

```

/*
* Code is of the file DepdenetPartitionsAR1LikeAR1Theta_sPPM2.c
* that one since the R function drpm_fit calls the function which
* is defined there:
  initial_partition <- 0
  C.out <- .C("drpm_ar1_sppm", // defined in that file
            as.integer(draws), as.integer(burn), as.integer(thin),
            as.integer(nsubject), as.i)
* so it should be the "main" file
*/

```

```

*****
* Copyright (c) 2018 Garritt Leland Page
*
* This file contains C code for an MCMC algorithm constructed
* to fit a hierarchical model that incorporates the idea of
* temporally dependent partitions.
*
* I will include model details at a later date
*
*****
```

```

#include "matrix.h"
#include "Rutil.h"
#include <R_ext/Lapack.h>
#include <R.h>
#include <Rmath.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
*****
* The following are the inputs of the function that are read from R
*
* draws = total number of MCMC draws
* burn = number of MCMC draws discarded as burn-in
* thin = indicates how much MCMC chain should be thinned
* nsubject = integer for the number of subjects in data set
* ntime = integer for the number of time points
* y = double nsubject x ntime matrix containing response for each
  subject at time t
* s1 = nsubject x 1 vector containing spatial coordinate one
* s2 = nsubject x 1 vector containing spatial coordinate two
*
* M = double indicating value of M associated with cohesion (scale
  parameter of DP).
* alpha = double - prior probability of being pegged, starting value
```

Estime dataset well look like this:

```

only if update_alpha is TRUE
* priorvals = vector containing values for prior distributions as
  follows
*
* time_specific_alpha = integer - logical indicating whether to make
  alpha time-specific or one global alpha.
* update_alpha = integer - logical indicating whether to update alpha
  or not.
* update_eta1 = integer - logical indicating whether to update eta1
  or set it to zero for all subjects.
* update_phi1 = integer - logical indicating whether to update phi1
  or set it to zero.
* sPPM = integer - logical indicating whether to use spatial
  information or not
*
* SpatialCohesion = integer indicating which cohesion to use
* 1 -Auxiliary
* 2 - Double dipper
*
* cParms - vector holding values employed in the cohesion
*
* OUTPUT
* Si -
* mu -
* sig2 -
* eta1 -
* theta -
* tau2 -
* phi0 -
* phi1 -
* gamma -
* alpha.out -
* like -
* lpml -
* waic -
*****
void drpm_ar1_sppm(int *draws, int *burn, int *thin, int *nsubject,
int *ntime,
  double *y, double *s1, double *s2, double *M,
  double *alpha, double *modelPriors, double *alphaPriors,
  int *time_specific_alpha,
  int *update_alpha, int *update_eta1, int *update_phi1,
  int *sPPM, int *SpatialCohesion, double *cParms, double *mh,
  int *space_1, int *simpleModel, double *theta_tau2,
  int *Si, double *mu, double *sig2, double *eta1, double
  *theta, double *tau2,
  double *phi0, double *phi1, double *lam2, int *gamma, double
  *alpha_out,
```

*parameters in results
impostos in FALSE
impostos wrong in
the code/written in*

```

double *fitted, double *llike, double *lpml, double *waic){

// i - MCMC iterate
// ii - MCMC iterate that is saved
// j - subject iterate
// jj - second subject iterate
// t - time iterate
// k - cluster iterate
// kk - second cluster iterate
// p - prediction iterate

int i, ii, j, jj, t, k, kk;
ii = 0;

int nout = (*draws - *burn) / (*thin);
Rprintf("nsubject = %d\n", *nsubject);
Rprintf("ntime = %d\n", *ntime);
Rprintf("nout = %d\n", nout);
Rprintf("update_alpha = %d\n", *update_alpha);
Rprintf("update_eta1 = %d\n", *update_eta1);
Rprintf("update_phi1 = %d\n", *update_phi1);

=====
// =====
// Memory vectors to hold MCMC iterates for non cluster specific
parameters
//
// This variable is used to create a "buffer" zone of memory so
that updating
// things on time boundary do not need special attention in the
algorithm since
// I have to look at time period before and after when updating
partitions
int ntime1 = *ntime + 1;
// I am adding one more year as an empty vector
// so that the C program does not crash.
int gamma_iter[(*nsubject)*(ntime1)];
int Si_iter[(*nsubject)*(ntime1)];
int nclus_iter[ntime1];
double *eta1_iter = R_VecorInit(*nsubject, runif(0,1));
double *theta1_iter = R_VecorInit(ntime1, rnorm(0,3));
double *tau2_iter = R_VecorInit(ntime1, runif(0,
modelParams[?]*modelParams[?]));

```

```

// Initialize Si according to covariates
// I am adding one time period here to have
// scratch memory (never filled in) so that
// I can avoid dealing with boundaries in algorithm
for(j = 0; j < *nsubject; j++){
    for(t = 0; t < ntime1; t++){ // Note I am not initializing the
"added time memory"
        Si_iter[j*(ntime1) + t] = 1;
        gamma_iter[j*(ntime1) + t] = 0;
        nh[j*(ntime1) + t] = 0;
        if(t==1) Si_iter[j*ntime1 + t] = 1;
        if(t==ntime) Si_iter[j*(ntime1) + t] = 0;
    }
}

// Initial enumeration of number of subjects per cluster;
for(j = 0; j < *nsubject; j++){
    for(t = 0; t < ntime; t++){
        nh[(Si_iter[j*(ntime1)+t]-1)*(ntime1) + t] = nh[(Si_iter[j*(ntime1)+t-1]*(ntime1) + t) + 1];
    }
}

```

$t=1 \Rightarrow$ two > values
 $t=2 \Rightarrow$ 2 clusters
 $t=2 \Rightarrow$ three > values
 $t=2 \Rightarrow$ 3 clusters

$$m_{\theta} = \begin{pmatrix} 8 & 6 & 6 \\ 2 & 3 & 2 \\ 0 & 4 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

```

} // Initialize the number of clusters
for(t = 0; t < *ntime; t++){
  nclus_iter[t] = 0;
  for(j = 0; j < *nsubject; j++){
    if(nh[j*(ntime1) + t] > 0) nclus_iter[t] = nclus_iter[t] + 1;
  }
  nclus_iter[t] += (me[j, t] > 0);
}
nclus_iter[*ntime] = 0;

// =====
// scratch vectors of memory needed to update parameters
// =====
// stuff needed to update gamma vectors
int nclus_red=0, nh_red[*nsubject], n_red=0, gt, n_red_1=0, cit_1;
int nh_redtmp[*nsubject], nh_tmp[*nsubject];
int nh_redtmp_no_zero[*nsubject],
nh_tmp_no_zero[*nsubject], nh_red_no_zero[*nsubject];
int nh_red_1[*nsubject];
// int nclus_red_1;
int nh_redtmp_1[*nsubject], nh_tmp_1[*nsubject];
int nh_redtmp_no_zero_1[*nsubject],
nh_red_no_zero_1[*nsubject], nh_tmp_no_zero_1[*nsubject];
double *s1_red = R_VectorInit(*nsubject, 0.0);
double *s2_red = R_VectorInit(*nsubject, 0.0);
for(j=0; j<*nsubject; j++){
  nh_tmp[j] = 0; nh_red[j] = 0; nh_redtmp[j] = 0;
  nh_redtmp_no_zero[j] = 0; nh_tmp_no_zero[j] = 0;
  nh_red_no_zero[j] = 0; nh_red_no_zero_1[j] = 0;
  nh_tmp_1[j] = 0; nh_red_1[j] = 0; nh_redtmp_1[j] = 0;
  nh_redtmp_no_zero_1[j] = 0; nh_red_no_zero_1[j] = 0;
  nh_tmp_no_zero_1[j] = 0;
}
// stuff that I need to update Si (the partition);
int compit[(*nsubject)], comptm1[(*nsubject)], comp2t[(*nsubject)], comptp1[(*
int rho_tmp[*nsubject], Si_tmp[*nsubject], Si_tmp2[*nsubject];
int Si_red[*nsubject], Si_red_1[*nsubject];
int oldLab[*nsubject], reorder[*nsubject];
int iaux, Rindx1, Rindx2, n_tmp, nclus_tmp, rho_comp, indx;
double auxm, auxs, mudraw, sigdraw, maxph, denph, cprobh, uu, lCo,
lCn, lCn_1, lpp;
  
```

```

double *ph = R_VectorInit(*nsubject, 0.0);
double *phtmp = R_VectorInit(*nsubject, 0.0);
double *probh = R_VectorInit(*nsubject, 0.0);
double *lgweight = R_VectorInit(*nsubject, 0.0);
double *s1o = R_Vector(*nsubject);
double *s2o = R_Vector(*nsubject);
double *s1n = R_Vector(*nsubject);
double *s2n = R_Vector(*nsubject);
for(j=0; j<(*nsubject); j++){
  comp1t[j] = 0; comptm1[j] = 0, comp2t[j]=0, comptp1[j]=0;
}
// stuff I need to update eta1
double e1o, e1n, logito, logitn, one_phisq;
// stuff I need to update muh and sig2h
double mstar, s2star, sumy, sume2;
double nsig, osig, llo, lln, llr;
double *mu_tmp = R_VectorInit(*nsubject, 0.0);
double *sig2_tmp = R_VectorInit(*nsubject, 1.0);

// stuff that I need for theta and lam2
double summu, nt, ot, lam2tmp, phi1sq, sumt, op1, np1, ol, nl;
// double ssq;

// stuff that I need to update alpha
int sumg;
double astar, bstar, alpha_tmp;

// Stuff to compute lpml, likelihood, and WAIC
int like0, nout_0=0;
double lpml_iter, elppdWAIC;
double *CPO = R_VectorInit((*nsubject)*(ntime1), 0.0);
double *like_iter = R_VectorInit((*nsubject)*(ntime1), 0.0);
double *fitted_iter = R_VectorInit((*nsubject)*(ntime1), 0.0);
double *mnlike = R_VectorInit((*nsubject)*(ntime1), 0.0);
double *mnllike = R_VectorInit((*nsubject)*(ntime1), 0.0);
// stuff to predict
// int gpred[*nsubject], nh_pred[*nsubject];
// =====
// Prior parameter values
// =====
// prior values for sig2
double Asig=modelPriors[2];
double Atau=modelPriors[3];
double Alam=modelPriors[4];
  
```

```

// priors for phi0
double m0 = modelPriors[0], s00 = modelPriors[1];
// priors for alpha
double a = alphaPriors[0], b = alphaPriors[1];
// priors for eta1
double b_eta1 = modelPriors[5];

// DP weight parameter
double Mdp = *M;
Rprintf("Prior values: Asig = %.2f, Atau = %.2f, Alam = %.2f, \n
m0 = %.2f, s00 = %.2f\n", Asig, Atau, Alam, m0, s00);
// Cohesion auxiliary model parameters for Cohesions 3 and 4
double k0=cParms[1], v0=cParms[2];
double *mu0 = R_VectorInit(2,cParms[0]);
double *L0 = R_VectorInit(2*2,0.0);
L0[0] = cParms[3]; L0[3] = cParms[3];

if(*sPPM==1){
  RprintVecAsMat("mu0", mu0, 1, 2);
  Rprintf("k0 = %f\n", k0);
  Rprintf("v0 = %f\n", v0);
  RprintVecAsMat("L0", L0, 2, 2);
}

// M-H step tuning parameter
double csigSIG=mh[0], csigTAU=mh[1], csigLAM=mh[2], csigETA1=mh[3],
csigPHI1=mh[4];
GetRNstate();
// =====
// start of the mcmc algorithm;
// =====
=====

for(i = 0; i < *draws; i++){
  if((i+1) % 10000 == 0){
    time_t now;
    time(&now);
    Rprintf("mcmc iter = %d\n", i+1);
    Rprintf("%s", ctime(&now));
  }
  // Start updating gamma and partition for each time period
  for(t = 0; t < *ntime; t++){
    // =====

