```
##
#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))
#--------------------------------------------------------------------------------
# naming is important to have enough resemblance to writing
#================================================================================
T = 60 # total duration of our sample

#--------------------------------------------------------------------------------
# first: simulate state space of category in a for loop for j=1...J (HB)+ complementarity
# HB includes: popularity of category
#--------------------------------------------------------------------------------
nzcat=1         #category popularity as explanator (J)
ncat=10       # number of categories under study
Zcat=matrix(runif(nzcat*ncat),ncol=nzcat)
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((ncat+2))*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(ncat+2)*1e-6,rooti=diag(rep(1,ncat+2)*1e6))
compscat[[2]]=list(mu=runif(ncat+2)*1e-7,rooti=diag(rep(1,ncat+2)*1e6))
compscat[[3]]=list(mu=runif(ncat+2)*1e-9,rooti=diag(rep(1,ncat+2)*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 = matrix(5*runif(ncat*T),ncol=T);   # initialize state and allocate space
thetacatj = matrix(rep(1,ncat*(ncat+2)),ncol=ncat+2)
colnames(thetacatj) <- c("p","q","Cj",c(1:(ncat-1)))
for (t in 2:T){
```

```
   # treat first eelement differently
   i = 1
   if (t==2){   # only generate thetaj=(pj,qj,Cj,lambdakj) once
      thetacatj[i,]=Deltacat%*%Zcat[i,]+as.vector(rmixture(1,pveccat,compscat)$x)
      # make sure that market size (M), p and q are positive for the sake of simulation
      thetacatj[i,1]=abs(thetacatj[i,1]);
      thetacatj[i,2]=abs(thetacatj[i,2]);
      thetacatj[i,3]=abs(thetacatj[i,3]);
   }
   catlatent[i,t]=catlatent[i,t-1]+
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(catlatent[i,t-1]/thetacatj[i,"Cj"])+
          thetacatj[i,(i+3):(ncat+2)]%*%catlatent[(i+1):ncat,t-1])*
      (thetacatj[i,"Cj"]-catlatent[i,t-1])+ewcat[i,t];
   catlatent[i,t] = abs( catlatent[i,t])   # to make sure it is positive per theory
   #catlatent[i,t]=abs(catlatent[i,t]) #make sure generated state is not negative
   # treat element in the middle
   for (i in 2:(ncat-1)){
      if (t==2){   # only generate thetaj=(pj,qj,Cj,lambdakj) once
         thetacatj[i,]=Deltacat%*%Zcat[i,]+as.vector(rmixture(1,pveccat,compscat)$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]);
      }
      catlatent[i,t]=catlatent[i,t-1]+
         (thetacatj[i,"p"]+thetacatj[i,"q"]*(catlatent[i,t-1]/thetacatj[i,"Cj"])+
             thetacatj[i,4:(i+2)]%*%catlatent[1:(i-1),t-1]+
             thetacatj[i,(i+3):(ncat+2)]%*%catlatent[(i+1):ncat,t-1])*
         (thetacatj[i,"Cj"]-catlatent[i,t-1])+ewcat[i,t];
      #catlatent[i,t]=abs(catlatent[i,t]) #make sure generated state is not negative
      catlatent[i,t] = abs( catlatent[i,t])   # to make sure it is positive per theory
   }
   # treat last element differently
   i = ncat;
   if (t==2){   # only generate thetaj=(pj,qj,Cj,lambdakj) once
      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]);
   }
   catlatent[i,t]=catlatent[i,t-1]+
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(catlatent[i,t-1]/thetacatj[i,"Cj"])+
          thetacatj[i,4:(i+2)]%*%catlatent[1:i-1,(t-1)])*
      (thetacatj[i,"Cj"]-catlatent[i,t-1])+ewcat[i,t];
   catlatent[i,t] = abs( catlatent[i,t])   # to make sure it is positive per theory
   #catlatent[i,t]=abs(catlatent[i,t]) #make sure generated state is not negative
}
# 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)
```

```
# }



#-------------------------------------------------------------------------------
# second: simulate state space of mobile app in a for loop for j=1...J (HB) + complementarity
# HB includes: continent decomposition of app (asia, europe, africa, US)
#-------------------------------------------------------------------------------
numappInCateg=c(rep(1,9),2);   # number of app's in each category
maxappIncat   = max(numappInCateg)    # to balance the data on complement and substitute within
the category
appCategoryIndex=c(0,cumsum(numappInCateg)) # for indexing in using latent
nzapps=4         #search continent decomposition (asia, europe, africa, US) (A)
napps=10         # number of apps under study
nappdiffcoeff = 3; #p, q, alpha, because we are not modeling complementarity within the app
category
Zapp=matrix(runif(nzapps*napps),ncol=nzapps)
Zapp=t(t(Zapp)-apply(Zapp,2,mean))          # demean Zapp, popularity explanator of apps
ncompapp= 2                                 # no of mixture components of app is consider 2 to
allow multimode
Deltaapp=matrix(runif((3)*nzapps)*1e-20,ncol=nzapps) # generate Delta for
thetacat=Deltacat*Zcat+ujcat
Deltaapp[1,]=0.003 # set p's mean data
Deltaapp[2,]=0.01 # set q's mean data
Deltaapp[3,]=0.5# set alpha a's mean data
compsapp=NULL
for (i in 1:ncompapp){
      compsapp[[i]]=list(mu=runif(nappdiffcoeff)*(10**(-23)),rooti=diag(rep(1,nappdiffcoeff)*(10
      **(23))))
}
pvecapp=rep(1/ncompapp,ncompapp) # assume equal probability of mixture

# error of the state equationn for the diffusion of apps
ewapps=matrix(rep(0,T*napps),ncol=T)
for (a in 1:napps){
   wapps = 0.5*runif(1) # as I am treating each differently then they are not correlated
   ewapps[a,] = wapps%*%matrix(rnorm(T,mean=0,sd=1),ncol=T)
}



appslatent = matrix(5*runif(napps*T),ncol=T);   # initialize state and allocate space
thetaappa = matrix(rep(1,napps*(3)),ncol=3)
colnames(thetaappa) <- c("p","q","alphaj")

# addressing appslatent[a,t]
#appslatent = matrix(5*runif(napps*T),ncol=T);   # initialize state and allocate space
# only normal diffusion and no complementarity within the app category
for (t in 2:T){
  for (a in 1:napps) { #given category
      # treat first eelement differently
      i = 1
      if (t==2){   # only generate thetaj=(pj,qj,Cj,lambdakj) once
         thetaappa[a,1:nappdiffcoeff]=Deltaapp[1:nappdiffcoeff,]%*%Zapp[a,]+as.vector(rmixture(1
         ,pvecapp,compsapp)$x)[1:nappdiffcoeff]
         # make sure that alpha, p and q are positive for the sake of simulation
```

```
         thetaappa[a,1]=abs(thetaappa[a,1]) # set p's mean data to positive number
         thetaappa[a,2]=abs(thetaappa[a,2]) # set q's mean data to positive number
         thetaappa[a,3]=abs(thetaappa[a,3]) # set alpha a's mean data to positive number
      }
         appslatent[a,t]=appslatent[a,t-1]+
            (thetaappa[a,"p"]+thetaappa[a,"q"]*(appslatent[a,t-1]/(thetaappa[a,"alphaj"]*
            catlatent[a,t])))*
            ((thetaappa[a,"alphaj"]* catlatent[a,t])-appslatent[a,t-1])+
            ewapps[a,t]
         appslatent[a,t] = abs(appslatent[a,t])
         while (appslatent[a,t]>10**3){
            appslatent[a,t]=appslatent[a,t]-abs(ewapps[a,t])
         }
   }

}
plot( appslatent[1,],type="l")
plot( appslatent[2,],type="l")
# for (a in 1:napps){
#     plot(appslatent[a,],type="l")
#     par(ask=TRUE)
# }



#--------------------------------------------------------------------------------
# Third: simulate individual perception of latent of category
#--------------------------------------------------------------------------------
nzIndv=1         #individual's tenure as explanator (L)
nIndv=50        # number of individuals under study
ZIndv=matrix(10*runif(nzIndv*nIndv),ncol=nzIndv)
ZIndv=t(t(ZIndv)-apply(ZIndv,2,mean))      # demean ZIndv, population explanator of Individuals
ncompIndv= 5                                # no of mixture components of Individuals is
consider 5 to allow for multimodal
DeltaIndvcat=matrix(runif(1)*1e-1,ncol=1)      # generate Delta for
catloclatent=Deltaloc*Zloc+ujloc
compsIndvcat=NULL
for (i in 1:ncompIndv){
   compsIndvcat[[i]]=list(mu=runif(1)*1e-9,rooti=diag(1)*1e9)
}
pvecIndvcat=rep(1/ncompIndv,ncompIndv)    # equally likely

# error of the observation equationn for the diffusion of category, locations across individuals
# I can not use cholesky of big matrix so assume that they are independent
evIndvcat = array(rep(0,nIndv*ncat*T),dim=c(nIndv,ncat,T))
# assume at each point in time the misperceptions are uncorrelated like normal learning theories
for (i in 1:nIndv){
   for (j in 1:ncat){
      print(paste("individual:",i,"category:",j))
      vIndvcat = 0.01*diag(1)   # assuming mispercpetions are uncorrelated (later it could be
      block diagonal)
      evIndvcat[i,j,]=t(chol(vIndvcat))%*%matrix(rnorm(T,mean=0,sd=1),ncol=T)

   }
}
```

```
catIndvlatent= array(5*runif(ncat*nIndv*T),dim=c(nIndv,ncat,T))
gammaIndv = matrix(rep(1,nIndv),ncol=1)
# first set the coefficients
#foreach (j=1:ncat,.packages=c("bayesm"),.combine=cbind) %dopar%{
temp=foreach (i=1:nIndv,.packages=c("bayesm"),.combine=cbind) %dopar%{
   #cat(paste("time:",t,"category:",j,"location:",l,"individual:",i),fill=TRUE)
   if ( i!=1){   # only generate thetaj=(pj,qj,Cj,lambdakj) once
      #gammaIndv[locIndvDistIndex[l]+i,]=
         DeltaIndvcat%*%ZIndv[i,]+as.vector(rmixture(1,pvecIndvcat,compsIndvcat)$x)
   }else{
      1  # normalization for identification
   }

   #catlocIndvlatent[locIndvDistIndex[l]+i,t]=gammaIndv[locIndvDistIndex[l]+i,]*catloclatent[(j-1
   )*nloc+l,t]+
   #   evIndv[locIndvDistIndex[l]+i,t]
}
gammaIndv[1:nIndv,]=temp


# second set the latent measure of misperception of individuals
for (t in 1:T){
   for (j in 1:ncat){
      cat(paste("time:",t,"category",j),fill=TRUE)
      catIndvlatent[1:nIndv,j,t]=
         gammaIndv[1:nIndv,]*catlatent[j,t]+evIndvcat[1:nIndv,j,t]
   }
}

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

#--------------------------------------------------------------------------------
# Fourth: simulate individual perception of latent of apps
#--------------------------------------------------------------------------------
DeltaIndvapp=matrix(runif(1)*1e-1,ncol=1)      # generate Delta for
catloclatent=Deltaloc*Zloc+ujloc
compsIndvapp=NULL
for (i in 1:ncompIndv){
   compsIndvapp[[i]]=list(mu=runif(1)*1e-9,rooti=diag(1)*1e9)
}
pvecIndvapp=rep(1/ncompIndv,ncompIndv)    # equally likely

# error of the observation equationn for the diffusion of category, locations across individuals
# I can not use cholesky of big matrix so assume that they are independent
evIndvapp = array(rep(0,nIndv*napps*T),dim=c(nIndv,napps,T))
# assume at each point in time the misperceptions are uncorrelated like normal learning theories
```

```
for (i in 1:nIndv){
   for (a in 1:napps){
      print(paste("individual:",i,"app:",a))
      vIndvapp = 0.01*diag(1)   # assuming mispercpetions are uncorrelated (later it could be
      block diagonal)
      evIndvapp[i,a,]=t(chol(vIndvapp))%*%matrix(rnorm(T,mean=0,sd=1),ncol=T)

   }
}
appIndvlatent= array(5*runif(napps*nIndv*T),dim=c(nIndv,napps,T))
etaIndvapp = matrix(rep(1,nIndv),ncol=1)
# first set the coefficients
#foreach (j=1:ncat,.packages=c("bayesm"),.combine=cbind) %dopar%{
temp=foreach (i=1:nIndv,.packages=c("bayesm"),.combine=cbind) %dopar%{
   #cat(paste("time:",t,"category:",j,"location:",l,"individual:",i),fill=TRUE)
   if ( i!=1){   # only generate thetaj=(pj,qj,Cj,lambdakj) once
      #gammaIndv[locIndvDistIndex[l]+i,]=
      DeltaIndvapp%*%ZIndv[i,]+as.vector(rmixture(1,pvecIndvapp,compsIndvapp)$x)
   }else{
      1  # normalization for identification
   }

   #catlocIndvlatent[locIndvDistIndex[l]+i,t]=gammaIndv[locIndvDistIndex[l]+i,]*catloclatent[(j-1
   )*nloc+l,t]+
   #   evIndv[locIndvDistIndex[l]+i,t]
}
etaIndvapp[1:nIndv,]=temp


# second set the latent measure of misperception of individuals
for (t in 1:T){
   for (a in 1:napps){
      cat(paste("time:",t,"app",a),fill=TRUE)
      appIndvlatent[1:nIndv,a,t]=
         etaIndvapp[1:nIndv,]*appslatent[a,t]+evIndvapp[1:nIndv,a,t]
   }
}

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

#--------------------------------------------------------------------------------
# Fifth: simulate app level data (tenure, feature, size)
#--------------------------------------------------------------------------------
appchar = array(c(rep(0,3*napps*T)),dim=c(napps,T,3))   # 3 for tenure, feature, size
for (j in 1:napps) { #given category
   tenure=round(runif(numappInCateg[j])*347)
```

```
   dayincrement<-as.vector(sapply(1:T, function (x) rep(x,numappInCateg[j]))) # day increment
   of tenure
   appchar[,,1]=t(matrix(tenure+dayincrement,ncol=T))                #Tenure
   appchar[,,2]=t(matrix(round(runif(numappInCateg[j]*T)),ncol=T))    # Feature
   appchar[,,3]= t(matrix(runif(numappInCateg[j]*T)*80,ncol=T))      #Size
}

#--------------------------------------------------------------------------------
# Sixth: simulate category level parameters ( (1) Variance of price,
#(2) mean of file size, (3) mean of tenure of apps inside, (4) total number of free apps
# (5) total number of paid apps)
#--------------------------------------------------------------------------------
catchar = array(rep(1,T*ncat*5),dim=c(ncat,T,5)); # characteristics of an app
for (j in 1:ncat) { #given category
   catchar[j,,1]=t(matrix(runif(T)*39,ncol=T))    #Variance of price
   catchar[j,,2]=t(matrix(appchar[j,,3]+runif(T)*20,ncol=T))    #file size with pertubration
   catchar[j,,3]=t(matrix(appchar[j,,1]+sort(floor(runif(T)**80)),ncol=T))    #tenure with
   pertubration
   catchar[j,,4]=t(matrix(round(runif(T)*100),ncol=T))    #total number of free apps
   catchar[j,,5]=t(matrix(round(runif(T)*20),ncol=T))    #total number of free
   apps
}

#--------------------------------------------------------------------------------
# Seventh: simulate individual with HB of tenure of each individual
# start at moment 1 and create choice and state space for individual
# state of individual for category includes: number of app's in each category
# state of individual for app includes downloads of app in each category [nc*na(c)]
#--------------------------------------------------------------------------------
dummyNoPurchaseCat=c(rep(0,T)); # for no cat download option
dummyNoPurchaseApp=c(rep(0,T)); # for no app download option
indvCatState = array(rep(0,T*nIndv*ncat),dim=c(T,nIndv,ncat))   # start with no prior download
as it is starting point of this app store
indvAppState = array(rep(0,T*nIndv*napps),dim=c(T,nIndv,napps))   # start with no prior
download as it is starting point of this app store

#---------------------------------------------------
#              HB for individual for category choice
#---------------------------------------------------
ncatcoefficients = 7 +ncat           # for all 7 coefficients of utility (for fixed effects)
DeltaIndv3=matrix(runif(ncatcoefficients)*1e-1,ncol=1)
compsIndv3=NULL
for (i in 1:ncompIndv){
   compsIndv3[[i]]=list(mu=runif(ncatcoefficients)*1e-9,rooti=diag(ncatcoefficients)*1e9)
}
pvecIndv3=rep(1/ncompIndv,ncompIndv)    # equally likely
#---------------------------------------------------
#              HB for individual for app choice
#---------------------------------------------------
nappcoefficients = 5 +napps           # for all 5 coefficients of utility
DeltaIndv4=matrix(runif(nappcoefficients)*1e-1,ncol=1)
compsIndv4=NULL
for (i in 1:ncompIndv){
   compsIndv4[[i]]=list(mu=runif(nappcoefficients)*1e-9,rooti=diag(nappcoefficients)*1e9)
```

```
}
pvecIndv4=rep(1/ncompIndv,ncompIndv)    # equally likely

#-----------------------------------------------------
#                  Simulating Choice
#-----------------------------------------------------
# individual coefficients for category and app
alphai = matrix(rep(0,nIndv*ncatcoefficients),ncol=ncatcoefficients)
betai  = matrix(rep(0,nIndv*nappcoefficients),ncol=nappcoefficients)
simmnlwX= function(n,X,beta) {
   ##  simulate from MNL model conditional on X matrix
   k=length(beta)
   Xbeta=X%*%beta
   j=nrow(Xbeta)/n
   Xbeta=matrix(Xbeta,byrow=TRUE,ncol=j)
   Prob=exp(Xbeta)
   iota=c(rep(1,j))
   denom=Prob%*%iota
   Prob=Prob/as.vector(denom)
   y=vector("double",n)
   ind=1:j
   for (i in 1:n)
   {yvec=rmultinom(1,1,Prob[i,]); y[i]=ind%*%yvec}
   return(list(y=y,X=X,beta=beta,prob=Prob))
}
simCatChoice=NULL
simAppChoice=NULL
Xcattemp=matrix(rep(0,T*(ncat+1)*ncatcoefficients),nrow=T*(ncat+1))   #as ncat is number of
choices
ycatTemp       = c(rep(0,T))
Xapptemp=matrix(rep(0,T*2*(nappcoefficients)),nrow=T*2)   #as ncat is number of choices, as
only one fixed effect per item
yappTemp       = c(rep(0,T))

#app data available information
appDataAvail   = array(rep(0,napps*T*nIndv),dim=c(napps,T,nIndv))

for (i in 1:nIndv){       # given individual
   # create individual coefficients of utility for category and app choice
   alphai[i,]=DeltaIndv3%*%ZIndv[i,]+as.vector(rmixture(1,pvecIndv3,compsIndv3)$x)
   betai[i,] =DeltaIndv4%*%ZIndv[i,]+as.vector(rmixture(1,pvecIndv4,compsIndv4)$x)
   for (t in 1:T){         # given time
      #-----------------------------------------------------
      #        Choice of Category
      #-----------------------------------------------------
      # include outside option as vector of zeros
      Xacat=matrix(c(c(0,indvCatState[t,i,]),c(0,catIndvlatent[i,,t]),as.vector(rbind(rep(0,dim(
      catchar)[3]),catchar[,t,]))),nrow=1);

      Xcat=createX(p=ncat+1,na=ncatcoefficients-ncat,nd=NULL,Xa=Xacat,Xd=NULL,base=1)
      outa=simmnlwX(1,Xcat,alphai[i,])
      Xcattemp[(((t-1)*(ncat+1)+1):(t*(ncat+1))),]=Xcat #pooling choice situations
      ycatTemp[t]=outa$y
      # Update state vector of individual choic
```

```
      if ((t<T)&&(outa$y!=1)){ # the first choice is no purchase
         indvCatState[(t+1):T,i,(outa$y-1)]=indvCatState[(t+1):T,i,(outa$y-1)]+1; # keep total
         number of purchases within the category
       }
      #----------------------------------------------------
      #        Choice of app | Category (Given category)
      #----------------------------------------------------
      selectedCat=outa$y-1                     # to make sure that choice 0 is no-category
      purchase
      if (selectedCat!=0){
         #define for which item, I have the data
         appDataAvail[selectedCat,t,i]=1;

         # include outside option as vector of zeros
         Xaapp=matrix(c(c(0,indvAppState[t,i,selectedCat]),c(0,appIndvlatent[i,selectedCat,t]),
         as.vector(rbind(rep(0,dim(appchar)[3]),appchar[selectedCat,t,]))),nrow=1);

         #reateX(p=2,na=nappcoefficients -napps ,nd=NULL,Xa=Xaapp,Xd=NULL,base=1)  # 2 choice,
         either outside or dominant app
         AppfixedEffects = matrix(rep(0,2*napps),ncol=napps);
         AppfixedEffects [2,selectedCat]=1;
         Xapp=cbind(AppfixedEffects,matrix(Xaapp,ncol=(nappcoefficients -napps)))

         outapp=simmnlwX(1,Xapp,c(betai[i,]))      # to allow for fixed effect for all apps
         Xapptemp[(((t-1)*2+1):(t*2)),]=Xapp #pooling choice situations
         yappTemp[t]=outapp$y
         # Update state vector of individual choic
         if ((t<T) && (outapp$y!=1)){
            indvAppState[(t+1):T,i,(outapp$y-1)]=indvAppState[(t+1):T,i,(outapp$y-1)]+1; # keep
            total number of purchases within the category
         }
      }else{ # as I am conditioning on the category, I would need to (be careful in likelhood
      implementation to do not use this dummy data)
         Xaapp=matrix(c(c(0,0),c(0,0),as.vector(rbind(rep(0,dim(appchar)[3]),rep(0,dim(appchar)[
         3])))),nrow=1);

         #  createX(p=2,na=nappcoefficients -napps ,nd=NULL,Xa=Xaapp,Xd=NULL,base=1)  # 2
         choice, either outside or dominant app
         AppfixedEffects = matrix(rep(0,2*napps),ncol=napps);
         AppfixedEffects [2,selectedCat]=1;
         Xapp=cbind(AppfixedEffects,matrix(Xaapp,ncol=(nappcoefficients -napps)))

         outapp=simmnlwX(1,Xapp,c(betai[i,]))      # to allow for fixed effect for all apps
         Xapptemp[(((t-1)*2+1):(t*2)),]=Xapp #pooling choice situations
         yappTemp[t]=outapp$y
      }
   }
   simCatChoice[[i]]=list(y=ycatTemp,X=Xcattemp,beta=alphai[i,])
   simAppChoice[[i]]=list(y=yappTemp,X=Xapptemp,beta=betai[i,]) #be careful in coding to
   extract related items
}
```

```
#=======================================================================================
#                             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);
   colnames(thetacatj) <- c("p","q","Cj",c(1:(ncat-1)))
   i = 1   # for the first category
   newmean[i]=ctbar[i]+
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"])+
          thetacatj[i,(i+3):(ncat+2)]%*%ctbar[(i+1):ncat])*
      (thetacatj[i,"Cj"]-ctbar[i]);
    if (abs(thetacatj[i,"Cj"])<abs(ctbar[i])){
        newmean[i]=ctbar[i]
    }

   # 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,4:(i+2)]%*%ctbar[1:(i-1)]+
             thetacatj[i,(i+3):(ncat+2)]%*%ctbar[(i+1):ncat])*
         (thetacatj[i,"Cj"]-ctbar[i]);
    if (abs(thetacatj[i,"Cj"])<abs(ctbar[i])){
        newmean[i]=ctbar[i]
    }
   }
   # treat last element differently
   i = ncat;
   newmean[i]=ctbar[i]+
      (thetacatj[i,"p"]+thetacatj[i,"q"]*(ctbar[i]/thetacatj[i,"Cj"])+
          thetacatj[i,4:(i+2)]%*%ctbar[1:(i-1)])*
      (thetacatj[i,"Cj"]-ctbar[i]);
    if (abs(thetacatj[i,"Cj"])<abs(ctbar[i])){
        newmean[i]=ctbar[i]
    }
```

```
   return(list(newmean=newmean))
}

#----------------------------
#   Jacobian of Bass for Category
#----------------------------
# for test:
#ctbar=catlatent[,1]
Jccat= 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 = dim(thetacatj)[1]
   newJacob = matrix(rep(0,ncat*ncat),ncol=ncat);
   colnames(thetacatj) <- c("p","q","Cj",c(1:(ncat-1)))
   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"])+
          thetacatj[i,(i+3):(ncat+2)]%*%ctbar[(i+1):ncat])

   newJacob[i,(i+1):ncat]=thetacatj[i,(i+3):(ncat+2)]*(thetacatj[i,"Cj"]-ctbar[i])

   # treat element in the middle
   for (i in 2:(ncat-1)){
      newJacob[i,1:(i-1)]=thetacatj[i,4:(i+2)]*(thetacatj[i,"Cj"]-ctbar[i])

      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"])+
             thetacatj[i,4:(i+2)]%*%ctbar[1:(i-1)]+
             thetacatj[i,(i+3):(ncat+2)]%*%ctbar[(i+1):ncat])

      newJacob[i,(i+1):ncat]=thetacatj[i,(i+3):(ncat+2)]*(thetacatj[i,"Cj"]-ctbar[i])
   }
   # treat last element differently
   i = ncat;
   newJacob[i,1:(i-1)]=thetacatj[i,4:(i+2)]*(thetacatj[i,"Cj"]-ctbar[i])

   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"])+
          thetacatj[i,4:(i+2)]%*%ctbar[1:(i-1)])

   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)[3]
   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,nIndv*T*ncat)),dim=c(nIndv,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){
      gcat = Jccat(mtcat,thetacatj)$newJacob
      acat = fccat(mtcat,thetacatj)$newmean
      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){
      result = foreach(j=1:pcat,.combine=rbind,.packages=c("MASS","corpcor")) %dopar%{
