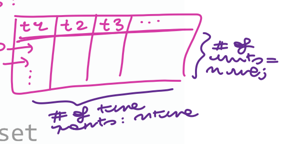
```c
/*
 * Code is of the file DepdenetPartitionsAR1LikeAR1Theta_sPPM2.c
 * that one since the R function drpm_fit calls the function which
 * is defined there:
     initial_partition <- 0
     C.out <- .C("drpm_ar1_sppm", // defined in that file
               as.integer(draws), as.integer(burn),as.integer(thin),
               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
only if update_alpha is TRUE
* priorvals = vector containing values for prior distributions as
follows
*
* time_specific_alpha = integer - logical indicating wether to make
alpha time-specific or one global alpha.
* update_alpha = integer - logical indicating wether 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 indication 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,
```

estimate dataset will look like this:



parametro in realtà impostato su FALSE nel codice/wrapper in R

```
      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_VectorInit(*nsubject, runif(0,1));
  double *theta_iter = R_VectorInit(ntime1, rnorm(0,3));
  double *tau2_iter = R_VectorInit(ntime1, runif(0,
modelPriors[3]*modelPriors[3]));
```

```
  double phi0_iter = rnorm(0,3);
  double phi1_iter = runif(0,1);
  double lam2_iter = runif(0, modelPriors[4]*modelPriors[4]);
  double *alpha_iter = R_VectorInit(ntime1, *alpha);

  //
=========================================================================
  //
  // Memory vectors to hold MCMC iterates for cluster specific
parameters
  //
  //
=========================================================================
  double *muh = R_VectorInit((*nsubject)*(ntime1), 0.0);
  double *sig2h = R_VectorInit((*nsubject)*(ntime1), 1.0);
    if(*simpleModel==1){
      for(t = 0; t < ntime1; t++){
        theta_iter[t] = theta_tau2[0];
        tau2_iter[t] = theta_tau2[1];
      }
    }
  int nh[(*nsubject)*(ntime1)];
  //
=========================================================================
  //
  // Initialize a few parameter vectors
  //
  //
=========================================================================
  // 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;
```

at t=1 ⇒ two >0 values ⇒ 2 clusters

at t=2 ⇒ three >0 values ⇒ 3 clusters

$$nh = \begin{pmatrix} 8 & 6 & 6 \\ 2 & 3 & 2 \\ 0 & 4 & 2 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \end{pmatrix}$$

scorre numero unit (nelle righe) e per ogni subjects (le colonne) quando vede questo la conseguenza di numero nella struttura di nh

```
    }
  }
  // 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[*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 parition);
  int
comp1t[(*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;
```

⟸ nclus_iter[t] += (nh[j,t] > 0)

```
  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];
```

```c
  // priors for phi0
  double m0 = modelPriors[0], s20 = 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, s20 = %.2f\n\n", Asig, Atau, Alam, m0, s20);
  // Cohesion auxiliary model paramaters 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 tunning parameter
  double csigSIG=mh[0], csigTAU=mh[1], csigLAM=mh[2], csigETA1=mh[3],
csigPHI1=mh[4];
  GetRNGstate();
  //
==========================================================================
  //
  // 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++){

//////////////////////////////////
      //
```

In ogni iterata non si re-inizializza tutto a zero perché il nuovo ciclo MCMC continua dalla iterata precedente

Per partire con la nuova iterata usiamo quelli calcolati nella iterata precedente

