Code to fit the Blocked Gibbs Sampler

Simulated data

The data required by our method consist of a matrix with the number of times each location was visited by each individual. In this matrix, rows are individuals and columns are locations. For instance, if we had four locations and locations 1-4 were visited 2, 3, 1, and 1 times, respectively, our observation for individual j would consist of the following vector $w_j = [2, 3, 1, 1]$.

The data required for the network analysis methods consist of a matrix with the amount of movement between all pairwise combinations of locations. To generate this input, we randomly sample without replacement the locations visited by each individual to then determine the amount of movement between the different locations. For instance, using the example above, we start by creating a derived vector with the explicit identifiers of each location $r_j = [1, 1, 2, 2, 2, 3, 4]$. We then randomly sample these location identifiers without replacement. Say that this process generates the vector $r_j^* = [2, 2, 1, 1, 3, 2, 4]$, where now the elements in this vector represent the location visited at each time point. This vector can then be represented by the following matrix

$$R_j = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where numbers refer to the amount of movement departing from location i (row i) and arriving in location s (column s). The input matrix required for the network analysis methods is obtained by summing these matrices over all individuals j. To illustrate this process, we provide below the code we used to create the simulated data shown in panel D of Fig. 1 in the main manuscript.

```
rm(list=ls(all=TRUE))
set.seed(1)

#set up the initial parameter values
nloc=50 #number of locations
inds=list(ind1=c(1:12,19:24,50),ind2=c(13:24,31:36,50),
          ind3=c(25:36,43:50),  ind4=c(6:12,37:50)) #locations visited by each group
psi=numeric()
for (i in 1:4){
  tmp=runif(nloc,min=0.5,max=1.1)
  base1=rep(0.001,nloc)
  base1[inds[[i]]]=tmp[inds[[i]]]
  psi=rbind(psi,base1/sum(base1))
}

beta=c(0.2,0.3,0.3,0.2)
nind=10000 #number of individuals
nmov=rpois(nind,lambda=10) #number of times each individual is seen
psi=t(psi)

#generate the fake data
res=matrix(NA,nind,nrow(psi))
locloc=matrix(0,nloc,nloc)
resz=rep(NA,nind)
```

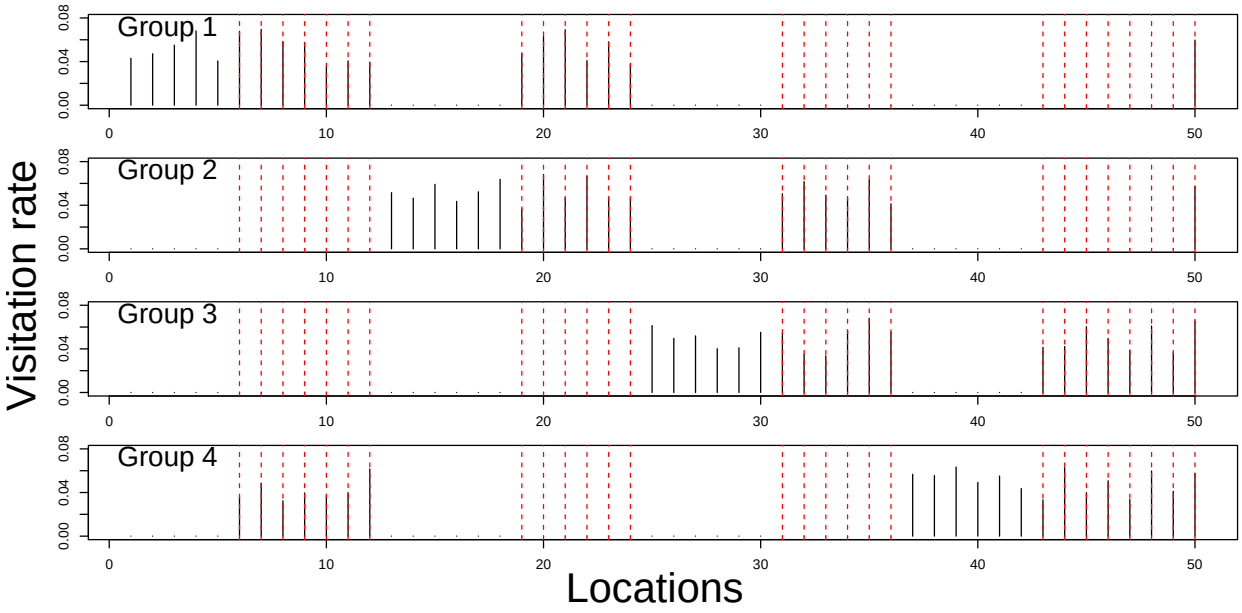
```
for (i in 1:nind){
  #generate z's
  tmp=rmultinom(1,size=1,prob=beta) #sample group membership of each indiv
  z=which(tmp==1)
  resz[i]=z

  tmp=rmultinom(1,size=nmov[i],prob=psi[,z]) #sample where each indiv is seen
  res[i,]=tmp

  #summarize this information in a location-by-location table
  tmp1=numeric()
  for (j in 1:nloc) tmp1=c(tmp1,rep(j,tmp[j]))
  tmp2=sample(tmp1)
  for (j in 2:length(tmp2)){
    ind1=tmp2[j-1]
    ind2=tmp2[j]
    locloc[ind1,ind2]=locloc[ind1,ind2]+1
    locloc[ind2,ind1]=locloc[ind2,ind1]+1
  }
}

#output the results
write.csv(res,'sim1 indiv data.csv',row.names=F)
write.csv(locloc,'sim1 locloc data.csv',row.names=F)
write.csv(psi,'sim1 parameter.csv',row.names=F)
```