         hcat  = Fcat%*%acat[j]         #linear as no state equation (element by element)
         Fcat  = Jacat           # for readability only
         forecastcat  = hcat     # for readability only
         Y1cattemp = forecastcat  # step ahead forecast saving
         # variance is sigma22 - sigma21*sigma11Inv*sigma12
         sigma21 = rcat[j,]
         sigma21 = sigma21[-j] # remove the elment
         sigma12 = rcat[,j]
         sigma12 = sigma12[-j] # remove the elment
         rcatTemp = rcat[j,j]- (sigma21%*%rcat[-j,-j]%*%sigma12) #using multivariate normal
         theory
         qcat = Fcat%*%rcatTemp%*%t(Fcat) + vcat[,,j]    #variance of one step ahead forecast
         ecat    = ycat[,j,t] - forecastcat             # error of forecast
```

```
         if (is.positive.definite(qcat)){
        Acat = rcatTemp%*%t(Fcat)%*%chol2inv(chol(qcat))
       }else{
        Acat = rcatTemp%*%t(Fcat)%*%ginv(qcat)

       }
         MSEcattemp =sum(ecat**2)
         MADcattemp = sum(abs(ecat))
         mtcat = acat[j] + Acat %*% ecat
         Ctcat = rcatTemp - Acat%*%qcat%*%t(Acat)
         Ctcat = (Ctcat+t(Ctcat))/2
       if (Ctcat <0){
            Ctcat  = 1e-6
       }
         # save mean and variance of posterior at time t
         #Ccat[j,j,t]   = Ctcat
         #mcat[j,t]    = mtcat
         list(Ctcat=Ctcat, mtcat=mtcat, MSE=MSEcattemp,MAD=MADcattemp,Y1cat=Y1cattemp)
      }
      Ctcat = diag(result[,1])
      Ccat[,,t] =Ctcat
      mtcat    = diag(diag(result[,2])) # I don't know why, but it works this way only
      mcat[,t] = mtcat
      MSEcat[t,]= diag(diag(result[,3]))
      MADcat[t,]= diag(diag(result[,4]))
      for (j in 1:ncat){
         Y1cat[,j,t] =cbind(result[j,5][[1]])
      }

      cat(t, ",")
   }

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

      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)%*%chol2inv(chol(rcat))
      ucat = Ccat[,,t]%*%t(gcat)
```

```
      Cmcat = Ccat[,,t] - ucat%*%chol2inv(chol(rcat))*t(ucat)
      tmcat = mcat[,t]  + bcat%*%(tt1cat[,t+1]-acat)

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

   }

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

   return(list(mcat=mcat,Ccat=Ccat,m0cat=m0cat,C0cat=C0cat,tt1cat=tt1cat,tt0cat=tt0cat,Y1cat=
   Y1cat,MADcat=MADcat,MSEcat=MSEcat))
}

#---------------------------------------------------------------------------------------
#                              Extended Kalman Filter for Apps
#---------------------------------------------------------------------------------------

#-----------------------------
#   Bass function for the app
#-----------------------------
# for test:
# mtbar=appslatent[,1]
fmapp= function(mtbar,thetaappa,catlatent,t) {
   # mtbar is vector of napps latent (mean of previous period)
   ##  Bass diffusion function for apps, getting old vector of latent mean and returning the
   next latent mean
   napps = length(mtbar)
   thetaappa =matrix(as.numeric(thetaappa),nrow=napps)
   newmean = rep(0,napps)
   maxmarketsize = rep(0,napps)

   colnames(thetaappa) <- c("p","q","alphaj")

   for (a in 1:napps) { #given category
      # treat first eelement differently
    if (abs(thetaappa[a,"alphaj"]* catlatent[a,t])<abs(mtbar[a])){
        mtbar[a]= thetaappa[a,"alphaj"]* catlatent[a,t]
    }
      newmean[a]=mtbar[a]+
```

```
         (thetaappa[a,"p"]+thetaappa[a,"q"]*(mtbar[a]/(thetaappa[a,"alphaj"]* catlatent[a,t])))*
         ((thetaappa[a,"alphaj"]* catlatent[a,t])-mtbar[a])
    maxmarketsize [a] = (thetaappa[a,"alphaj"]* catlatent[a,t])
   }
   return(list(newmean=newmean,maxmarketsize =maxmarketsize ))
}

#----------------------------
#   Jacobian of Bass for Category
#----------------------------
# for test:
# mtbar=appslatent[,1]
Jmapp= function(mtbar,thetaappa,catlatent,t) {
   # 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
   napps = dim(thetaappa)[1]
   newJacob = matrix(rep(0,napps),ncol=napps);
   colnames(thetaappa) <- c("p","q","alphaj")

   for (a in 1:napps) { #given category
    if (abs(thetaappa[a,"alphaj"]* catlatent[a,t])<abs(mtbar[a])){
        mtbar[a]= thetaappa[a,"alphaj"]* catlatent[a,t]
    }
      # treat first eelement differently
      newJacob[a]=1+(thetaappa[a,"q"]/(thetaappa[a,"alphaj"]* catlatent[a,t]))* ((thetaappa[a,
      "alphaj"]* catlatent[a,t])-mtbar[a])-
         (thetaappa[a,"p"]+thetaappa[a,"q"]*(mtbar[a]/(thetaappa[a,"alphaj"]* catlatent[a,t])))
   }

   return(list(newJacob=newJacob))
}



#----------------------------
# EKF of the mobile apps
# 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:
# yapp = appIndvlatent
# Fapp = etaIndvapp
# papp = 1
# m0app = 3*matrix(c(rep(1,napps)),ncol=napps)
# C0app = 2*(c(rep(1,napps)))
# vapp = matrix(rep(0,napps*nIndv*nIndv),ncol=nIndv)
# for (j in 1:napps){
#       vapp[(nIndv*(j-1)+1):(nIndv*j),] = 0.01*diag(nIndv)
# }
# wapp = 0.5*c(rep(1,napps))
appEKF= function(yapp,Fapp,papp,m0app,C0app,vapp,wapp,thetaappa,catlatent) {
   # Definition of Variables
   # yapp: [I*A*T]  data to use as observation equation
   # Fapp   : [I*1]  it is the same across time
```

```
    # papp   : 1 for now as only one state is running EKF
    # m0app: [p*A] for the mean of the state of current app
    # C0app: [p*p*A] for the variance of state equation at each point in time
    # vapp : [(A*I)xI]   for simplicity it could be diagonal but general case is also possible
    # wapp : [A*1]   for the variance of state equation of current app
    # thetaappa: [A*3] for the coefficients, generaly it is GT

    T    = dim(yapp)[3]
    napps = dim(yapp)[2]
    nIndv = dim(yapp)[1]
    MSEapp = matrix(rep(0,T*napps),ncol=T)                                    # in each loop sum
    for all the individuals
    MADapp = matrix(rep(0,T*napps),ncol=T)                                     # in each loop
    sum for all the individuals
    Y1app = array(c(rep(0,nIndv*T*napps)),dim=c(nIndv,napps,T))    # T*J matrix
    mapp = matrix(rep(m0app,T),ncol=T)
    Capp = matrix(rep(0,napps*T),ncol=T)
    Capp[,1]=C0app
    mapp[,1]=m0app
    mtapp = m0app
    Ctapp = C0app
    # Kalman Filtering
    for (t in 1:T){
       gapp = Jmapp(mtapp,thetaappa,catlatent,t)$newJacob
       aapp = fmapp(mtapp,thetaappa,catlatent,t)$newmean
     maxmarketsize =  fmapp(mtapp,thetaappa,catlatent,t)$maxmarketsize

       Jaapp = Fapp                      # a vector as there is no nonlinearity
       Fapp  = Jaapp           # for readability only

       rapp = gapp*Ctapp*gapp+wapp      #variance of prior t-1

#       happ  = Fapp%x%aapp         #linear as no state equation (element by element)
#
#       forecastapp  = happ     # for readability only
#       Y1app[,,t] = as.matrix(forecastapp,ncol=napps)  # step ahead forecast saving
#       eapp    = as.vector(yapp[,,t]) - forecastapp             # error of forecast
#       MSEapp[t] =sum(eapp**2)
#       MADapp[t] = sum(abs(eapp))
#       qapp = t(rapp)%x%(Fapp%*%t(Fapp)) + vapp[,]

       #for (a in 1:napps){
       result = foreach(a=1:napps,.combine=rbind,.packages=c("corpcor","MASS")) %dopar%{
              #variance of one step ahead forecast
#          qapptemp=(qapp[((a-1)*nIndv+1):(a*nIndv),]+t(qapp[((a-1)*nIndv+1):(a*nIndv),]))/2
#          Aapp = rapp[a]%*%t(Fapp)%*%chol2inv(chol(qapptemp))
#
#          mtapp = aapp[a] + Aapp %*% eapp[((a-1)*nIndv+1):(a*nIndv)]
#          Ctapp[a] = rapp[a] - Aapp%*%qapptemp%*%t(Aapp)

          happ  = Fapp%*%aapp[a]
          forecastapp  = happ     # for readability only
          forecastTemp= as.matrix(forecastapp,ncol=napps)  # step ahead forecast saving
          eapp    = as.vector(yapp[,a,t]) - forecastapp             # error of forecast
```

```
        MSEapptemp =sum(eapp**2)
        MADapptemp = sum(abs(eapp))
        qapp = (Fapp%*%rapp[a]%*%t(Fapp)) + vapp[(nIndv*(a-1)+1):(nIndv*a),]
        qapptemp=qapp
      if (is.positive.definite(qapptemp)){
            Aapp = rapp[a]%*%t(Fapp)%*%chol2inv(chol(qapptemp))
      }else{
       Aapp = rapp[a]%*%t(Fapp)%*%diag(rep(1e-6,sqrt(length(qapptemp))))

      }
      if (maxmarketsize [a]<aapp[a] ){
       aapp[a] = maxmarketsize [a]
      }
        mtapp = aapp[a] + Aapp %*% eapp
       if (maxmarketsize [a]<mtapp  ){
       mtapp  = maxmarketsize [a]
      }

      if (rapp[a] - Aapp%*%qapptemp%*%t(Aapp) >0){
         Ctapp[a] = rapp[a] - Aapp%*%qapptemp%*%t(Aapp)
      }else{
         Ctapp[a] = 1e-6
      }

        # save mean and variance of posterior at time t
        #Ccat[j,j,t]    = Ctcat
        #mcat[j,t]     = mtcat
        list(Ctapp=Ctapp[a], mtapp=mtapp,MSE=MSEapptemp,MAD=MADapptemp,forecast=forecastTemp)

     }
     Ctapp = diag(diag(result[,1]))
     Ctapp = (Ctapp+t(Ctapp))/2
     Capp[,t] =Ctapp
     mtapp    = diag(diag(result[,2])) # I don't know why, but it works this way only
     mapp[,t] = mtapp
     MSEapp[,t]=diag(diag(result[,3]))
     MADapp[,t]=diag(diag(result[,4]))
     for (a in 1:napps){
        Y1app[,a,t] =cbind(result[a,5][[1]])
     }
     cat(t, ",")
  }

  #backward smoothing
  tt1app = matrix(rep(0,napps*T),ncol=T)
  tt1app[,T] = mapp[,T]+sqrt(abs(Capp[,T]))*as.matrix(rnorm(napps))
  for (t in (T-1):1){
     gapp = Jmapp(mapp[,t],thetaappa,catlatent,t)$newJacob
     aapp = fmapp(mapp[,t],thetaappa,catlatent,t)$newmean
     maxmarketsize =  fmapp(mapp[,t],thetaappa,catlatent,t)$maxmarketsize
   aapp =pmin(aapp,maxmarketsize)

     rapp = as.vector(gapp)*Capp[,t]*as.vector(gapp)+wapp      #variance of prior t-1
```

```
      bapp =(as.matrix((Capp[,t]*as.vector(gapp)),ncol=1)[,1])/(as.matrix(rapp,ncol=1)[,1])
      uapp = (as.matrix((Capp[,t]*as.vector(gapp)),ncol=1)[,1])

      Cmapp = Capp[,t] - as.matrix(uapp,ncol=1)[,1]/(as.matrix(rapp,ncol=1)[,1])*as.matrix(uapp,
      ncol=1)[,1]
      tmapp = mapp[,t]  + bapp*(tt1app[,t+1]-aapp)

      if (min(diag(Cmapp))<0){
#         stop("Negative Variance Found in C Matrix",Call.=FALSE)
#       Cmapp = diag(make.positive.definite(diag(Cmapp)))
        Cmapp= pmax(Cmapp,1e-6)

      }

      # save mean and variance of posterior at time t
      Capp[,t]   = Capp[,t] - bapp*(rapp-Capp[,t+1])*bapp
      mapp[,t]    = mapp[,t]  + bapp*(mapp[,t+1]-aapp)
     mapp[,t] =pmin( mapp[,t],maxmarketsize)

      tt1app[,t]  = tmapp     + sqrt(abs(Cmapp))*as.matrix(rnorm(napps))

   }

   #now ad-hoc treatment of start value
   m0app  = mapp[,1]
   C0app  = Capp[,1]
   tt0app    = m0app     + sqrt(abs(C0app))*as.matrix(rnorm(napps))

   return(list(mapp=mapp,Capp=Capp,m0app=m0app,C0app=C0app,tt1app=tt1app,tt0app=tt0app,Y1app=
   Y1app,MADapp=MADapp,MSEapp=MSEapp))
}

#-------------------------------------------------------------------------------------
#                             Metrapolis RW first step
#-------------------------------------------------------------------------------------
#           Inputs
#---------------------------------------------------
#Data:list(p=p,lgtdata=simlgtdata,Z=Z)
#---------------------------------------------------
#pc   = ncat+1        # number of choices alternatives of categories (one for outside option)
#pa =2                # number of choices alternatives of apps (outside option + dominant app)
#lgtdatac   =simCatChoice    #data of choice of cat (y=catTemp, X=Xcattemp,beta=betai)
                            # one list per unit (which here is individual during all days)
                            # X is a (length(y)*pc) x nvarcat (y: selected alternative of cat)
#lgtdataa   =simAppChoice    #data of choice of app (y=yappTemp,X=Xapptemp,beta=alphai)
                            # one list per unit (which here is individual during all days)
                            # X is a (length(y)*pa) x nvarapp (y: selected alternative of app)

#Zc =Zcat       # pc x nzcat matrix
               # to explain heterogeneity in category(with no intercept)
#Za =Zapp       # pa x nzapp matrix
               #  to explain heterogeneity in apps (with no intercept)
#ZI =ZIndv      # length(lgtdataa) x nzIndv matrix, but (lgtdataa=lgtdatac)
                  #  to explain heterogeneity in apps (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)
#----------------------------------------------------
#ncomIndv:   number of components in normal mixture of Individual coefficients
#ncomcat:    number of components in normal mixture of category coefficients
#ncomapp:    number of components in normal mixture of app coefficients


#----------------------------------------------------
#  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

#----------------------------------------------------
#  Explanation:
#----------------------------------------------------
#  weighting scheme: (1-w)logl_i + w*Lbar (normalized)
#  run hierarchical mnl logit model incorporating DNLM (dynamic non-linear model) with mixture
of normals
#   using RW and cov(RW inc) = (hess_i + Vbeta^-1)^-1
#   uses normal approximation to pooled likelihood



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

#------------------------------
#prepare Data
#------------------------------
Data = list(pc=ncat+1,pa=2,lgtdatac = simCatChoice,lgtdataa=simAppChoice, Zc=Zcat, Za=Zapp, ZI=
ZIndv)
jumps      =    1
idx = 0
jp = 0
ndraw=1000;
ndraw0 = 500
burnIn = ndraw0
McMc = list(R=ndraw,keep=jumps)
Prior= list(ncomIndv=ncompIndv,ncomcat=ncompcat,ncomapp=ncompapp)
ncat = ncat
napp = napps
nIndv = nIndv

# Priors for alphai,betai,thetacatj,thetaappa,etaIndvapp,gammaIndv)
#    beta_i = D %*% z[i,] + u_i
#       u_i ~ N(mu_ind[i],Sigma_ind[i])
#       ind[i] ~multinomial(p)
#       p ~ dirichlet (a)
#       D is a k x nz array
#          delta= vec(D) ~ N(deltabar,A_d^-1)
#    mu_j ~ N(mubar,A_mu^-1(x)Sigma_j)
#    Sigma_j ~ IW(nu,V^-1)

#----------------------------------------------------
#  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
#------------------------------

if(missing(Data)) {pandterm("Requires Data argument -- list of pc,pa,lgtdatac,lgtdataa, and 
Zc,Za,ZI")}
if(is.null(Data$pc)) {pandterm("Requires Data element pc (# chce alternatives of category)") }
if(is.null(Data$pa)) {pandterm("Requires Data element pa (# chce alternatives of apps)") }
pc=Data$pc
pa=Data$pa
if(is.null(Data$lgtdatac)) {pandterm("Requires Data element lgtdatac (list of choice data for 
category for each unit)")}
if(is.null(Data$lgtdataa)) {pandterm("Requires Data element lgtdataa (list of choice data for 
app for each unit)")}
lgtdatac=Data$lgtdatac
lgtdataa=Data$lgtdataa
nlgt=length(lgtdataa)
#-------------------------------
# Component heterogeneity data check
#-------------------------------
drawdeltaalphai=TRUE
drawdeltabetai=TRUE
drawdeltathetacatj=TRUE
drawdeltathetaappa=TRUE
drawdeltaetaIndvapp=TRUE
drawdeltagammaIndv=TRUE
if(is.null(Data$Zc)) { cat("Zc not specified",fill=TRUE); fsh() ; drawdelta=FALSE} else {Zc=Data
$Zc}
if(is.null(Data$Za)) { cat("Za not specified",fill=TRUE); fsh() ; drawdelta=FALSE} else {Za=Data
$Za}
if(is.null(Data$ZI)) { cat("ZI not specified",fill=TRUE); fsh() ; drawdelta=FALSE}else {
   if (nrow(Data$ZI) != nlgt) {pandterm(paste("Nrow(Z) ",nrow(ZI),"ne number logits ",nlgt))}
   else {ZI=Data$ZI}}
if(drawdeltaalphai) {
   nzalphai=ncol(ZI)
   colmeans=apply(ZI,2,mean)
   if(sum(colmeans) > .00001)
   {pandterm(paste("ZI does not appear to be de-meaned: colmeans= ",colmeans))}
}
if(drawdeltabetai) {
   nzbetai=ncol(ZI)
   colmeans=apply(ZI,2,mean)
   if(sum(colmeans) > .00001)
   {pandterm(paste("ZI does not appear to be de-meaned: colmeans= ",colmeans))}
}
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))}
```

```
}
if(drawdeltathetaappa) {
   nzthetaappa=ncol(Za)
   colmeans=apply(Za,2,mean)
   if(sum(colmeans) > .00001)
   {pandterm(paste("Za does not appear to be de-meaned: colmeans= ",colmeans))}
}
if(drawdeltaetaIndvapp) {
   nzetaIndvapp=ncol(ZI)
   colmeans=apply(ZI,2,mean)
   if(sum(colmeans) > .00001)
   {pandterm(paste("ZI does not appear to be de-meaned: colmeans= ",colmeans))}
}
if(drawdeltagammaIndv) {
   nzgammaIndv=ncol(ZI)
   colmeans=apply(ZI,2,mean)
   if(sum(colmeans) > .00001)
   {pandterm(paste("ZI does not appear to be de-meaned: colmeans= ",colmeans))}
}



#-----------------------------------
# check lgtdatac for validity for format of (y,X)
#-----------------------------------
ypooledcat=NULL
Xpooledcat=NULL
if(!is.null(lgtdatac[[1]]$X)) {oldncol=ncol(lgtdatac[[1]]$X)}
for (i in 1:nlgt)
{
   if(is.null(lgtdatac[[i]]$y)) {pandterm(paste("Requires element y of lgtdatac[[",i,"]]"))}
   if(is.null(lgtdatac[[i]]$X)) {pandterm(paste("Requires element X of lgtdatac[[",i,"]]"))}
   ypooledcat=c(ypooledcat,lgtdatac[[i]]$y)
   nrowX=nrow(lgtdatac[[i]]$X)
   if((nrowX/pc) !=length(lgtdatac[[i]]$y)) {pandterm(paste("nrow(X) ne pc*length(yi);
   exception at unit",i))}
   newncol=ncol(lgtdatac[[i]]$X)
   if(newncol != oldncol) {pandterm(paste("All X elements must have same # of cols; exception
   at unit(category)",i))}
   Xpooledcat=rbind(Xpooledcat,lgtdatac[[i]]$X)
   oldncol=newncol
}
nvarcat=ncol(Xpooledcat)
levelycat=as.numeric(levels(as.factor(ypooledcat)))
if(max(length(levelycat)) != pc) {pandterm(paste("y takes on ",length(levelycat)," values --
must be = pc"))}
badycat=FALSE
for (i in 1:pc )
{
   if(levelycat[i] != i) badycat=TRUE
}
cat("Table of Yc values pooled over all units",fill=TRUE)
print(table(ypooledcat))
if (badycat)
```

```
{pandterm("Invalid Yc")}


#---------------------------------
# check lgtdatac for validity for format of (y,X)
#---------------------------------
ypooledapp=NULL
Xpooledapp=NULL
if(!is.null(lgtdataa[[1]]$X)) {oldncol=ncol(lgtdataa[[1]]$X)}
for (i in 1:nlgt)
{
   if(is.null(lgtdataa[[i]]$y)) {pandterm(paste("Requires element y of lgtdataa[[",i,"]]"))}
   if(is.null(lgtdataa[[i]]$X)) {pandterm(paste("Requires element X of lgtdataa[[",i,"]]"))}
   ypooledapp=c(ypooledapp,lgtdataa[[i]]$y)
   nrowX=nrow(lgtdataa[[i]]$X)
   if((nrowX/pa) !=length(lgtdataa[[i]]$y)) {pandterm(paste("nrow(X) ne pc*length(yi);
   exception at unit",i))}
   newncol=ncol(lgtdataa[[i]]$X)
   if(newncol != oldncol) {pandterm(paste("All X elements must have same # of cols; exception
   at unit(apps)",i))}
   Xpooledapp=rbind(Xpooledapp,lgtdataa[[i]]$X)
   oldncol=newncol
}
nvarapp=ncol(Xpooledapp)
levelyapp=as.numeric(levels(as.factor(ypooledapp)))
if(max(length(levelyapp)) != pa) {pandterm(paste("y takes on ",length(levelyapp)," values --
must be = pa"))}
badyapp=FALSE
for (i in 1:pa )
{
   if(levelyapp[i] != i) badyapp=TRUE
}
cat("Table of Ya values pooled over all units",fill=TRUE)
print(table(ypooledapp))
if (badyapp)
{pandterm("Invalid Ya")}

#---------------------------------------------
#     Check McMc
#---------------------------------------------
if(missing(McMc)) {pandterm("Requires Mcmc list argument")}
if(!missing(McMc)){
   if(is.null(McMc$s)) {s=2.93/sqrt(nvarcat)} else {s=McMc$s}
   if(is.null(McMc$w)) {w=.1}  else {w=McMc$w}
   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$ncomIndv)) {pandterm("Requires Prior element ncomIndv (num of mixture
```

```
components for individuals")} else {ncomIndv=Prior$ncomIndv}
if(is.null(Prior$ncomcat)) {pandterm("Requires Prior element ncomcat (num of mixture components
for categories)")} else {ncomcat=Prior$ncomcat}
if(is.null(Prior$ncomapp)) {pandterm("Requires Prior element ncomapp (num of mixture components
for apps)")} else {ncomapp=Prior$ncomapp}

# prior for mubar across 6 HB
# number of coefficients should be set for the number of means
#alphai
nvaralphai = nvarcat
if(is.null(Prior$mubaralphai)) {mubaralphai=matrix(rep(0,nvarcat),nrow=1)} else { mubaralphai=
matrix(Prior$mubaralphai,nrow=1)}
if(ncol(mubaralphai) != nvarcat) {pandterm(paste("mubarmubaralphai must have ncomp cols,
ncol(mubaralphai)= ",ncol(mubaralphai)))}
if(is.null(Prior$Amualphai)) {Amualphai=matrix(.01,ncol=1)} else {Amualphai=matrix(Prior$
Amualphai,ncol=1)}
if(ncol(Amualphai) != 1 | nrow(Amualphai) != 1) {pandterm("Am alphaimust be a 1 x 1 array")}
if(is.null(Prior$nualphai)) {nualphai=nvarcat+3}  else {nualphai=Prior$nualphai}
if(nualphai < 1) {pandterm("invalid nualphai value")}
if(is.null(Prior$Valphai)) {Valphai=nualphai*diag(nvarcat)} else {Valphai=Prior$Valphai}
if(sum(dim(Valphai)==c(nvarcat,nvarcat)) !=2) pandterm("Invalid Valphai in prior")
if(is.null(Prior$Adalphai) & drawdeltaalphai) {Adalphai=.01*diag(nvarcat*nzalphai)} else {
Adalphai=Prior$Adalphai}
if(drawdeltaalphai) {if(ncol(Adalphai) != nvarcat*nzalphai | nrow(Adalphai) != nvarcat*nzalphai)
 {pandterm("Adalphai must be nvarcat*nzalphai x nvarcat*nzalphai")}}
if(is.null(Prior$deltabaralphai)& drawdeltaalphai) {deltabaralphai=rep(0,nzalphai*nvarcat)} else
 {deltabaralphai=Prior$deltabaralphai}
if(drawdeltaalphai) {if(length(deltabaralphai) != nzalphai*nvarcat) {pandterm("deltabar must be
of length nvarcat*nzalphai")}}
if(is.null(Prior$aalphai)) { aalphai=rep(5,ncomIndv)} else {aalphai=Prior$aalphai}
if(length(aalphai) != ncomIndv) {pandterm("Requires dim(aalphai)= ncomp (no of components)")}
badaalphai=FALSE
for(i in 1:ncomIndv) { if(aalphai[i] < 1) badaalphai=TRUE}
if(badaalphai) pandterm("invalid values in a vector in alphai")

#betai
nvarbetai =nvarapp
if(is.null(Prior$mubarbetai)) {mubarbetai=matrix(rep(0,nvarapp),nrow=1)} else { mubarbetai=
matrix(Prior$mubarbetai,nrow=1)}
if(ncol(mubarbetai) != nvarapp) {pandterm(paste("mubar must have ncomp cols, ncol(mubarbetai)= "
,ncol(mubarbetai)))}
if(is.null(Prior$Amubetai)) {Amubetai=matrix(.01,ncol=1)} else {Amubetai=matrix(Prior$Amubetai,
ncol=1)}
if(ncol(Amubetai) != 1 | nrow(Amubetai) != 1) {pandterm("Ambetai must be a 1 x 1 array")}
if(is.null(Prior$nubetai)) {nubetai=nvarapp+3}  else {nubetai=Prior$nubetai}
if(nubetai < 1) {pandterm("invalid nubetai value")}
if(is.null(Prior$Vbetai)) {Vbetai=nubetai*diag(nvarapp)} else {Vbetai=Prior$Vbetai}
if(sum(dim(Vbetai)==c(nvarapp,nvarapp)) !=2) pandterm("Invalid Vbetai in prior")
if(is.null(Prior$Adbetai) & drawdeltabetai) {Adbetai=.01*diag(nvarapp*nzbetai)} else {Adbetai=
Prior$Adbetai}
if(drawdeltabetai) {if(ncol(Adbetai) != nvarapp*nzbetai | nrow(Adbetai) != nvarapp*nzbetai) {
pandterm("Adbetai must be nvarapp*nzbetai x nvarapp*nzbetai")}}
if(is.null(Prior$deltabarbetai)& drawdeltabetai) {deltabarbetai=rep(0,nzbetai*nvarapp)} else {
deltabarbetai=Prior$deltabarbetai}
```

```
if(drawdeltabetai) {if(length(deltabarbetai) != nzbetai*nvarapp) {pandterm("deltabar must be of
length nvarapp*nzbetai")}}
if(is.null(Prior$abetai)) { abetai=rep(5,ncomIndv)} else {abetai=Prior$abetai}
if(length(abetai) != ncomIndv) {pandterm("Requires dim(abetai)= ncomp (no of components)")}
badabetai=FALSE
for(i in 1:ncomIndv) { if(abetai[i] < 1) badabetai=TRUE}
if(badabetai) pandterm("invalid values in a vector in betai")

#thetacatj
nvarthetacatj = ncat+2
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")

#thetaappa
nvarthetaappa=3