$S_i^{TMP} \to S_i^{RED}$, $S_i^{TMP2} \to S_i^{RED1}$ ($\rho_t^{R_t}$, $\rho_t^{R_t(+j)}$)

example

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

```c
      // begin by updating gamma (pegged) parameters
      //

//////////////////////////////////////////////////////////////////////
      for(j = 0; j < *nsubject; j++){
        // at time period one, all gammas are zero (none are
``pegged'')
        if(t == 0){
          gamma_iter[j*(ntime1) + t] = 0;
        } 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){
              if(jj != j){
                Si_tmp[Rindx1] = Si_iter[jj*ntime1 + (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;
              }
            }
          }
          Si_tmp2[Rindx1] = Si_iter[j*ntime1 + (t)];
          comptm1[Rindx1] = Si_iter[j*ntime1 + (t-1)];
          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];
```

pegged

FOR j in 1:nsubjects

Cluster = ... $t=1$ ... for $t>1$ enter the else and decides how to update the γ values

FOR jj in nsubjects $\setminus \{j\}$ (we are here if $\gamma_{jj,t}=1$)

$S_i^{TMP} = \rho_t^{R_t}$, $S_i^{TMP2} = \rho_t^{R_t}$, $comp_{t-1} = \rho_{t-1}$ and $S_i^{RED} = $ last R_t, $S_i^{RED1} = $ long R_t

$S_i^{TMP2} = \rho_t^{R_t(+j)}$, $comp_{t-1} = \rho_{t-1}^{R_t(+j)}$

$n_{RED} = |\rho_t^{R_t}| = R_x$, $n_{RED1} = |\rho_t^{R_t(+j)}| = R_x + 1$

Filter the units which have $\gamma_{jt}=1$, ie the units composing R_t

in jj loop we are extracting $R_t = \{jj : \gamma_{jj,t} = 1\}$

well actually, we are directly getting ρ_t / R_t and ρ_{t-1}/R_t

that because we treat here separately the case about the current subject j (not jj)

we now also add subject j to both, in them to get out $\rho_t^{R_t(+j)}$ and $\rho_{t-1}^{R_t(+j)}$ to compare

$S_{i}\_TMP = \rho_t^{R_t(-j)} \leftarrow S_i^{RED}$

$S_{i}\_TMP2 = \rho_t^{R_t(+j)} \leftarrow S_i^{RED1}$

$Comp\ TM1 = \rho_{t-1}^{R_t(+j)}$

subjects in the reduced partition

n_{RED} in relevance a $S_i^{TMP} = \rho_t^{R_t(-j)} \Rightarrow$ the R_{indx} subjects

n_{RED1} // // $S_i^{TMP1} = \rho_t^{R_t(+j)} \Rightarrow$ $R_{indx}+1$

$cit1 = S_i^{RED1}[end] = $ cluster label of unit j in $\rho_t^{R_t(+j)}$

this should be the cluster label of the last inserted subject, ie the j on which we are focusing on

$\underset{\text{relabeled}}{S_i^{TMP}} = [\underbrace{1\ 2\ 3\ 2\ 2\ 1\ 2}_{\text{units} \neq j \text{ that were pegged}}\ 3]$

last place is for the last added unit, j

Definition 1. We say that partitions ρ_{t-1} and ρ_t are compatible with respect to $\boldsymbol{\gamma}_t$, if ρ_t may be obtained from ρ_{t-1} by reallocating items as indicated by $\boldsymbol{\gamma}_t$, that is, those items i such that $\gamma_{it} = 0$ for $i = 1, \ldots, m$. 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 $\boldsymbol{\gamma}_t$. 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 $\boldsymbol{\gamma}_t$. Then ρ_{t-1} and ρ_t are compatible with respect to $\boldsymbol{\gamma}_t$ if and only if $\rho_{t-1}^{\mathfrak{R}_t} = \rho_t^{\mathfrak{R}_t}$.

$$\Pr(\gamma_{it} = 1 \mid -) = \frac{\alpha_t}{\alpha_t + (1-\alpha_t)\Pr(\rho_t^{\mathfrak{R}_t^{(-i)}})/\Pr(\rho_t^{\mathfrak{R}_t^{(+i)}})} \boxed{I[\rho_{t-1}^{\mathfrak{R}_t^{(+i)}} = \rho_t^{\mathfrak{R}_t^{(+i)}}]}. \quad (S.8)$$