The outputs from this code are:

1. the individual level data "sim1 indiv data.csv" used by our method IBC;
2. the spatially aggregated data "sim1 locloc data.csv" used by common network analysis methods; and
3. the underlying visitation rate parameters for each group "sim1 parameter.csv".

The figure below shows the assumed visitation rate for each group, where dotted red lines indicate locations that are frequently visited by individuals from various groups.

```
setwd('U:/independent studies/movement LDA/appendix code')
psi=read.csv('sim1 parameter.csv')
ind=c(19:24,31:36,43:50,6:12)
par(mfrow=c(4,1),mar=c(2,2,1,1),oma=c(3,3,0,0))
psi=t(psi)
for (i in 1:4) {
  plot(psi[i,],type='h',ylim=c(0,0.08))
  abline(v=ind,col='red',lty=2) #locations used by multiple groups
  text(0,0.07,paste('Group',i),pos=4,cex=2)
}
mtext(side=1,at=0.5,outer=T,line=1,'Locations',cex=2)
mtext(side=2,at=0.5,outer=T,line=1,'Visitation rate',cex=2)
```



Blocked Gibbs sampler

Our algorithm is primarily implemented in R, with some auxiliary functions implemented in C++ through the use of the R package "Rcpp". To fit this model, we used the following code:

```
set.seed(10)
library(gtools)
library(Rcpp)

setwd('U:/independent studies/movement LDA/appendix code')
source('gibbs functions.R')
sourceCpp("gibbs_functions_cpp.cpp")

#import data (rows are individuals, columns are locations, cells contain the number
#of times each individual was seen at each location)
dat=data.matrix(read.csv('sim1 indiv data.csv',as.is=T))

ngroups=25 #maximum number of groups
alpha=0.1 #hyper prior parameter
epsilon=0.1 #hyper prior parameter
ngibbs=1000 #number of iterations for the Gibbs Sampler

#run Gibbs sampler
results=gibbs.sampler(ngroups,ngibbs,dat,alpha,epsilon)

write.csv(results$loglikel,'loglikel.csv',row.names=F)
write.csv(results$beta,'betas.csv',row.names=F)
write.csv(results$psi,'psi.csv',row.names=F)
```

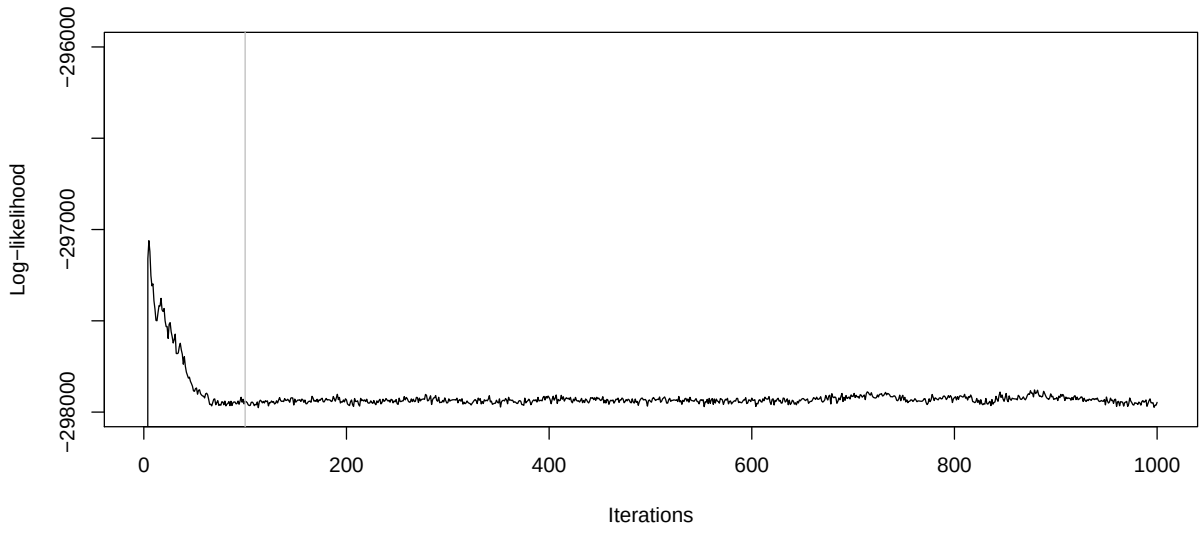
This code imports the individual level data "sim1 indiv data.csv" and relies on the wrapper function "gibbs.sampler" (contained in the file "gibbs functions.R") to fit the model. The arguments of this function are:

1. ngroups: the maximum number of groups;
2. ngibbs: number of iterations for the Gibbs sampler;
3. dat: the data; and
4. alpha and epsilon: values for the hyper-prior parameters α and ϵ.

The output of this function consists in a list containing the following parameters and results for each Gibbs iteration: $\mathbf{z}$,$\mathbf{V}$,β,ψ and the log-likelihood (a useful metric to assess algorithm convergence).

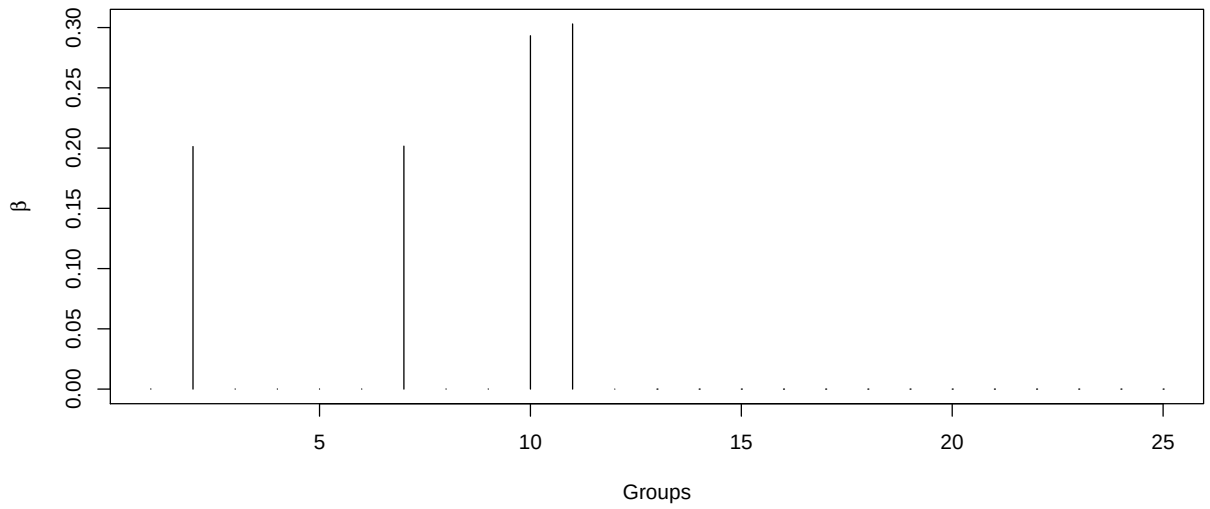
To assess if the algorithm has converged, we examine the trace-plot of the log-likelihood. The code below suggests that the algorithm has converged in 100 iterations. It is important to note that our algorithm does not necessarily result in a model with the highest likelihood because our truncated stick-breaking prior will attempt to enforce a more parsimonius representation (i.e., a smaller number of groups).

```
loglikel=read.csv('loglikel.csv',as.is=T)
plot(loglikel$x,type='l',ylim=c(-298000,-296000),ylab='Log-likelihood',xlab='Iterations')
abline(v=100,col='grey')
```



