```
##
#if(nchar(Sys.getenv("LONG_TEST")) != 0) {R=10000} else {R=10}

#-------------------------------------------------------------------------------
# simulation of app-store project data to show identification
# By: Meisam Hejazi Nia
# Date July 7th
#-------------------------------------------------------------------------------
rm(list=ls(pattern="^tmp"))
rm(list=ls())
library(bayesm)
library(foreach)
library(abind)
library(doSNOW)
library(DEoptim)
library(MASS)
library(numDeriv)
library("corpcor")
cl=makeCluster(7)
registerDoSNOW(cl)
set.seed(66)
par(mfrow=c(3,1))
#---------------------------------------------------------------------------
# Read data of category diffusion
#===========================================================================
ncat = 10
T    = 258
categoryDiffData=read.csv(
"C:/Users/mxh109420/Desktop/MobileAppProject/GlobalCategoryDiffusion.csv",header=T)
categoryDiff= matrix(rep(0,ncat*T),ncol=ncat)
for (i in 1:ncat){
   categoryDiff[,i]=categoryDiffData[[i+1]]
}
categoryDiff=t(categoryDiff)
plot(categoryDiff[1,])

catlatent = categoryDiff
totalForce = catlatent # b/c in FFBS we only would need aggregate
categoryDiffWeekly = matrix(rep(0,((T-(T%%7))/7*ncat)),ncol=ncat)
# prepare data for FFBS by aggregating
for (i in 1:ncat){
   currentCat = i
   currentSales = totalForce[currentCat,]
   currentSalesTemp = t(matrix(currentSales[1:(T-(T%%7))],nrow=7))
   categoryDiffWeekly[,i]=rowMeans(currentSalesTemp)
}
categoryDiffWeekly = t(categoryDiffWeekly)
T = dim(categoryDiffWeekly)[2]
# set the scale
categoryDiffWeekly = categoryDiffWeekly /10
#---------------------------------------------------------------------------
# Read CategoryHB data
#===========================================================================
nzcat = 1
```

```
CategoryHBData=read.csv("C:/Users/mxh109420/Desktop/MobileAppProject/CategoryHB.csv",header=T)
CategoryHB= matrix(rep(0,ncat*nzcat),ncol=nzcat)
CategoryHB[,1]=CategoryHBData[[3]]


#--------------------------------------------------------------------------------
# first: simulate state space of category in a for loop for j=1...J (HB)+ complementarity
# HB includes: popularity of category
#--------------------------------------------------------------------------------
Zcat=CategoryHB
Zcat=t(t(Zcat)-apply(Zcat,2,mean))          # demean Zcat, popularity explanator of category
ncompcat= 3                                 # no of mixture components of category is consider 3
Deltacat=matrix(runif(3)*1e-20,ncol=1) # generate Delta for thetacat=Deltacat*Zcat+ujcat
Deltacat[1,]=0.0003 # set p's mean data
Deltacat[2,]=0.001 # set q's mean data
Deltacat[3,]=1000 # set Cj's mean data
compscat=NULL
compscat[[1]]=list(mu=runif(3)*1e-6,rooti=diag(rep(1,3)*1e6))
compscat[[2]]=list(mu=runif(3)*1e-7,rooti=diag(rep(1,3)*1e6))
compscat[[3]]=list(mu=runif(3)*1e-9,rooti=diag(rep(1,3)*1e6))
pveccat=c(.4,.2,.4)

# error of the state equationn for the diffusion of category
wcat = 0.5*diag(ncat)
ewcat = t(chol(wcat))%*%matrix(rnorm(ncat*T,mean=0,sd=1),ncol=T)

catlatent = categoryDiffWeekly;   # initialize state and allocate space
thetacatj = matrix(rep(1,ncat*(3)),ncol=3)
colnames(thetacatj) <- c("p","q","Cj")
for (i in 1:ncat){
   thetacatj[i,]=Deltacat%*%Zcat[i,]+as.vector(rmixture(1,pveccat,compscat)$x)
   # make sure that market size (M), p and q are positive for the sake of simulation
   thetacatj[i,1]=abs(thetacatj[i,1]);
   thetacatj[i,2]=abs(thetacatj[i,2]);
   thetacatj[i,3]=abs(thetacatj[i,3]);
}

# check the data generated
plot( catlatent[1,], type="l")
plot( catlatent[2,], type="l")
# for (i in 1:ncat){
#    plot(catlatent[i,],type="l")
#    par(ask=TRUE)
# }

#=====================================================================================
#                             End of Simulating the data
#=====================================================================================

#=====================================================================================
#                             Beginning  of Estimation
#=====================================================================================

#-------------------------------------------------------------------------------------
```

```
#                             Extended Kalman Filter for Category
#------------------------------------------------------------------------------------

#----------------------------
#   Bass function for the Category
#----------------------------
# for test:
#ctbar=catlatent[,1]
fccat= function(ctbar,thetacatj) {
   # ctbar is vector of ncat latent (mean of previous period)
   ##  Bass diffusion function for category, getting old vector of latent mean and returning
   the next latent mean
   ncat = length(ctbar)
   thetacatj =matrix(as.numeric(thetacatj),nrow=ncat)
   newmean = rep(0,ncat);
   maxDiff = rep(0,ncat)
   colnames(thetacatj) <- c("p","q","Cj")
   i = 1   # for the first category
   newmean[i]=ctbar[i]+
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"]))*
      (thetacatj[i,"Cj"]-ctbar[i]);
    if (abs(thetacatj[i,"Cj"])<abs(ctbar[i])){
        newmean[i]=ctbar[i]
    }
   maxDiff[i] = thetacatj[i,"Cj"]
   # treat element in the middle
   for (i in 2:(ncat-1)){
      newmean[i]=ctbar[i]+
         (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"]))*
         (thetacatj[i,"Cj"]-ctbar[i]);
    if (abs(thetacatj[i,"Cj"])<abs(ctbar[i])){