Def 2

FOR ∀ label k in 1 : n_RED compute

$nh^{RED}(k) = |\rho_t^{R_t} == k| = |S_w^{RED} == k|$

$nh^{RED1}(k) = |\rho_t^{R_t(+i)} == k| = |S_i^{RED1} == k|$

$nclus_{RED} = \max_k(\rho_t^{R_k})$

update nh^{RED1} because it has one unit more (the "target" j)

actually it loops on subjects jj in 1 : n_RED, cos code

```
for(jj = 0; jj < n_red_1; jj++){
  nh_red[jj]=0; nh_red_1[jj]=0;
}
nclus_red = 0;
for(jj = 0; jj < n_red; jj++){
  nh_red[Si_red[jj]-1] = nh_red[Si_red[jj]-1] + 1;
  nh_red_1[Si_red_1[jj]-1] = nh_red_1[Si_red_1[jj]-1] + 1;
  if(Si_red[jj] > nclus_red) nclus_red = Si_red[jj];
}

nh_red_1[Si_red_1[n_red]-1]= nh_red_1[Si_red_1[n_red]-1] +
1;
  // this may need to be updated depending on if the value of
gamma changes
  //nclus_red_1 = nclus_red;
  //if(Si_red_1[n_red] > nclus_red) nclus_red_1 =
Si_red_1[n_red];
  lCo=0.0, lCn=0.0;
  for(k = 0; k < nclus_red; k++){
    if(*sPPM==1){
      // Note that if space is only included for first time
point
      // then it does not inform gamma.
      if((*space_1==1 & t == 0) | (*space_1==0)){
        indx = 0;
        for(jj = 0; jj < n_red; jj++){
          if(Si_red[jj] == k+1){
            s1o[indx] = s1_red[jj];
            s2o[indx] = s2_red[jj];

            s1n[indx] = s1_red[jj];
            s2n[indx] = s2_red[jj];
            indx = indx+1;
          }
        }
        s1n[indx] = s1[j];
        s2n[indx] = s2[j];

        lCo = Cohesion3_4(s1o, s2o, mu0, k0, v0, L0,
nh_red[k], *SpatialCohesion, 1);
        lCn = Cohesion3_4(s1n, s2n, mu0, k0, v0, L0,
nh_red[k]+1,*SpatialCohesion, 1);
      }
    }

    lgweight[k] = log(nh_red[k]) + lCn - lCo;
    Si_red_1[Rindx1] = k+1;
    rho_comp = compatibility(Si_red_1, comptm1, Rindx1+1);
//    if(rho_comp==0) lgweight[k] = log(0);
  }
```

qui invece valutiamo l'unità j ad entrare nei diversi (nclusRED) cluster presenti

FOR k in 1 : nclusRED

- get spatial coords of units (jj) inside cluster k → s1,2o (original)
- update to add there the target unit j, so we get the variants s1,2n (new)
- compute the log cohesions lCo and lCn
- assign to cluster k the log-weight $\ln(nh_{RED}^{(k)}) + lCn - lCo$ ($|S_{t,k}|$)

qui contiamo la numerosità di ogni cluster

w label partono da 1 ma C indicizza da 0

= max(S_RED) → conta quanti cluster ci sono in tutto, e' il max label

since labels were just 1:n_RED

we access the last element (j) and update the count

we always enter here since space 1 is set to FALSE in R code

loop starts from k=0 but clusters labels go from 1

this ones were the same as the original coords (s-o) but become new (s-n) since we now add the unit j in them

cohesions of the original and new configurations

compute the log weight for the cluster k

useless check now

TP-2

simulate to assign the last unit (i.e. j, at index R-indx1) to the current cluster k(+1)

check if this assignment would be compatible with comptm1, i.e. the configuration