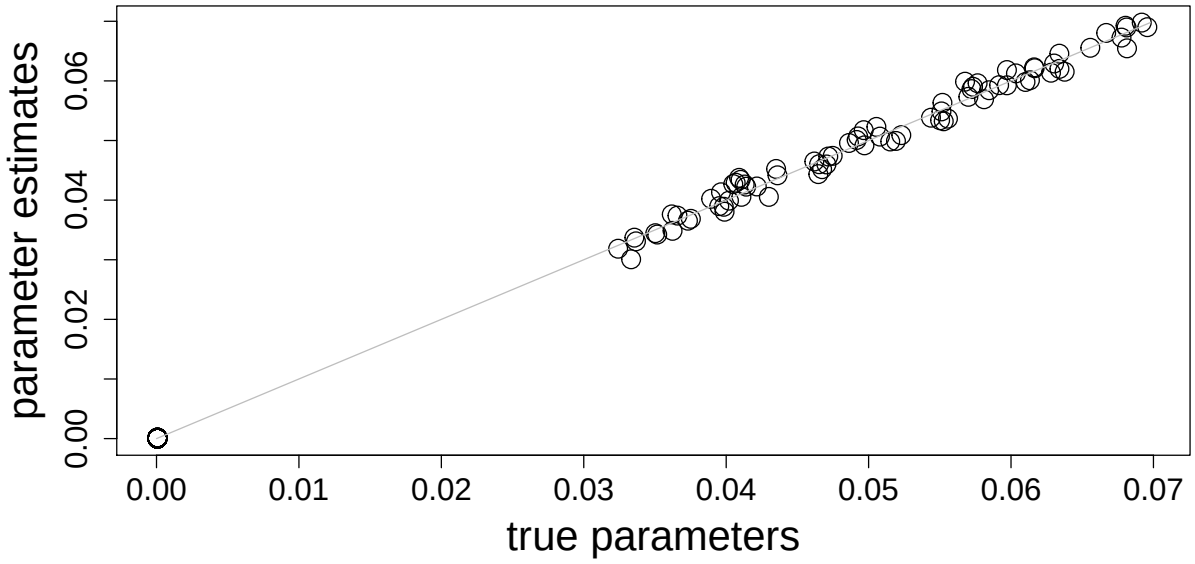
Next, we determine which are the main groups detected by the algorithm. The code below suggests that there are only 4 major groups: groups 2, 7, 10, and 11.

```
betas=read.csv('betas.csv',as.is=T)
seq1=100:nrow(betas)
betas1=apply(betas[seq1,],2,mean)
plot(betas1,type='h',xlab='Groups',ylab=expression(beta))
```



Finally, we compare the estimated visitation rate of each group with the true model parameters in "sim1 parameter.csv". This comparison reveals that our model is able to estimate the true visitation rate ψ parameters very well:

```
psi=data.matrix(read.csv('psi.csv',as.is=T))
seq1=100:nrow(psi)
psi1=apply(psi[seq1,],2,mean)
ngroups=25
nloc=50
psi2=matrix(psi1,ngroups,nloc)[c(2,7,10,11),]

#re-arrange groups to match true configuration of groups
psi3=psi2[c(2,3,4,1),]
# for (i in 1:4) plot(psi3[i,],type='h')

true=data.matrix(read.csv('sim1 parameter.csv',as.is=T))
par(mfrow=c(1,1),mar=c(6,6,1,1))
max1=max(c(true,psi3))
rango=c(0,max1)
plot(t(true),psi3,xlim=rango,ylim=rango,xlab='true parameters',ylab='parameter estimates',
     cex.axis=1.5,cex.lab=2,cex=2)
lines(rango,rango,col='grey')
```



To fit our IBC model, we relied on two auxiliary files:

1. "gibbs functions.R": this file contains the wrapper function "gibbs.sampler", functions to sample from the full conditional distributions of each set of parameters ("update.z", "update.vh", "update.psi"), and a function to calculate the log-likelihood "calc.loglikel"; and
2. "gibbs_functions_cpp.cpp": this file contains functions written in C++ that speed up some of the calculations required in the "update.z" function.