```

Original labels (Si): 1 3 3 5 5 4 4 1 2 1
 Relabeled groups (Sirelab): 1 2 2 3 3 4 4 1 5 1
 Reordered cluster sizes (nhrelab): 3 2 2 2 1 0 0 0 0 0
 Old labels (oldLab): 1 3 5 4 2 0 0 0 0 0

```

    // begin by updating gamma (pegged) parameters
    //
    // FOR j w/ nsubject
    // at time period one, all gammas are zero (none are
    ``pegged'')
    if(t == 0){
      gamma_iter[j*(ntime1) + t] = 0; Initial = (0 0 0 0) at t=1 we can decide to update the low values
    } else {
      // find the reduced partition information
      // i.e., vector of cluster labels;
      // =====
      Rindx1 = 0;
      for(jj = 0; jj < *nsubject; jj++){
        if(gamma_iter[jj*ntime1 + (t)] == 1){ Filter the units which have stat = 1, we can units comprising R^t
          if(jj != j){ not yet see we are extracting one extra one
            Si_tmp[Rindx1] = Si_iter[jj*ntime1 + (t)]; R^t = {jj : O_{j,t} = 1} well overall we are selecting current cluster and R^t
            Si_tmp2[Rindx1] = Si_iter[jj*ntime1 + (t)];
            comptm1[Rindx1] = Si_iter[jj*ntime1 + (t-1)];
            // Also get the reduced spatial coordinates if space is included
            if(*sPPM==1){
              if(*space_1==1 & t == 0) | (*space_1==0){
                s1_red[Rindx1] = s1[jj];
                s2_red[Rindx1] = s2[jj];
              }
            }
            Rindx1 = Rindx1 + 1; first because we treat the subject i to case about the current subject j (not w)
          }
        }
        Si_tmp2[Rindx1] = Si_iter[j*ntime1 + (t)]; we now also add subject i to some w terms to set and compare
        comptm1[Rindx1] = Si_iter[j*ntime1 + (t-1)]; case i+1 and R^t
        n_red = Rindx1; #subjects in the reduced partition
        n_red_1 = Rindx1 + 1; reduced partition
        relabel(Si_tmp, *nsubject, Si_red, reorder, oldLab); Si_red = Sirelab
        relabel(Si_tmp2, *nsubject, Si_red_1, reorder, oldLab); Si_red_1 = Sirelab
        // I need to keep the relabeled cluster label for the
        // individual so that I know what lgweight to keep in the
        // full conditional.
        cit_1 = Si_red_1[Rindx1];
        this should be the cluster label of the last inserted subject, i.e. the one on which we are focusing on
        Si_TMP = [4 2 3 2 2 4 2] (3) last place is on the last added subject
        cotw = Si_TMP[7] = clustering label of unit j that were inserted
      }
    }
  }
  // =====

```

Definition 1. We say that partitions ρ_{t-1} and ρ_t are compatible with respect to γ_t , if ρ_t may be obtained from ρ_{t-1} by reallocating items as indicated by γ_t , that is, those items i such that $\gamma_{it} = 0$ for $i = 1, \dots, m$. Note that the compatibility relation is an equivalence relation.

There is a simple way to check if ρ_{t-1} is compatible with ρ_t with respect to γ_t . Let $\mathfrak{R}_t = \{i : \gamma_{it} = 1\}$ be the collection of units that remain fixed when moving from time $t-1$ to time t , and $\mathfrak{R}_t^C = \{i : \gamma_{it} = 0\}$ is the collection of units that do not. Next denote with $\rho_t^{\mathfrak{R}_t}$ the "reduced" partition at time t that remains after removing all items in \mathfrak{R}_t^C from the subsets of ρ_t . Similarly, let $\rho_{t-1}^{\mathfrak{R}_t}$ be the reduced partition at time $t-1$ based on γ_t . Then ρ_{t-1} and ρ_t are compatible with respect to γ_t if and only if $\rho_{t-1}^{\mathfrak{R}_t} = \rho_t^{\mathfrak{R}_t}$.

$$\Pr(\gamma_{it} = 1 | \rho_t) = \frac{\alpha_i}{\alpha_i + (1 - \alpha_i)\Pr(\rho_t^{\mathfrak{R}_t})/\Pr(\rho_t)} \quad (S.8)$$

$$\Pr(\rho_t^{\mathfrak{R}_t} = \rho_t^{\mathfrak{R}_{t-1}} | \rho_{t-1}) = \frac{\Pr(\rho_t^{\mathfrak{R}_{t-1}})}{\Pr(\rho_t^{\mathfrak{R}_{t-1}}) + \Pr(\rho_t^{\mathfrak{R}_{t-1}})} \quad (S.9)$$

$$\begin{aligned} Si_tmp &= P_t^{R+t-1} \rightarrow Si_red \\ Si_tmp2 &= P_t^{R+t-1} \rightarrow Si_red_1 \\ Comptm1 &= P_t^{R+t-1} \end{aligned}$$

$$n_red = Rindx1;$$

$$n_red_1 = Rindx1 + 1;$$

$$relabel(Si_tmp, *nsubject, Si_red, reorder, oldLab);$$

$$relabel(Si_tmp2, *nsubject, Si_red_1, reorder, oldLab);$$

$$// I need to keep the relabeled cluster label for the$$

$$// individual so that I know what lgweight to keep in the$$

$$// full conditional.$$

$$cit_1 = Si_red_1[Rindx1];$$

$$\text{this should be the cluster label of the}$$

$$\text{last inserted subject, i.e. the one on which}$$

$$\text{we are focusing on}$$

$$Si_TMP = [4 2 3 2 2 4 2] (3)$$

$$\text{last place is on the last added subject}$$

$$cotw = Si_TMP[7] = \text{clustering label of unit j that were inserted}$$

// What if pegged subject creates a singleton in the reduced partition?

```

    lCn_1=0.0;
    if(*sPPM==1){
        if((*space_1==1 & t == 0) | (*space_1==0)){
            s1o[0] = s1[j];
            s2o[0] = s2[j];
            lCn_1 = Cohesion3_4(s1o, s2o, mu0, k0, v0, L0,
1,*SpatialCohesion, 1);  

        }
    }
    lgweight[nclus_red] = Log(Mdp) + lCn_1;
}

```

Si_red_1[Rindx1] = nclus_red+1;
rho_comp = compatibility(Si_red_1, comptm1, Rindx1+1);
if(rho_comp == 0) lgweight[nclus_red] = log(0);

denph = 0.0;
for(k = 0; k < nclus_red + 1; k++){
 phtmp[k] = lgweight[k];
}
R_rsort(phtmp, nclus_red + 1);
maxph = phtmp[nclus_red];
denph = 0.0;
for(k = 0; k < nclus_red + 1; k++){
 lgweight[k] = exp(lgweight[k] - maxph);
 denph = denph + lgweight[k];
}
for(k = 0; k < nclus_red + 1; k++){
 lgweight[k] = lgweight[k]/denph;
}

probh[1] = alpha_iter[t]/(alpha_iter[t] + (1-
alpha_iter[t])*lgweight[cit_1-1]);
// If gamma is 1 at current MCMC iterate, then there are no
// concerns about partitions being incompatible as gamma

```

// Note that if time_specific_alpha is false, then
// alpha[t] is the same for all value of t
if(*unit_specific_alpha==1){
    probh[1] = alpha_iter[j*(ntime1) + t]/
        (alpha_iter[j*(ntime1) + t] + (1-alpha_iter[j*(ntime1) + t])*lgweight[cit_1-1]);
} else {
    probh[1] = alpha_iter[t]/(alpha_iter[t] + (1-alpha_iter[t])*lgweight[cit_1-1]);
}

```

```

changes
    // from 1 to 0.
    //
    // However, if gamma's current value is 0, then care must
be taken when
    // trying to change from gamma=0 to gamma=1 as the
partitions may
    // no longer be compatible
    if(gamma_iter[j*(ntime1) + t] == 0){

        // To determine compatibility, I need to make sure that
        // comparison of the reduced partitions is being made with
        // correct cluster labeling. I try to do this by
identifying
        // the sets of units and sequentially assigning "cluster
labels"
        // starting with set that contains the first unit. I
wonder if
        // there is a way to do this in C without using loops?

        // can I ask about this?
        // Get rho_t | gamma_t = 1 and rho_{t-1} | gamma_t = 1
        // when gamma_{it} = 1;
        Rindx1 = 0;
        for(jj = 0; jj < *nsubject; jj++){
            if(gamma_iter[jj*ntime1 + (t)] == 1){
                comptm1[Rindx1] = Si_iter[jj*ntime1 + (t-1)];
                comp1[Rindx1] = Si_iter[jj*ntime1 + (t)];
                Rindx1 = Rindx1 + 1;
            }
            // I need to include this because determine what
happens when
            // none code therefore we
            // unless we
            // well where test
            // we update up be
            // we assume to
            // insert unit i
            // as the last one
        }
        rho_comp = compatibility(comptm1, comp1, Rindx1);
        // If rho_comp = 0 then not compatible and probability of
        // pegging subject needs to be set to 0;
        if(rho_comp==0){
            probh[1] = 0;
        }
    }
}

```

*- recompute
 $\text{COMP}_{t-1} = P_{t-1}$
 $\text{COMP}_t = P_t$
 leaving on all
 subjects in our
 2: subjects
 - check convergence
 and if not met
 set $\text{Rho}_{t+1} = 0$
 (to enable to
 handle ∞)*

*well where test
 we update up be
 we assume to
 insert unit i
 as the last one*

```

 $\tilde{\gamma}_t^{(1)} \sim \text{Bin}(1, \text{probh}[1])$ 
gt = rbinom(1, probh[1]);
gamma_iter[j*(ntime1) + t] = gt;
} // end of the else case of  $t > 1$ 
} // end of the for loop
////////////////////////////////////////////////////////////////
// update partition
//
////////////////////////////////////////////////////////////////
// The cluster probabilities depend on four partition
probabilities
//
// rho_t
// rho_t.REDUCED
// rho_t+1
// rho_t+1.REDUCED
//
// I have switched a number of times on which of these needs to
be computed
// and which one can be absorbed in the normalizing constant.
Right now I am
// leaning towards  $\text{Pr}(\rho_{t+1})$  and  $\text{Pr}(\rho_{t+1}.R)$  can be
absorbed. But I need
// to use  $\rho_t.R$  and  $\rho_{t+1}.R$  to check compatibility as I
update rho_t.
//

```

*Only handle the
 new σ_t value*

```

// The cluster probabilities depend on four partition probabilities
// rho_t
// rho_t.REDUCE
// rho_t+1
// rho_t+1.REDUCE
//
// I have switched a number of times on which of these needs to be computed
// and which one can be absorbed in the normalizing constant. Right now I am
// leaning towards Pr(rho_t+1) and Pr(rho_t+1.R) can be absorbed. But I need
// to use rho_t.R and rho_t+1.R to check compatibility as I update rho_t.
//
for(jj = 0; jj < *nsubject; jj++){
    rho_tmp[jj] = Si_iter[jj*(ntime1) + t]; Ptmp = Pt
}

// It seems to me that I can use some of the structure used to carry
// out Algorithm 8 from previous code to keep track of empty clusters
// etc.
for(j = 0; j < *nsubject; j++){
    // Only need to update partition relative to units that are not pegged
    //
    // Note that if a centering partition is supplied then gamma = 1 for all units and the
    // first entry of Si_iter is never updated and so this part of codes is never executed
    // for t=0 when rho0 is supplied. (we do this for t=0)
    if(gamma_iter[j*(ntime1) + t] == 0){
        if(nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] > 1){
            [rhot, t] = rhot at time t
            // Observation belongs to a non-singleton ...
            nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] = nh[(Si_iter[j*(ntime1) + t]-1)*
            (ntime1) + t] [t] [we remove (unlike?) unit i from the
            center to which we belong, so we update one more]
        } else{
            // Observation is a member of a singleton cluster ...
            iaux = Si_iter[j*(ntime1) + t]; label of unit j(t)
            if(iaux < nclus_iter[t]){
                // Need to relabel clusters. I will do this by swapping cluster labels
                // Si_iter[j] and nclus_iter along with cluster specific parameters;
                [we swap the current label (the one of j)
                and the last cluster label]
                // All members of last cluster will be assigned subject j's label
                for(jj = 0; jj < *nsubject; jj++){
                    if(Si_iter[jj*(ntime1) + t] == nclus_iter[t]){
                        Si_iter[jj*(ntime1) + t] = iaux;
                    }
                }
                [subject j
                new: 1 2 3 2 3 2 2
                label: 1 2 3 2 3 2 2] [cluster label: 1 2 3 2 3 2 2] [here we end the swap
                because nclus_iter[t]=a]
            }
            Si_iter[j*(ntime1) + t] = nclus_iter[t];
            // The following steps swaps order of cluster specific parameters
            // so that the newly labeled subjects from previous step retain
            // their correct cluster specific parameters
            auxs = sig2h[(iaux-1)*ntime1 + t];
            sig2h[(iaux-1)*ntime1 + t] = sig2h[(nclus_iter[t]-1)*(ntime1)+t];
            sig2h[(nclus_iter[t]-1)*(ntime1)+t] = auxs;

            auxm = muh[(iaux-1)*ntime1 + t];
            muh[(iaux-1)*ntime1 + t] = muh[(nclus_iter[t]-1)*(ntime1)+t];
            muh[(nclus_iter[t]-1)*(ntime1)+t] = auxm;
        }
    }
}