$\rho_t^{R_t(+i)} \circledcirc (label_{k+1})$ $\rho_{t-1}^{R_t(+i)}$

actually, I think we should check that, to access the computation on $P(\gamma_{it}=1\mid-)$ to return just the ρ_t which are to the "feasible" support

he writes (in another file "correct-solution.c") that "compatibility wasn't checked for reasons he says"

per questo è commentato via quindi forse quel controllo di compatibilità è inutile (perché magari è sempre vero?)

secondo me ci perché li noi stiamo considerando j, e contando "cose" per la singola della γ_{it}, ma qui per ora non c'è nessuna γ coinvolta, quindi un qualunque cluster mettiamo la j va bene, anche non coppiamo se $\gamma=0$ o $\gamma=1$ (cioè deve stare così)

```
//     if(rho_comp==0) lgweight[k] = log(0);
   }

   // 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
```

qui consideriamo solo un nuovo cluster se il subject j va a formare un nuovo cluster da solo

- compute the log weight for the case cluster of a singleton for unit j

cose uguali ma singoletta: qui c'è solo un elemento nel cl e non c'è nessun lc da contare

sort (in R) di default ordina in modo crescente (sx→dx)

this shift is just to center the log weights around 0 and avoid numerical issues

quindi come fin qui è conveniente

logs = 20 30 70 ⇒ exps = $\approx 10^{10,13,30}$

logs − max(logs) = $\approx 10^{-22,-13,0}$

therefore now in lgweight[k] we actually get the real weights: lgw[k] = $\frac{w_k}{\max(w_k)}$

denph = $\sum w_k$

so now here we normalize the weights

phTMP = copy(lgweight)

sort!(phTMP, ascending)

FOR k in 1 : nclusRED

- shifts the logweights and exps them
- accumulate their sum into denph

- normalize the weights

- compute probh[1] using the weight of cluster cit1 and the appropriate αs

```
// 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]);
}
```

$$\Pr(\gamma_{it}=1\mid -) = \frac{\alpha_t}{\alpha_t + (1-\alpha_t)\Pr(\rho_t^{\mathcal{R}_t^{(+i)}})/\Pr(\rho_t^{\mathcal{R}_t^{(-i)}})} I[\rho_{t-1}^{\mathcal{R}_t^{(+i)}} = \rho_t^{\mathcal{R}_t^{(+i)}}]. \quad (S.8)$$

```
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 partitios 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 with out using loops?
Who
            // 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)];
                comp1t[Rindx1] = Si_iter[jj*ntime1 + (t)];
                Rindx1 = Rindx1 + 1;
              }
              // I need to include this because determine what
happens when
              // gamma goes from 0 to 1;
              if(jj == j){
                comptm1[Rindx1] = Si_iter[jj*ntime1 + (t-1)];
                comp1t[Rindx1] = Si_iter[jj*ntime1 + (t)];
                Rindx1 = Rindx1 + 1;
              }
            }
            rho_comp = compatibility(comptm1, comp1t, 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
$COMP_{t-1} = \rho_{t-1}^{Rt}$
$COMP_{1t} = \rho_t^{Rt}$
looping on all subjects in 1:nsubjects

- check compatibility and if not met set probh[1]=0 (to force to sample a 1)

some code therefore is useless

well with that separate if we ensuring to insert unit j as the last one

= 1 w/ wt's a Ber(p) actually

n   p

```
              gt = rbinom(1,probh[1]);

              gamma_iter[j*(ntime1) + t] = gt;
            }
          }
```

end of the else for the cases of t>1

end of the FOR j in 1:nsubjects

Finally sample the new γ_{it} value

```
/////////////////////////////////////////////////////////////////////////,
      //
      // 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 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.
      //
```