        newmean[i]=ctbar[i]
    }
       maxDiff[i] = thetacatj[i,"Cj"]
   }
   # treat last element differently
   i = ncat;
   newmean[i]=ctbar[i]+
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"]))*
      (thetacatj[i,"Cj"]-ctbar[i]);
    if (abs(thetacatj[i,"Cj"])<abs(ctbar[i])){
        newmean[i]=ctbar[i]
    }
    maxDiff[i] = thetacatj[i,"Cj"]
   newmean = pmin(newmean,maxDiff)
   newmean = pmax(0,newmean)
   return(list(newmean=newmean,maxDiff=maxDiff))
}

#----------------------------
#   Jacobian of Bass for Category
#----------------------------
# for test:
#ctbar=catlatent[,1]
```

```
Jccat= function(ctbar,thetacatj,maxDiff) {
   ctbar= pmin(ctbar,maxDiff)
   ctbar= pmax(0,ctbar)

   # ctbar is vector of ncat latent (mean of previous period)
   ##  Bass diffusion function for category, getting old vector of latent mean and returning
   the next latent mean
   ncat = dim(thetacatj)[1]
   newJacob = matrix(rep(0,ncat*ncat),ncol=ncat);
   colnames(thetacatj) <- c("p","q","Cj")
   i = 1   # for the first category
   newJacob[i,i]=1+(thetacatj[i,"q"]/thetacatj[i,"Cj"])*(thetacatj[i,"Cj"]-ctbar[i])-
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"]))

   # treat element in the middle
   for (i in 2:(ncat-1)){
      newJacob[i,i]=1+(thetacatj[i,"q"]/thetacatj[i,"Cj"])*(thetacatj[i,"Cj"]-ctbar[i])-
         (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"]))
   }
   # treat last element differently
   i = ncat;
   newJacob[i,i]=1+(thetacatj[i,"q"]/thetacatj[i,"Cj"])*(thetacatj[i,"Cj"]-ctbar[i])-
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"]))

   return(list(newJacob=newJacob))
}



#----------------------------
# EKF of the app categories
# for now assume v is diagonal, so I do not use matrix form as it  is complex (simplification)
# As vectorization was not possible I will use parallelization
#----------------------------
# to test:
# ycat = catIndvlatent
# Fcat = gammaIndv
# pcat = ncat
# m0cat = 3*matrix(c(rep(1,pcat)),ncol=ncat)
# C0cat = 2*diag(c(rep(1,pcat)))
# vcat = array(rep(0,nIndv*nIndv*ncat),dim=c(nIndv,nIndv,ncat))
# for (j in 1:ncat){
#       vcat[,,j] = 0.1*diag(nIndv)
# }
# wcat = 0.5*diag(c(rep(1,pcat)))
# thetacatj = thetacatjest
catEKF= function(ycat,Fcat,pcat,m0cat,C0cat,vcat,wcat,thetacatj) {
   # Definition of Variables
   # ycat: [I*J*T]  data to use as observation equation
   # Fcat   : [I*1]  it is the same across time
   # pcat   : 1 for now as only one state is running EKF
   # m0cat: [p*J] for the mean of the state of current category
   # C0cat: [p*p*J] for the variance of state equation at each point in time
   # vcat : [I*I*J]   for simplicity it could be diagonal but general case is also possible
   # wcat : [J*J]   for the variance of state equation of current category
```

```
  # thetacatj: [J*(2+J)] for the coefficients, generaly it is GT
  T    = dim(ycat)[1]
  ncat = dim(ycat)[2]
  MSEcat = matrix(c(rep(0,T*ncat)),ncol=ncat)                        # in each loop sum for
  all the individuals
  MADcat = matrix(c(rep(0,T*ncat)),ncol=ncat)                                # in each
  loop sum for all the individuals
  Y1cat = array(c(rep(0,T*ncat)),dim=c(ncat,T))    # T*J matrix
  mcat = matrix(rep(m0cat,T),ncol=T)
  Ccat = array(rep(0,pcat*pcat*T),dim=c(pcat,pcat,T))
  Ccat[,,1]=C0cat
  mcat[,1]=m0cat
  mtcat = m0cat
  Ctcat = C0cat
  # Kalman Filtering
  for (t in 1:T){
     acat = fccat(mtcat,thetacatj)$newmean
     maxDiff= fccat(mtcat,thetacatj)$maxDiff
   gcat = Jccat(mtcat,thetacatj,maxDiff)$newJacob
   acat = pmin(acat ,maxDiff)
     acat = pmax(0,acat )
     rcat = gcat%*%Ctcat%*%t(gcat)+wcat      #variance of prior t-1
     Jacat = Fcat           # a vector as there is no nonlinearity
     #for (j in 1:pcat){
     hcat  = Fcat*acat
     Fcat  = Jacat          # for readability only
     forecastcat  = hcat     # for readability only
     Y1cat[,t]= forecastcat  # step ahead forecast saving
     qcat = Fcat*rcat*Fcat + vcat  #variance of one step ahead forecast
     ecat    = ycat[t,] - forecastcat   # error of forecast
     Acat = Fcat*rcat%*%ginv(qcat)
     MSEcat[t,]=sum(ecat**2)
     MADcat[t,]= sum(abs(ecat))
     mtcat = acat + Acat %*% ecat
     Ctcat = rcat - Acat%*%qcat%*%t(Acat)
     Ctcat = (Ctcat+t(Ctcat))/2
   #catharsis
     Ctcat[is.infinite(Ctcat)]=10
   Ctcat[is.nan(Ctcat)]=10

     Ccat[,,t] =Ctcat
     mcat[,t] = mtcat
       mcat[,t]= pmin(mcat[,t],maxDiff)
     mcat[,t]= pmax(0,mcat[,t])
     mtcat = mcat[,t]
     cat(t, ",")
  }


   cat("\n")
  #backward smoothing
  tt1cat = matrix(rep(0,pcat*T),ncol=T)
   if (!is.positive.definite(Ccat[,,T])){
#          stop("Negative Variance Found in C Matrix",Call.=FALSE)
```

```
        Ccat[,,T]= diag(rep(1e-6,sqrt(length(Ccat[,,T]))))
      }

   tt1cat[,T] = mcat[,T]+t(chol(Ccat[,,T]))%*%as.matrix(rnorm(pcat))
   tt1cat[,T]= pmin( tt1cat[,T],maxDiff)
   tt1cat[,T]= pmax(0, tt1cat[,T])


   for (t in (T-1):1){
      acat = fccat(mcat[,t],thetacatj)$newmean
    maxDiff= fccat(mcat[,t],thetacatj)$maxDiff
      gcat = Jccat(mcat[,t],thetacatj,maxDiff)$newJacob
    acat = pmin(acat ,maxDiff)
      acat = pmax(0,acat )


      rcat = gcat%*%Ccat[,,t]%*%t(gcat)+wcat      #variance of prior t-1

     if (!is.positive.definite(rcat)){
#         stop("Negative Variance Found in C Matrix",Call.=FALSE)
        rcat= diag(rep(1e-6,sqrt(length(rcat))))