```

```

// the number of members in cluster is also swapped with the last
nh[(iaux-1)*(ntime1)+t] = nh[(nclus_iter[t]-1)*(ntime1)+t];
nh[(nclus_iter[t]-1)*(ntime1)+t] = 1;
}

// Now remove the ith obs and last cluster;
nh[(nclus_iter[t]-1)*(ntime1)+t] = nh[(nclus_iter[t]-1)*(ntime1)+t] - 1;
nclus_iter[t] = nclus_iter[t] - 1;
} end of the ELSE

for(jj = 0; jj < *nsubject; jj++){
    rho_tmp[jj] = Si_iter[jj*(ntime1) + t]; recompute Ptmp or
    the updated by with
    unit j removed and
    clusters corrected
}
for(k = 0; k < nclus_iter[t]; k++){
    rho_tmp[j] = K[k]; for each cluster w we taxonomy,
    we have to assign unit; to them
}

// First need to check compatibility
Rindx2=0;
for(jj = 0; jj < *nsubject; jj++){
    if(gamma_iter[jj*(ntime1) + (t+1)] == 1){
        comp2t[Rindx2] = rho_tmp[jj]; Pt (#j=a)
        comptp1[Rindx2] = Si_iter[jj*(ntime1) + (t+1)];
        Rindx2 = Rindx2 + 1; Pt
    }
}
// check for compatibility
rho_comp = compatibility(comp2t, comptp1, Rindx2);
if(rho_comp != 1){
    ph[k] = log(0); // Not compatible
} else {
    // Need to compute Pr(rhot), Pr(rhot.R), Pr(rhot+1), Pr(rhot+1.R)

    for(jj = 0; jj < *nsubject; jj++){
        nh_tmp[jj] = 0;
    }
    n_tmp = 0;
    for(jj = 0; jj < *nsubject; jj++){
        nh_tmp[rho_tmp[jj]-1] = nh_tmp[rho_tmp[jj]-1]+1;
    }
    n_tmp=n_tmp+1;
}

nclus_tmp=0;
for(jj = 0; jj < *nsubject; jj++){
    if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
}

lpp = 0.0;
for(kk = 0; kk < nclus_tmp; kk++){
    // Beginning of spatial part
    lCn = 0.0;
    if(*sPPM==1){
        if((*space_1==1 & t == 1) | (*space_1==0)){
            indx = 0;
            for(jj = 0; jj < *nsubject; jj++){
                if(rho_tmp[jj] == kk+1){
                    s1n[indx] = s1[jj];
                    s2n[indx] = s2[jj];
                    indx = indx+1;
                }
            }
            take the max of cluster in
            the units of cluster h
        }
    }
}

```

```

        lCn = Cohesion3_4(s1n, s2n, mu0, k0, v0, L0, nh_tmp[kk], *SpatialCohesion, 1);
    }
}

// End of spatial part

// lpp = lpp + nclus_tmp*log(Mdp) + lgamma((double) nh_tmp[kk]) + lCn;
lpp = lpp + nh_tmp[kk]*log(Mdp) + lgamma((double) nh_tmp[kk]) + lCn;
lpp = lpp + (Log(Mdp) + lgamma((double) nh_tmp[kk]) + lCn);

}

if(t==1){// t=1 is first time point after centering partition
    ph[k] = dnorm(y[j*(ntime) + t],
    muh[k*(ntime1) + t],
    sqrt(sig2h[k*(ntime1) + t]), ① + the last argument can ignore "one-sec"
    lpp; centering is "one-sec"

}

if(t > 1){// Do I want there to be temporal correlation between rho0 and
rho1?
    ph[k] = dnorm(y[j*(ntime) + t],
    muh[k*(ntime1) + t] + eta1_iter[j]*y[j*(ntime) + t-1],
    sqrt(sig2h[k*(ntime1) + t]*(1-eta1_iter[j]*eta1_iter[j])), 1)+
    lpp;
}

// use this to test if MCMC draws from prior are correct
ph[k] = lpp;

}

see we ended the loop on the  

clusters you have now: nclus  

so now we can consider the case  

of just 1 cluster as we have seen  

↑  

// Determine if E_U gets allocated to a new cluster  

// Need to check compatibility first  

evaluated unit?  

rho_tmp[j] = nclus_iter[t]+1;

// First need to check compatibility
Rindx1 = 0, Rindx2=0;
for(jj = 0; jj < *nsubject; jj++){
    if(gamma_iter[jj*ntime1 + (t+1)] == 1){
        comp2t[Rindx2] = rho_tmp[jj];
        comptp1[Rindx2] = Si_iter[jj*ntime1 + (t+1)];
        Rindx2 = Rindx2 + 1;
    }
}
// check for compatibility
rho_comp = compatibility(comp2t, comptp1, Rindx2);

if(rho_comp != 1){
    ph[nclus_iter[t]] = Log(0); // going to own cluster is not compatible;
} else {
    mudraw = rnorm(theta_iter[t], sqrt(tau2_iter[t]));
    sigdraw = runif(0, Asig);
    for(jj = 0; jj < *nsubject; jj++){
        nh_tmp[jj] = 0;
    }
    n tmp = 0;
}

```

```

for(jj = 0; jj < *nsubject; jj++){
    nh_tmp[rho_tmp[jj]-1] = nh_tmp[rho_tmp[jj]-1]+1;
    n_tmp=n_tmp+1;
}

nclus_tmp=0;
for(jj = 0; jj < *nsubject; jj++){
    if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
}

lpp = 0.0;
for(kk = 0; kk < nclus_tmp; kk++){

// Beginning of spatial part
lCn = 0.0;
if(*sPPM==1){
    if((*space_1==1 & t == 1) | (*space_1==0)){
        indx = 0;
        for(jj = 0; jj < *nsubject; jj++){
            if(rho_tmp[jj] == kk+1){

                s1n[indx] = s1[jj];
                s2n[indx] = s2[jj];
                indx = indx+1;
            }
        }
        lCn = Cohesion3_4(s1n, s2n, mu0, k0, v0, L0, nh_tmp[kk],*SpatialCohesion, 1);
    }
}
// End of spatial part

lpp = lpp + (Log(Mdp) + lgamma((double) nh_tmp[kk]) + lCn);
lpp = lpp + nh_tmp[kk]*log(Mdp) + lgamma((double) nh_tmp[kk]) + lCn;
}

if(t==1){
    ph[nclus_iter[t]] = dnorm(y[j*(*ntime) + t], mudraw, sigdraw, 1) +
        lpp;
}
if(t > 1){
    ph[nclus_iter[t]] = dnorm(y[j*(*ntime) + t],
        mudraw + eta1_iter[j]*y[j*(*ntime) + t-1],
        sigdraw*sqrt(1-eta1_iter[j]*eta1_iter[j]), 1) +
        lpp;
}
// Now compute the probabilities
for(k = 0; k < nclus_iter[t]+1; k++) phtmp[k] = ph[k];

R_rsort(phtmp, nclus_iter[t]+1) ;

maxph = phtmp[nclus_iter[t]];

denph = 0.0;
for(k = 0; k < nclus_iter[t]+1; k++){
    ph[k] = exp(ph[k] - maxph); ~~~ convert to weights (from
    denph = denph + ph[k]; tile 0s - weights) and the
} ~~~ center them using the
      max weight moves
} ~~~ accumulate the sum

```

```

for(k = 0; k < nclus_iter[t]+1; k++){
    probh[k] = ph[k]/denph; normalize all the molecules
}

uu = runif(0.0,1.0); we take in mu(0,4)
cprobh = 0.0; initialise the cumulative prob
for(k = 0; k < nclus_iter[t]+1; k++){
    cprobh = cprobh + probh[k];
    if(uu < cprobh){
        iaux = k+1; we can tell label w/t
        break;
    }
}
if we are assigned to one exisiting cluster we update the related stuff (mu and sigma)
if(iaux <= nclus_iter[t]){
    Si_iter[j*(ntime1) + t] = iaux;
    nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1)+t] = nh[(Si_iter[j*(ntime1) + t]-1)*
(ntime1)+t] + 1;
    rho_tmp[j] = iaux; we are assigned to a new cluster
    }else{
        nclus_iter[t] = nclus_iter[t] + 1; - inc by 1
        Si_iter[j*(ntime1) + t] = nclus_iter[t]; - update mu
        nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1)+t] = 1; - set to the new cluster
        rho_tmp[j] = nclus_iter[t]; - assigns to the drawn values
    }

    muh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] = mudraw;
    sig2h[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] = sigdraw*sigdraw;
    if(*simpleModel==1) sig2h[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] = 1.0;
}
end of the wif(t==0)

for(jj = 0; jj < *nsubject; jj++){
    Si_tmp[jj] = Si_iter[jj*(ntime1) + t]; assign pmk to the current position
    Si_tmp2[jj] = 0;
    reorder[jj] = 0;
}
I believe that I have to make sure that groups are order so that EU one is always in the group one, and then the smallest index not with group 1 anchors group 2 etc.

relabel(Si_tmp, *nsubject, Si_tmp2, reorder, oldLab); relabel/connect S1 to S2
(as the new label? connects next molecule to the previous of C?)

for(jj=0; jj<*nsubject; jj++){
    Si_iter[jj*(ntime1) + t] = Si_tmp2[jj];
}
re-set the original pmk to the water with the connected S1/S2

for(k = 0; k < nclus_iter[t]; k++){
    mu_tmp[k] = muh[k*(ntime1)+t];
    sig2_tmp[k] = sig2h[k*(ntime1)+t];
}
int usage in old the code
=> we can (uncomment), the whole?
the current clusters > assigns ya and he
for(k = 0; k < nclus_iter[t]; k++){
    nh[k*(ntime1)+t] = reorder[k];
    muh[k*(ntime1)+t] = mu_tmp[(oldLab[k]-1)];
    sig2h[k*(ntime1)+t] = sig2_tmp[(oldLab[k]-1)];
}
end of the for j with o: nsubjects

for(j = 0; j < *nsubject; j++){
    Si_tmp[j] = Si_iter[j*(ntime1) + t];
    Si_tmp2[j] = 0;
    reorder[j] = 0;
}
I believe that I have to make sure that groups are order so that EU one is always in the group one, and then the smallest index not with group 1 anchors group 2 etc.

