameters
check4$pwaic
```

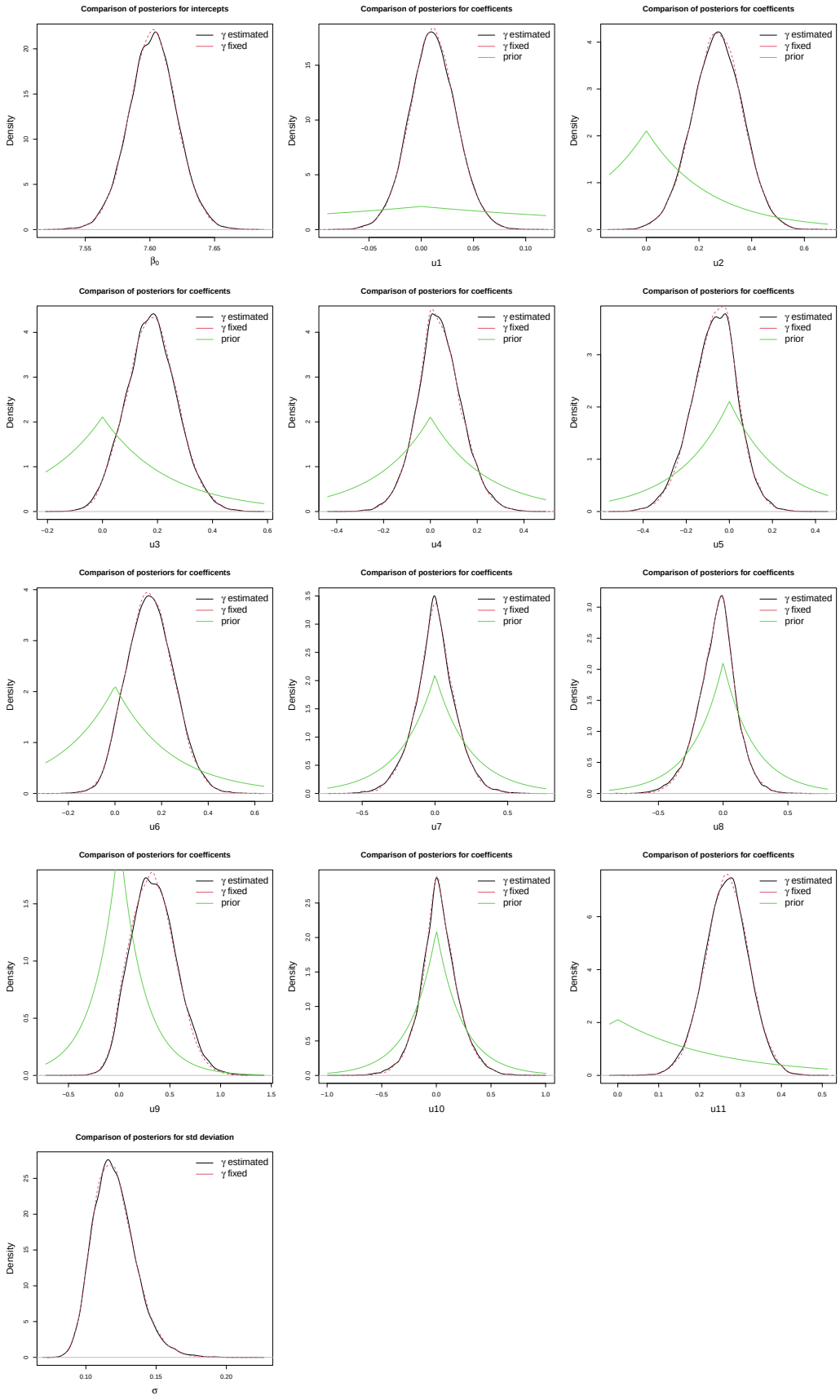
```
## [1] 9.147985
```

```
check5$pwaic
```

```
## [1] 9.094478
```

```
check6$pwaic
```

```
## [1] 9.147622
```


Comparison of posteriors for intercepts
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for coefficents
Comparison of posteriors for std deviation
γ estimated
γ fixed
prior
Density
β0
u1
u2
u3
u4
u5
u6
u7
u8
u9
u10
u11
σ