if(is.null(Prior$mubarthetaappa)) {mubarthetaappa=matrix(rep(0,nvarthetaappa),nrow=1)} else {
mubarthetaappa=matrix(Prior$mubarthetaappa,nrow=1)}
if(ncol(mubarthetaappa) != nvarthetaappa) {pandterm(paste("mubar must have ncomp cols,
ncol(mubarthetaappa)= ",ncol(mubarthetaappa)))}
if(is.null(Prior$Amuthetaappa)) {Amuthetaappa=matrix(.01,ncol=1)} else {Amuthetaappa=matrix(
Prior$Amuthetaappa,ncol=1)}
if(ncol(Amuthetaappa) != 1 | nrow(Amuthetaappa) != 1) {pandterm("Amthetaappa must be a 1 x 1
array")}
if(is.null(Prior$nuthetaappa)) {nuthetaappa=nvarthetaappa+3}  else {nuthetaappa=Prior$
nuthetaappa}
if(nuthetaappa < 1) {pandterm("invalid nuthetaappa value")}
```

```
if(is.null(Prior$Vthetaappa)) {Vthetaappa=nuthetaappa*diag(nvarthetaappa)} else {Vthetaappa=
Prior$Vthetaappa}
if(sum(dim(Vthetaappa)==c(nvarthetaappa,nvarthetaappa)) !=2) pandterm("Invalid Vthetaappa in
prior")
if(is.null(Prior$Adthetaappa) & drawdeltathetaappa) {Adthetaappa=.01*diag(nvarthetaappa*
nzthetaappa)} else {Adthetaappa=Prior$Adthetaappa}
if(drawdeltathetaappa) {if(ncol(Adthetaappa) != nvarthetaappa*nzthetaappa | nrow(Adthetaappa) !=
 nvarthetaappa*nzthetaappa) {pandterm("Adthetaappa must be nvarthetaappa*nzthetaappa x
nvarthetaappa*nzthetaappa")}}
if(is.null(Prior$deltabarthetaappa)& drawdeltathetaappa) {deltabarthetaappa=rep(0,nzthetaappa*
nvarthetaappa)} else {deltabarthetaappa=Prior$deltabarthetaappa}
if(drawdeltathetaappa) {if(length(deltabarthetaappa) != nzthetaappa*nvarthetaappa) {pandterm(
"deltabar must be of length nvarthetaappa*nzthetaappa")}}
if(is.null(Prior$athetaappa)) { athetaappa=rep(5,ncomapp)} else {athetaappa=Prior$athetaappa}
if(length(athetaappa) != ncomapp) {pandterm("Requires dim(athetaappa)= ncomp (no of components)"
)}
badathetaappa=FALSE
for(i in 1:ncomapp) { if(athetaappa[i] < 1) badathetaappa=TRUE}
if(badathetaappa) pandterm("invalid values in a vector in thetaappa")

#etaIndvapp
nvaretaIndvapp = 1
if(is.null(Prior$mubaretaIndvapp)) {mubaretaIndvapp=matrix(rep(0,nvaretaIndvapp),nrow=1)} else {
 mubaretaIndvapp=matrix(Prior$mubaretaIndvapp,nrow=1)}
if(ncol(mubaretaIndvapp) != nvaretaIndvapp) {pandterm(paste("mubar must have ncomp cols,
ncol(mubaretaIndvapp)= ",ncol(mubaretaIndvapp)))}
if(is.null(Prior$AmuetaIndvapp)) {AmuetaIndvapp=matrix(.01,ncol=1)} else {AmuetaIndvapp=matrix(
Prior$AmuetaIndvapp,ncol=1)}
if(ncol(AmuetaIndvapp) != 1 | nrow(AmuetaIndvapp) != 1) {pandterm("AmetaIndvapp must be a 1 x 1
array")}
if(is.null(Prior$nuetaIndvapp)) {nuetaIndvapp=nvaretaIndvapp+3}  else {nuetaIndvapp=Prior$
nuetaIndvapp}
if(nuetaIndvapp < 1) {pandterm("invalid nuetaIndvapp value")}
if(is.null(Prior$VetaIndvapp)) {VetaIndvapp=nuetaIndvapp*diag(nvaretaIndvapp)} else {VetaIndvapp
=Prior$VetaIndvapp}
if(sum(dim(VetaIndvapp)==c(nvaretaIndvapp,nvaretaIndvapp)) !=2) pandterm("Invalid VetaIndvapp
in prior")
if(is.null(Prior$AdetaIndvapp) & drawdeltaetaIndvapp) {AdetaIndvapp=.01*diag(nvaretaIndvapp*
nzetaIndvapp)} else {AdetaIndvapp=Prior$AdetaIndvapp}
if(drawdeltaetaIndvapp) {if(ncol(AdetaIndvapp) != nvaretaIndvapp*nzetaIndvapp | nrow(
AdetaIndvapp) != nvaretaIndvapp*nzetaIndvapp) {pandterm("AdetaIndvapp must be
nvaretaIndvapp*nzetaIndvapp x nvaretaIndvapp*nzetaIndvapp")}}
if(is.null(Prior$deltabaretaIndvapp)& drawdeltaetaIndvapp) {deltabaretaIndvapp=rep(0,
nzetaIndvapp*nvaretaIndvapp)} else {deltabaretaIndvapp=Prior$deltabaretaIndvapp}
if(drawdeltaetaIndvapp) {if(length(deltabaretaIndvapp) != nzetaIndvapp*nvaretaIndvapp) {pandterm
("deltabaretaIndvapp must be of length nvaretaIndvapp*nzetaIndvapp")}}
if(is.null(Prior$aetaIndvapp)) { aetaIndvapp=rep(5,ncomIndv)} else {aetaIndvapp=Prior$
aetaIndvapp}
if(length(aetaIndvapp) != ncomIndv) {pandterm("Requires dim(aetaIndvapp)= ncomp (no of
components)")}
badaetaIndvapp=FALSE
for(i in 1:ncomIndv) { if(aetaIndvapp[i] < 1) badaetaIndvapp=TRUE}
if(badaetaIndvapp) pandterm("invalid values in a vector in etaIndvapp")
```

```
#gammaIndv
nvargammaIndv = 1
if(is.null(Prior$mubargammaIndv)) {mubargammaIndv=matrix(rep(0,nvargammaIndv),nrow=1)} else {
mubargammaIndv=matrix(Prior$mubargammaIndv,nrow=1)}
if(ncol(mubargammaIndv) != nvargammaIndv) {pandterm(paste("mubar must have ncomp cols,
ncol(mubargammaIndv)= ",ncol(mubargammaIndv)))}
if(is.null(Prior$AmugammaIndv)) {AmugammaIndv=matrix(.01,ncol=1)} else {AmugammaIndv=matrix(
Prior$AmugammaIndv,ncol=1)}
if(ncol(AmugammaIndv) != 1 | nrow(AmugammaIndv) != 1) {pandterm("AmgammaIndv must be a 1 x 1
array")}
if(is.null(Prior$nugammaIndv)) {nugammaIndv=nvargammaIndv+3}  else {nugammaIndv=Prior$
nugammaIndv}
if(nugammaIndv < 1) {pandterm("invalid nugammaIndv value")}
if(is.null(Prior$VgammaIndv)) {VgammaIndv=nugammaIndv*diag(nvargammaIndv)} else {VgammaIndv=
Prior$VgammaIndv}
if(sum(dim(VgammaIndv)==c(nvargammaIndv,nvargammaIndv)) !=2) pandterm("Invalid VgammaIndv in
prior")
if(is.null(Prior$AdgammaIndv) & drawdeltagammaIndv) {AdgammaIndv=.01*diag(nvargammaIndv*
nzgammaIndv)} else {AdgammaIndv=Prior$AdgammaIndv}
if(drawdeltagammaIndv) {if(ncol(AdgammaIndv) != nvargammaIndv*nzgammaIndv | nrow(AdgammaIndv) !=
 nvargammaIndv*nzgammaIndv) {pandterm("Ad must be nvargammaIndv*nzgammaIndv x
nvargammaIndv*nzgammaIndv")}}
if(is.null(Prior$deltabargammaIndv)& drawdeltagammaIndv) {deltabargammaIndv=rep(0,nzgammaIndv*
nvargammaIndv)} else {deltabargammaIndv=Prior$deltabargammaIndv}
if(drawdeltagammaIndv) {if(length(deltabargammaIndv) != nzgammaIndv*nvargammaIndv) {pandterm(
"deltabargammaIndv must be of length nvargammaIndv*nzgammaIndv")}}
if(is.null(Prior$agammaIndv)) { agammaIndv=rep(5,ncomIndv)} else {agammaIndv=Prior$agammaIndv}
if(length(agammaIndv) != ncomIndv) {pandterm("Requires dim(agammaIndv)= ncomp (no of
components)")}
badagammaIndv=FALSE
for(i in 1:ncomIndv) { if(agammaIndv[i] < 1) badagammaIndv=TRUE}
if(badagammaIndv) pandterm("invalid values in a vector in gammaIndv")


#--------------------------------------------------------------------------
# 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",ncomIndv,"components of individuals,",
    ncomcat,"components of categories,",
    ncomapp,"components of apps,"," for first and second stage prior",fill=TRUE)
cat(paste("   ",pc," alternatives of categories and ",pa,"alternatives of apps;", nvarcat,
          " variables in X of category", nvarapp," variables in X of apps"),fill=TRUE)
cat(paste("   for ",nlgt," cross-sectional units(Individuals who selected an app)"),fill=TRUE)
#alphai
cat(" ",fill=TRUE)
cat("Prior Parms for alphai: ",fill=TRUE)
cat("nualphai =",nualphai,fill=TRUE)
cat("Valphai ",fill=TRUE)
print(Valphai)
cat("mubaralphai ",fill=TRUE)
print(mubaralphai)
```

```
cat("Amualphai ", fill=TRUE)
print(Amualphai)
cat("aalphai ",fill=TRUE)
print(aalphai)
if(drawdeltaalphai)
{
   cat("deltabaralphai",fill=TRUE)
   print(deltabaralphai)
   cat("Adalphai",fill=TRUE)
   print(Adalphai)
}

#betai
cat("Prior Parms for betai: ",fill=TRUE)
cat("nubetai =",nubetai,fill=TRUE)
cat("Vbetai ",fill=TRUE)
print(Vbetai)
cat("mubarbetai ",fill=TRUE)
print(mubarbetai)
cat("Amubetai ", fill=TRUE)
print(Amubetai)
cat("abetai ",fill=TRUE)
print(abetai)
if(drawdeltabetai)
{
   cat("deltabarbetai",fill=TRUE)
   print(deltabarbetai)
   cat("Adbetai",fill=TRUE)
   print(Adbetai)
}

#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)
}

#thetaappa
cat("Prior Parms for thetaappa: ",fill=TRUE)
cat("nuthetaappa =",nuthetaappa,fill=TRUE)
cat("Vthetaappa ",fill=TRUE)
```

```
print(Vthetaappa)
cat("mubarthetaappa ",fill=TRUE)
print(mubarthetaappa)
cat("Amuthetaappa ", fill=TRUE)
print(Amuthetaappa)
cat("athetaappa ",fill=TRUE)
print(athetaappa)
if(drawdeltathetaappa)
{
   cat("deltabarthetaappa",fill=TRUE)
   print(deltabarthetaappa)
   cat("Adthetaappa",fill=TRUE)
   print(Adthetaappa)
}

#etaIndvapp
cat("Prior Parms for etaIndvapp: ",fill=TRUE)
cat("nuetaIndvapp =",nuetaIndvapp,fill=TRUE)
cat("VetaIndvapp ",fill=TRUE)
print(VetaIndvapp)
cat("mubaretaIndvapp ",fill=TRUE)
print(mubaretaIndvapp)
cat("AmuetaIndvapp ", fill=TRUE)
print(AmuetaIndvapp)
cat("aetaIndvapp ",fill=TRUE)
print(aetaIndvapp)
if(drawdeltaetaIndvapp)
{
   cat("deltabaretaIndvapp",fill=TRUE)
   print(deltabaretaIndvapp)
   cat("AdetaIndvapp",fill=TRUE)
   print(AdetaIndvapp)
}

#gammaIndv
cat("Prior Parms for gammaIndv: ",fill=TRUE)
cat("nugammaIndv =",nugammaIndv,fill=TRUE)
cat("V gammaIndv",fill=TRUE)
print(VgammaIndv)
cat("mubar gammaIndv",fill=TRUE)
print(mubargammaIndv)
cat("Amu gammaIndv", fill=TRUE)
print(AmugammaIndv)
cat("a gammaIndv",fill=TRUE)
print(agammaIndv)
if(drawdeltagammaIndv)
{
   cat("deltabargammaIndv",fill=TRUE)
   print(deltabargammaIndv)
   cat("AdgammaIndv",fill=TRUE)
   print(AdgammaIndv)
}

cat(" ",fill=TRUE)
```

```
cat("MCMC Parms: ",fill=TRUE)
cat(paste("s=",round(s,3)," w= ",w," R= ",R," keep= ",keep),fill=TRUE)
cat("",fill=TRUE)


#-------------------------------------------------------------------
# allocate space for draws
#-------------------------------------------------------------------
#alphai
if(drawdeltaalphai) Deltadrawalphai=matrix(double((floor((R-burnIn)/keep))*nzalphai*nvaralphai),
ncol=nzalphai*nvaralphai)
alphaidraw=array(double((floor((R-burnIn)/keep))*nlgt*nvaralphai),dim=c(nlgt,nvaralphai,floor((R
-burnIn)/keep)))
probdrawalphai=matrix(double((floor((R-burnIn)/keep))*ncomIndv),ncol=ncomIndv)
oldalphai=matrix(double(nlgt*nvaralphai),ncol=nvaralphai)
oldllcat=double(nlgt)
loglikecat=double(floor((R-burnIn)/keep))
oldcomalphai=NULL
compdrawalphai=NULL

#betai
if(drawdeltabetai) Deltadrawbetai=matrix(double((floor((R-burnIn)/keep))*nzbetai*nvarbetai),ncol
=nzbetai*nvarbetai)
betaidraw=array(double((floor((R-burnIn)/keep))*nlgt*nvarbetai),dim=c(nlgt,nvarbetai,floor((R-
burnIn)/keep)))
probdrawbetai=matrix(double((floor((R-burnIn)/keep))*ncomIndv),ncol=ncomIndv)
oldbetai=matrix(double(nlgt*nvarbetai),ncol=nvarbetai)
oldllapp=double(nlgt)
loglikeapp=double(floor((R-burnIn)/keep))
oldcombetai=NULL
compdrawbetai=NULL

#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

#thetaappa
if(drawdeltathetaappa) Deltadrawthetaappa=matrix(double((floor((R-burnIn)/keep))*nzthetaappa*
nvarthetaappa),ncol=nzthetaappa*nvarthetaappa)
thetaappadraw=array(double((floor((R-burnIn)/keep))*napps*nvarthetaappa),dim=c(napps,
nvarthetaappa,floor((R-burnIn)/keep)))
probdrawthetaappa=matrix(double((floor((R-burnIn)/keep))*ncomapp),ncol=ncomapp)
oldcomthetaappa=NULL
compdrawthetaappa=NULL

#etaIndvapp
if(drawdeltaetaIndvapp) DeltadrawetaIndvapp=matrix(double((floor((R-burnIn)/keep))*nzetaIndvapp*
nvaretaIndvapp),ncol=nzetaIndvapp*nvaretaIndvapp)
etaIndvappdraw=array(double((floor((R-burnIn)/keep))*nlgt*nvaretaIndvapp),dim=c(nlgt,
```

```
nvaretaIndvapp,floor((R-burnIn)/keep)))
probdrawetaIndvapp=matrix(double((floor((R-burnIn)/keep))*ncomIndv),ncol=ncomIndv)
oldcometaIndvapp=NULL
compdrawetaIndvapp=NULL

#gammaIndv
if(drawdeltagammaIndv) DeltadrawgammaIndv=matrix(double((floor((R-burnIn)/keep))*nzgammaIndv*
nvargammaIndv),ncol=nzgammaIndv*nvargammaIndv)
gammaIndvdraw=array(double((floor((R-burnIn)/keep))*nlgt*nvargammaIndv),dim=c(nlgt,nvargammaIndv
,floor((R-burnIn)/keep)))
probdrawgammaIndv=matrix(double((floor((R-burnIn)/keep))*ncomIndv),ncol=ncomIndv)
oldcomgammaIndv=NULL
compdrawgammaIndv=NULL

#===================================================================================================
==
##  create functions of mixture and likelihoods
#===================================================================================================
==
# as given the choice of category probability of making choice of category and making choice of
app is independent
# I do not need to do a lot in here as I run this function seperately for choice of app and for
choice of category
# as conditioning has already been done in my data
# log likelihood of multinomial logit, using weighting scheme
llmnlFract=
function(beta,y,X,betapooled,rootH,w,wgt){
   z=as.vector(rootH%*%(beta-betapooled))
   return((1-w)*llmnl(beta,y,X)+w*wgt*(-.5*(z%*%z)))
}

# new likelihood to find the mode of misperception of individual about the latent diffusion
# to test latentit = latentitold; beta = betadraw;
# to test: latentit = latentitnew;

#to test for category
#latentit=latentitold
#X=X
#y=y
#beta=betadraw
#priorLatentMean = priorLatentMean
#priorLatentVar = priorLatentVar
#distortionIndv = distortionIndv

llmnllatent=
function(latentit,y,X,beta,priorLatentMean,priorLatentVar,distortionIndv){
   #------------------------------------------------------------
   # Purpose:evaluate log-like for MNL with mispercepted latent
   #------------------------------------------------------------
   # Arguments:
   #   y   :  n vector with element = 1,...,j indicating which alt chosen
   #   X   : nj x k matrix of xvalues for each of j alt on each of n occasions
   #   beta: k vector of coefs
   #   priorLatentMean : the correct latent state of diffusion as prior
```

```
   #   priorLatentVar  : misperception variance for consumers
   #   distortionIndv  : scale factor for the misperception of consumers
   # Output: value of loglike

   # first replace latentit inside X
   latenttemp = matrix(latentit,byrow=T,nrow=length(y))   # to change J*T to J x T
   tempcol    = ncol(latenttemp)                        # it will be J for category and 1 for app
   temprow    = nrow(latenttemp)                        # it will return T
   latenttemp = cbind(c(rep(0,length(y))),latenttemp)     # to add no purchase's zero latent (T
   times)
   latenttemp = as.vector(t(latenttemp))

   # se the new X to calculate likelihood
   X[,tempcol+2] = latenttemp                           # 10+2 = 12 for cat and 1+2 =3 for
   apps


   prior        = -0.5 * sum(log(2*priorLatentVar*pi))-0.5*
      sum(((latentit - priorLatentMean)*(latentit - priorLatentMean))/priorLatentVar);

   # calculate likelihood of MNL
   n=length(y)
   j=nrow(X)/n
   Xbeta=X%*%beta
   Xbeta=matrix(Xbeta,byrow=T,ncol=j)
   ind=cbind(c(1:n),y)
   xby=Xbeta[ind]
   Xbeta=exp(Xbeta)
   iota=c(rep(1,j))
   denom=log(Xbeta%*%iota)
   denom[is.infinite(denom)==TRUE]=1e305
   return(-sum(xby-denom)-sum(prior))
}
#----------------------------------------------------------------
# Calculate expected  hessian for independent metrapolist chain
#----------------------------------------------------------------
# to test latentit = latentitold; beta = betadraw
mnlHesslatent =
function(latentit,y,X,beta,priorLatentMean,priorLatentVar,distortionIndv)
{
   # Purpose: compute mnl -Expected[Hessian] for latent misperception
   #
   # Arguments:
   #   beta is k vector of coefs
   #   y is n vector with element = 1,...,j indicating which alt chosen
   #   X is nj x k matrix of xvalues for each of j alt on each of n occasions
   #
   # Output:  -Hess evaluated at beta
   # first replace latentit inside X
   latenttemp = matrix(latentit,byrow=T,nrow=length(y))   # to change J*T to J x T
   tempcol    = ncol(latenttemp)                          # it will be J for category and 1 for
   app
   temprow    = nrow(latenttemp)                          # it will return T
   if (tempcol==0){
```

```
        tempcol=1
  }
  latenttemp = cbind(c(rep(0,length(y))),latenttemp)     # to add no purchase's zero latent (T
  times)
  latenttemp = as.vector(t(latenttemp))

  # se the new X to calculate likelihood
  X[,tempcol+2] = latenttemp                           # 10+2 = 12 for cat and 1+2 =3 for apps

  # calculate likelihood of MNL
  n=length(y)
  j=nrow(X)/n
  Xbeta=X%*%beta
  Xbeta=matrix(Xbeta,byrow=T,ncol=j)
  Xbetatemp = Xbeta;
  ind=cbind(c(1:n),y)
  xby=Xbeta[ind]
  Xbeta=exp(Xbeta)
  iota=c(rep(1,j))
  denom=log(Xbeta%*%iota)

  k     = length(latentit)
  Prob  = exp(Xbetatemp-as.vector(denom))
  Prob  = Prob[,-1]              # remove the first row which is irrelevant
  betatemp =  beta[tempcol+2]    # it will include the coefficient of the latent 10+2 = 12 for
  cat and 1+2=3 for apps
  Xtemp    = diag(rep(betatemp,tempcol*temprow)) # create a matrix jT*jT
  Probtemp     = matrix(c(rep(0,temprow*temprow*tempcol)),ncol=temprow*tempcol)
  Prob =as.matrix(Prob,ncol=tempcol-1)
  for (t in 1:temprow){
     Probtemp[t,((t-1)*tempcol+1):(t*tempcol)]=Prob[t,]
  }

  Hess= matrix(double(k*k),ncol=k)
  j   = tempcol
  for (i in 1:n) {
     Xt=Xtemp[(j*(i-1)+1):(j*i),]
     if (length(Xt)/j==length(Xt)){
        Xt=t(as.matrix(Xt,nrow=j))
        p=as.vector(Prob[i,])
        A = matrix(p - p*p,ncol=1)
     }else{
        p=as.vector(Prob[i,])
        A=diag(p)-outer(p,p)
     }

     Hess=Hess+crossprod(Xt,A)%*%Xt
  }
  return(Hess)
}

#-------------------------------------------------------------------------------------------------
----
#draw non-state and misperception of latent state
```

```
# the function works for both category and app as the structure is the same regarding state
# First it draws from RW-MH the non-state parameters for each individual
# Second conditional on non-state parameters it draws misperception of latent state for
individual
# The reason for conditioning is to account for
#-------------------------------------------------------------------------------------------------
----
# to test for the mobile app
# y=lgtdataa[[lgt]]$y
# X=lgtdataa[[lgt]]$X
# oldbeta=oldbetai[lgt,]
# oldll=oldllapp[lgt]
# inc.root=inc.rootapp
# betabar=betabarbetai
# rootpi=rootpibetai
# pc=pcapp[lgt]
# latentitold=indvPerceptTemp
# cumj=cumjapp[lgt]
# priorLatentMean=latentapprealTemp
# priorLatentVar=vappmispercept[lgtdatac[[lgt]]$y-1]
# distortionIndv=Fapp[lgt,]
# iter=iterrep

#test for category
#y=lgtdatac[[lgt]]$y
#X=lgtdatac[[lgt]]$X
#oldbeta=oldalphai[lgt,]
#oldll=oldllcat[lgt]
#inc.root=inc.rootcat
#betabar=betabaralphai
#rootpi=rootpialphai
#pc =pccat[lgt]
#latentitold= as.vector(indvperceptioncat[lgt,,])
#cumj=cumjcat[lgt]
#priorLatentMean=as.vector(cbind(tt0cat,tt1cat[,-T]))
#priorLatentVar=rep(vcatmispercept,T)
#distortionIndv=Fcat[lgt,]
#iter=iterrep

mnlRwMetropOnce=
function(y,X,oldbeta,oldll,s,inc.root,betabar,rootpi,
         pc,latentitold,cumj,priorLatentMean,priorLatentVar,distortionIndv,iter){
   #
   # function to execute RW-MH (metrapolist hasting) for the MNL
   #     and MH-M (around the mode)  for the misperception of latent variable

   # y                 :  n vector with element = 1,...,j indicating which alt chosen
   (alternative could either be category or app)
   # X                 :  nj x k matrix of xvalues for each of j alt on each of n occasions
   # RW increments     :  N(0,s^2*t(inc.root)%*%inc.root)
   # prior on beta     :  N(betabar,Sigma)  Sigma^-1=rootpi*t(rootpi)
   # inc.root, rootpi :  upper triangular (this means that we are using the UL decomp of
   Sigma^-1 for prior)
   # oldbeta           :  the last beta (current) which has wone MH-RW MNL
```

```
   # oldll            :  old likelihood of winner of MH-RW MNL
   # betabar          :  mean of the sensitivities

   #----------------------------------------------------------------
   # list of new parameters added:
   # pc               : size of increment of metrapolist around the mode (updated automatically 
   and returned)
   # latentitold      : the last (current) misperception about the latent MH-M (around the mode)
   # cmuj             : for the total acceptance rate
   # priorLatentMean  : is true current state of diffusion of an app or a category
   # priorLatentVar   : is the variance of the misperception across all individuals
   # distortionIndv   : is the coefficient of diffusion distortion by individual
   # iter             : is the iteration number in MCMC
   #----------------------------------------------------------------

   #---------------------------------------------------------------
   # First step in drawing from M-HRW
   #---------------------------------------------------------------
   stay  =0
   betac =oldbeta + s*t(inc.root)%*%(matrix(rnorm(ncol(X)),ncol=1))
   cll   =llmnl(betac,y,X)
   clpost=cll+lndMvn(betac,betabar,rootpi)
   ldiff =clpost-oldll-lndMvn(oldbeta,betabar,rootpi)
   alpha =min(1,exp(ldiff))
   if(alpha < 1) {
      unif=runif(1)
   } else {
      unif=0
   }
   if (unif <= alpha){
      betadraw=betac; oldll=cll
   }else{
      betadraw=oldbeta; stay=1
   }
   #---------------------------------------------------------------
   # First step in drawing from M-HRW
   #---------------------------------------------------------------
   T             = length(y)
   alpha         = 0.5
   rndacceptance = 0.8
   #---------------------
   # to test
   #---------------------
#    latentitold     =  as.vector(catIndvlatent[1,,])
#    y               =  simCatChoice[[1]]$y
#    X               =  simCatChoice[[1]]$X
#    betadraw            =  simCatChoice[[1]]$beta
#    priorLatentMean =  as.vector(catlatent)
#    priorLatentVar  =  rep(vIndvcat,ncat)
#    distortionIndv  =  gammaIndv[1,1]
#    perturbation    =  runif(10*(length(lower)))
#    latentitoldpop  =  matrix(perturbation%x%latentitold,ncol=length(lower))
#end test
   # test whether function itself works
```

```
#    llmnllatent(latentitold,X=X,y=y,beta=betadraw,priorLatentMean = priorLatentMean,
#                priorLatentVar = priorLatentVar,distortionIndv = distortionIndv)
   # Constraints
   upper           =  rep(1000,length(latentitold))
   lower           =  rep(-1000,length(latentitold))
   bounds <- cbind(lower,upper)
   colnames(bounds) <- c("lower", "upper")
   # Convert the constraints to the ui and ci matrices
   n <- nrow(bounds)
   ui <- rbind( diag(n), -diag(n) )
   ci <- c( bounds[,1], - bounds[,2] )
   starttime = proc.time()[3]
   out             =  optim(latentitold,llmnllatent,method="SANN",control=list( fnscale=-1,trace
   =1,reltol=1e-4,maxit=1000), #hessian = TRUE
                                    ,X=X,y=y,beta=betadraw,priorLatentMean = priorLatentMean,
                                    priorLatentVar = priorLatentVar,distortionIndv =
                                    distortionIndv)
   endtime = proc.time()[3]
   durationtime= endtime-starttime
   modelatent = out$par
   latenthess  = mnlHesslatent(modelatent,X=X,y=y,beta=betadraw,priorLatentMean =
   priorLatentMean,
                   priorLatentVar = priorLatentVar,distortionIndv = distortionIndv)
   if (is.positive.definite(latenthess)){
      latentvar  = diag(chol2inv(chol(latenthess)))
   }else{
      latentvar  = diag(ginv(make.positive.definite(latenthess)))
   }

   latentvar  = pmax(latentvar,c(rep(1e-6,length(latentvar))))

   # to test
   #pc = 1e-8
   #end test
   j = 1
   cat("M-H loop to find the appropriate parameters")
   while (alpha < rndacceptance){
      j              = j+1
      cat(j,",")
      wpm            = pc*latentvar
      wpm            = pmax(wpm,c(rep(1e-6,length(wpm))))
      latentitnew    = latentitold + wpm*rnorm(length(latentitold))
      postPlatentNew = -llmnllatent(latentitnew,X=X,y=y,beta=betadraw,priorLatentMean =