```

this part seems useless to do wt should be stored all right from the iteration on the last subject

```

    relabel(Si_tmp, *nsubject, Si_tmp2, reorder, oldLab);
    for(j=0; j<*nsubject; j++){
        Si_iter[j*(ntime1) + t] = Si_tmp2[j];
    }
    for(k = 0; k < nclus_iter[t]; k++){
        muh[k] = muh[k*(ntime1)+t];
        sig2h[k] = sig2h[k*(ntime1)+t];
    }
    for(k = 0; k < nclus_iter[t]; k++){
        nh[k*(ntime1)+t] = reorder[k];
        muh[k*(ntime1)+t] = muh[(oldLab[k]-1)];
        sig2h[k*(ntime1)+t] = sig2h[(oldLab[k]-1)];
    }
}

// for(k = 0; k < nclus_iter[t]; k++) sig2h[k*(ntime1)+t] = 1.0;
for(k = 0; k < nclus_iter[t]; k++){
    //////////////////////////////////////////////////////////////////
    // update muh
    //////////////////////////////////////////////////////////////////
    if(t==1){
        sumy = 0.0;
        for(j = 0; j < *nsubject; j++){
            if(Si_iter[j*(ntime1) + t] == k+1){
                sumy = sumy + y[j*(ntime)+t];
            }
        }
        s2star = 1/((double) nh[k*(ntime1)+t]/sig2h[k*(ntime1) + t] + 1/tau2_iter[t]);
        mstar = s2star*( (1/sig2h[k*(ntime1) + t])*sumy + (1/tau2_iter[t])*theta_iter[t]);
    }
    if(t > 1){
        sumy = 0.0;
        sume2 = 0.0;
        for(j = 0; j < *nsubject; j++){
            if(Si_iter[j*(ntime1) + t] == k+1){
                sume2 = sume2 + 1.0/(1-eta1_iter[j]*eta1_iter[j]);
                sumy = sumy + (y[j*(ntime)+t] - eta1_iter[j]*y[j*(ntime)+t-1])/
                    (1-eta1_iter[j]*eta1_iter[j]);
            }
        }
        s2star = 1/((1.0/sig2h[k*(ntime1) + t])*sume2 + 1/tau2_iter[t]);
        mstar = s2star*( (1.0/sig2h[k*(ntime1) + t])*sumy + (1/tau2_iter[t])*theta_iter[t]);
    }
    muh[k*(ntime1) + t] = rnorm(mstar, sqrt(s2star));
    //////////////////////////////////////////////////////////////////
    // update sig2h
    //////////////////////////////////////////////////////////////////
    osig = sqrt(sig2h[k*(ntime1) + t]);
    nsig = rnorm(osig,csigSIG);
    if(nsig > 0.0 & nsig < Asig){

        lln = 0.0;
        llo = 0.0;
        if(t == 1){
            for(j = 0; j < *nsubject; j++){
                if(Si_iter[j*(ntime1) + t] == k+1){
                    llo = llo + dnorm(y[j*(ntime)+t], muh[k*(ntime1) + t], osig,1);
                }
            }
        }
    }
}

```

```

    lln = lln + dnorm(y[j*(ntime)+t], muh[k*(ntime1) + t], nsig,1);
}
}

if(t > 1){
    for(j = 0; j < *nsubject; j++){
        if(Si_iter[j*(ntime1) + t] == k+1){
            llo = llo + dnorm(y[j*(ntime)+t], muh[k*(ntime1) + t] +
                eta1_iter[j]*y[j*(ntime) + t-1],
                osig*sqrt(1-eta1_iter[j]*eta1_iter[j]),1);
            lln = lln + dnorm(y[j*(ntime)+t], muh[k*(ntime1) + t] +
                eta1_iter[j]*y[j*(ntime) + t-1],
                nsig*sqrt(1-eta1_iter[j]*eta1_iter[j]),1);
        }
    }
}

llo = llo + dunif(osig, 0.0, Asig, 1);
lln = lln + dunif(nsig, 0.0, Asig, 1);
// llo = llo + dgamma(osig*osig, 10, 0.1, 1);
// lln = lln + dgamma(nsig*nsig, 10, 0.1, 1);

llr = lln - llo;
uu = runif(0,1);

if(log(uu) < llr){
    sig2h[k*(ntime1) + t] = nsig*nsig;
}
if(*simpleModel==1) sig2h[k*(ntime1) + t] = 1.0;
}

//////////////////////////////          //
// update theta (mean of mh)      //
//                                //
//////////////////////////////          //
summu = 0.0;
for(k = 0; k < nclus_iter[t]; k++){
    summu = summu + muh[k*(ntime1) + t];
}

phi1sq = phi1_iter*phi1_iter;
lam2tmp = lam2_iter*(1.0 - phi1sq);

if(t==1){
    s2star = 1.0/((double) nclus_iter[t]/tau2_iter[t] + 1.0/lam2_iter + phi1sq/lam2tmp);
    mstar = s2star*( (1.0/tau2_iter[t])*summu +
        (1.0/lam2_iter)*phi0_iter +
        (1.0/lam2tmp)*phi1_iter*(theta_iter[t+1]-phi0_iter*(1-phi1_iter)));
} else if(t==(*ntime-1)){
    s2star = 1.0/((double) nclus_iter[t]/tau2_iter[t] + 1.0/lam2tmp);
    mstar = s2star*((1.0/tau2_iter[t])*summu +
        (1.0/lam2tmp)*(phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-1]));
} else {
    s2star = 1.0/((double) nclus_iter[t]/tau2_iter[t] + (1.0 + phi1sq)/lam2tmp);
    mstar = s2star*( (1.0/tau2_iter[t])*summu +
        (1.0/lam2tmp)*(phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-1]));
}

(theta_iter[t] = rnorm(mstar, sqrt(s2star));
if(*simpleModel==1) theta_iter[t] = theta_tau2[0];
//////////////////////////////          //
//                                //
// update tau2 (variance of mh)   //
//                                //
//////////////////////////////          //
ot = sqrt(tau2_iter[t]);
nt = rnorm(ot,csigTAU);
if(nt > 0){

    lln = 0.0;
    llo = 0.0;
    for(k = 0; k < nclus_iter[t]; k++){
        llo = llo + dnorm(muh[k*(ntime1) + t], theta_iter[t], ot,1);
        lln = lln + dnorm(muh[k*(ntime1) + t], theta_iter[t], nt,1);
    }

    llo = llo + dunif(nt, 0.0, Atau, 1);
    lln = lln + dunif(nt, 0.0, Atau, 1);

    llr = lln - llo;
    uu = runif(0,1);

    if(log(uu) < llr){
        tau2_iter[t] = nt*nt;
    }
    if(*simpleModel==1) tau2_iter[t] = theta_tau2[1];
}
}

//////////////////////////////          //
//                                //
// update eta1 (temporal correlation parameter at likelihood)  //
//                                //
//////////////////////////////          //
if(*update_eta1==1){
    for(j = 0; j < *nsubject; j++){
        e1o = eta1_iter[j];
        e1n = rnorm(e1o, csigETA1);

        if(e1n < 1 & e1n > -1){

            llo=lln=0.0;
            for(t=1; t<ntime; t++){// need to skip the first "Y" as it is a column of zeros
                llo = llo + dnorm(y[j*(ntime)+t],
                    muh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] + e1o*y[j*(ntime)+t-1],
                    sqrt(sig2h[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t]*(1-e1o*e1o)), 1);
            }
            lln = lln + dnorm(y[j*(ntime)+t],
                muh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] + e1n*y[j*(ntime)+t-1],
                sqrt(sig2h[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t]*(1-e1n*e1n)), 1);
        }
    }
}

```

```

logito = Log(0.5*(e1o + 1)) - Log(1 - 0.5*(e1o+1));
logitn = Log(0.5*(e1n + 1)) - Log(1 - 0.5*(e1n+1));

llo = llo + -Log(2*b_eta1) - (1/b_eta1)*fabs(logito - 0.0);
lln = lln + -Log(2*b_eta1) - (1/b_eta1)*fabs(logitn - 0.0);
llr = lln - llo;
uu = runif(0,1);

    if(llr > Log(uu)) eta1_iter[j] = e1n;
}
}

///////////////////////////////
//                                //
// update alpha                  //
//                                //
///////////////////////////////
if(*update_alpha == 1){
if(*time_specific_alpha == 0 & *unit_specific_alpha==0){ // global time and unit

sumg = 0;
for(j = 0; j < *nsubject; j++){
    for(t = 1; t < *ntime; t++){
        sumg = sumg + gamma_iter[j*ntime1 + t];
    }
}
astar = (double) sumg + alphaPriors[0];
bstar = (double) ((*nsubject)*(*ntime-1) - sumg) + alphaPriors[1];

alpha_tmp = rbeta(astar, bstar);
for(t=1;t<*ntime;t++){alpha_iter[t] = alpha_tmp;}
alpha_iter[0] = 1.0;
}
if(*time_specific_alpha == 1 & *unit_specific_alpha==0){ // local time and global
unit
    for(t = 1; t < *ntime; t++){
        sumg = 0;
        for(j = 0; j < *nsubject; j++){
            sumg = sumg + gamma_iter[j*ntime1 + t];
        }
        astar = (double) sumg + alphaPriors[0];
        bstar = (double) ((*nsubject) - sumg) + alphaPriors[1];
        alpha_iter[t] = rbeta(astar, bstar);
    }
    alpha_iter[0] = 1.0;
}
if(*time_specific_alpha == 0 & *unit_specific_alpha==1){ // global time and local
unit
    for(j = 0; j < *nsubject; j++){
        sumg = 0;
        for(t = 1; t < *ntime; t++){
            sumg = sumg + gamma_iter[j*ntime1 + t];
        }
        astar = (double) sumg + alphaPriors[j*2 + 0];
        bstar = (double) ((*ntime-1) - sumg) + alphaPriors[j*2 + 1];
        alpha_iter[j*ntime1 + 1] = rbeta(astar, bstar);
    }
}
if(*time specific alpha == 1 & *unit specific alpha==1){ // local time and local unit
}
}

```

```

for(j = 0; j < *nsubject; j++){
    for(t = 1; t < *ntime; t++){
        sumg = gamma_iter[j*ntime1 + t];
        astar = (double) sumg + alphaPriors[j*2 + 0];
        bstar = (double) ((*ntime-1) - sumg) + alphaPriors[j*2 + 1];
        alpha_iter[j*ntime1 + t] = rbeta(astar, bstar);
    }
}
}

if(*ntime>2){
    ///////////////////////////////////////////////////
    //
    // update phi0
    //
    ///////////////////////////////////////////////////
    phi1sq = phi1_iter*phi1_iter;
    one_phisq = (1-phi1_iter)*(1-phi1_iter);
    lam2tmp = lam2_iter*(1.0 - phi1sq);

    sumt = 0.0;
    for(t=2; t<*ntime; t++){
        sumt = sumt + (theta_iter[t] - phi1_iter*theta_iter[t-1]);
    }

    s2star = 1.0/((*ntime-1)*(one_phisq/lam2tmp) + (1/lam2_iter) + (1/s20));
    mstar = s2star*((1.0-phi1_iter)/lam2tmp)*sumt + (1/lam2_iter)*theta_iter[0] + (1/s20)*m0;

    phi0_iter = rnorm(mstar, sqrt(s2star));

    ///////////////////////////////////////////////////
    //
    // update phi1
    //
    ///////////////////////////////////////////////////
}

if(*update_phi1==1){
    op1 = phi1_iter;
    np1 = rnorm(op1, csigPHI1);

    if(np1 > -1 & np1 < 1){
        llo = 0.0, lln = 0.0;
        for(t=2; t < *ntime; t++){//}

        llo = llo + dnorm(theta_iter[t], phi0_iter*(1-op1) + op1*theta_iter[t-1],
                            sqrt(lam2_iter*(1.0 - op1*op1)), 1);
        lln = lln + dnorm(theta_iter[t], phi0_iter*(1-np1) + np1*theta_iter[t-1],
                            sqrt(lam2_iter*(1.0 - np1*np1)), 1);
    }

    llo = llo + dunif(op1, -1, 1);
    lln = lln + dunif(np1, -1, 1);

    llr = lln - llo;
    if(llr > log(runif(0,1))) phi1_iter = np1;
}
}

```

```

////////// //////////////////////////////////////////////////////////////////
// // update lam2 // //////////////////////////////////////////////////////////////////
// //////////////////////////////////////////////////////////////////
// Update lambda with a MH step
phi1sq = phi1_iter*phi1_iter;

ol = sqrt(lam2_iter);
nl = rnorm(ol, csigLAM);
if(nl > 0.0){
  lln = 0.0;
  llo = 0.0;
  for(t=2; t<*ntime; t++){
    llo = llo + dnorm(theta_iter[t],
                       phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-1], ol*sqrt(1-
phi1sq),1);
    lln = lln + dnorm(theta_iter[t],
                       phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-1], nl*sqrt(1-
phi1sq),1);
  }
  llo = llo + dnorm(theta_iter[0], phi0_iter, ol, 1) + dunif(ol, 0.0, Alam, 1);
  lln = lln + dnorm(theta_iter[0], phi0_iter, nl, 1) + dunif(nl, 0.0, Alam, 1);

  llr = lln - llo;
  uu = runif(0,1);

  if(log(uu) < llr){
    lam2_iter = nl*nl;
  }
}

/*
phi1sq = phi1_iter*phi1_iter;
ssq = 0.0;
for(t=1; t<*ntime; t++){
  ssq = ssq + (theta_iter[t] - (phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-1]))*
  (theta_iter[t] - (phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-
1]));
}
ssq = 1.0/(1.0 - phi1sq)*ssq + (theta_iter[0]-phi0_iter)*(theta_iter[0]-phi0_iter);

astar = 0.5*(*ntime) + 1;
bstar = 0.5*ssq + 1/1;

lam2_iter = 1.0/rgamma(astar, 1/bstar);

//////////////////////////////////////////////////////////////// //////////////////////////////////////////////////////////////////
// // predict partition for new time period // //////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
for(p = 0; p < *npred; p++){
  for(j=0; j<*nsubject; j++){
    nh_pred[j] = 0;
    predSi_iter[j*(*npred) + p] = 0;
  }
}