      }

      bcat = Ccat[,,t]%*%t(gcat)%*%ginv(rcat)
      ucat = Ccat[,,t]%*%t(gcat)

      Cmcat = Ccat[,,t] - ucat%*%ginv(rcat)*t(ucat)
      tmcat = mcat[,t]  + bcat%*%(tt1cat[,t+1]-acat)
    Cmcat=(Cmcat+t(Cmcat))/2

    #catharsis
      Cmcat[is.infinite(Cmcat)]=10
    Cmcat[is.nan(Cmcat)]=10

      if (!is.positive.definite(Cmcat)){
#         stop("Negative Variance Found in C Matrix",Call.=FALSE)
        Cmcat= diag(rep(1e-6,sqrt(length(Cmcat))))
      }

      # save mean and variance of posterior at time t
      Ccat[,,t]   = Ccat[,,t] - bcat%*%(rcat-Ccat[,,t+1])%*%t(bcat)
       mcat[,t]    = mcat[,t]  + bcat%*%(mcat[,t+1]-acat)
     mcat[,t]= pmin( mcat[,t],maxDiff)
       mcat[,t]= pmax(0, mcat[,t])

      tt1cat[,t]  = tmcat     + t(chol(Cmcat))%*%as.matrix(rnorm(pcat))
     tt1cat[,t]= pmin( tt1cat[,t],maxDiff)
       tt1cat[,t]= pmax(0, tt1cat[,t])

   }

   #now ad-hoc treatment of start value
   m0cat  = mcat[,1]
   C0cat  = Ccat[,,1]
```

```
   C0cat  = (C0cat  +t(C0cat  ))/2
     if (!is.positive.definite(C0cat)){
#         stop("Negative Variance Found in C Matrix",Call.=FALSE)
        C0cat  = diag(rep(1e-6,sqrt(length(C0cat  ))))
      }
     tt0cat    =m0cat     + t(chol(C0cat))%*%as.matrix(rnorm(pcat))
     tt0cat    = pmin( tt0cat    ,maxDiff)
       tt0cat    = pmax(0, tt0cat)
     tt0cat    = matrix(tt0cat,ncol=1)
   return(list(mcat=mcat,Ccat=Ccat,m0cat=m0cat,C0cat=C0cat,tt1cat=tt1cat,tt0cat=tt0cat,Y1cat=
   Y1cat,MADcat=MADcat,MSEcat=MSEcat))
}

#--------------------------------------------------------------------------------------
#                            FFBS estimation
#--------------------------------------------------------------------------------------
#           Inputs
#----------------------------------------------------
#Data:list(p=p,lgtdata=simlgtdata,Z=Z)
#----------------------------------------------------
#Zc =Zcat       # pc x nzcat matrix
               # to explain heterogeneity in category(with no intercept)
#----------------------------------------------------
# McMc = list(R=R,keep=keep)
#----------------------------------------------------
#R: number of iterations of draw
#keep : thining (1 for no thinning)

#----------------------------------------------------
# Prior = list(ncomp=5)
#----------------------------------------------------
#ncomcat:    number of components in normal mixture of category coefficients



#----------------------------------------------------
#  Output: out$betadraw, out$nmix
#----------------------------------------------------
#out$thetacatdraw:    [ncat x nvaretaIndvapp x (R/keep)]   coefficient draws for #units (nlgt)
                  #  and for #nvaretaIndvapp (number of var relevant to choice)


#out$DeltaCatdraw:   [(R/keep)x(nzalphai*nvaralphai)]
                  #  Delta draws, with first row as initial value

#out$nmixDiffcat     :    list of list of lists (length: R/keep)
                     # out$nmixDiffcat[[i]]: i's draw of component of mixture

#out$llikeDiffcat           :    loglikelihood at each kept draw

#----------------------------------------------------
#  Code:
#----------------------------------------------------

#------------------------------
```

```
#prepare Data
#------------------------------
Data = list( Zc=Zcat)
jumps      =    1
idx = 0
jp = 0
ndraw=10000;
ndraw0 = 5000
burnIn = ndraw0
McMc = list(R=ndraw,keep=jumps)
Prior= list(ncomcat=ncompcat)
ncat = ncat

#----------------------------------------------------
#  check arguments of DGP, or real data to make sure conformity
#----------------------------------------------------
#  function to through error and stop
pandterm=function(message) { stop(message,call.=FALSE) }

#------------------------------
#check the Data
#------------------------------

#-------------------------------
# Component heterogeneity data check
#-------------------------------
drawdeltathetacatj=TRUE
if(is.null(Data$Zc)) { cat("Zc not specified",fill=TRUE); fsh() ; drawdelta=FALSE} else {Zc=Data
$Zc}

if(drawdeltathetacatj) {
   nzthetacatj=ncol(Zc)
   colmeans=apply(Zc,2,mean)
   if(sum(colmeans) > .00001)
   {pandterm(paste("Zc does not appear to be de-meaned: colmeans= ",colmeans))}
}

#----------------------------------------------
#     Check McMc
#----------------------------------------------
if(missing(McMc)) {pandterm("Requires Mcmc list argument")}
if(!missing(McMc)){
   if(is.null(McMc$keep)) {keep=1} else {keep=McMc$keep}
   if(is.null(McMc$R)) {pandterm("Requires R argument in Mcmc list")} else {R=McMc$R}
}
#----------------------------------------------
# check on priors
#----------------------------------------------
if(missing(Prior))
{pandterm("Requires Prior list argument (at least ncompIndv,ncomcat,ncomapp)")}
if(is.null(Prior$ncomcat)) {pandterm("Requires Prior element ncomcat (num of mixture components
for categories)")} else {ncomcat=Prior$ncomcat}

# prior for mubar across 6 HB
```

```
# number of coefficients should be set for the number of means

#thetacatj
nvarthetacatj = 3
if(is.null(Prior$mubarthetacatj)) {mubarthetacatj=matrix(rep(0,(nvarthetacatj)),nrow=1)} else {
mubarthetacatj=matrix(Prior$mubarthetacatj,nrow=1)}
if(ncol(mubarthetacatj) != (nvarthetacatj)) {pandterm(paste("mubar must have ncomp cols,
ncol(mubarthetacatj)= ",ncol(mubarthetacatj)))}
if(is.null(Prior$Amuthetacatj)) {Amuthetacatj=matrix(.01,ncol=1)} else {Amuthetacatj=matrix(