      priorLatentMean,
                                    priorLatentVar = priorLatentVar,distortionIndv =
                                    distortionIndv)
      postPlatentOld = -llmnllatent(latentitold,X=X,y=y,beta=betadraw,priorLatentMean =
      priorLatentMean,
                                    priorLatentVar = priorLatentVar,distortionIndv =
                                    distortionIndv)
      alpha          = postPlatentNew - postPlatentOld
      rndacceptance  = log(runif(1))
      if (j > 100){
         pc = pc/10;
```

```
         break;
      }
   }
   cumj = cumj + j          # to keep cumulative value
   accptrate = iter/cumj    # acceptance rate until now
   if (floor (iter/5) == iter/5){
      if (accptrate > 0.15){
         pc = pc*3;
         cumj = iter/0.15
      }else{
         if(accptrate < 0.01){
            pc = pc/3
            cumj = iter/0.01
         }
      }
   }
   postPlatentOld = postPlatentNew
   latentitold    = latentitnew

# following was very slow, so I used the analytical form
#    latentvar  = fdHess(modelatent,llmnllatent,X=X,y=y,beta=betadraw,priorLatentMean =
priorLatentMean,
#                         priorLatentVar = priorLatentVar,distortionIndv = distortionIndv)
#
# following are list of functions that I tried but did not work
#       constrOptim(latentitold,llmnllatent,grad=NULL,ci=ci, ui=ui,# method="BFGS"
#                                method="Nelder-Mead",control=list(
fnscale=-1,trace=0,reltol=1e-6,maxit=2),# hessian = TRUE
#                               X=X,y=y,beta=betadraw,priorLatentMean = priorLatentMean,
#                                priorLatentVar = priorLatentVar,distortionIndv = distortionIndv)


#       DEoptim(llmnllatent, lower, upper #initialpop=latentitoldpop, # irecieved wiered error
Error in match.fun(FUN) : '1'
#                              ,DEoptim.control( reltol=1e-6,
itermax=2),
#                                     X=X,y=y,beta=betadraw,priorLatentMean = priorLatentMean,
#                                     priorLatentVar = priorLatentVar,distortionIndv =
distortionIndv)
#

#       nlm(llmnllatent,latentitold, hessian = TRUE, print.level = 0,steptol=1e-6,
#                           gradtol=1e-6, X=X,y=y,beta=betadraw,priorLatentMean =
priorLatentMean,
#                              priorLatentVar = priorLatentVar,distortionIndv = distortionIndv)

#       optim(latentitold,llmnllatent,method="BFGS",control=list(
fnscale=-1,trace=0,reltol=1e-6),
#                           hessian = TRUE,X=X,y=y,beta=betadraw,priorLatentMean =
priorLatentMean,
#                           priorLatentVar = priorLatentVar,distortionIndv = distortionIndv)

   return(list(betadraw=betadraw,stay=stay,oldll=oldll,cumj = cumj,latentitold=latentitold,pc=pc
   ))
```

```
   # new return items:
   # pc         : for the size of the increment
   # latentitold: the last (current) misperception about the latent MH-M (around the mode)
   # cumj       : for the total acceptance rate

}

#-----------------------------------------------------------
#           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 and 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",c(1:(ncat-1)))

if (i ==1){
   predictedcat[1]=tt0cat[i,1]+
      (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/thetacatjest[i,"Cj"])+
          thetacatjest[i,(i+3):(ncat+2)]%*%tt0cat[(i+1):ncat,1])*
      (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,4:(i+2)]%*%tt0cat[1:(i-1),1]+
             thetacatjest[i,(i+3):(ncat+2)]%*%tt0cat[(i+1):ncat,1])*
         (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,4:(i+2)]%*%tt0cat[1:(i-1),1])*
         (thetacatjest[i,"Cj"]-tt0cat[i,1]);
   }
}

predictedcat[2:T]=foreach (t=2:T,.combine=cbind)%dopar%{
   if (i ==1){
      predictedcattemp=tt1cat[i,t]+
         (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/thetacatjest[i,"Cj"])+
             thetacatjest[i,(i+3):(ncat+2)]%*%tt1cat[(i+1):ncat,t])*
         (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,4:(i+2)]%*%tt1cat[1:(i-1),t]+
                thetacatjest[i,(i+3):(ncat+2)]%*%tt1cat[(i+1):ncat,t])*
            (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,4:(i+2)]%*%tt1cat[1:(i-1),t])*
            (thetacatjest[i,"Cj"]-tt1cat[i,t]);
      }
   }

   return(predictedcattemp)
}
errortempcat = tt1cat[i,]-predictedcat
return (errortempcat)
```

```
}

basslikelihoodCat= function(thetacatjestcur,thetacatjest,tt0cat,tt1cat,wcat,oldcompthetacatj,
indthetacatj,curcat,meansuntilcur){
   #meansuntilcur is an array but I will use until curcat of it
   ncat =  dim(tt1cat)[1]
   T    =  dim(tt1cat)[2]
   thetacatjest[curcat,]=thetacatjestcur
   if (curcat == 1){
      thetacatjest=matrix(thetacatjest,nrow=ncat)
      errortempcat = t(bassErrorsCat(thetacatjest,tt0cat,tt1cat,wcat,curcat))
      wcatconditional = abs(wcat[curcat,curcat])
      LLcat        = -0.5*sum(crossprod(errortempcat,errortempcat)/sqrt(wcatconditional))-0.5*T*
      log(2*pi)-
         0.5*T*wcatconditional
   }else{
      # to test:
      # meansuntilcur=list(thetacatjest[1,],thetacatjest[2,])
    #  delta21 = wcat[curcat,(1:(curcat-1))]%*%wcat[(1:(curcat-1)),(1:(curcat-1))]
      thetacatjconditional = thetacatjest
    #  thetacatjconditional[curcat,] = thetacatjest[curcat,] -
    delta21%*%meansuntilcur[(1:(curcat-1)),]
      errortempcat = t(bassErrorsCat(thetacatjconditional,tt0cat,tt1cat,wcat,curcat))
      wcatconditional = abs(wcat[curcat,curcat] - wcat[curcat,(1:(curcat-1))]%*%wcat[(1:(curcat-
      1)),(1:(curcat-1))]%*%
         wcat[(1:(curcat-1)),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]]]$rooti
   betabarthetacatj= oldcompthetacatj[[indthetacatj[clt]]]$mu
   LLpriorcat = lndMvn(thetacatjest[clt,],betabarthetacatj,rootpithetacatj)
   return(sum(LLpriorcat)+LLcat)
}

#Hessian
bassHessianCat= function(thetacatjest,tt0cat,tt1cat,wcat,curcat){
   ncat = dim(tt1cat)[1]
   T    = dim(tt1cat)[2]
   heessianElement = array(rep(0,(T*(ncat+2)*(ncat+2))),dim=c(T,(ncat+2),(ncat+2)))
   i = curcat

   thetacatjest =matrix(as.numeric(thetacatjest),nrow=ncat)
   colnames(thetacatjest) <- c("p","q","Cj",c(1:(ncat-1)))

   if (i ==1){
      e1primee2primetemp = matrix(c(-(thetacatjest[i,"Cj"]-tt0cat[i,1]),-(tt0cat[i,1]/
      thetacatjest[i,"Cj"])*
                                  (thetacatjest[i,"Cj"]-tt0cat[i,1]),(thetacatjest[i,"Cj"]-tt0cat[i
```

```
                              ,1])*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[i,"Cj"])**2)-
          (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/thetacatjest[i,"Cj"])+
              thetacatjest[i,(i+3):(ncat+2)]%*%tt0cat[(i+1):ncat,1]),-tt0cat[(i+1):ncat,1]*(
              thetacatjest[i,"Cj"]-tt0cat[i,1])),
          nrow=1)
    e12zegondetemp = matrix(rep(0,(ncat+2)*(ncat+2)),ncol=(ncat+2))
    e12zegondetemp[1,3]=-1
    e12zegondetemp[2,3]=-((tt0cat[i,1]**2)/(thetacatjest[i,"Cj"])**2)
    e12zegondetemp[3,3]=-2*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[i,"Cj"]**3))*
       (thetacatjest[i,"Cj"]-tt0cat[i,1])+2*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[i,
       "Cj"]**2))
    e12zegondetemp[3,4:(2+ncat)]=tt0cat[(i+1):ncat,1]
    e12zegondetemp[lower.tri(e12zegondetemp)] = e12zegondetemp[upper.tri(e12zegondetemp)]
    heessianElement[1,,]=c(tt0cat[i,1]+
       (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/thetacatjest[i,"Cj"])+
           thetacatjest[i,(i+3):(ncat+2)]%*%tt0cat[(i+1):ncat,1])*
       (thetacatjest[i,"Cj"]-tt0cat[i,1]))*e12zegondetemp+crossprod(e1primee2primetemp,
       e1primee2primetemp)
  }else{
    # treat element in the middle
    if (i>1 && i<ncat){
       e1primee2primetemp = matrix(c(-(thetacatjest[i,"Cj"]-tt0cat[i,1]),-(tt0cat[i,1]/
       thetacatjest[i,"Cj"])*
                                        (thetacatjest[i,"Cj"]-tt0cat[i,1]),(thetacatjest[i,
                                        "Cj"]-tt0cat[i,1])*thetacatjest[i,"q"]*(tt0cat[i,1]/(
                                        thetacatjest[i,"Cj"])**2)-
                                        (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/
                                        thetacatjest[i,"Cj"])+
                                            thetacatjest[i,4:(i+2)]%*%tt0cat[1:(i-1),1]+
                                            thetacatjest[i,(i+3):(ncat+2)]%*%tt0cat[(i+1):ncat
                                            ,1]),
                                     -c(tt0cat[1:(i-1),1],tt0cat[(i+1):ncat,1])*(thetacatjest[
                                     i,"Cj"]-tt0cat[i,1])),
                                   nrow=1)
       e12zegondetemp = matrix(rep(0,(ncat+2)*(ncat+2)),ncol=(ncat+2))
       e12zegondetemp[1,3]=-1
       e12zegondetemp[2,3]=-((tt0cat[i,1]**2)/(thetacatjest[i,"Cj"])**2)
       e12zegondetemp[3,3]=-2*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[i,"Cj"]**3))*
          (thetacatjest[i,"Cj"]-tt0cat[i,1])+2*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[
          i,"Cj"]**2))
       e12zegondetemp[3,4:(2+ncat)]=c(tt0cat[1:(i-1),1],tt0cat[(i+1):ncat,1])
       e12zegondetemp[lower.tri(e12zegondetemp)] = e12zegondetemp[upper.tri(e12zegondetemp)]
       heessianElement[1,,]=c(tt0cat[i,1]+
                                 (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/
                                 thetacatjest[i,"Cj"])+
                                     thetacatjest[i,4:(i+2)]%*%tt0cat[1:(i-1),1]+
                                     thetacatjest[i,(i+3):(ncat+2)]%*%tt0cat[(i+1):ncat,1])*
                                 (thetacatjest[i,"Cj"]-tt0cat[i,1]))*e12zegondetemp+crossprod(
                                 e1primee2primetemp,e1primee2primetemp)
    }else{
       e1primee2primetemp = matrix(c(-(thetacatjest[i,"Cj"]-tt0cat[i,1]),-(tt0cat[i,1]/
       thetacatjest[i,"Cj"])*
                                        (thetacatjest[i,"Cj"]-tt0cat[i,1]),(thetacatjest[i,
                                        "Cj"]-tt0cat[i,1])*thetacatjest[i,"q"]*(tt0cat[i,1]/(
```

```
                                                thetacatjest[i,"Cj"])**2)-
                                                (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/
                                                thetacatjest[i,"Cj"])+
                                                    thetacatjest[i,4:(i+2)]%*%tt0cat[1:(i-1),1]),
                                             -tt0cat[1:(i-1),1]*(thetacatjest[i,"Cj"]-tt0cat[i,1])),
                                           nrow=1)
          e12zegondetemp = matrix(rep(0,(ncat+2)*(ncat+2)),ncol=(ncat+2))
          e12zegondetemp[1,3]=-1
          e12zegondetemp[2,3]=-((tt0cat[i,1]**2)/(thetacatjest[i,"Cj"])**2)
          e12zegondetemp[3,3]=-2*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[i,"Cj"]**3))*
             (thetacatjest[i,"Cj"]-tt0cat[i,1])+2*thetacatjest[i,"q"]*(tt0cat[i,1]/(thetacatjest[
             i,"Cj"]**2))
          e12zegondetemp[3,4:(2+ncat)]=tt0cat[1:(i-1),1]
          e12zegondetemp[lower.tri(e12zegondetemp)] = e12zegondetemp[upper.tri(e12zegondetemp)]
          heessianElement[1,,]=c(tt0cat[i,1]+
                                         (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt0cat[i,1]/
                                         thetacatjest[i,"Cj"])+
                                             thetacatjest[i,4:(i+2)]%*%tt0cat[1:(i-1),1])*
                                         (thetacatjest[i,"Cj"]-tt0cat[i,1]))*e12zegondetemp+crossprod(
                                         e1primee2primetemp,e1primee2primetemp)
       }
    }

    heessianElement[2:T,,]=foreach (t=2:T,.combine=cbind)%dopar%{
       if (i ==1){
          e1primee2primetemp = matrix(c(-(thetacatjest[i,"Cj"]-tt1cat[i,t]),-(tt1cat[i,t]/
          thetacatjest[i,"Cj"])*
                                                (thetacatjest[i,"Cj"]-tt1cat[i,t]),(thetacatjest[i,
                                                "Cj"]-tt1cat[i,t])*
                                                thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[i,"Cj"
                                                ])**2)-
                                                (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/
                                                thetacatjest[i,"Cj"])+
                                                    thetacatjest[i,(i+3):(ncat+2)]%*%tt1cat[(i+1):ncat
                                                    ,t]),
                                             tt1cat[(i+1):ncat,t]* (thetacatjest[i,"Cj"]-tt1cat[i,t])),
                                           nrow=1)
          e12zegondetemp = matrix(rep(0,(ncat+2)*(ncat+2)),ncol=(ncat+2))
          e12zegondetemp[1,3]=-1
          e12zegondetemp[2,3]=-((tt1cat[i,t]**2)/(thetacatjest[i,"Cj"])**2)
          e12zegondetemp[3,3]=-2*thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[i,"Cj"]**3))*
             (thetacatjest[i,"Cj"]-tt1cat[i,t])+2*thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[
             i,"Cj"]**2))
          e12zegondetemp[3,4:(2+ncat)]=tt1cat[(i+1):ncat,t]
          e12zegondetemp[lower.tri(e12zegondetemp)] = e12zegondetemp[upper.tri(e12zegondetemp)]
          heessianElementtemp=c(tt1cat[i,t]+
                                        (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/
                                        thetacatjest[i,"Cj"])+
                                            thetacatjest[i,(i+3):(ncat+2)]%*%tt1cat[(i+1):ncat,t])*
                                        (thetacatjest[i,"Cj"]-tt1cat[i,t]))*e12zegondetemp+crossprod(
                                        e1primee2primetemp,e1primee2primetemp)
       }else{
          # treat element in the middle
          if (i>1 && i<ncat){
```

```
        e1primee2primetemp = matrix(c(-(thetacatjest[i,"Cj"]-tt1cat[i,t]),-(tt1cat[i,t]/
        thetacatjest[i,"Cj"])*
                                          (thetacatjest[i,"Cj"]-tt1cat[i,t]),(thetacatjest[i,
                                          "Cj"]-tt1cat[i,t])*
                                          thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[i,
                                          "Cj"])**2)-
                                          (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,
                                          t]/thetacatjest[i,"Cj"])+
                                              thetacatjest[i,4:(i+2)]%*%tt1cat[1:(i-1),t]+
                                              thetacatjest[i,(i+3):(ncat+2)]%*%tt1cat[(i+1):
                                              ncat,t]),
                                       c(tt1cat[1:(i-1),t],tt1cat[(i+1):ncat,t])* (
                                       thetacatjest[i,"Cj"]-tt1cat[i,t])),
                                     nrow=1)
        e12zegondetemp = matrix(rep(0,(ncat+2)*(ncat+2)),ncol=(ncat+2))
        e12zegondetemp[1,3]=-1
        e12zegondetemp[2,3]=-((tt1cat[i,t]**2)/(thetacatjest[i,"Cj"])**2)
        e12zegondetemp[3,3]=-2*thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[i,"Cj"]**3))*
           (thetacatjest[i,"Cj"]-tt1cat[i,t])+2*thetacatjest[i,"q"]*(tt1cat[i,t]/(
           thetacatjest[i,"Cj"]**2))
        e12zegondetemp[3,4:(2+ncat)]=c(tt1cat[1:(i-1),t],tt1cat[(i+1):ncat,t])
        e12zegondetemp[lower.tri(e12zegondetemp)] = e12zegondetemp[upper.tri(e12zegondetemp)]
        heessianElementtemp=c(tt1cat[i,t]+
                                 (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/
                                 thetacatjest[i,"Cj"])+
                                     thetacatjest[i,4:(i+2)]%*%tt1cat[1:(i-1),t]+
                                     thetacatjest[i,(i+3):(ncat+2)]%*%tt1cat[(i+1):ncat,t])*
                                 (thetacatjest[i,"Cj"]-tt1cat[i,t]))*e12zegondetemp+
                                 crossprod(e1primee2primetemp,e1primee2primetemp)

     }else{

        e1primee2primetemp = matrix(c(-(thetacatjest[i,"Cj"]-tt1cat[i,t]),-(tt1cat[i,t]/
        thetacatjest[i,"Cj"])*
                                          (thetacatjest[i,"Cj"]-tt1cat[i,t]),(thetacatjest[i,
                                          "Cj"]-tt1cat[i,t])*
                                          thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[i,
                                          "Cj"])**2)-
                                          (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,
                                          t]/thetacatjest[i,"Cj"])+
                                              thetacatjest[i,4:(i+2)]%*%tt1cat[1:(i-1),t]),
                                       tt1cat[1:(i-1),t]* (thetacatjest[i,"Cj"]-tt1cat[i,t])),
                                     nrow=1)
        e12zegondetemp = matrix(rep(0,(ncat+2)*(ncat+2)),ncol=(ncat+2))
        e12zegondetemp[1,3]=-1
        e12zegondetemp[2,3]=-((tt1cat[i,t]**2)/(thetacatjest[i,"Cj"])**2)
        e12zegondetemp[3,3]=-2*thetacatjest[i,"q"]*(tt1cat[i,t]/(thetacatjest[i,"Cj"]**3))*
           (thetacatjest[i,"Cj"]-tt1cat[i,t])+2*thetacatjest[i,"q"]*(tt1cat[i,t]/(
           thetacatjest[i,"Cj"]**2))
        e12zegondetemp[3,4:(2+ncat)]=tt1cat[1:(i-1),t]
        e12zegondetemp[lower.tri(e12zegondetemp)] = e12zegondetemp[upper.tri(e12zegondetemp)]
        heessianElementtemp=c(tt1cat[i,t]+
                                 (thetacatjest[i,"p"]+thetacatjest[i,"q"]*(tt1cat[i,t]/
                                 thetacatjest[i,"Cj"])+
```

```
                                        thetacatjest[i,4:(i+2)]%*%tt1cat[1:(i-1),t])*
                                       (thetacatjest[i,"Cj"]-tt1cat[i,t]))*e12zegondetemp+
                                       crossprod(e1primee2primetemp,e1primee2primetemp)
         }
      }

      return(heessianElementtemp)
   }

   return (colMeans(heessianElement,dims=1))
}




#-------------------------
# Apps
#-------------------------
bassErrorsApp= function(thetaappaest,tt0app,tt1app,wapp,curapp,tt1cat){
   napp = dim(tt1app)[1]
   T    = dim(tt1app)[2]
   predictedapp = matrix(rep(0,(T)),ncol=T)
   a = curapp

   thetaappaest =matrix(as.numeric(thetaappaest),nrow=napps)
   colnames(thetaappaest) <- c("p","q","alphaj")


   # treat first eelement differently
   predictedapp[1]=tt0app[a]+
      (thetaappaest[a,"p"]+thetaappaest[a,"q"]*(tt0app[a]/(thetaappaest[a,"alphaj"]* tt1cat[a,1
      ])))*
      ((thetaappaest[a,"alphaj"]* tt1cat[a,1])-tt0app[a])

   predictedapp[2:T]=foreach (t=2:T,.combine=cbind)%dopar%{
      predictedapptemp = tt1app[a,t]+
         (thetaappaest[a,"p"]+thetaappaest[a,"q"]*(tt1app[a,t]/(thetaappaest[a,"alphaj"]* tt1cat
         [a,t])))*
         ((thetaappaest[a,"alphaj"]* tt1cat[a,t])-tt1app[a,t])
      return(predictedapptemp)
   }
   errortempapp = tt1app[a,]-predictedapp
   return (errortempapp)
}

basslikelihoodApp= function(thetaappaestcur,thetaappaest,tt0app,tt1app,wapp,oldcompthetaappa,
indthetaappa,curapp,meansuntilapp,tt1cat){
   #meansuntilcur is an array but I will use until curcat of it
   napp = dim(tt1app)[1]
   T    = dim(tt1app)[2]
   thetaappaest[curapp,]=thetaappaestcur
   thetaappaest=matrix(thetaappaest,nrow=napp)
   errortempapp = t(bassErrorsApp(thetaappaest,tt0app,tt1app,wapp,curapp,tt1cat))
   wappconditional = abs(wapp[curapp])
```

```
   LLapp        = -0.5*sum(crossprod(errortempapp,errortempapp)/sqrt(wappconditional))-0.5*T*log
   (2*pi)-
      0.5*T*wappconditional

   # to test
   rootpithetaappa=oldcompthetaappa[[indthetaappa[curapp]]]$rooti
   betabarthetaappa= oldcompthetaappa[[indthetaappa[curapp]]]$mu
   LLpriorapp = lndMvn(thetaappa[curapp,],betabarthetaappa,rootpithetaappa)
   return(LLpriorapp+LLapp)
}

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

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

#alphai
#-------------------------
indalphai=NULL
nincalphai=floor(nlgt/ncomIndv)
for (i in 1:(ncomIndv-1)) {indalphai=c(indalphai,rep(i,nincalphai))}
if(ncomIndv != 1) {indalphai = c(indalphai,rep(ncomIndv,nlgt-length(indalphai)))} else {
indalphai=rep(1,nlgt)}
# initialize delta
if (drawdeltaalphai) olddeltaalphai=rep(0,nzalphai*nvaralphai)
# initialize probs
oldprobalphai=rep(1/ncomIndv,ncomIndv)
# initialize comps
tcompalphai=list(list(mu=rep(0,nvaralphai),rooti=diag(nvaralphai)))
oldcompalphai=rep(tcompalphai,ncomIndv)

#betai
#-------------------------
indbetai=NULL
nincbetai=floor(nlgt/ncomIndv)
for (i in 1:(ncomIndv-1)) {indbetai=c(indbetai,rep(i,nincbetai))}
if(ncomIndv != 1) {indbetai = c(indbetai,rep(ncomIndv,nlgt-length(indbetai)))} else {indbetai=
rep(1,nlgt)}
# initialize delta
if (drawdeltabetai) olddeltabetai=rep(0,nzbetai*nvarbetai)
# initialize probs
oldprobbetai=rep(1/ncomIndv,ncomIndv)
# initialize comps
tcompbetai=list(list(mu=rep(0,nvarbetai),rooti=diag(nvarbetai)))
oldcompbetai=rep(tcompbetai,ncomIndv)

#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)

#thetaappa
#-------------------------
indthetaappa=NULL
nincthetaappa=floor(napps/ncomapp)
for (i in 1:(ncomapp-1)) {indthetaappa=c(indthetaappa,rep(i,nincthetaappa))}
if(ncomapp != 1) {indthetaappa = c(indthetaappa,rep(ncomapp,napps-length(indthetaappa)))} else {
indthetaappa=rep(1,napps)}
# initialize delta
if (drawdeltathetaappa) olddeltathetaappa=rep(0,nzthetaappa*nvarthetaappa)
# initialize probs
oldprobthetaappa=rep(1/ncomapp,ncomapp)
# initialize comps
tcompthetaappa=list(list(mu=rep(0,nvarthetaappa),rooti=diag(nvarthetaappa)))
oldcompthetaappa=rep(tcompthetaappa,ncomapp)

#etaIndvapp
#-------------------------
indetaIndvapp=NULL
ninCetaIndvapp=floor(nIndv/ncomIndv)
for (i in 1:(ncomIndv-1)) {indetaIndvapp=c(indetaIndvapp,rep(i,nincetaIndvapp))}
if(ncomIndv != 1) {indetaIndvapp = c(indetaIndvapp,rep(ncomIndv,nIndv-length(indetaIndvapp)))}
else {indetaIndvapp=rep(1,nIndv)}
# initialize delta
if (drawdeltaetaIndvapp) olddeltaetaIndvapp=rep(0,nzetaIndvapp*nvaretaIndvapp)
# initialize probs
oldprobetaIndvapp=rep(1/ncomIndv,ncomIndv)
# initialize comps
tcompetaIndvapp=list(list(mu=rep(0,nvaretaIndvapp),rooti=diag(nvaretaIndvapp)))
oldcompetaIndvapp=rep(tcompetaIndvapp,ncomIndv)

#gammaIndv
#-------------------------
indgammaIndv=NULL
nincgammaIndv=floor(nIndv/ncomIndv)
for (i in 1:(ncomIndv-1)) {indgammaIndv=c(indgammaIndv,rep(i,nincgammaIndv))}
if(ncomIndv != 1) {indgammaIndv = c(indgammaIndv,rep(ncomIndv,nIndv-length(indgammaIndv)))} else
 {indgammaIndv=rep(1,nIndv)}
# initialize delta
if (drawdeltagammaIndv) olddeltagammaIndv=rep(0,nzgammaIndv*nvargammaIndv)
# initialize probs
```

```
oldprobgammaIndv=rep(1/ncomIndv,ncomIndv)
# initialize comps
tcompgammaIndv=list(list(mu=rep(0,nvargammaIndv),rooti=diag(nvargammaIndv)))
oldcompgammaIndv=rep(tcompgammaIndv,ncomIndv)

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