```

```

if(*update_alpha == 0){
  n_red = 0;
  for(j=0;j<*nsubject;j++){
    gpred[j] = rbinom(1,*alpha);

    if(gpred[j] == 1){
      nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] = nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] + 1;
      n_red = n_red + 1;
    }

    predSi_iter[j*(*npred) + p] = Si_iter[j*(ntime1)+(*ntime)-1];
  }
}

if(*update_alpha == 1){
  if(*time_specific_alpha == 1){

    n_red = 0;
    for(j=0;j<*nsubject;j++){

      gpred[j] = rbinom(1,alpha_iter[1]);

      if(gpred[j] == 1){
        nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] = nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] + 1;
        n_red = n_red + 1;
      }

      predSi_iter[j*(*npred) + p] = Si_iter[j*(ntime1)+(*ntime)-1];
    }
  }
}

remove_zero(nh_pred, *nsubject, nh_tmp_no_zero);

nclus_tmp = 0;
for(j=0; j<*nsubject;j++){
  if(nh_tmp_no_zero[j] > 0){
    nclus_tmp = nclus_tmp + 1;
  }else{
    break;
  }
}

for(j=0;j<*nsubject;j++){
  if(gpred[j] == 0){
    for(k = 0; k < nclus_tmp; k++){
      probh[k] = nh_pred[k]/(n_red + Mdp);
    }
    probh[nclus_tmp] = Mdp/(n_red + Mdp);
  }
}

```

```

uu = runif(0.0,1.0);

cprobh= 0.0;
for(k = 0; k < nclus_tmp+1; k++){
    cprobh = cprobh + probh[k];
    if (uu < cprobh){
        iaux = k+1;
        break;
    }
}

if(iaux <= nclus_tmp){

    predSi_iter[j*(npred) + p] = iaux;
    nh_pred[iaux-1] = nh_pred[iaux-1] + 1;
}else{
    nclus_tmp = nclus_tmp + 1;
    predSi_iter[j*(npred) + p] = nclus_tmp;
    nh_pred[(predSi_iter[j*(npred) + p]-1)*(npred)+p] = 1;
}

n_red = n_red + 1;

}
}

// evaluating likelihood that will be used to calculate LPML and WAIC?
// (see page 81 Christensen Hansen and Johnson)
//



if(i > (*burn-1) & i % (*thin) == 0){

    like0=0;
    for(j = 0; j < *nsubject; j++){
        for(t = 1; t < *ntime; t++){

            mudraw = muh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t];
            sigdraw = sqrt(sig2h[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t]);

            if(t == 1){

                like_iter[j*(ntime)+t] = dnorm(y[j*(ntime)+t], mudraw, sigdraw, 1);
                fitted_iter[j*(ntime)+t] = mudraw;

            }if(t > 1){

                like_iter[j*(ntime)+t] = dnorm(y[j*(ntime)+t],
                    mudraw + eta1_iter[j]*y[j*(ntime)+t-1],
                    sigdraw*sqrt(1-eta1_iter[j]*eta1_iter[j]), 1);

                fitted_iter[j*(ntime)+t] = mudraw + eta1_iter[j]*y[j*(ntime)+t-1];
            }
        }
    }

    // These are needed for WAIC
    mnlike[j*(ntime)+t] = mnlike[j*(ntime)+t] + exp(like_iter[j*(ntime)+t])/(double)nout;
    mnllike[j*(ntime)+t] = mnllike[j*(ntime)+t] + (like_iter[j*(ntime)+t])/(double)nout;

    if(exp(like_iter[j*(ntime)+t]) < 1e-320) like0=1;
}

if(like0==1) nout_0 = nout_0 + 1;

if(like0==0){
    for(j = 0; j < *nsubject; j++){
        for(t = 1; t < *ntime; t++){
            CPO[j*(ntime)+t] = CPO[j*(ntime)+t] + (1/(double) nout)*(1/exp(like_iter[j*(ntime)+t]));
        }
    }
}

// Save MCMC iterates
if((i > (*burn-1)) & ((i+1) % *thin == 0)){
    // Notice that I am not saving the "first" time as it belongs to the
    // vector of zeros added to the data and nothing is updated.
    for(t = 1; t < *ntime; t++){
        if(*unit_specific_alpha==0) alpha_out[ii*(ntime) + t-1] = alpha_iter[t];
        theta[ii*(ntime) + t-1] = theta_iter[t];
        tau2[ii*(ntime) + t-1] = tau2_iter[t];

        for(j = 0; j < *nsubject; j++){
            if(*unit_specific_alpha==1) alpha_out[(ii*(nsubject) + j)*(ntime) + t-1] =
alpha_iter[j*ntime1 + t];
            sig2[(ii*(nsubject) + j)*(ntime) + t-1] = sig2h[(Si_iter[j*(ntime1) + t-1]*(ntime1) + t];
            mu[(ii*(nsubject) + j)*(ntime) + t-1] = muh[(Si_iter[j*(ntime1) + t-1]*(ntime1) + t];
            Si[(ii*(nsubject) + j)*(ntime) + t-1] = Si_iter[j*ntime1 + t];
            gamma[(ii*(nsubject) + j)*(ntime) + t-1] = gamma_iter[j*ntime1 + t];

            llike[(ii*(nsubject) + j)*(ntime) + t-1] = like_iter[j*(ntime)+t];
            fitted[(ii*(nsubject) + j)*(ntime) + t-1] = fitted_iter[j*(ntime)+t];
        }
    }
}

for(j=0; j<*nsubject; j++){
    eta1[ii*(nsubject) + j] = eta1_iter[j];
}
}

```

```

phi1[ii] = phi1_iter;
phi0[ii] = phi0_iter;
lam2[ii] = lam2_iter;

ii = ii+1;

}
/**/
}

lpml_iter=0.0;
for(t = 1; t < *ntime; t++){
for(j = 0; j < *nsubject; j++){

lpml_iter = lpml_iter + Log(1/CP0[j*(*ntime)+t]);

}
}
lpml[0] = lpml_iter;
elppdWAIC = 0.0;

for(j = 0; j < *nsubject; j++){
    for(t = 1; t < *ntime; t++){
        elppdWAIC = elppdWAIC + (2*mnllike[j*(*ntime)+t] - Log(mnllike[j*(*ntime)+t]));
    }
}
waic[0] = -2*elppdWAIC;
PutRNGstate();
}

```


INPUT: draws, draw, tdraw

subject (m)

time (T)

ξ_t, μ_t, σ_t (data and model counts)

X (tile coordinate matrix)

M (bracelet mass parameter)

α (start alpha)

model - draws = [$m_0, \beta_0, A_0, A_1, b_0, \gamma_0$]

α - priors = [a_0, b_0]

time - specific - α

unit - specific - α

update - α

update - γ

update - b

#PPM (true if we remove the model counts)

model X (true if we remove the coordinates)

model - colors

coordinates - unification

colors (vector for the model colors)

zeros: [$\xi_0, \mu_0, \sigma_0, b_0$]

ma (vector for relatives stays update)

cat [0, 1, 2, 3, 4, 5, 6, 7]

space - 4 (more in False alarm, see for

use to expand into a $t=4$ or beyond)

model - model

model if none model was true)

Ω_{22} (all)

ALGORITMO

initialize none zeros

out = (draws - ξ_0)

ww = 0 (no more none iterations)

Sw - wter = 4 (0 ... T)

D - wter = 4 (0 ... T)

ma = 4 (0 ... T)

inclus - wter = 4 (0 ... T)

other core variables are zero

(variables auxiliary for loop)

iteration delle priori

| measure la moltiplicazione

start effettivo delle elaborazioni

for ($i=0, i < t$)

for ($t=0, t < T$)

update α

update ρ

update γ

update β

update θ

update φ

update η

update ω

update τ

update μ

update σ

update β_0

update α_0

update γ_0

update b_0

update ω_0

update τ_0

update μ_0

update σ_0

update β_1

update α_1

update γ_1

update b_1

update ω_1

update τ_1

update μ_1

update σ_1

update β_2

update α_2

update γ_2

update b_2

update ω_2

update τ_2

update μ_2

update σ_2

update β_3

update α_3

update γ_3

update b_3

update ω_3

update τ_3

update μ_3

update σ_3

update β_4

update α_4

update γ_4

update b_4

update ω_4

update τ_4

update μ_4

update σ_4

update β_5

update α_5

update γ_5

update b_5

update ω_5

update τ_5

update μ_5

update σ_5

update β_6

update α_6

update γ_6

update b_6

update ω_6

update τ_6

update μ_6

update σ_6

update β_7

update α_7

update γ_7

update b_7

update ω_7

update τ_7

update μ_7

update σ_7

update

$\text{P TMP} = \text{copy}(\text{C}_t) = \text{copy}(\text{S}_t \text{ with } t, t+7)$

$\text{Cat} (j=0 \cup \text{non subjects})$

we only update the revision relatively to the units team can move (we update $\text{B}_t=0$)

if ($\text{B}_t=0$) \curvearrowright means categories can run for the next 7 days

we now update to remove the unit j to the cluster she was belonging to

if ($\text{unit } j \in \text{a non-modeler cl}$)

well not anymore, we decide now to leave the cluster

but only one day left

else ($\text{unit } j \in \text{a modeler cl}$)

else (err) because the unit j is the last one

remove it as in the if

$\text{P TMP} = \text{copy}(\text{C}_t) = \text{copy}(\text{S}_t \text{ with } t, t+7)$

we now update the unit to be moved

to one of the existing clusters...

$\text{Cat} (h=0, h \in \text{inclusor}, t+7)$

$\text{P TMP}[j, t] = h$

$\text{COMP}_{2t} = \text{P TMP}_{R+t} = \text{P}_t^{R+t} | \text{local}; h = \text{COMP}_{t+4}$

$\text{COMP}_{t+4} = \text{S}_t$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

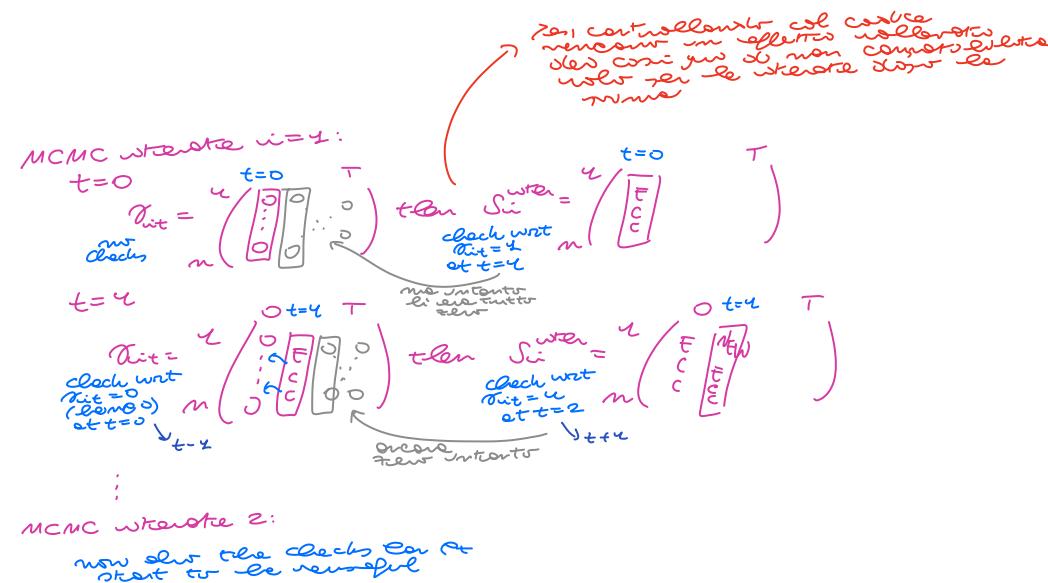
$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