Prior$Amuthetacatj,ncol=1)}
if(ncol(Amuthetacatj) != 1 | nrow(Amuthetacatj) != 1) {pandterm("Amthetacatj must be a 1 x 1
array")}
if(is.null(Prior$nuthetacatj)) {nuthetacatj=nvarthetacatj+3}  else {nuthetacatj=Prior$
nuthetacatj}
if(nuthetacatj < 1) {pandterm("invalid nuthetacatj value")}
if(is.null(Prior$Vthetacatj)) {Vthetacatj=nuthetacatj*diag(nvarthetacatj)} else {Vthetacatj=
Prior$Vthetacatj}
if(sum(dim(Vthetacatj)==c(nvarthetacatj,nvarthetacatj)) !=2) pandterm("Invalid Vthetacatj in
prior")
if(is.null(Prior$Adthetacatj) & drawdeltathetacatj) {Adthetacatj=.01*diag(nvarthetacatj*
nzthetacatj)} else {Adthetacatj=Prior$Adthetacatj}
if(drawdeltathetacatj) {if(ncol(Adthetacatj) != nvarthetacatj*nzthetacatj | nrow(Adthetacatj) !=
 nvarthetacatj*nzthetacatj) {pandterm("Adthetacatj must be nvarthetacatj*thetacatj x
nvarthetacatj*thetacatj")}}
if(is.null(Prior$deltabarthetacatj)& drawdeltathetacatj) {deltabarthetacatj=rep(0,nzthetacatj*
nvarthetacatj)} else {deltabarthetacatj=Prior$deltabarthetacatj}
if(drawdeltathetacatj) {if(length(deltabarthetacatj) != nzthetacatj*nvarthetacatj) {pandterm(
"deltabar must be of length nvarthetacatj*nzthetacatj")}}
if(is.null(Prior$athetacatj)) { athetacatj=rep(5,ncomcat)} else {athetacatj=Prior$athetacatj}
if(length(athetacatj) != ncomcat) {pandterm("Requires dim(athetacatj)= ncomp (no of components)"
)}
badathetacatj=FALSE
for(i in 1:ncomcat) { if(athetacatj[i] < 1) badathetacatj=TRUE}
if(badathetacatj) pandterm("invalid values in a vector in thetacatj")

#----------------------------------------------------------------------------
# print out problem description
#----------------------------------------------------------------------------
cat(" ",fill=TRUE)
cat("Starting MCMC Inference for Hierarchical Logit with Dynamic Non-Linear Model (Bass Model):"
,fill=TRUE)
cat("   Normal Mixture with",
    ncomcat,"components of categories, for first stage prior",fill=TRUE)

#thetacatj
cat("Prior Parms for thetacatj: ",fill=TRUE)
cat("nuthetacatj =",nuthetacatj,fill=TRUE)
cat("Vthetacatj ",fill=TRUE)
print(Vthetacatj)
cat("mubarthetacatj ",fill=TRUE)
print(mubarthetacatj)
cat("Amuthetacatj ", fill=TRUE)
print(Amuthetacatj)
cat("athetacatj ",fill=TRUE)
```

```
print(athetacatj)
if(drawdeltathetacatj)
{
   cat("deltabarthetacatj",fill=TRUE)
   print(deltabarthetacatj)
   cat("Adthetacatj",fill=TRUE)
   print(Adthetacatj)
}

cat(" ",fill=TRUE)
cat("MCMC Parms: ",fill=TRUE)
cat(paste(" R= ",R," keep= ",keep),fill=TRUE)
cat("",fill=TRUE)


#---------------------------------------------------------------------
# allocate space for draws
#---------------------------------------------------------------------

#thetacatj
if(drawdeltathetacatj) Deltadrawthetacatj=matrix(double((floor((R-burnIn)/keep))*nzthetacatj*
nvarthetacatj),ncol=nzthetacatj*nvarthetacatj)
thetacatjdraw=array(double((floor((R-burnIn)/keep))*ncat*nvarthetacatj),dim=c(ncat,nvarthetacatj
,floor((R-burnIn)/keep)))
probdrawthetacatj=matrix(double((floor((R-burnIn)/keep))*ncomcat),ncol=ncomcat)
oldcomthetacatj=NULL
compdrawthetacatj=NULL

#-----------------------------------------------------------
#           Draw from the hierarchy
#-----------------------------------------------------------
drawDelta=
   function(x,y,z,comps,deltabar,Ad){
      # delta = vec(D)
      #  given z and comps (z[i] gives component indicator for the ith observation,
      #   comps is a list of mu and rooti)
      #y is n x p
      #x is n x k
      #y = xD' + U , rows of U are indep with covs Sigma_i given by z and comps
      p=ncol(y)
      k=ncol(x)
      xtx = matrix(0.0,k*p,k*p)
      xty = matrix(0.0,p,k) #this is the unvecced version, have to vec after sum
      for(i in 1:length(comps)) {
         nobs=sum(z==i)
         if(nobs > 0) {
            if(nobs == 1)
            { yi = matrix(y[z==i,],ncol=p); xi = matrix(x[z==i,],ncol=k)}
            else
            { yi = y[z==i,]; xi = x[z==i,]}

            yi = t(t(yi)-comps[[i]][[1]])
            sigi = crossprod(t(comps[[i]][[2]]))
            xtx = xtx + crossprod(xi) %x% sigi
```

```
            xty = xty + (sigi %*% crossprod(yi,xi))
         }
      }
   xty = matrix(xty,ncol=1)

   # then vec(t(D)) ~ N(V^{-1}(xty + Ad*deltabar),V^{-1}) V = (xtx+Ad)
   cov=chol2inv(chol(xtx+Ad))
   return(cov%*%(xty+Ad%*%deltabar) + t(chol(cov))%*%rnorm(length(deltabar)))
}

#-----------------------------------------------------------------------------------------------
------
#  Likelihood for app category diffusion level
#-----------------------------------------------------------------------------------------------
------

#-------------------------
# Categories
#-------------------------
bassErrorsCat= function(thetacatjest,tt0cat,tt1cat,wcat,curcat){
   ncat = dim(tt1cat)[1]
   T    = dim(tt1cat)[2]
   predictedcat = matrix(rep(0,(T)),ncol=T)
   i = curcat

   thetacatjest =matrix(as.numeric(thetacatjest),nrow=ncat)
   colnames(thetacatjest) <- c("p","q","Cj")

   if (i ==1){
      predictedcat[1]=tt0cat[i,1]+
         (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/thetacatjest[i,"Cj"]))*
         (thetacatjest[i,"Cj"]-tt0cat[i,1]);
   }else{
      # treat element in the middle
      if (i>1 && i<ncat){
         predictedcat[1]=tt0cat[i,1]+
            (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/thetacatjest[i,"Cj"]))*