#app categories
#---------------------------------------
ycat = array(rep(0,nIndv*ncat*T), dim=c(nIndv,ncat,T))
pcat = ncat
Fcat = matrix(rep(1,nIndv),ncol=1)
thetacatjest = matrix(rep(0.01,ncat*(ncat+2)),ncol=ncat+2)
m0cat        = 0.01*matrix(c(rep(1,ncat)),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)))
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,nIndv*ncat*T),dim=c(nIndv,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 = (length(vcat)+length(wcat))*(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*nIndv*(R-burnIn)),dim=c(nIndv,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     = 100
Svivcat     = diag(rep(0.1,nIndv))


# mobile apps
#---------------------------------------
yapp = array(rep(0,nIndv*napps*T), dim=c(nIndv,napps,T))
papp = 1
Fapp  = matrix(rep(1,nIndv),ncol=1)
thetaappaest = matrix(rep(0.01,napps*3),ncol=3)
m0app        = 0.01*matrix(c(rep(1,napps)),ncol=napps)
C0app = 2*(c(rep(1,napps)))
vapp = matrix(rep(0,napps*nIndv*nIndv),ncol=nIndv)
for (j in 1:napps){
```

```
      vapp[(nIndv*(j-1)+1):(nIndv*j),] = 0.01*diag(nIndv)
}
wapp = 0.5*c(rep(1,napps))
tt0app = 0.01*matrix(c(rep(1,napp)),nrow=napp)
tt1app = 0.01*matrix(c(rep(1,napp*T)),ncol=T)
#Y1app  = array(rep(0,nIndv*napp*T),dim=c(nIndv,napp,T)) #memory explosion so it does not work
MADapp = matrix(rep(0,T*napp),ncol=napp)
MSEapp = matrix(rep(0,T*napp),ncol=napp)
# i did not save v because of its large size
crossseclengthapp = (length(vapp)+length(wapp))*(R-burnIn)
capp_=matrix(rep(0,crossseclengthapp),ncol=(R-burnIn))
bapp_ = matrix(rep(0,(length(tt0app)+length(tt1app))*(R-burnIn)),ncol=(R-burnIn))
llapp_ = rep(0,(R-burnIn))
Y1app_ = rep(0,(R-burnIn))
MADapp_ = rep(0,(R-burnIn))
MSEapp_ = rep(0,(R-burnIn))
b0appst  = rep(0, nvarthetaappa)
S0appstInv  = diag(rep(1.5e-7,nvarthetaappa))
b0appobs = 0
S0appobsInv = 1.5e-7
v0ivapp     = 100
Svivapp     = diag(rep(0.1,nIndv))

#-----------------------------------
# initial values for the pooled
#-----------------------------------
alphainit     = c(rep(0,nvaralphai))
betainit      = c(rep(0,nvarbetai))
oldthetacatj  = thetacatj
oldthetaappa  = thetaappa
oldetaIndvapp = etaIndvapp
oldgammaIndv  = gammaIndv

# mean of the coefficient (prior mean)
#-------------------
betabargammaIndv = rep(0, nIndv)
betabaretaIndvapp = rep(0, nIndv)
betabarthetacatj = matrix(rep(0, ncat*nvarthetacatj),ncol=nvarthetacatj)
betabarthetaappa = matrix(rep(0, napps*nvarthetaappa),ncol=nvarthetaappa)

#definition of metrapolist hasting and misperception parameters
#--------------------------------------
# category
#-------------
pccat = rep(1e-20,nIndv)
cumjcat = rep(1, nIndv)
vcatmispercept = rep(1,ncat)
indvperceptioncat = catIndvlatent

# app
#-------------
pcapp = rep(1e-20,nIndv)
cumjapp = rep(1, nIndv)
vappmispercept = rep(1,napps)
```

```
indvperceptionapp = abind(array(rep(0,nIndv*T),dim=c(nIndv,1,T)),appIndvlatent,along=2)

#variance of misperception for individuals
catvarmisperceptindv = rep(0, ncat)
appvarmisperceptindv = rep(0, napp)

# estimate non-state parameters of state equation
#-------------------------------------------------------
#cat
#---------------------
pcThetacatj = 1e-20
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))

#app
#----------------------
pcThetaappa = 1e-20
cumjThetaappa = 0
thetaappaNew = matrix(rep(0,nvarthetaappa*napp), ncol=nvarthetaappa)
varModeApp  = matrix(rep(0,nvarthetaappa*napp), ncol=nvarthetaappa)
errorwtempapp = matrix(rep(0,T*napp), ncol=T)
w0ivapp = napp
Swivapp =diag(rep(0.1,napp))

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

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 Metropolis candidate densities for ",nlgt," units ...",fill=TRUE)
fsh()

#-------------------------------
#  compute pooled optimum
#-------------------------------

# for category:
#----------------
outcat  =  optim(alphainit,llmnl,method="BFGS",control=list( fnscale=-1,trace=1,reltol=1e-3),
              X=Xpooledcat,y=ypooledcat)
betapooledcat =   outcat$par
Hcat          =   mnlHess(betapooledcat,ypooledcat,Xpooledcat)
rootHcat      =   chol(Hcat)

# for apps:
#----------------
outapp  =  optim(betainit,llmnl,method="BFGS",control=list( fnscale=-1,trace=1,reltol=1e-3),
                 X=Xpooledapp,y=ypooledapp)
betapooledapp =   outapp$par
Happ          =   mnlHess(betapooledapp,ypooledapp,Xpooledapp)
rootHapp      =   chol(Happ)

#--------------------------------------------------------------------
#  now go thru and computed fraction likelihood estimates and hessians
#       Lbar=log(pooled likelihood^(n_i/N))
#       fraction loglike = (1-w)*loglike_i + w*Lbar
#--------------------------------------------------------------------
for (i in 1:nlgt)
{
   wgtcat = length(lgtdatac[[i]]$y)/length(ypooledcat)
   wgtapp = length(lgtdataa[[i]]$y)/length(ypooledapp)
   outcat=optim(betapooledcat,llmnlFract,method="BFGS",control=list( fnscale=-1,trace=0,
   reltol=1e-2),
             X=lgtdatac[[i]]$X,y=lgtdatac[[i]]$y,betapooled=betapooledcat,rootH=rootHcat,w=w,
             wgt=wgtcat)
   outapp=optim(betapooledapp,llmnlFract,method="BFGS",control=list( fnscale=-1,trace=0,
   reltol=1e-2),
             X=lgtdataa[[i]]$X,y=lgtdataa[[i]]$y,betapooled=betapooledapp,rootH=rootHapp,w=w,
             wgt=wgtapp)
   # for cat
   if(outcat$convergence == 0)
   { hesscat=mnlHess(outcat$par,lgtdatac[[i]]$y,lgtdatac[[i]]$X)
     lgtdatac[[i]]=c(lgtdatac[[i]],list(converge=1,betafmle=outcat$par,hess=hesscat))
   }else
   { lgtdatac[[i]]=c(lgtdatac[[i]],list(converge=0,betafmle=c(rep(0,nvaralphai)),
                                        hess=diag(nvaralphai))) }
   # for app
   if(outapp$convergence == 0)
```

```
    { hessapp=mnlHess(outapp$par,lgtdataa[[i]]$y,lgtdataa[[i]]$X)
      lgtdataa[[i]]=c(lgtdataa[[i]],list(converge=1,betafmle=outapp$par,hess=hessapp))
    }else
    { lgtdataa[[i]]=c(lgtdataa[[i]],list(converge=0,betafmle=c(rep(0,nvarbetai)),
                                         hess=diag(nvarbetai))) }

    oldalphai[i,]=lgtdatac[[i]]$betafmle
    oldbetai [i,]=lgtdataa[[i]]$betafmle
    if(i%%50 ==0) cat("  completed unit #",i,fill=TRUE)
    fsh()
 }

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

 #alphai
 #----------
 if(drawdeltaalphai)
 {mgoutalphai=rmixGibbs(oldalphai-ZIndv%*%t(matrix(olddeltaalphai,ncol=nzalphai)),
                  mubaralphai,Amualphai,nualphai,Valphai,aalphai,oldprobalphai,indalphai,
                  oldcompalphai)
 } else
 {mgoutalphai=rmixGibbs(oldalphai,
                  mubaralphai,Amualphai,nualphai,Valphai,aalphai,oldprobalphai,indalphai,
                  oldcompalphai)}
 oldprobalphai=mgoutalphai[[1]]
 oldcompalphai=mgoutalphai[[3]]
 indalphai=mgoutalphai[[2]]
 #--------------------------------------------------------------
 # now draw deltaalphai | {alphai}, indalphai, compsalphai
 #--------------------------------------------------------------
 if(drawdeltaalphai) {olddeltaalphai=drawDelta(ZIndv,oldalphai,indalphai,oldcompalphai,
 deltabaralphai,Adalphai)}

 #betai
 #----------
 if(drawdeltabetai)
 {mgoutbetai=rmixGibbs(oldbetai-ZIndv%*%t(matrix(olddeltabetai,ncol=nzbetai)),
                  mubarbetai,Amubetai,nubetai,Vbetai,abetai,oldprobbetai,indbetai,oldcompbetai)
 }else
 {mgoutbetai=rmixGibbs(oldbetai,
                  mubarbetai,Amubetai,nubetai,Vbetai,abetai,oldprobbetai,indbetai,oldcompbetai
                  )}
 oldprobbetai=mgoutbetai[[1]]
 oldcompbetai=mgoutbetai[[3]]
 indbetai=mgoutbetai[[2]]
 #--------------------------------------------------------------
 # now draw delta | {beta_i}, ind, comps
 #--------------------------------------------------------------
 if(drawdeltabetai) {olddeltabetai=drawDelta(ZIndv,oldbetai,indbetai,oldcompbetai,
 deltabarbetai,Adbetai)}
```

```
#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)}

#thetaappa
#----------
if(drawdeltathetaappa)
{mgoutthetaappa=rmixGibbs(oldthetaappa-Zapp%*%t(matrix(olddeltathetaappa,ncol=nzthetaappa)),
                 mubarthetaappa,Amuthetaappa,nuthetaappa,Vthetaappa,athetaappa,
                 oldprobthetaappa,indthetaappa,
                 oldcompthetaappa)
}else
{mgoutthetaappa=rmixGibbs(oldthetaappa,
                 mubarthetaappa,Amuthetaappa,nuthetaappa,Vthetaappa,athetaappa,
                 oldprobthetaappa,indthetaappa,
                 oldcompthetaappa)}
oldprobthetaappa=mgoutthetaappa[[1]]
oldcompthetaappa=mgoutthetaappa[[3]]
indthetaappa=mgoutthetaappa[[2]]
#---------------------------------------------------------------
# now draw delta | {beta_i}, ind, comps
#---------------------------------------------------------------
if(drawdeltathetaappa) {olddeltathetaappa=drawDelta(Zapp,oldthetaappa,indthetaappa,
oldcompthetaappa,deltabarthetaappa,
                                                   Adthetaappa)}

#etaIndvapp
#----------
if(drawdeltaetaIndvapp)
{mgoutetaIndvapp=rmixGibbs(oldetaIndvapp-ZIndv%*%t(matrix(olddeltaetaIndvapp,ncol=
nzetaIndvapp)),
                 mubaretaIndvapp,AmuetaIndvapp,nuetaIndvapp,VetaIndvapp,aetaIndvapp,
                 oldprobetaIndvapp,
                 indetaIndvapp,oldcompetaIndvapp)
}else
{mgoutetaIndvapp=rmixGibbs(oldetaIndvapp,
                 mubaretaIndvapp,AmuetaIndvapp,nuetaIndvapp,VetaIndvapp,aetaIndvapp,
```

```
                  oldprobetaIndvapp,
                  indetaIndvapp,oldcompetaIndvapp)}
oldprobetaIndvapp=mgouteteaIndvapp[[1]]
oldcompetaIndvapp=mgouteteaIndvapp[[3]]
indetaIndvapp=mgouteteaIndvapp[[2]]
#---------------------------------------------------------------
# now draw delta | {beta_i}, ind, comps
#---------------------------------------------------------------
if(drawdeltaetaIndvapp) {olddeltaetaIndvapp=drawDelta(ZIndv,oldetaIndvapp,indetaIndvapp,
oldcompetaIndvapp,
                                                      deltabaretaIndvapp,AdetaIndvapp)}

#gammaIndv
#----------
if(drawdeltagammaIndv)
{mgoutgammaIndv=rmixGibbs(oldgammaIndv-ZIndv%*%t(matrix(olddeltagammaIndv,ncol=nzgammaIndv)),
                  mubargammaIndv,AmugammaIndv,nugammaIndv,VgammaIndv,agammaIndv,
                  oldprobgammaIndv,
                  indgammaIndv,oldcompgammaIndv)
}else
{mgoutgammaIndv=rmixGibbs(oldgammaIndv,
                  mubargammaIndv,AmugammaIndv,nugammaIndv,VgammaIndv,agammaIndv,
                  oldprobgammaIndv,indgammaIndv,
                  oldcompgammaIndv)}
oldprobgammaIndv=mgoutgammaIndv[[1]]
oldcompgammaIndv=mgoutgammaIndv[[3]]
indgammaIndv=mgoutgammaIndv[[2]]
#---------------------------------------------------------------
# now draw delta | {beta_i}, ind, comps
#---------------------------------------------------------------
if(drawdeltagammaIndv) {olddeltagammaIndv=drawDelta(ZIndv,oldgammaIndv,indgammaIndv,
oldcompgammaIndv,
                                                    deltabargammaIndv,AdgammaIndv)}
#-------------------------------------------------------------------------------------
#  loop over all lgt equations drawing beta_i, alphai | ind[i],z[i,],mu[ind[i]],rooti[ind[i]]
#-------------------------------------------------------------------------------------
itime=proc.time()[3]
result = foreach(lgt=1:nlgt,.packages=c("MASS","corpcor"),.combine=rbind,.export=c("llmnl",
"lndMvn")) %dopar%{
   rootpialphai=oldcompalphai[[indalphai[lgt]]]$rooti
   rootpibetai=oldcompbetai[[indbetai[lgt]]]$rooti

   #  note: alpha_i = Deltaalphai*zIndv_i + u_i  Deltaalphai is nvaralphai x nzalphai
   if(drawdeltaalphai) {
      betabaralphai=oldcompalphai[[indalphai[lgt]]]$mu+matrix(olddeltaalphai,ncol=nzalphai)
      %*%as.vector(ZIndv[lgt,])
   }else {
      betabaralphai=oldcompalphai[[indalphai[lgt]]]$mu }

   #  note: beta_i = Deltabetai*zIndv_i + u_i  Deltabetai is nvarbetai x nzbetai
   if(drawdeltabetai) {
      betabarbetai=oldcompbetai[[indbetai[lgt]]]$mu+matrix(olddeltabetai,ncol=nzbetai)%*%
      as.vector(ZIndv[lgt,])
   }else {
```

```
        betabarbetai=oldcompbetai[[indbetai[lgt]]]$mu }

      #etaIndvapp
      #-------------------------------
      #  note: etaIndvapp = DeltaetaIndvapp*zIndv_i + u_i  DeltaetaIndvapp is nvaretaIndvapp x
      nzetaIndvapp
      if(drawdeltaetaIndvapp) {
         betabaretaIndvapptemp=oldcompetaIndvapp[[indetaIndvapp[lgt]]]$mu+matrix(
         olddeltaetaIndvapp,ncol=nzetaIndvapp)%*%
            as.vector(ZIndv[lgt,])
      }else {
         betabaretaIndvapptemp=oldcompetaIndvapp[[indetaIndvapp[lgt]]]$mu }

      #gammaIndv
      #-------------------------------
      #  note: beta_i = Deltabetai*zIndv_i + u_i  Deltabetai is nvarbetai x nzbetai
      if(drawdeltagammaIndv) {
         betabargammaIndvtemp=oldcompgammaIndv[[indgammaIndv[lgt]]]$mu+matrix(olddeltagammaIndv,
         ncol=nzgammaIndv)%*%
            as.vector(ZIndv[lgt,])
      }else {
         betabargammaIndvtemp=oldcompgammaIndv[[indgammaIndv[lgt]]]$mu }

      etagammalist = list(eta=betabaretaIndvapptemp,gamma=betabargammaIndvtemp)

      # cat (alphai)
      #----------
      if (iterrep == 1)
      { oldllcat[lgt]=llmnl(oldalphai[lgt,],lgtdatac[[lgt]]$y,lgtdatac[[lgt]]$X)}
      #   compute inc.root
      inc.rootcat=chol(chol2inv(chol(lgtdatac[[lgt]]$hess+rootpialphai%*%t(rootpialphai))))
      # variance of misperception
      for (clgt in 1:ncat){
            tempmisperceptcat = matrix(vcat[,,clgt],ncol=nlgt)
            vcatmispercept[clgt]    = abs(tempmisperceptcat[clgt,clgt]-
            tempmisperceptcat[clgt,-clgt]%*%tempmisperceptcat[-clgt,-clgt]%*%
            as.matrix(tempmisperceptcat[-clgt,clgt],nrow=1))
      }
      metropoutcat=mnlRwMetropOnce(lgtdatac[[lgt]]$y,lgtdatac[[lgt]]$X,oldalphai[lgt,],
                                oldllcat[lgt],s,inc.rootcat,betabaralphai,rootpialphai,pccat[lgt
                                ],
                                as.vector(indvperceptioncat[lgt,,]),cumjcat[lgt],as.vector(cbind
                                (tt0cat,tt1cat[,-T])),
                                rep(vcatmispercept,T), Fcat[lgt,],iterrep)

#       oldalphai[lgt,]= metropoutcat$betadraw
#       oldllcat[lgt]  = metropoutcat$oldll
#       cumjcat[lgt]   = metropoutcat$cumj
#       pccat[lgt]     = metropoutcat$pc
#       indvperceptioncat[lgt,,]=matrix(metropoutcat$latentitold,byrow=T,nrow=ncat)
      catlist=list(betadraw=metropoutcat$betadraw,oldll=metropoutcat$oldll,cumj=metropoutcat$
      cumj,pc=metropoutcat$pc,
                   indvperceptioncat=matrix(metropoutcat$latentitold,byrow=T,nrow=ncat))
```

```
      # app (betai)
      #----------
      if (iterrep == 1)
      { oldllapp[lgt]=llmnl(oldbetai[lgt,],lgtdataa[[lgt]]$y,lgtdataa[[lgt]]$X)}
      #   compute inc.root
      inc.rootapp=chol(chol2inv(chol(lgtdataa[[lgt]]$hess+rootpibetai%*%t(rootpibetai))))
      # variance of misperception
      for (algt in 1:napps){
         tempmisperceptapp = matrix(vapp[(((algt-1)*nIndv+1):(algt*nIndv)),],ncol=nlgt)
         vappmispercept[algt]    = abs(tempmisperceptapp[algt,algt]-
            tempmisperceptapp[algt,-algt]%*%tempmisperceptapp[-algt,-algt]%*%
            as.matrix(tempmisperceptapp[-algt,algt],nrow=1))
      }
      indvPerceptTemp = as.vector(diag(indvperceptionapp[lgt,as.vector(lgtdatac[[lgt]]$y),1:T]))
      latentapprealTemp = rbind(rep(0,T),cbind(tt0app,tt1app[,-T]))
      latentapprealTemp = as.vector(diag(latentapprealTemp[as.vector(lgtdatac[[lgt]]$y),1:T]))
      metropoutapp    = mnlRwMetropOnce(lgtdataa[[lgt]]$y,lgtdataa[[lgt]]$X,oldbetai[lgt,],
                             oldllapp[lgt],s,inc.rootapp,betabarbetai,rootpibetai,pcapp[
                             lgt],
                             indvPerceptTemp,cumjapp[lgt],
                             latentapprealTemp,
                             vappmispercept[lgtdatac[[lgt]]$y-1], Fapp[lgt,],iterrep)
#       oldbetai[lgt,]= metropoutapp$betadraw
#       oldllapp[lgt]  = metropoutapp$oldll
#       cumjapp[lgt]   = metropoutapp$cumj
#       pcapp[lgt]     = metropoutapp$pc
#
diag(indvperceptionapp[lgt,as.vector(lgtdatac[[lgt]]$y),1:T])=matrix(metropoutapp$latentitold,byr
ow=T,nrow=1)
#       indvperceptionapp[lgt,1,1:T]=rep(0,T)

      applist=list(betadraw=metropoutapp$betadraw,oldll=metropoutapp$oldll,cumj=metropoutapp$
      cumj,pc=metropoutapp$pc,
                   indvperceptioncat=matrix(metropoutapp$latentitold,byrow=T,nrow=1))
      return(list(etagammalist=etagammalist, catlist=catlist, applist=applist))
   }
    ctime=proc.time()[3]
    timetoend=(ctime-itime)
    cat ('time it takes to go over all lgt elements:')
    cat(timetoend)

   # this assignment is very quick
   for (lgt in 1:nlgt){
      betabaretaIndvapp[lgt]=result[[lgt,1]]$eta
      betabargammaIndv[lgt]=result[[lgt,1]]$gamma
      oldalphai[lgt,]= result[[lgt,2]]$betadraw
      oldllcat[lgt]  = result[[lgt,2]]$oldll
      cumjcat[lgt]   = result[[lgt,2]]$cumj
      pccat[lgt]     = result[[lgt,2]]$pc
      indvperceptioncat[lgt,,]=result[[lgt,2]]$indvperceptioncat
      oldbetai[lgt,]= result[[lgt,3]]$betadraw
      oldllapp[lgt]  = result[[lgt,3]]$oldll
      cumjapp[lgt]   = result[[lgt,3]]$cumj
      pcapp[lgt]     = result[[lgt,3]]$pc
```

```
      diag(indvperceptionapp[lgt,as.vector(lgtdatac[[lgt]]$y),1:T])=result[[lgt,3]]$
      indvperceptioncat
      indvperceptionapp[lgt,1,1:T]=rep(0,T)
   }
    result = NULL

   #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 }

   }

   #thetaappa
   #-------------------------------
   #  note: beta_i = Deltabetai*zIndv_i + u_i  Deltabetai is nvarbetai x nzbetai
   for (algt in 1:napps) {
      if(drawdeltathetaappa) {
         betabarthetaappa[algt,]=oldcompthetaappa[[indthetaappa[algt]]]$mu+matrix(
         olddeltathetaappa,ncol=nzthetaappa)%*%
            as.vector(Zapp[algt,])
      }else {
         betabarthetaappa[algt,]=oldcompthetaappa[[indthetaappa[algt]]]$mu }
   }
 cat('\nEstimation of alpha and beta parameters done successfully...\n')
#-------------------------------------------------------------------------------------
#                             EKF for the app categories
#-------------------------------------------------------------------------------------
   #itime=proc.time()[3]
   outcatEKF = catEKF(ycat=indvperceptioncat,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')

#-------------------------------------------------------------------------------------
#                             EKF for the apps
#-------------------------------------------------------------------------------------
```

```
#    itime=proc.time()[3]
   outappEKF = appEKF(yapp=indvperceptionapp[,2:(1+napps),],Fapp=Fapp,papp=1,m0app=m0app,C0app=
   C0app,
                       vapp=vapp,wapp=wapp,thetaappa=thetaappaest,catlatent = tt1cat)
#    ctime=proc.time()[3]
#    timetoend=(ctime-itime)
   mapp   = outappEKF$mapp
   Capp   = outappEKF$Capp
   m0app  = outappEKF$m0app
   C0app  = outappEKF$C0app
   tt1app = outappEKF$tt1app
   tt0app = outappEKF$tt0app
   Y1app  = outappEKF$Y1app
   MADapp = outappEKF$MADapp
   MSEapp = outappEKF$MSEapp
   if (max(mapp)>1e5){
    cat('\n a problem is found in the appEKF\n')
    break;
    }
  cat('\nEKF of app done successfully...\n')
#-------------------------------------------------------------------------------------
#   Misperception mean and Variance for Category (mean and variance of observation equation)
#-------------------------------------------------------------------------------------
# be careful as the first coefficient is set to one for identification
# pull observations across categories

#    itime=proc.time()[3]
   for (lgt in 2:nlgt){
      #individual misperception
      for (clgt in 1:ncat){
         catvarmisperceptindv[clgt] =   abs(vcat[lgt,lgt,clgt] - vcat[lgt,-lgt,clgt]%*%vcat[-lgt
         ,-lgt,clgt]%*%vcat[-lgt,lgt,clgt])
      }
      Ytempcat = t(indvperceptioncat[lgt,,])
      Xtempcat = t(tt1cat[,])
      S1 = chol2inv(chol(sum(diag(crossprod(Xtempcat,Xtempcat))/catvarmisperceptindv)+
      S0catobsInv))
      b1 = S1*(sum(diag(crossprod(Xtempcat,Ytempcat))/catvarmisperceptindv)+b0catobs*S0catobsInv)
      Fcat[lgt,] = b1 + t(chol(S1))%*%rnorm(1)

   }
   llcatObsEqCur = 0
   for (clt in 1:ncat){
      # IW to find the misperception variance
      viwmucat = v0ivcat + T
      errortempcat = t(matrix(indvperceptioncat[,clt,],ncol = T) -Fcat%*% tt1cat[clt,])
      #dim=c(nIndv,T)
      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)
      vcat[,,clt] = rwishart(viwmucat,sigmaiwmucat)$IW
```

```
      llcatObsEqCur = llcatObsEqCur - 0.5*sum(diag(chol2inv(chol(vcat[,,clt]))%*%crossprod(
      errortempcat,errortempcat)))-
         0.5*T*log(2*pi)-0.5*T*det(vcat[,,clt])

   }
#   ctime=proc.time()[3]
#   timetoend=(ctime-itime)
    cat('\nestimation of misperception parameters for category......successfully done\n')
#-------------------------------------------------------------------------------------
# Misperception mean and Variance for Apps (mean and variance of observation equation)
#-------------------------------------------------------------------------------------
# be careful as the first coefficient is set to one for identification
# pull observations across apps

 #   itime=proc.time()[3]
   for (lgt in 2:nlgt){
      #individual misperception
      appvarmisperceptindv = foreach (alt = 1:napp, .combine=rbind) %dopar%{
         sigmacurcur = vapp[(((alt-1)*nlgt+1):(alt*nlgt)),]
         abs(sigmacurcur[lgt,lgt] - sigmacurcur[lgt,-lgt]%*%sigmacurcur[-lgt,-lgt]%*%sigmacurcur
         [-lgt,lgt])
      }
      Ytempapp = t(indvperceptionapp[lgt,,])
      Xtempapp = t(tt1app[,])
      S1 = chol2inv(chol(sum(diag(crossprod(Xtempapp,Xtempapp))/appvarmisperceptindv)+
      S0appobsInv))
      b1 = S1*(sum(diag(crossprod(Xtempapp,Ytempapp))/appvarmisperceptindv)+b0appobs*S0appobsInv)
      Fapp[lgt,] = b1 + t(chol(S1))%*%rnorm(1)

   }
   llappObsEqCur =0
   for (alt in 1:napp){
      # IW to find the misperception variance
      viwmuapp = v0ivapp + T
      errortempapp = t(matrix(indvperceptionapp[,alt,],ncol = T) - Fapp%*%tt1app[alt,])
      #dim=c(nIndv,T)
    if (is.positive.definite(Svivapp+crossprod(errortempapp,errortempapp))) {
        sigmaiwmuapp = chol2inv(chol(Svivapp+crossprod(errortempapp,errortempapp)))
    }else{
        sigmaiwmuapp = ginv(Svivapp)
    }
      # draw from IW(sigmaiwmu,viwmu)
      vapp[(((alt-1)*nlgt+1):(alt*nlgt)),] = rwishart(viwmuapp,sigmaiwmuapp)$IW
      llappObsEqCur = llappObsEqCur - 0.5*sum(diag(chol2inv(chol(vapp[(((alt-1)*nlgt+1):(alt*
      nlgt)),]))%*%
            crossprod(errortempapp,errortempapp)))-0.5*T*log(2*pi)-0.5*T*det(vapp[(((alt-1)*nlgt
            +1):(alt*nlgt)),])
   }
   ctime=proc.time()[3]
   timetoend=(ctime-itime)
  cat('\nestimation of misperception parameters for apps......successfully done\n')


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