$\Pr(c_{it} = h) = \frac{N(Y_{it} | \mu_{c_{it}, h}, \sigma_{c_{it}, h}^2) \Pr(c_{it} = h)}{N(Y_{it} | \mu_{\text{new}, t}, \sigma_{\text{new}, t}^2) \Pr(c_{it} = h)} \mu_t^{h-1}$

INPUT: draws, draw, term
 subjects (m)
 movie (T)
 ξ, ρ_1, ρ_2 (date and spatial coords)
 X (the covariate matrix)
 M (wavelet mass parameter)
 α (starting alpha)
 $\text{model_prior} = [m_0, \rho_0, A_0, A_1, b_0]$
 $\alpha_prior = [\alpha_0, \beta_0]$
 $\text{true_specific_}\alpha$
 $\text{init_wavelet_}\alpha$
 $\text{update_}\alpha$
 $\text{update_}\gamma_0$
 $\text{update_}\gamma_1$
 $\text{update_}\gamma_2$
 PPM (true if we remove the spatial coords)
 model X (true if we remove the covariates)
 spatial_coherency
 $\text{coherency_uncertainty}$
 cRoms (vector for the spatial coherency parameters: $[y_0, k_0, b_0, b_1]$)
 mRoms (vector for networks stays update
 mt (vector for networks stays update
 $\text{mt} = [0^2, \varepsilon, \gamma_1, \gamma_2, \beta + 7]$
 $\text{space_}\gamma$ (noise in False alarm, true for $t=2$ or regime)
 $\text{use_}\gamma$ (true for regime after $t=2$ or regime)
 single_model
 θ_{err} (only used if single model is true)

ALGORITHM
 $\#$ initialise wave parameters
 $\text{mout} = (\frac{\text{draws} - \text{mean}}{\text{term}})$
 $\text{ww} = 0$ (real movie movie iterations)
 $\text{Sw_iter} = \frac{u}{n} \begin{pmatrix} 0 & \dots & T \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix}$
 $\alpha_iter = \frac{u}{n} \begin{pmatrix} 0 & \dots & T \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix}$
 $\text{me} = \frac{u}{n} \begin{pmatrix} u & u & \dots & u \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$
 $\text{winter_cone} = \text{label area}$
 $\text{inclus_iter} = (0 \ 0 \ \dots \ 0)$
 active cone unselected more times $\rightarrow 0$
 (variables auxiliary per layer)
 $\#$ estensione delle movie
 movies = la movies in input
 $\#$ start effettivo dell'algoritmo
 $\text{for } (i=0, i < \text{draws})$
 $\quad \text{for } (t=0, t < T)$
 $\quad \quad \text{update } \alpha$
 $\quad \quad \text{update } \rho$
 $\quad \quad \text{update } \gamma_0$
 $\quad \quad \text{update } \gamma_1$
 $\quad \quad \text{update } \theta$
 $\quad \quad \text{update } \theta'$
 $\quad \quad \text{update } \gamma_2$
 $\quad \quad \text{update } \theta''$
 $\quad \quad \text{update } \alpha'$
 $\quad \quad \text{update } \rho'$
 $\quad \quad \text{update } \theta_0'$
 $\quad \quad \text{update } \theta_1'$
 $\quad \quad \text{update } \theta_2'$
 $\quad \quad \text{update } \gamma_3'$



INPUT: draws, draw, turn
 subjects (m)
 movie (T)
 $\xi, \gamma_{11}, \gamma_{21}$ (dots and update counts)
 X (the covariate matrix)
 M (drawslet mass parameter)
 α (starting alpha)
 α -prior = $[m_0, s_0, A_0, A_0, b_0]$
 model - priors = $[a_0, b_0]$
 α -priors = $[a_0, b_0]$
 tune - specific - α
 init - specific - α
 update - α
 update - γ
 update - β
 update - θ
 update - η
 update - ψ
 PPM (true if we move the model counts)
 model X (true if we move the covariates)
 specific - covariates
 covariates - unchanged
 covariates - vector (vector for the specific covariates)
 covariates: $[\gamma_{01}, k_0, b_0, b_0]$
 m0 (vector for parameters stays update
 can $\in \mathbb{R}^2, \mathbb{C}, \mathbb{R}, \gamma_0, \beta \in \mathbb{R}$)
 space - α (mass in false mass, true for
 uses the prior after $t=2$ or regime)
 uses the prior after $t=2$ or regime
 single - model
 σ_{ϵ^2} (only used if single model is true)

ALGORITMO

initialize new torsos
 mout = draws - turn
 $w = 0$ real name new variables
 $\Sigma_w = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$
 $\Omega_w = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$
 $m = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$
 winter covar
 labels are
 $\text{label } \gamma_{11} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$
 inclus - wker = $(0 \ 0 \ \dots \ 0)$
 other covar unchanged from turn to 0
 (variables unchanged for draw)

estrazione delle nuove variabili da input
 | usiamo le variabili da input

start effettuando delle elaborazioni

Car ($i=0, i < \text{draws}$)

Car ($t=0, t < T$)

update α

update β

update γ

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

update β

update θ

update η

update ψ

update Ω

update Σ

update m

update γ

INPUT: draws, burn / thin
 subjects (n)
 time (T)
 $\delta, \gamma_1, \gamma_2$ (dose and survival counts)
 X (the covariate matrix)
 M (burninlet mass parameter)
 d (starting alpha)
 model - priors = [$m_0, s_0, A_0, A_1, b_{q+}$]
 α - priors = [a_0, b_0]
 time - specific - α
 unit - specific - α
 update - α
 update - γ_1
 update - γ_2
 update - δ
 SPPM (true w/ we remove the survival counts)
 model X (true w/ we remove the covariates)
 spatial - cohesion
 covariates - interaction
 cRoms (vector on the spatial cohesion
 parameters: [$\gamma_0, k_0, \nu_0, l_0$])
 mta (vector on metabolites steps update
 'car [$\delta^2, \gamma_1, \gamma_2, b_{q+}, p \pm T$])
 space - α (noise in FALSE w/ m2, see 'for
 noise to appear after a $t=2$ or before)
 single - model
 α_{22} (old) used w/ single model as true)

initializza navi toroas
 $m_{out} = (\underbrace{\text{draws}}_{\text{item}} - \underbrace{\text{bumm}}_{\text{item}})$

$w_0 = 0$ (non hanno navi interate)

$S_{w_inter} = \begin{matrix} & & \\ & 4 & \\ & \vdots & \end{matrix} \left(\begin{matrix} 4 \\ 0 \end{matrix} \right)$

$\emptyset_{w_inter} = \begin{matrix} & & \\ & 0 & \\ & \vdots & \end{matrix} \left(\begin{matrix} 0 \\ 0 \end{matrix} \right)$

$m_{in} = \begin{matrix} & & \\ & m & \\ & \vdots & \end{matrix} \left(\begin{matrix} m & \dots & m \\ 0 & & 0 \end{matrix} \right)$

water come
 land area

inclus_water = $(\underbrace{0}_{\text{water}} \underbrace{4}_{\text{land}} \dots \underbrace{4}_{\text{land}})$

altre care variabili esterne furse a 0
 (variabili esterne per forza)

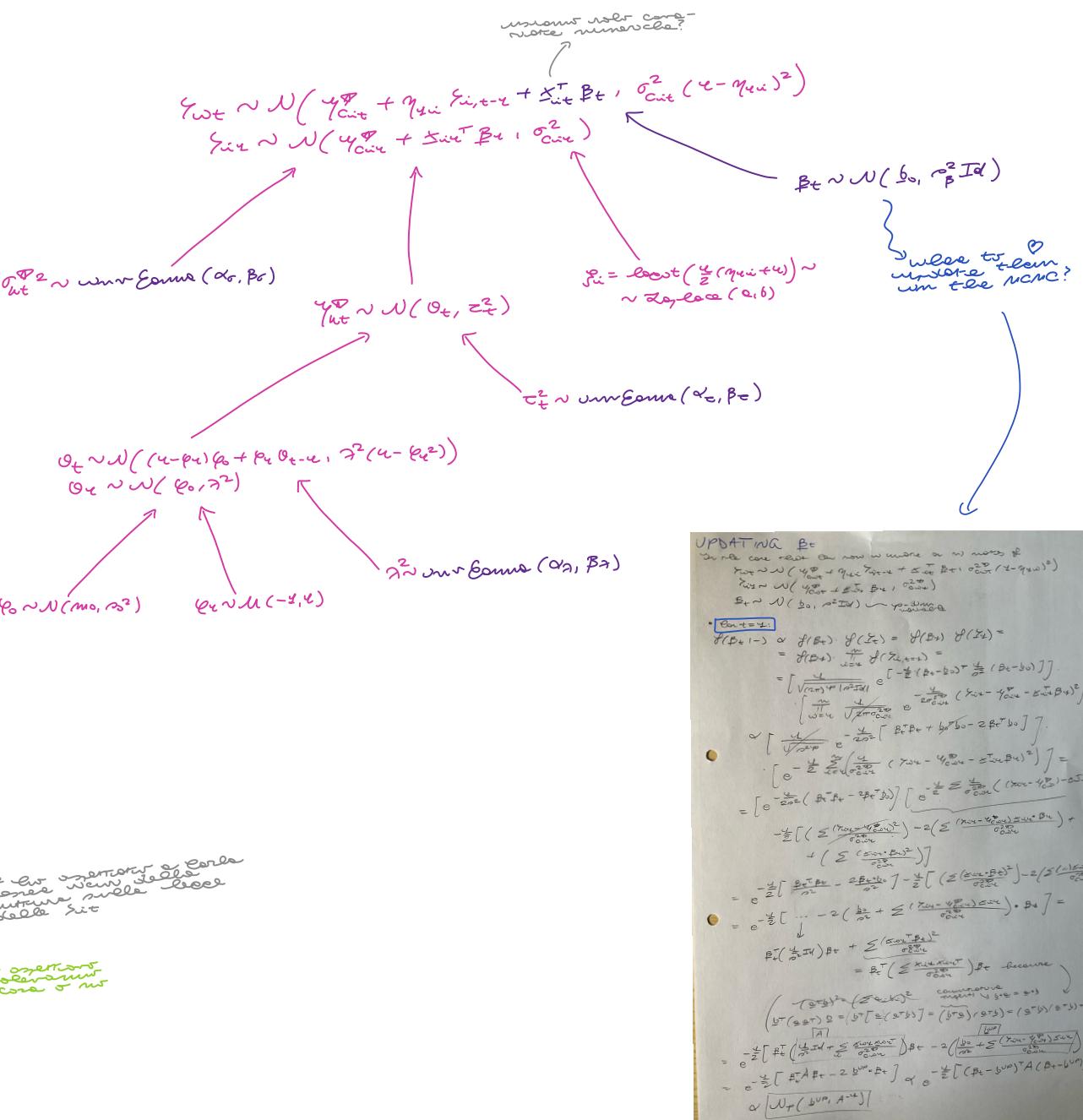
estensione delle navi
 inserendo le navi interate in input

start effettuare delle elezioni

Car ($w=0$, $i \leftarrow \text{draws}$)
 Car ($t=0$, $t \leftarrow T$)

- + update r^i ✓
- + update p^i ✓
- + update y^i ✓ slides
- + update θ^i 0 (es) now
- + update ϕ^i ✓ slides
- + update ψ^i 0 (es) now
- + update η^i_1 memoryless
- + update η^i_2 ✓ slides
- + update η^i_3 ✓ slides
- + update η^i_4 memoryless
- + update η^i_5 memoryless
- + update η^i_6 memoryless
- + update η^i_7 memoryless
- + update η^i_8 memoryless
- + update η^i_9 now