            (thetacatjest[i,"Cj"]-tt0cat[i,1]);
      }else{
         predictedcat[1]=tt0cat[i,1]+
            (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/thetacatjest[i,"Cj"]))*
            (thetacatjest[i,"Cj"]-tt0cat[i,1]);
      }
   }

  for(t in 2:T){
      if (i ==1){
         predictedcattemp=tt1cat[i,t]+
            (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/thetacatjest[i,"Cj"]))*
            (thetacatjest[i,"Cj"]-tt1cat[i,t]);
      }else{
         # treat element in the middle
         if (i>1 && i<ncat){
            predictedcattemp=tt1cat[i,t]+
```

```
              (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/thetacatjest[i,"Cj"]))*
              (thetacatjest[i,"Cj"]-tt1cat[i,t]);
        }else{
           predictedcattemp=tt1cat[i,t]+
              (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/thetacatjest[i,"Cj"]))*
              (thetacatjest[i,"Cj"]-tt1cat[i,t]);
        }
     }
     predictedcat[t]=predictedcattemp
   }
   errortempcat = tt1cat[i,]-predictedcat
   return (errortempcat)
}

basslikelihoodCat= function(thetacatjestcur,thetacatjest,tt0cat,tt1cat,wcat,oldcompthetacatj,
indthetacatj,curcat,meansuntilcur,betabarthetacatj){
   #meansuntilcur is an array but I will use until curcat of it
   ncat =  dim(tt1cat)[1]
   T    =  dim(tt1cat)[2]
   thetacatjest[curcat,]=thetacatjestcur
   thetacatjest[curcat,3]=exp(thetacatjest[curcat,3])
   if (curcat == 1){
      thetacatjest=matrix(thetacatjest,nrow=ncat)
      errortempcat = t(bassErrorsCat(thetacatjest,tt0cat,tt1cat,wcat,curcat))
      wcatconditional = abs(wcat[curcat,curcat] - wcat[curcat,-curcat]%*%ginv(wcat[-curcat,-
      curcat])%*%
         wcat[-curcat,curcat])
      LLcat        = -0.5*sum(crossprod(errortempcat,errortempcat)/sqrt(wcatconditional))-0.5*T*
      log(2*pi)-
         0.5*T*wcatconditional
   }else{
       thetacatjconditional = thetacatjest
      errortempcat = t(bassErrorsCat(thetacatjconditional,tt0cat,tt1cat,wcat,curcat))
      wcatconditional = abs(wcat[curcat,curcat] - wcat[curcat,-curcat]%*%ginv(wcat[-curcat,-
      curcat])%*%
         wcat[-curcat,curcat])
      LLcat        = -0.5*sum(diag(chol2inv(chol(wcatconditional))%*%crossprod(errortempcat,
      errortempcat)))-0.5*T*ncat*log(2*pi)-
         0.5*T*det(wcatconditional)
   }

   LLpriorcat = 0;
   # to test
   clt = curcat
   rootpithetacatj=oldcompthetacatj[[indthetacatj[clt]]]$rooti
   betabarthetacatj= betabarthetacatj
   # as the prior is over the exponent item rather than main
   LLpriorcat = lndMvn(thetacatjest[curcat,],betabarthetacatj,rootpithetacatj)
   output = sum(LLpriorcat)+LLcat
   if (is.nan(output)){
       output = -Inf
   }
   if (is.infinite(output)){
      output = -1e40
```

```
   }
   return(-output) #make sure I send back negative likelihood
}

#--------------------------------------------------------------------------------------------------
-------
#  initialize values
#--------------------------------------------------------------------------------------------------
-------

#---------------------------------------------------------------
# set initial values for the indicators
#     ind is of length(nlgt) and indicates which mixture component this obs
#     belongs to.
#---------------------------------------------------------------

#thetacatj
#-------------------------
indthetacatj=NULL
nincthetacatj=floor(ncat/ncomcat)
for (i in 1:(ncomcat-1)) {indthetacatj=c(indthetacatj,rep(i,nincthetacatj))}
if(ncomcat != 1) {indthetacatj = c(indthetacatj,rep(ncomcat,ncat-length(indthetacatj)))} else {
indthetacatj=rep(1,ncat)}
# initialize delta
if (drawdeltathetacatj) olddeltathetacatj=rep(0,nzthetacatj*nvarthetacatj)
# initialize probs
oldprobthetacatj=rep(1/ncomcat,ncomcat)
# initialize comps
tcompthetacatj=list(list(mu=rep(0,nvarthetacatj),rooti=diag(nvarthetacatj)))
oldcompthetacatj=rep(tcompthetacatj,ncomcat)

#---------------------------------------------------------------
# set initial values for the state space portion of the model
#---------------------------------------------------------------

#app categories
#--------------------------------------
ycat = array(rep(0,ncat*T), dim=c(ncat,T))
pcat = ncat
Fcat = 1
thetacatjest = matrix(rep(0.01,ncat*3),ncol=3)
m0cat        = 0.01*matrix(c(rep(1,ncat)),ncol=ncat)
C0cat        = 2*diag(c(rep(1,pcat)))
vcat = 0.1*rep(1,ncat)

wcat = 0.5*diag(c(rep(1,pcat)))
tt0cat = 0.01*matrix(c(rep(1,ncat)),nrow=ncat)
tt1cat = 0.01*matrix(c(rep(1,ncat*T)),ncol=T)
Y1cat  = array(rep(0,ncat*T),dim=c(ncat,T))
MADcat = matrix(rep(0,T*ncat),ncol=ncat)
MSEcat = matrix(rep(0,T*ncat),ncol=ncat)
# i did not save v because of its large size
crossseclengthcat = (ncat*ncat*2)*(R-burnIn)
ccat_=matrix(rep(0,crossseclengthcat),ncol=(R-burnIn))
```

```
bcat_ = matrix(rep(0,(length(tt0cat)+length(tt1cat))*(R-burnIn)),ncol=(R-burnIn))
llcat_ = rep(0,(R-burnIn))
Y1cat_ = array(rep(0,ncat*T*(R-burnIn)),dim=c(ncat,T,(R-burnIn))) #memory explosion so it does
not work
MADcat_ = rep(0,(R-burnIn))
MSEcat_ = rep(0,(R-burnIn))
b0catst  = rep(0, nvarthetacatj)
S0catstInv  = diag(rep(1.5e-7,nvarthetacatj))
b0catobs = 0
S0catobsInv = 1.5e-7
v0ivcat     = 30
Svivcat     = diag(rep(0.1,ncat))

#-----------------------------------
# initial values for the pooled
#-----------------------------------
oldthetacatj  = thetacatj