```
//
/////////////////////////////////////////////////////////////////////////////////
// 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 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];      ρ^TMP = ρ_t
}
// 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.
                                        alter only the units with γ_it = 0
                                        (i.e. the ones free to move)
    if(gamma_iter[j*(ntime1) + t] == 0){
  if(nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] > 1){
    nh[ρ_jt, t] = nh[cluster label of unit j 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] - 1;
                                    for now it's not modified (ok) but
                                    it will be (conceptually) removed
      we remove (update?) unit j from the
      cluster to which it belongs,
      so we subtract one there
                                          Si^OLD | 4 2 3 2 2 3 6 4 2
                                          Si^NEW | 4 2 3 2 2 3 6 4 2
  }else{                                  nh^OLD | 2 4 2 4
                                          nh^NEW | 2 3 2 4
                                                   4 2 3 4
    // Observation is a member of a singleton cluster ...
                           label of unit j (at t)
    iaux = Si_iter[j*(ntime1) + t];
    if(iaux < nclus_iter[t]){     in this case unit j was part of a cl that
                                  was not the last one
      // 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]){      the units which had
        Si_iter[jj*(ntime1) + t] = iaux;                  the max/last label
      }                                                   get re-labelled
    }       Si_iter
            labels: 4 4 2 3 4 3 4 4
            new:    4 4 4 3 2 3 2 2
            (iaux = 2, nclus_iter[t] = 4)
                          here we end the swap assigning
                          to unit j the last/max label
    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                  we update
       auxs = sig2h[(iaux-1)*ntime1 + t];                            μ_h*, σ_h^2*, nh
       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 ρ_TMP as the updated ρ_t
  }                                            with unit j removed and clusters corrected
    for(k = 0; k < nclus_iter[t]; k++){
  rho_tmp[j] = k+1;     for each cluster k we temporarily
                        update to assign unit j to them
    // First need to check compatability
    Rindx2=0;
    for(jj = 0; jj < *nsubject; jj++){
      if(gamma_iter[jj*ntime1 + (t+1)] == 1){
        comp2t[Rindx2] = rho_tmp[jj];   ρ_t (#j=k)
        comptp1[Rindx2] = Si_iter[jj*ntime1 + (t+1)];
        Rindx2 = Rindx2 + 1;       ρ_t
      }
    }                                  perché non dovrebbe essere
    // check for compatibility         compatibile? se ha γ=0
                                       può andare ovunque
    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;                           count the clusters numerosity
    for(jj = 0; jj < *nsubject; jj++){   (useless, indeed we have it
      nh_tmp[rho_tmp[jj]-1] = nh_tmp[rho_tmp[jj]-1]+1;   above as nh but for the cluster k)
    n_tmp=n_tmp+1;
    }

    nclus_tmp=0;                         compute the # of
    for(jj = 0; jj < *nsubject; jj++){   (non empty) clusters
      if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
    }
                 log prob-
                 ability
    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 spatial coords of
                              the units of cluster kk
        }
```

```
        }
        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) +
        lpp;
```

the last argument for dnorm function is "give-log"

double dnorm4(double x, double mu, double sigma, int give_log)

```
      }
        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;

        }
      }
```

here we ended the loop on the clusters for k in 0: nclus_iter

so now we also consider the case of unit j creating a new cluster

```
    // 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 compatability
    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);
```

draw new params for the new cluster

```
        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;
        }
```

end of the else about the compatibility)

```
//        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];
      }
```

convert to weights (from the log-weights) and center them using the max weight maxph

accumulate the sum

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

normalize all the probabilities

```c
    uu = runif(0.0,1.0);
```

we take u ~ U(0,1)

```c
    cprobh= 0.0;;
```

initialize the cumulative probabilities

```c
    for(k = 0; k < nclus_iter[t]+1; k++){
      cprobh = cprobh + probh[k];
      if (uu < cprobh){
        iaux = k+1;
        break;
      }
    }
```

we assign the label k+1 based on when u is less than cprob

if we are assigned to an existing cluster we update the related stuff (nh and ρTMP)

```c
    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;
    }
```

we are assigned to a new cluster:
- nclus is incremented
- assign the label
- update nh
- update ρTMP
- set the new cluster params to the drawn values

end of the if(γ_{jt}==0)

```c
  }
  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.
```

assign ρTMP to the current partition assignments

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

relabel/correct S_i^1 into S_i^2

(as the new label assignment next might have changed the coherence of ρ?)