```
#                              Category Latent Coefficient mean and Variance
#---------------------------------------------------------------------------------------
   alpha         = 0.5
   rndacceptance = 0.8
   meansuntilcur = thetacatjest
   # using conditioning technique (cannot be parallelized due to dependance)
     itime=proc.time()[3]
   result = foreach (clt=1:ncat, .packages=c("numDeriv","foreach","bayesm","corpcor","MASS"), .
   combine=rbind) %dopar%{
      cat(clt)
   #   itime=proc.time()[3]
      outcatst=optim(thetacatjest[clt,],basslikelihoodCat,method="BFGS",control=list( fnscale=-1
      ,trace=1,reltol=1e-2),
                     thetacatjest=thetacatjest, tt0cat=tt0cat,tt1cat=tt1cat,
                     wcat=wcat,oldcompthetacatj=oldcompthetacatj,
                     indthetacatj=indthetacatj,curcat=clt,meansuntilcur=meansuntilcur)
     # ctime=proc.time()[3]
    #   timetoend=(ctime-itime)
      meansuntilcur[clt,]=outcatst$par
      thetacatjestTemp   = thetacatjest
      thetacatjestTemp[clt,]=meansuntilcur[clt,]
      hessiancatTemp=bassHessianCat(thetacatjestTemp,tt0cat,tt1cat,wcat,clt)
      if (is.positive.definite(hessiancatTemp)){
         varcattemp  = diag(chol2inv(chol(hessiancatTemp)))
      }else{
         varcattemp  = diag(ginv(make.positive.definite(hessiancatTemp)))
      }
      varcattemp  = pmax(varcattemp,c(rep(1e-6,length(varcattemp))))
      list(mode= meansuntilcur[clt,], variance=varcattemp)
      #hessiancatconditional[[clt]]=outcatst$hessian
   }