# estimate non-state parameters of state equation
#------------------------------------------------------
#cat
#--------------------
pcThetacatj = 6e-5
cumjThetacatj = 0
thetacatjNew = matrix(rep(0,nvarthetacatj*ncat), ncol=nvarthetacatj)
varModeCat = matrix(rep(0,nvarthetacatj*ncat), ncol=nvarthetacatj)
errorwtempcat =matrix(rep(0,T*ncat), ncol=T)
w0ivcat = ncat
Swivcat = diag(rep(0.1,ncat))

# mean of the coefficient (prior mean)
#------------------
betabarthetacatj = matrix(rep(0, ncat*nvarthetacatj),ncol=nvarthetacatj)

#--------------------------------------------------------------------------
#            Main iterations of MCMC
#--------------------------------------------------------------------------
#   start main iteration loop
itime=proc.time()[3]
cat("MCMC Iteration (est time to end - min) ",fill=TRUE)
fsh()

thetacatjest = thetacatj

for(iterrep in 1:R){
   cat('\nStarted new iteration.....\n')
   cat (iterrep)
   cat ('\n')
   itimetest=proc.time()[3]
   #-----------------------------------------------------------------
   #parameters to set burn in
   #------------------------------------------------------------------
   sw=0;
```

```
  if   (iterrep                >     ndraw0) { #  Discarding burnin
     idx               =     idx + 1

  }

  if   (idx  ==     jumps){
     idx   =     0;
     jp    =     jp + 1
     sw    =     1
  }

  #-----------------------------------------------------------------
  # intialize compute quantities for Metropolis (pooled)
  #------------------------------------------------------------------
  cat("initializing loop for  ",ncat," app category units ...",fill=TRUE)
  fsh()


#---------------------------------------------------------------------
# first draw comps,ind,p | {beta_i}, delta
#       ind,p need initialization comps is drawn first in sub-Gibbs
#---------------------------------------------------------------------

  #thetacatj
  #----------
  if(drawdeltathetacatj)
  {mgoutthetacatj=rmixGibbs(oldthetacatj-Zcat%*%t(matrix(olddeltathetacatj,ncol=nzthetacatj)),
                   mubarthetacatj,Amuthetacatj,nuthetacatj,Vthetacatj,athetacatj,
                   oldprobthetacatj,indthetacatj,
                   oldcompthetacatj)
  }else
  {mgoutthetacatj=rmixGibbs(oldthetacatj,
                   mubarthetacatj,Amuthetacatj,nuthetacatj,Vthetacatj,athetacatj,
                   oldprobthetacatj,indthetacatj,
                   oldcompthetacatj)}
  oldprobthetacatj=mgoutthetacatj[[1]]
  oldcompthetacatj=mgoutthetacatj[[3]]
  indthetacatj=mgoutthetacatj[[2]]
  #-----------------------------------------------------------
  # now draw delta | {beta_i}, ind, comps
  #-----------------------------------------------------------
  if(drawdeltathetacatj) {olddeltathetacatj=drawDelta(Zcat,oldthetacatj,indthetacatj,
  oldcompthetacatj,deltabarthetacatj,Adthetacatj)}



  #thetacatj
  #--------------------------------
  #  note: beta_i = Deltabetai*zIndv_i + u_i  Deltabetai is nvarbetai x nzbetai
  for (clgt in 1:ncat) {
     if(drawdeltathetacatj) {
        betabarthetacatj[clgt,]=oldcompthetacatj[[indthetacatj[clgt]]]$mu+matrix(
        olddeltathetacatj,ncol=nzthetacatj)%*%
           as.vector(Zcat[clgt,])
```

```
      }else {
         betabarthetacatj[clgt,]=oldcompthetacatj[[indthetacatj[clgt]]]$mu }

   }

  cat('\nEstimation of HB of thetacatj parameters done successfully...\n')
#--------------------------------------------------------------------------------------
#                              EKF for the  categories
#--------------------------------------------------------------------------------------
   #itime=proc.time()[3]
   outcatEKF = catEKF(ycat=t(categoryDiffWeekly),Fcat=Fcat,pcat=ncat,m0cat=m0cat,C0cat=C0cat,
   vcat=vcat,wcat=wcat
                     ,thetacatj=thetacatjest)
   #ctime=proc.time()[3]
   #timetoend=(ctime-itime)
   mcat = outcatEKF$mcat
   Ccat = outcatEKF$Ccat
   m0cat = outcatEKF$m0cat
   C0cat = outcatEKF$C0cat
   tt1cat = outcatEKF$tt1cat
   tt0cat = outcatEKF$tt0cat
   Y1cat  = outcatEKF$Y1cat
   MADcat = outcatEKF$MADcat
   MSEcat = outcatEKF$MSEcat
  cat('\nEKF of category done successfully...\n')


#--------------------------------------------------------------------------------------
#   Estimating Observation equation's variance
#--------------------------------------------------------------------------------------

   # IW to find the misperception variance
   viwmucat = v0ivcat + T
   errortempcat = t(categoryDiffWeekly - tt1cat[,])  #dim=c(nIndv,T)
   errortempcat[is.nan(errortempcat )]=1e-6
   if (is.positive.definite(Svivcat+crossprod(errortempcat,errortempcat))){
    sigmaiwmucat = chol2inv(chol(Svivcat+crossprod(errortempcat,errortempcat)))
    }else{
    sigmaiwmucat = chol2inv(chol(Svivcat))
    }
   # draw from IW(sigmaiwmu,viwmu)
   vcatTemp = rwishart(viwmucat,sigmaiwmucat)$IW
   llcatObsEqCur =  - 0.5*sum(diag(chol2inv(chol(vcatTemp))%*%crossprod(errortempcat,
   errortempcat)))-
      0.5*ncat*T*log(2*pi)-0.5*T*det(vcatTemp)
   cat('\nestimation of misperception parameters for category......successfully done\n')
    # new form joint estimation of state variables
   vcat = vcatTemp
   cat('\n Maximum Variance of observation equation is:\n')
   cat(max(vcat))
   cat('\n')
#--------------------------------------------------------------------------------------