(Car $t=0$, $t \leftarrow T$)



```

nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] - 1;

} else{

    // Observation is a member of a singleton cluster ...

    iaux = Si_iter[j*(ntime1) + t];
    if(iaux < nclus_iter[t]){

        // Need to relabel clusters. I will do this by swapping
        cluster labels
        // Si_iter[j] and nclus_iter along with cluster
        specific parameters;

        // All members of last cluster will be assigned subject
        j's label
        for(jj = 0; jj < *nsubject; jj++){
            if(Si_iter[jj*(ntime1) + t] == nclus_iter[t]){
                Si_iter[jj*(ntime1) + t] = iaux;
            }
        }

        Si_iter[j*(ntime1) + t] = nclus_iter[t];
        // The following steps swaps order of cluster specific
        parameters
        // so that the newly labeled subjects from previous
        step retain
        // their correct cluster specific parameters
        auxs = sig2h[(iaux-1)*ntime1 + t];
        sig2h[(iaux-1)*ntime1 + t] = sig2h[(nclus_iter[t]-1)*
        (ntime1)+t];
        sig2h[(nclus_iter[t]-1)*(ntime1)+t] = auxs;
        auxm = muh[(iaux-1)*ntime1 + t];
        muh[(iaux-1)*ntime1 + t] = muh[(nclus_iter[t]-1)*
        (ntime1)+t];
        muh[(nclus_iter[t]-1)*(ntime1)+t] = auxm;
        // the number of members in cluster is also swapped
        with the last
        nh[(iaux-1)*(ntime1)+t] = nh[(nclus_iter[t]-1)*
        (ntime1)+t];
        nh[(nclus_iter[t]-1)*(ntime1)+t] = 1;
    }
}

```

```

}

// Now remove the ith obs and last cluster;
nh[(nclus_iter[t]-1)*(ntime1)+t] = nh[(nclus_iter[t]-1)*
(ntime1)+t] - 1;
nclus_iter[t] = nclus_iter[t] - 1;

}

for(jj = 0; jj < *nsubject; jj++){

    rho_tmp[jj] = Si_iter[jj*(ntime1) + t];
}

for(k = 0; k < nclus_iter[t]; k++){
    rho_tmp[j] = k+1;

    // First need to check compatibility
    Rindx2=0;
    for(jj = 0; jj < *nsubject; jj++){
        if(gamma_iter[jj*ntime1 + (t+1)] == 1){
            comp2t[Rindx2] = rho_tmp[jj];
            comptp1[Rindx2] = Si_iter[jj*ntime1 + (t+1)];
            Rindx2 = Rindx2 + 1;
        }
    }
    // check for compatibility
    rho_comp = compatibility(comp2t, comptp1, Rindx2);
    if(rho_comp != 1){
        ph[k] = Log(0); // Not compatible
    } else {
        // Need to compute Pr(rhot), Pr(rhot.R), Pr(rhot+1),
        Pr(rhot+1.R)

        for(jj = 0; jj < *nsubject; jj++){
            nh_tmp[jj] = 0;
        }
        n_tmp = 0;
        for(jj = 0; jj < *nsubject; jj++){
            nh_tmp[rho_tmp[jj]-1] = nh_tmp[rho_tmp[jj]-1]+1;
            n_tmp=n_tmp+1;
        }

        nclus_tmp=0;
        for(jj = 0; jj < *nsubject; jj++){
            if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
        }
    }
}

```

```

}

lpp = 0.0;
for(kk = 0; kk < nclus_tmp; kk++){
    // Beginning of spatial part
    lCn = 0.0;
    if(*sPPM==1){
        if((*space_1==1 & t == 0) | (*space_1==0)){
            indx = 0;
            for(jj = 0; jj < *nsubject; jj++){
                if(rho_tmp[jj] == kk+1){

                    s1n[indx] = s1[jj];
                    s2n[indx] = s2[jj];
                    indx = indx+1;
                }
            }
        }
        nh_tmp[kk]*SpatialCohesion, 1);
    }
    // End of spatial part
    lpp = lpp + nclus_tmp*log(Mdp) + lgamma((double)
nh_tmp[kk]) + lCn;
    lpp = lpp + nh_tmp[kk]*log(Mdp) + lgamma((double)
nh_tmp[kk]) + lCn;
    lpp = lpp + (Log(Mdp) + lgamma((double) nh_tmp[kk]) +
lCn);

    if(t==0){
        // //
        ph[k] = dnorm(y[j*ntime] + t,
                        sqrt(sig2h[k*(ntime1) + t]), 1));
        lpp;
    }
    if(t > 0){
        muh[k*(ntime1) + t] +
            eta1_iter[j]*y[j*(ntime) + t-1],
            sqrt(sig2h[k*(ntime1) + t]*
//                                         (1-eta1_iter[j])*eta1_iter[j])),

ph[k] = dnorm(y[j*ntime] + t,
               muh[k*(ntime1) + t] +
                   eta1_iter[j]*y[j*(ntime) + t-1],
                   sqrt(sig2h[k*(ntime1) + t]*
(1-eta1_iter[j])*eta1_iter[j])), 1)+

lpp;
}
// use this to test if MCMC draws from prior are
correct
// ph[k] = lpp;

}

// Determine if E.U. gets allocated to a new cluster
// Need to check compatibility first

rho_tmp[j] = nclus_iter[t]+1;

// First need to check compatibility
Rindx1 = 0, Rindx2=0;
for(jj = 0; jj < *nsubject; jj++){
    if(gamma_iter[jj*ntime1 + (t+1)] == 1){
        comp2t[Rindx2] = rho_tmp[jj];
        comptp1[Rindx2] = Si_iter[jj*ntime1 + (t+1)];
        Rindx2 = Rindx2 + 1;
    }
}
// check for compatibility
rho_comp = compatibility(comp2t, comptp1, Rindx2);
if(rho_comp != 1){
    ph[nclus_iter[t]] = Log(0); // going to own cluster is
not compatible;
} else {

    mudraw = rnorm(theta_iter[t], sqrt(tau2_iter[t]));
    sigdraw = runif(0, Asig);

    for(jj = 0; jj < *nsubject; jj++){
        nh_tmp[jj] = 0;
    }
    n_tmp = 0;
    for(jj = 0; jj < *nsubject; jj++){
        nh_tmp[rho_tmp[jj]-1] = nh_tmp[rho_tmp[jj]-1]+1;
        n_tmp=n_tmp+1;
    }
}

```

```

}

nclus_tmp=0;
for(jj = 0; jj < *nsubject; jj++){
  if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
}

lpp = 0.0;
for(kk = 0; kk < nclus_tmp; kk++){
  // Beginning of spatial part
  lCn = 0.0;
  if(*sPPM==1){
    if((*space_1==1 & t == 0) | (*space_1==0)){
      indx = 0;
      for(jj = 0; jj < *nsubject; jj++){

        if(rho_tmp[jj] == kk+1){

          s1n[indx] = s1[jj];
          s2n[indx] = s2[jj];
          indx = indx+1;
        }

      }
      lCn = Cohesion3_4(s1n, s2n, mu0, k0, v0, L0,
nh_tmp[kk],*SpatialCohesion, 1);
    }
  }
  // End of spatial part

  lpp = lpp + (Log(Mdp) + Lgamma((double) nh_tmp[kk]) +
lCn);
  // lpp = lpp + nh_tmp[kk]*log(Mdp) + lgamma((double)
nh_tmp[kk]) + lCn;
}

if(t==0){
  ph[nclus_iter[t]] = dnorm(y[j*(*ntime) + t], mudraw,
sigdraw, 1) +
lpp;
}
if(t > 0){

  ph[nclus_iter[t]] = dnorm(y[j*(*ntime) + t],
mudraw + eta1_iter[j]*y[j*(*ntime) + t-1],
sigdraw*sqrt(1-

```

```

eta1_iter[j]*eta1_iter[j]), 1) +
lpp;
}

// ph[nclus_iter[t]] = lpp;

}

// Now compute the probabilities
for(k = 0; k < nclus_iter[t]+1; k++) phtmp[k] = ph[k];

R_rsort(phtmp, nclus_iter[t]+1);

maxph = phtmp[nclus_iter[t]];

denph = 0.0;
for(k = 0; k < nclus_iter[t]+1; k++){

  ph[k] = exp(ph[k] - maxph);
  denph = denph + ph[k];

}

for(k = 0; k < nclus_iter[t]+1; k++){
  probh[k] = ph[k]/denph;
}

uu = runif(0.0,1.0);
cprobh= 0.0;;
for(k = 0; k < nclus_iter[t]+1; k++){
  cprobh = cprobh + probh[k];
  if (uu < cprobh){

    iaux = k+1;
    break;
  }
}

if(iaux <= nclus_iter[t]){
  Si_iter[j*(ntime1) + t] = iaux;
  nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1)+t] =
nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1)+t] + 1;
  rho_tmp[j] = iaux;
} else{

  nclus_iter[t] = nclus_iter[t] + 1;
  Si_iter[j*(ntime1) + t] = nclus_iter[t];
}
```

```

nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1)+t] = 1;
rho_tmp[j] = nclus_iter[t];

muh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] = mudraw;
sig2h[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] =
sigdraw*sigdraw;
    if(*simpleModel==1) sig2h[(Si_iter[j*(ntime1) +
t]-1)*(ntime1) + t] = 1.0;
}
}

for(jj = 0; jj < *nsubject; jj++){
    Si_tmp[jj] = Si_iter[jj*(ntime1) + t];
    Si_tmp2[jj] = 0;
    reorder[jj] = 0;
}
// I believe that I have to make sure that groups are order
so that
// EU one is always in the group one, and then the smallest
index not
// with group 1 anchors group 2 etc.

relabel(Si_tmp, *nsubject, Si_tmp2, reorder, oldLab);

for(jj=0; jj<*nsubject; jj++){
    Si_iter[jj*(ntime1) + t] = Si_tmp2[jj];
}
for(k = 0; k < nclus_iter[t]; k++){
    mu_tmp[k] = muh[k*(ntime1)+t];
    sig2_tmp[k] = sig2h[k*(ntime1)+t];
}
for(k = 0; k < nclus_iter[t]; k++){
    nh[k*(ntime1)+t] = reorder[k];
    muh[k*(ntime1)+t] = mu_tmp[(oldLab[k]-1)];
    sig2h[k*(ntime1)+t] = sig2h[(oldLab[k]-1)];
}
}

for(j = 0; j < *nsubject; j++){
    Si_tmp[j] = Si_iter[j*(ntime1) + t];
    Si_tmp2[j] = 0;
    reorder[j] = 0;
}
// I believe that I have to make sure that groups are order so
that
// EU one is always in the group one, and then the smallest
index not
// with group 1 anchors group 2 etc.

relabel(Si tmp, *nsubject, Si tmp2, reorder, oldLab);

```

```

for(j=0; j<*nsubject; j++){
    Si_iter[j*(ntime1) + t] = Si_tmp2[j];
}
for(k = 0; k < nclus_iter[t]; k++){
    mu_tmp[k] = muh[k*(ntime1)+t];
    sig2_tmp[k] = sig2h[k*(ntime1)+t];
}
for(k = 0; k < nclus_iter[t]; k++){
    nh[k*(ntime1)+t] = reorder[k];
    muh[k*(ntime1)+t] = mu_tmp[(oldLab[k]-1)];
    sig2h[k*(ntime1)+t] = sig2_tmp[(oldLab[k]-1)];
}

// for(k = 0; k < nclus_iter[t]; k++) sig2h[k*(ntime1)+t] = 1.0;
for(k = 0; k < nclus_iter[t]; k++){
    ///////////////////////////////////////////////////////////////////
    // update yt ∈ Rdy, but to one / store the
    // result in one a certain yi to each
    // iteration in case there will be obs under
    // they have yt & to the same cluster
    ///////////////////////////////////////////////////////////////////  $\text{sum} = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ n \end{pmatrix} \Rightarrow y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$ 
    if(t==0){
        sumy = 0.0;
        for(j = 0; j < *nsubject; j++){
            if(Si_iter[j*(ntime1) + t] == k+1){
                sumy = sumy + y[j*(ntime1)+t];
            }
        }
        s2star = 1/((double) nh[k*(ntime1)+t]/sig2h[k*(ntime1) + t]
+ 1/tau2_iter[t]);
        mstar = s2star*( (1/sig2h[k*(ntime1) + t])*sumy +
(1/tau2_iter[t])*theta_iter[t]);
    }
    if(t > 0){
        sumy = 0.0;
        sume2 = 0.0;
        for(j = 0; j < *nsubject; j++){
            if(Si_iter[j*(ntime1) + t] == k+1){
                sume2 = sume2 + 1.0/(1-eta1_iter[j]*eta1_iter[j]);
                sumy = sumy + (y[j*(ntime1)+t] - eta1_iter[j]*y[j*
(ntime1)+t-1])/
(1-eta1_iter[j]*eta1_iter[j]);
            }
        }
        s2star = 1/(( 1.0/sig2h[k*(ntime1) + t])*sume2 +