```c
  for(jj=0; jj<*nsubject; jj++){
    Si_iter[jj*(ntime1) + t] = Si_tmp2[jj];
  }
```

re-set the actual ρ (we switch) with the corrected Si^{TMP2}

```c
  for(k = 0; k < nclus_iter[t]; k++){
    mu_tmp[k] = muh[k*(ntime1)+t];
    sig2_tmp[k] = sig2h[k*(ntime1)+t];
  }
```

1st usage in all the code

=> we save (momentarily, the idea?) the current clusters params μ and σ^2_h

```c
  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)];
  }
```

we reassign the params based on the relabeling

end of the for j in 0:nsubjects

```c
}
  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

it should be already all right

(given the relabeling on the last j subject)

```c
  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 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)*(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] = 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(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;
      }
      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
to accomodate rho0
              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, 1);
        lln = lln + dunif(np1, -1,1, 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];
      }
    }

  }else {

  }
}


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/CPO[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(mnlike[j*(*ntime)+t]));
  }
  }
  waic[0] = -2*elppdWAIC;
  PutRNGstate();
}
```

INPUT: draws, burnin, thin
nsubjects (n)
ntime (T)
y, s_1, s_2 (data and spatial coords)
X (the covariate matrix)
M (Dirichlet mass parameter)
α (starting alpha)
model_priors = [m_0, s_0, A_σ, A_τ, A_λ, $b_{\eta 1}$]
α_priors = [a_α, b_α]
time_specific_α
unit_specific_α
update_α
update_η_1
update_φ_1
sPPM (true if we model the spatial coords)
model X (true if we model the covariates)
spatial_cohesion
covariates_similarity
cParms (vector for the spatial cohesion params: [μ_0, k_0, ν_0, L_0])
mh (vector for metropolis steps update for [σ^2, τ, λ, η_1, φ_1])
space_1 (messo su FALSE in R, era per usare lo spazio solo a $t=1$ o sempre)
simple_model
$\theta_{\tau 2}$ (only used if simple model was TRUE)

OUTPUT: S_i
μ
σ^2
η_1
θ
τ^2
φ_0
φ_1
λ^2
γ
α_{out}
fitted (y_{est})
llike
LPML
WAIC

RESPECTIVES IN THE CODE:

ntime1 = $T+1 := T^*$
$\Rightarrow$ nel codice ne do $0 \le t \le T^*$ quindi le matrici hanno $T+1$ col, da 0 a T

initializations

S_i_iter	$n \times T^*$ of ones (tranne 0 al tempo 0 ...)
μ_{ec}	$n \times T^*$ of zeros
σ^2_{ec}	$n \times T^*$ of ones
η_1_iter	n of $U(0,1)$
θ_iter	T^* of $N(0,3^2)$
τ^2_iter	T^* of $U(0, A_\tau^2)$
φ_0_iter	scalar, $N(0,3^2)$
φ_1_iter	scalar, $U(0,1)$
λ^2_iter	scalar, $U(0, A_\lambda^2)$
γ_iter	$n \times T^*$ of zeros
α_iter	T^* of starting_α times ones

altre cose
nclus_iter | T^* (it counts the # of cl at time t)
nh | $n \times T^*$ (it counts the numerosity of each cl at each time t) ~ worst case

ALGORITMO

\# initialize some params
nout = (draws − burnin) / thin
iii = 0 (for saving MCMC iterates)

S_i_iter = $\begin{array}{c|cc} & 0 \ldots & T \\ 1 \ldots n & \boxed{1} & \boxed{0} \end{array}$