   for (clt in 1:ncat){
      meansuntilcur[clt,]      =  result[[clt,1]]
      varModeCat[clt,] = result[[clt,2]]
   }
      ctime=proc.time()[3]
      timetoend=(ctime-itime)

   # to test
   #end test
   j = 1
   cat("M-H loop to find the appropriate parameters for category latent state equation")
   while (alpha < rndacceptance){
      j              = j+1
      cat(j,",")

      result = foreach (clt= 1:ncat,.packages=c("numDeriv","foreach","bayesm","corpcor"), .
      combine=rbind) %dopar%{
         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)
         postPlatentOld = -basslikelihoodCat(thetacatjest[clt,], thetacatjest=thetacatjest,
         tt0cat=tt0cat,tt1cat=tt1cat,
                                                wcat=wcat,oldcompthetacatj=oldcompthetacatj,
                                                indthetacatj=indthetacatj,curcat=clt,meansuntilcur=
                                                meansuntilcur)
         list(postPlatentNew = postPlatentNew, postPlatentOld = postPlatentOld,thetacatjestNew=
         thetacatjestNew)
      }

      postPlatentNew = sum(as.numeric(as.matrix(result,ncol=3)[,2]))
      postPlatentOld = sum(as.numeric(as.matrix(result,ncol=3)[,1]))
      alpha          = postPlatentNew - postPlatentOld
      rndacceptance  = log(runif(1))
      if (j > 100){
         pcThetacatj = pcThetacatj /10;
      postPlatentNew=llcatStateEq
         break
      }
   }
   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
   if (j<100){
     for (clt in 1:ncat){
         thetacatjest[clt,]    = as.numeric(result[[clt,3]],ncol=3)
      }
    }
 cat('\nestimation of non-state prameters of state equation parameters for
 category......successfully done\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 = chol2inv(chol(Swivcat+crossprod(errorwtempcatt,errorwtempcatt)))
```