```

we have a symmetric multi-variate normal centered at the seed values

$\sim \mathcal{N}(\text{seed}, -)$

```

    1/tau2_iter[t]);
    mstar = s2star*( (1.0/sig2h[k*(ntime1) + t])*sumy +
    (1/tau2_iter[t])*theta_iter[t]);

}

// muh[k*(ntime1) + t] = rnorm(mstar, sqrt(s2star));
muh[k] = 0.0;
///////////////////////////////
// update sig2h
// /////////////////////
osig = sqrt(sig2h[k*(ntime1) + t]);
nsig = rnorm(osig, csigSIG);
if(nsig > 0.0 & nsig < Asig){

    lln = 0.0;
    llo = 0.0;
    if(t == 0){
        for(j = 0; j < *nsubject; j++){
            if(Si_iter[j*(ntime1) + t] == k+1){
                llo = llo + dnorm(y[j]*(ntime1)+t], muh[k*(ntime1) +
t], osig,1);
                lln = lln + dnorm(y[j]*(ntime1)+t], muh[k*(ntime1) +
t], nsig,1);
            }
        }
    }
    if(t > 0){
        for(j = 0; j < *nsubject; j++){
            if(Si_iter[j*(ntime1) + t] == k+1){
                llo = llo + dnorm(y[j]*(ntime1)+t], muh[k*(ntime1) +
t] +
eta1_iter[j]*y[j]*(ntime1) + t-1], osig*sqrt(1-
eta1_iter[j]*eta1_iter[j]),1);
                lln = lln + dnorm(y[j]*(ntime1)+t], muh[k*(ntime1) +
t] +
eta1_iter[j]*y[j]*(ntime1) + t-1], nsig*sqrt(1-
eta1_iter[j]*eta1_iter[j]),1);
            }
        }
    }
    llo = llo + dunif(osig, 0.0, Asig, 1);
    lln = lln + dunif(nsig, 0.0, Asig, 1);
}

```

```

llr = lln - llo;
uu = runif(0,1);

if(log(uu) < llr){
    sig2h[k*(ntime1) + t] = nsig*nsig;
}
if(*simpleModel==1) sig2h[k*(ntime1) + t] = 1.0;
}

///////////////////////////////
// update theta (mean of mh)
// /////////////////////
summu = 0.0;
for(k = 0; k < nclus_iter[t]; k++){
    summu = summu + muh[k*(ntime1) + t];
}

phi1sq = phi1_iter*phi1_iter;
lam2tmp = lam2_iter*(1.0 - phi1sq);

if(t==0){
    s2star = 1.0/((double) nclus_iter[t]/tau2_iter[t] +
1.0/lam2_iter + phi1sq/lam2tmp);
    mstar = s2star*( (1.0/tau2_iter[t])*summu +
(1.0/lam2_iter)*phi0_iter +
(1.0/lam2tmp)*phi1_iter*(theta_iter[t+1] -
phi0_iter*(1-phi1_iter)));
} else if(t==(*ntime-1)){
    s2star = 1.0/((double) nclus_iter[t]/tau2_iter[t] +
1.0/lam2tmp);
    mstar = s2star*((1.0/tau2_iter[t])*summu +
(1.0/lam2tmp)*(phi0_iter*(1-phi1_iter) +
phi1_iter*theta_iter[t-1]));
} else {
    s2star = 1.0/((double) nclus_iter[t]/tau2_iter[t] + (1.0 +
phi1sq)/lam2tmp);
    mstar = s2star*( (1.0/tau2_iter[t])*summu +
(1.0/lam2tmp)*(phi0_iter*(1-phi1_iter) +
phi1_iter*theta_iter[t-1]));
}

```

```

(1.0/lam2tmp)*(phi1_iter*(theta_iter[t-1] +
theta_iter[t+1]) +
phi0_iter*(1.0 - phi1_iter)*
(1.0 - phi1_iter)));
}

theta_iter[t] = rnorm(mstar, sqrt(s2star));
if(*simpleModel==1) theta_iter[t] = 0.0;

///////////////////////////////
// // update tau2 (variance of mh) //
// // //

/////////////////////////////
ot = sqrt(tau2_iter[t]);
nt = rnorm(ot,csigTAU);
if(nt > 0){

    lln = 0.0;
    llo = 0.0;
    for(k = 0; k < nclus_iter[t]; k++){
        llo = llo + dnorm(mu[k*(ntime1) + t], theta_iter[t],
ot,1);
        lln = lln + dnorm(mu[k*(ntime1) + t], theta_iter[t],
nt,1);
    }

    llo = llo + dunif(ot, 0.0, Atau, 1);
    lln = lln + dunif(nt, 0.0, Atau, 1);

    llr = lln - llo;
    uu = runif(0,1);

    if(Log(uu) < llr){
        tau2_iter[t] = nt*nt;
        tau2_iter[t] = 5*5;
    }
    if(*simpleModel==1) tau2_iter[t] = theta_tau2[1];
}
}

```

```

//                                         //
// update alpha                         //           //
//                                         //

////////////////////////////////////////////////////////////////

if(*update_alpha == 1){
    if(*time_specific_alpha != 1){
        sumg = 0;
        for(j = 0; j < *nsubject; j++){
            for(t = 1; t < *ntime; t++){
                sumg = sumg + gamma_iter[j*ntime1 + t];
            }
        }
        astar = (double) sumg + a;
        bstar = (double) ((*nsubject)*(*ntime-1) - sumg) + b;

        alpha_tmp = rbeta(astar, bstar);
        for(t=0;t<*ntime;t++){alpha_iter[t] = alpha_tmp;}
    } else {
        for(t = 0; t < *ntime; t++){
            sumg = 0;
            for(j = 0; j < *nsubject; j++){
                sumg = sumg + gamma_iter[j*ntime1 + t];
            }
            astar = (double) sumg + a;
            bstar = (double) ((*nsubject) - sumg) + b;
            alpha_iter[t] = rbeta(astar, bstar);
        }
    }
    alpha_iter[0] = 0.0;
}

////////////////////////////////////////////////////////////////
//                                         //
// update phi0                          //           //
//                                         //

```

```

    llr = lln - llo;
    if(llr > log(runif(0,1))) phi1_iter = np1;
}
}

//////////////////////////////          //
// update lam2          //
//          //

//////////////////////////////          //
// Update lambda with a MH step
phi1sq = phi1_iter*phi1_iter;
ol = sqrt(lam2_iter);
nl = rnorm(ol, csigLAM);
if(nl > 0.0){
    lln = 0.0;
    llo = 0.0;
    for(t=1; t<*ntime; t++){
        llo = llo + dnorm(theta_iter[t],
                            phi0_iter*(1-phi1_iter) +
                            phi1_iter*theta_iter[t-1], ol*sqrt(1-phi1sq),1);
        lln = lln + dnorm(theta_iter[t],
                            phi0_iter*(1-phi1_iter) +
                            phi1_iter*theta_iter[t-1], nl*sqrt(1-phi1sq),1);
    }
    llo = llo + dnorm(theta_iter[0], phi0_iter, ol, 1) + dunif(ol,
0.0, Alam, 1);
    lln = lln + dnorm(theta_iter[0], phi0_iter, nl, 1) + dunif(nl,
0.0, Alam, 1);
    llr = lln - llo;
    uu = runif(0,1);
    if(log(uu) < llr){
        lam2_iter = nl*nl;
    }
}
/*
phi1sq = phi1_iter*phi1_iter;
ssq = 0.0;
for(t=1; t<*ntime; t++){
    ssq = ssq + (theta_iter[t] - (phi0_iter*(1-phi1_iter) +
phi1_iter*theta_iter[t-1]))*
(theta_iter[t] - (phi0_iter*(1-phi1_iter) +
phi1_iter*theta_iter[t-1]));
}
ssq = 1.0/(1.0 - phi1sq)*ssq + (theta_iter[0]-phi0_iter)*
(theta_iter[0]-phi0_iter);
astar = 0.5*(*ntime) + 1;

```

```

bstar = 0.5*ssq + 1/1;
lam2_iter = 1.0/rgamma(astar, 1/bstar);
*/
//////////////////////////////          //
// predict partition for new time period          //
//          //

//////////////////////////////          //
/*          //
for(p = 0; p < *npred; p++){
    for(j=0; j<*nsubject; j++){
        nh_pred[j] = 0;
        predSi_iter[j*(*npred) + p] = 0;
    }
    if(*update_alpha == 0){
        n_red = 0;
        for(j=0;j<*nsubject;j++){
            gpred[j] = rbinom(1,*alpha);
            if(gpred[j] == 1){
                nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] =
nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] + 1;
                n_red = n_red + 1;
                predSi_iter[j*(*npred) + p] = Si_iter[j*(ntime1)+(*ntime)-1];
            }
        }
    }
    if(*update_alpha == 1){
        if(*time_specific_alpha == 1){
            n_red = 0;
            for(j=0;j<*nsubject;j++){
                gpred[j] = rbinom(1,alpha_iter[1]);
                if(gpred[j] == 1){
                    nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] =
nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] + 1;
                    n_red = n_red + 1;
                    predSi_iter[j*(*npred) + p] = Si_iter[j*(ntime1)+(*ntime)-1];
                }
            }
        }
    }
}

```



```

if(like0==0){
    for(j = 0; j < *nsubject; j++){
        for(t = 0; t < *ntime; t++){
            CPO[j*(*ntime)+t] = CPO[j*(*ntime)+t] +
(1/exp(like_iter[j*(*ntime)+t]));
        }
    }
}

///////////////////////////////
//                                //
// Save MCMC iterates          //
//                                //
///////////////////////////////

if((i > (*burn-1)) & ((i+1) % *thin == 0)){
    for(t = 0; t < *ntime; t++){
        alpha_out[ii*(*ntime) + t] = alpha_iter[t];
        theta[ii*(*ntime) + t] = theta_iter[t];
        tau2[ii*(*ntime) + t] = tau2_iter[t];
        for(j = 0; j < *nsubject; j++){
            sig2[(ii*(*nsubject) + j)*(*ntime) + t] = sig2h[(Si_iter[j*
(ntime1) + t]-1)*(ntime1) + t];
            mu[(ii*(*nsubject) + j)*(*ntime) + t] = muh[(Si_iter[j*
(ntime1) + t]-1)*(ntime1) + t];
            Si[(ii*(*nsubject) + j)*(*ntime) + t] = Si_iter[j*ntime1 +
t];
            gamma[(ii*(*nsubject) + j)*(*ntime) + t] =
gamma_iter[j*ntime1 + t];
            llike[(ii*(*nsubject) + j)*(*ntime) + t] = like_iter[j*
(*ntime)+t];
            fitted[(ii*(*nsubject) + j)*(*ntime) + t] = fitted_iter[j*
(*ntime)+t];
        }
    }
    for(j=0; j<*nsubject; j++){

        eta1[ii*(*nsubject) + j] = eta1_iter[j];
    }
    phi1[ii] = phi1_iter;
    phi0[ii] = phi0_iter;
    lam2[ii] = lam2_iter;
    ii = ii+1;
}
/**/
}

```

```

lpml_iter=0.0;
for(t = 0; t < *ntime; t++){
    for(j = 0; j < *nsubject; j++){

        lpml_iter = lpml_iter - Log((1/(double) nout-nout_0)*CPO[j*(*ntime)+t]);
    }
}
lpml[0] = lpml_iter;
elppdWAIC = 0.0;

for(j = 0; j < *nsubject; j++){
    for(t = 0; t < *ntime; t++){
        elppdWAIC = elppdWAIC + (2*mnllike[j*(*ntime)+t] -
Log(mnllike[j*(*ntime)+t]));
    }
}
waic[0] = -2*elppdWAIC;
PutRNGstate();

}

```