#                              Category Latent Coefficient mean and Variance
#--------------------------------------------------------------------------------------
```

```
alpha         = 0.5
rndacceptance = 0.8
meansuntilcur = thetacatjest
# using conditioning technique (cannot be parallelized due to dependance)
thetacatjest[,3]= pmin(10,log(abs(thetacatjest[,3])))
  itime=proc.time()[3]
for (clt in 1:ncat){
   cat(clt)
#   itime=proc.time()[3]
 #basslikelihoodCat(thetacatjest[clt,],thetacatjest=thetacatjest, tt0cat=tt0cat,tt1cat=tt1cat,
   #               wcat=wcat,oldcompthetacatj=oldcompthetacatj,
   #               indthetacatj=indthetacatj,curcat=clt,meansuntilcur=meansuntilcur,
 #                          betabarthetacatj=betabarthetacatj[clt,])
   outcatst=optim(thetacatjest[clt,],basslikelihoodCat,method="BFGS",control=list( fnscale=1,
   trace=1,reltol=1e-5),hessian=TRUE,
                  thetacatjest=thetacatjest, tt0cat=tt0cat,tt1cat=tt1cat,
                  wcat=wcat,oldcompthetacatj=oldcompthetacatj,
                  indthetacatj=indthetacatj,curcat=clt,meansuntilcur=meansuntilcur,
                           betabarthetacatj=betabarthetacatj[clt,])
  # ctime=proc.time()[3]
 #   timetoend=(ctime-itime)
   meansuntilcur[clt,]=outcatst$par
   thetacatjestTemp   = thetacatjest
   thetacatjestTemp[clt,]=meansuntilcur[clt,]
   #it seems it is hessian of function and not the negative value
   hessiancatTemp=outcatst$hessian
   hessiancatTemp=(hessiancatTemp+t(hessiancatTemp))/2
   if (is.positive.definite(hessiancatTemp)){
      varcattemp  = diag(chol2inv(chol(hessiancatTemp)))
   }else{
      varcattemp  = diag(ginv(make.positive.definite(hessiancatTemp)))
   }
   varcattemp  =pmax(1e-20,varcattemp  )
   varModeCat[clt,] = varcattemp
}
   ctime=proc.time()[3]
   timetoend=(ctime-itime)

# to test
#end test
j = 0
cat("\nM-H loop to find the appropriate parameters for category latent state equation\n")

postPlatentNewTemp=rep(0,ncat)
postPlatentOldTemp =rep(0,ncat)
thetacatjestNewTemp = matrix(rep(0,ncat*3),ncol=3)
while (alpha < rndacceptance){
   j              = j+1
   cat(j,",")

   for (clt in 1:ncat) {
      wpm            = pcThetacatj*varModeCat[clt,]
      wpm            = pmax(wpm,c(rep(1e-6,length(wpm))))
      thetacatjestNew=meansuntilcur[clt,] + wpm*rnorm(length(thetacatjest[clt,]))
```

```
        postPlatentNew = -basslikelihoodCat(thetacatjestNew, thetacatjest=thetacatjest, tt0cat=
        tt0cat,tt1cat=tt1cat,
                                    wcat=wcat,oldcompthetacatj=oldcompthetacatj,
                                    indthetacatj=indthetacatj,curcat=clt,meansuntilcur=
                                    meansuntilcur,
                            betabarthetacatj=betabarthetacatj[clt,])
        postPlatentOld = -basslikelihoodCat(thetacatjest[clt,], thetacatjest=thetacatjest,
        tt0cat=tt0cat,tt1cat=tt1cat,
                                    wcat=wcat,oldcompthetacatj=oldcompthetacatj,
                                    indthetacatj=indthetacatj,curcat=clt,meansuntilcur=
                                    meansuntilcur,
                            betabarthetacatj=betabarthetacatj[clt,])

        postPlatentNewTemp[clt] = postPlatentNew
     postPlatentOldTemp[clt] = postPlatentOld
        thetacatjestNewTemp[clt,] = thetacatjestNew

   }

     postPlatentNew = sum(postPlatentNewTemp)
     postPlatentOld = sum(postPlatentOldTemp)
     alpha          = postPlatentNew - postPlatentOld
    if (is.nan(alpha)){
       if (is.nan(postPlatentOld)){
           alpha=Inf
       }else{
           alpha=-Inf
       }
   }
     rndacceptance  = log(runif(1))
     if (j > 100){
        pcThetacatj = pcThetacatj /10;
     thetacatjestNewTemp = thetacatjest
        postPlatentNew = postPlatentOld
        break
     }
  }
  thetacatjest=thetacatjestNewTemp
  cumjThetacatj = cumjThetacatj + j          # to keep cumulative value
  accptrate = iterrep/cumjThetacatj    # acceptance rate until now
  if (floor (iterrep/5) == iterrep/5){
     if (accptrate > 0.15){
        pcThetacatj = pcThetacatj*3;
        cumjThetacatj = iterrep/0.15
     }else{
        if(accptrate < 0.01){
           pcThetacatj = pcThetacatj/3
           cumjThetacatj = iterrep/0.01
        }
     }
  }
  llcatStateEq = postPlatentNew
   #thetacatjest[,3]=exp(thetacatjest[,3])
```

```
    thetacatjest[,3]= exp(pmax(pmin(thetacatjest[,3],10),1))
    thetacatjest[is.nan(thetacatjest)] = 100
    oldthetacatj= thetacatjest
  cat('\nestimation of non-state prameters of state equation parameters for
  category......successfully done\n')
  #make sure that it never becomes zero
  pcThetacatj=max(pcThetacatj,1e-30)
  pcThetacatj=min(pcThetacatj,10)
  cat('\n The current value of pcThetacatj is..\n')
  cat(pcThetacatj)
  cat('\n')
#------------------------------------------------
# Inverse Wishart Variance of State Equation
#------------------------------------------------
   wiwmucat = w0ivcat + T
   for (clt in 1:ncat){
      errorwtempcat[clt,] = t(bassErrorsCat(thetacatjest,tt0cat,tt1cat,wcat,clt))
   }
   errorwtempcatt = t(errorwtempcat)

   sigmaiwmucat = ginv(Swivcat+crossprod(errorwtempcatt,errorwtempcatt))
   # draw from IW(sigmaiwmu,viwmu)
   sigmaiwmucat=(sigmaiwmucat+t(sigmaiwmucat))/2

   if (!is.positive.definite(sigmaiwmucat)){
   sigmaiwmucat = ginv(Swivcat)}
   wcat = rwishart(wiwmucat,sigmaiwmucat)$IW
   wcat  = (wcat +t(wcat ))/2
   if (!is.positive.definite(wcat)){