```
   # draw from IW(sigmaiwmu,viwmu)
   wcat = rwishart(wiwmucat,sigmaiwmucat)$IW
    if (!is.positive.definite(wcat)){
       wcat=make.positive.definite(wcat)
    }
 cat('\nestimation of variance prameters of state equation for category......successfully
 done\n')

#-------------------------------------------------------------------------------------
#                              App Latent Coefficient mean and Variance
#-------------------------------------------------------------------------------------
   alpha         = 0.5
   rndacceptance = 0.8
   meansuntilcur = thetaappaest
   # using conditioning technique (cannot be parallelized due to dependance)
        itime=proc.time()[3]
   result = NULL
   result = foreach (alt=1:napp, .packages=c("numDeriv","foreach","bayesm","corpcor","MASS"), .
   combine=rbind) %dopar%{
      cat(alt)
      outappst=optim(thetaappaest[alt,],basslikelihoodApp,method="BFGS",control=list( fnscale=-1
      ,trace=1,reltol=1e-2), hessian=TRUE,
                     thetaappaest=thetaappaest, tt0app=tt0app,tt1app=tt1app,
                     wapp=wapp,oldcompthetaappa=oldcompthetaappa,
                     indthetaappa=indthetaappa,curapp=alt,meansuntilapp=meansuntilcur,tt1cat=
                     tt1cat)
      meansuntilcur[alt,]=outappst$par
      if (is.positive.definite(outappst$hessian )){
         varapptemp  = diag(chol2inv(chol(outappst$hessian)))
      }else{
         varapptemp  = diag(ginv(make.positive.definite(outappst$hessian)))
      }
      varapptemp  = pmax(varapptemp,c(rep(1e-6,length(varapptemp))))
      list(mode= meansuntilcur[alt], variance=varapptemp)
      #hessiancatconditional[[clt]]=outcatst$hessian
   }

   for (alt  in 1:ncat){
      meansuntilcur[alt,]      =  result[[alt,1]]
      varModeApp[alt,] = result[[alt,2]]
   }
        ctime=proc.time()[3]
       timetoend=(ctime-itime)

   # to test
   #end test
   j = 1
   cat("M-H loop to find the appropriate parameters for category latent state equation")
   while (alpha < rndacceptance){
      j              = j+1
      cat(j,",")

      result = foreach (alt= 1:napp,.packages=c("numDeriv","foreach","bayesm","corpcor"), .
      combine=rbind) %dopar%{
```

```
        wpm            = pcThetaappa*varModeApp[alt,]
        wpm            = pmax(wpm,c(rep(1e-6,length(wpm))))
        thetaappaestNew=meansuntilcur[alt,] + wpm*rnorm(length(thetaappa[alt,]))
        postPlatentNew = -basslikelihoodApp(thetaappaestNew, thetaappaest=thetaappaest, tt0app=
        tt0app,tt1app=tt1app,
                                    wapp=wapp,oldcompthetaappa=oldcompthetaappa,
                                    indthetaappa=indthetaappa,curapp=alt,meansuntilapp=
                                    meansuntilcur,tt1cat=tt1cat)
        postPlatentOld = -basslikelihoodApp(thetaappaest[alt,], thetaappaest=thetaappaest,
        tt0app=tt0app,tt1app=tt1app,
                                    wapp=wapp,oldcompthetaappa=oldcompthetaappa,
                                    indthetaappa=indthetaappa,curapp=alt,meansuntilapp=
                                    meansuntilcur,tt1cat=tt1cat)
        list(postPlatentNew = postPlatentNew, postPlatentOld = postPlatentOld,thetaappaestNew=
        thetaappaestNew)
      }

      postPlatentNew = sum(as.numeric(as.matrix(result,ncol=3)[,2]))
      postPlatentOld = sum(as.numeric(as.matrix(result,ncol=3)[,1]))
      alpha          = postPlatentNew - postPlatentOld
      rndacceptance  = log(runif(1))
      if (j > 100){
        pcThetaappa = pcThetaappa /10;
       postPlatentNew=llappStateEq
        break;
      }
    }
    cumjThetaappa = cumjThetaappa + j          # to keep cumulative value
    accptrate = iterrep/cumjThetaappa    # acceptance rate until now
    if (floor (iterrep/5) == iterrep/5){
      if (accptrate > 0.15){
        pcThetaappa = pcThetaappa*3;
        cumjThetaappa = iterrep/0.15
      }else{
        if(accptrate < 0.01){
          pcThetaappa = pcThetaappa/3
          cumjThetaappa = iterrep/0.01
        }
      }
    }
    llappStateEq = postPlatentNew
    if (j<100){
     for (alt in 1:napp){
        thetaappaest[alt,]    = as.numeric(result[[alt,3]],ncol=3)
      }
     }
  cat('\nestimation of non-state prameters of state equation parameters for
  apps......successfully done\n')

#------------------------------------------------
# Inverse Wishart Variance of State Equation
#------------------------------------------------
    wiwmuapp = w0ivapp + T
    for (alt in 1:napp){
```

```
      errorwtempapp[alt,] = t(bassErrorsApp(thetaappaest,tt0app,tt1app,wapp,alt,tt1cat))
   }
   errorwtempappt = t(errorwtempapp)

   sigmaiwmuapp = chol2inv(chol(Swivapp+crossprod(errorwtempappt,errorwtempappt)))
   # draw from IW(sigmaiwmu,viwmu)
   wapptemp = rwishart(wiwmuapp,sigmaiwmuapp)$IW
   for (alt in 1:napp){
      wapp[alt] = wapptemp[alt,alt] - wapptemp[alt,-alt]%*%wapptemp[-alt,-alt]%*%wapptemp[-alt,
      alt]
    if (wapp[alt]<0){
        wapp[alt]=1e-6
    }
   }
cat('\nestimation of variance prameters of state equation for apps......successfully done\n')


#--------------------------------------------------------------------------
#       save every keepth draw
#--------------------------------------------------------------------------
   if (sw == 1)
   {
      #alphai
      alphaidraw[,,jp] = oldalphai
      probdrawalphai[jp,]=oldprobalphai
      loglikecat[jp] = sum(oldllcat)
      Deltadrawalphai[jp,]= olddeltaalphai
      compdrawalphai[[jp]]=oldcompalphai
      #betai
      betaidraw[,,jp] = oldbetai
      probdrawbetai[jp,]=oldprobbetai
      loglikeapp[jp] = sum(oldllapp)
      Deltadrawbetai[jp,]= olddeltabetai
      compdrawbetai[[jp]]=oldcompbetai
      # thetacatj
      thetacatjdraw[,,jp] = thetacatjest
      probdrawthetacatj[jp,]=oldprobthetacatj
      llcat_[jp] = llcatObsEqCur + llcatStateEq
      Deltadrawthetacatj[jp,]= olddeltathetacatj
      compdrawthetacatj[[jp]]=oldcompthetacatj
      # thetaappa
      thetaappadraw[,,jp] = thetaappaest
      probdrawthetaappa[jp,]=oldprobthetaappa
      llapp_[jp] = llappStateEq + llappObsEqCur
      Deltadrawthetaappa[jp,]= olddeltathetaappa
      compdrawthetaappa[[jp]]=oldcompthetaappa
      #gammaIndv
      gammaIndvdraw[,,jp] = Fcat
      probdrawgammaIndv[jp,] =oldprobgammaIndv
      DeltadrawgammaIndv[jp,]= olddeltagammaIndv
      compdrawgammaIndv[[jp]]=oldcompgammaIndv
      #etaIndvapp
      etaIndvappdraw[,,jp] = Fapp
      probdrawetaIndvapp[jp,] =oldprobetaIndvapp
```

```
      DeltadrawetaIndvapp[jp,]= olddeltaetaIndvapp
      compdrawetaIndvapp[[jp]]=oldcompetaIndvapp

      #variances
      ccat_[,jp]=c(as.vector(vcat),as.vector(wcat))
      capp_[,jp]=c(as.vector(vapp),as.vector(wapp))
      bcat_ [,jp] = c(as.vector(tt0cat),as.vector(tt1cat))
      bapp_ [,jp] = c(as.vector(tt0app),as.vector(tt1app))


      MADcat_[jp]  = sum(MADcat)/T/ncat/nIndv
      MADapp_[jp]  = sum(MADapp)/T/napp/nIndv

      MSEcat_[jp]  = sum(MSEcat)/T/ncat/nIndv
      MSEapp_[jp]  = sum(MSEapp)/T/napp/nIndv

      #Y1cat_  # I can not save due to memory explodes
      # Y1app_  # 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
#=====================================================================================
```