Here is the code within the file "gibbs functions.R":

```
gibbs.sampler=function(ngroups,ngibbs,dat,alpha,epsilon){
  nind=nrow(dat)
  nloc=ncol(dat)
  nvisits=apply(dat,1,sum)

  vh=runif(ngroups-1)
  vh=c(vh,1)
  z=sample(1:ngroups,nind,replace=T)
  param=list(z=z,vh=vh,psi=matrix(1/nloc,ngroups,nloc,byrow=T))

  #store results
  vec.psi=matrix(NA,ngibbs,ngroups*nloc)
  vec.z=matrix(NA,ngibbs,nind)
  vec.vh=matrix(NA,ngibbs,ngroups)
  vec.beta=matrix(NA,ngibbs,ngroups)
  vec.loglikel=rep(NA,ngibbs)
  for (i in 1:ngibbs){
    print(i)
    tmp=update.z(param,nind,ngroups,nloc)
    param$z=tmp$z
    param$pi.h=tmp$pi.h
    param$vh=update.vh(param,ngroups,alpha,nind)
    param$psi=update.psi(param,ngroups,nloc,epsilon)
```

```
    vec.z[i,]=param$z
    vec.vh[i,]=param$vh
    vec.beta[i,]=exp(param$pi.h[1,])
    vec.psi[i,]=param$psi
    vec.loglikel[i]=calc.loglikel(param,ngroups,nloc)
  }

  list(z=vec.z,vh=vec.vh,beta=vec.beta,psi=vec.psi,loglikel=vec.loglikel)
}

update.z=function(paramz,nind,ngroups,nloc){

  #calculate probabilities
  pi.h=loglikel=matrix(NA,nind,ngroups)
  for (j in 1:ngroups){
    if (j==1) pi.h[,j]=log(paramz$vh[j])
    if (j!=1) pi.h[,j]=log(paramz$vh[j])+sum(log(1-paramz$vh[1:(j-1)]))
    loglikel[,j]=(dat*matrix(log(paramz$psi[j,]),nind,nloc,byrow=T))%*%rep(1,nloc)
  }

  res=loglikel+pi.h
  max0=getmax(res,nrow(res),ncol(res))
  max1=matrix(max0,nind,ngroups)
  res1=exp(res-max1)
  res2=res1/matrix(res1%*%rep(1,ngroups),nind,ngroups)

  #multinomial draw
  cumsum1=cumsummat(res2,nrow(res2),ncol(res2))
  res3=cbind(0,cumsum1)
  uni=runif(nind)
  z=rep(NA,nind)
  for (i in 1:ngroups){
    cond=uni>res3[,i] & uni<res3[,i+1]
    z[cond]=i
  }
  list(z=z,pi.h=pi.h)
}
#----------------------------------
update.vh=function(paramz,ngroups,alpha,nind){
  ztab=rep(0,ngroups)
  tmp=table(paramz$z)
  ztab[as.numeric(names(tmp))]=tmp
  n.maior.h=nind-cumsum(ztab)
  tmp1=rbeta(ngroups-1,ztab[-ngroups]+1,alpha+n.maior.h[-ngroups])
  c(tmp1,1)
}
#----------------------------------
update.psi=function(paramz,ngroups,nloc,epsilon){
  res=matrix(NA,ngroups,nloc)
  for (j in 1:ngroups){
    cond=paramz$z==j
    tmp=dat[cond,]
    if (sum(cond)==0) tmp1=rep(0,nloc)
```

```
    if (sum(cond)==1) tmp1=tmp
    if (sum(cond)>1)  tmp1=apply(tmp,2,sum)
    res[j,]=rdirichlet(1,tmp1+epsilon)
  }
  res
}
#------------------------------
calc.loglikel=function(paramz,ngroups,nloc){
  loglikel=0
  for (i in 1:ngroups){
    cond=paramz$z==i
    if (sum(cond)>0){
      tmp=sum(dat[cond,]*matrix(log(paramz$psi[i,]),sum(cond),nloc,byrow=T))
      loglikel=loglikel+tmp
    }
  }
  loglikel
}
```

Here is the code within the file "gibbs_functions_cpp.cpp":

```
#include <Rcpp.h>
using namespace Rcpp;

// aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

//dat1z is a matrix of nrows (rows) vs ncols (columns)

//This function calculates the maximum of each line

// [[Rcpp::export]]

NumericVector getmax(NumericMatrix dat1z, int nrows, int ncols) {
  NumericVector res(nrows);
  double max1;

  for (int oo = 0; oo < nrows; oo++){
    max1=-std::numeric_limits<double>::infinity();
    for (int jj = 0; jj < ncols; jj++){
      if (dat1z(oo,jj)>max1) {
        res(oo)=dat1z(oo,jj);
        max1=dat1z(oo,jj);
      }
    }
  }

  return res;
}

// aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

//dat1z is a matrix of nrows (rows) vs ncols (columns)
```

```
//This function calculates cumsum over the rows of dat1z

// [[Rcpp::export]]

NumericVector cumsummat(NumericMatrix dat1z, int nrows, int ncols) {
  NumericMatrix res(nrows,ncols);

  for (int oo = 0; oo < nrows; oo++){
    res(oo,0)=dat1z(oo,0);
    double max1=dat1z(oo,0);
    for (int jj = 1; jj < ncols; jj++){
        max1=max1+dat1z(oo,jj);
        res(oo,jj)=max1;
    }
  }

  return res;
}
```