    wcat=make.positive.definite(wcat)
   }
 cat('\nestimation of variance prameters of state equation for category......successfully
 done\n')
 cat('\n Maximum Variance of state equation is:\n')
 cat(max(wcat))
 cat('\n')

#--------------------------------------------------------------------------
#       save every keepth draw
#--------------------------------------------------------------------------
   if (sw == 1)
   {

      # thetacatj
      thetacatjdraw[,,jp] = thetacatjest
      probdrawthetacatj[jp,]=oldprobthetacatj
      llcat_[jp] = llcatObsEqCur + llcatStateEq
      Deltadrawthetacatj[jp,]= olddeltathetacatj
      compdrawthetacatj[[jp]]=oldcompthetacatj

      #variances
      ccat_[,jp]=c(as.vector(vcatTemp),as.vector(wcat))
      bcat_ [,jp] = c(as.vector(tt0cat),as.vector(tt1cat))
```

```
      MADcat_[jp]  = sum(MADcat)/T/ncat

      MSEcat_[jp]  = sum(MSEcat)/T/ncat

      Y1cat_[,,jp]= Y1cat # I can not save due to memory explodes

   }
    ctimetest=proc.time()[3]
    cat('\nTotal loop duration:')
    ctimetest - itimetest
    cat('\nwhole loop ...................done\n')
}

#------------------------------------------------------
# end of iterations
#------------------------------------------------------



#----------------------------------------------------
#  Output: out$betadraw, out$nmix
#----------------------------------------------------
#out$alphadraw:    [nlgt x nvaralphai x (R/keep)]   coefficient draws for #units (nlgt)
#  and for #nvaralphai (number of var relevant to choice)
#out$betadraw:    [nlgt x nvarbetai x (R/keep)]   coefficient draws for #units (nlgt)
#  and for #nvarbetai of var relevant to choice)
#out$etaIndvdraw:    [nlgt x nvarthetacatj x (R/keep)]   coefficient draws for #units (nlgt)
#  and for #nvarthetacatj (number of var relevant to choice)
#out$gammaIndvdraw:    [nlgt x nvarthetaappa x (R/keep)]   coefficient draws for #units (nlgt)
#  and for #nvarthetaappa (number of var relevant to choice)
#out$thetacatdraw:    [ncat x nvaretaIndvapp x (R/keep)]   coefficient draws for #units (nlgt)
#  and for #nvaretaIndvapp (number of var relevant to choice)
#out$thetaappdraw:    [napp x nvargammaIndv x (R/keep)]   coefficient draws for #units (nlgt)
#  and for #nvargammaIndv (number of var relevant to choice)


#out$DeltaCatdraw:   [(R/keep)x(nzalphai*nvaralphai)]
#  Delta draws, with first row as initial value
#out$DeltaAppdraw:   [(R/keep)x(nzbetai*nvarbetai)]
#  Delta draws, with first row as initial value
#out$DeltaIndvcatdraw:   [(R/keep)x(nzthetacatj*nvarthetacatj)]
#  Delta draws, with first row as initial value
#out$DeltaIndvappdraw:   [(R/keep)x(nzthetaappa*nvaretaIndvapp)]
#  Delta draws, with first row as initial value
#out$DeltaIndv3draw:   [(R/keep)x(nzetaIndvapp*nvaretaIndvapp)]
#  Delta draws, with first row as initial value
#out$DeltaIndv4draw:   [(R/keep)x(nzgammaIndv*nvargammaIndv)]
#  Delta draws, with first row as initial value


#out$nmixcat         :    list of list of lists (length: R/keep)
# out$nmixcat[[i]]: i's draw of component of mixture
#out$nmixapp         :    list of list of lists (length: R/keep)
# out$nmixapp[[i]]: i's draw of component of mixture
#out$nmixIndvcat     :    list of list of lists (length: R/keep)
```

```
# out$nmixIndvcat[[i]]: i's draw of component of mixture
#out$nmixIndvapp     :    list of list of lists (length: R/keep)
# out$nmixIndvapp[[i]]: i's draw of component of mixture
#out$nmixDiffcat     :    list of list of lists (length: R/keep)
# out$nmixDiffcat[[i]]: i's draw of component of mixture
#out$nmixDiffapp     :    list of list of lists (length: R/keep)
# out$nmixDiffapp[[i]]: i's draw of component of mixture

#out$llikecat              :    loglikelihood at each kept draw
#out$llikeapp              :    loglikelihood at each kept draw
#out$llikeIndvcat          :    loglikelihood at each kept draw
#out$llikeIndvapp          :    loglikelihood at each kept draw
#out$llikeDiffcat          :    loglikelihood at each kept draw
#out$llikeDiffapp          :    loglikelihood at each kept draw


#==========================================================================================
#                             End of Estimation Procedures
#==========================================================================================

#Check convergance plots
plot(alphaidraw[1,4,])
plot(gammaIndvdraw[2,1,])
plot(thetacatjdraw[2,2,])

# test confidence interval
#thetacatj
apply(thetacatjdraw,c(2,1),quantile,probs=c(0.05,0.95))
apply(thetacatjdraw,c(1,2),mean)
apply(thetacatjdraw,c(1,2),sd)

#variances
apply(ccat_,1,quantile,probs=c(0.05,0.95))
apply(ccat_,1,mean)
apply(ccat_,1,sd)

#Check Forecast
ForecastCatMean= apply(Y1cat_,c(1,2),mean)
plot(ForecastCatMean[3,])
plot(categoryDiffWeekly[3,])

#Deltadrawthetacatj
apply(Deltadrawthetacatj,2,quantile,probs=c(0.05,0.95))
apply(Deltadrawthetacatj,2,mean)
apply(Deltadrawthetacatj,2,sd)

#bcat_
apply(bcat_,1,quantile,probs=c(0.05,0.95))
apply(bcat_,1,mean)
apply(bcat_,1,sd)


nmix=list(probdraw=probdrawthetacatj ,zdraw=NULL,compdraw=compdrawthetacatj         )
attributes(nmix)$class="bayesm.nmix"
```

```
png(filename="C:/Users/mxh109420/Desktop/MobileAppProject/heterogeneityLocalDiff.png")
plot(nmix)
dev.off()

# Test convergance one by one
iindx = 3
jindx = 1
quantile(thetacatjdraw[iindx,jindx,],probs=c(0.05,0.95))
mean(thetacatjdraw[iindx,jindx,])
```