γ_iter = $\begin{array}{c|c} & 0 \ldots T \\ 1 \ldots n & 0 \end{array}$

nh = $\begin{array}{c|cccc} & 0 & 1 & \ldots & T \\ 1 \ldots n & n & n & \ldots & n \end{array}$ (con 0) ~ unitari come label ora

nclus_iter = $\begin{array}{cccc} 0 & 1 & \ldots & T \\ (1 & 1 & \ldots & 1) \end{array}$ ~ sono tutti nello stesso cluster iniziale

altre cose inizializzate tutte a 0 (variabili ausiliarie per dopo)

\# estrazione delle mia... usando le variabili in input

\# start effettivo dell'algoritmo
for ($i=0$, $i <$ draws)
  for ($t=0$, $t \le T$)
    update γ
    update ρ
    update μ_{ec}
    update σ^2_{ec}
    update θ
    update τ^2
  update η_1
  update α
  update φ_0
  update φ_1
  update λ^2

\# update γ

$$\Pr(\gamma_{it} = 1 \mid -) = \frac{\alpha_t}{\alpha_t + (1-\alpha_t)\Pr(\rho_t^{\mathfrak{R}_t^{(+i)}})/\Pr(\rho_t^{\mathfrak{R}_t^{(-i)}})}\mathrm{I}[\rho_{t-1}^{\mathfrak{R}_t^{(+i)}} = \rho_t^{\mathfrak{R}_t^{(+i)}}].$$

for ($j=0$, $j < n$)
  if ($t==0$) $\gamma_{iter}[j,t] = 0$ ~ at the first time instant all units are free to move
  else
    S_i^{TMP} = $S_i$iter [jj : jj ≠ j AND $\gamma_{jjt}=1$, t] = $\rho_t^{R_t(-j)}$
    $S_i^{TMP_2}$ = $S_i$iter [jj : jj = j OR $\gamma_{jjt}=1$, t] = $\rho_t^{R_t(+j)}$
    COMP$_{t-1}$ = $S_i$iter [jj : jj = j OR $\gamma_{jjt}=1$, t−1] = $\rho_{t-1}^{R_t(+j)}$ ~ questa non serve ora
    s_1RED = s_1(e.g. lat) [jj : jj ≠ j AND $\gamma_{jjt}=1$]
    s_2RED = s_2(e.g. long) [jj : jj ≠ j AND $\gamma_{jjt}=1$]

    nRED = R_x = $|S_i^{TMP}| = |\rho_t^{R_t(-j)}|$ ~ i.e. they represent how many units we selected
    nRED1 = R_x+1 = $|S_i^{TMP_2}| = |\rho_t^{R_t(+j)}|$
    S_iRED = relabel(S_i^{TMP}) ~ li giro e relabel per tornare con indici bassi (1,2,...) e quindi comodo per gli array dopo
    S_iRED1 = relabel($S_i^{TMP_2}$)
    cit = S_iRED1[end] ~ we added j as last element, now here we have its label
    nclusRED = max(S_iRED) ~ nclusRED ≠ nclusRED1 possible
    nhRED(k) = [quante unità in S_i^{RED} hanno label k]
    nhRED1(k) = [quante unità in S_i^{RED1} hanno label k]

eg 1 2 1 2 2 3 3 1
1 2 1 2 2 3 3 1 $\boxed{4}$
the last added unit j may have a different label wrt units i≠j

    lCo = 0, lCn = 0
    \# unit j can enter an existing cluster ~ perché i label partono da 1, ma C da 0
    for ($k=0$, $k <$ nclusRED)
      s_{1o} = s_1RED [jj ≠ j : S_i^{RED}[jj] == k+1] ~ we take the spatial coords relative to label k
      s_{2o} = //
      s_{1n} = s_{1o} ∪ s_1[j] ~ original vs new (possible) coords
      s_{2n} = //

      lCo = spatial_cohesion(...)
      lCn = spatial_cohesion(...)
      lgweight[k] = ln(nhRED[k]) + lCn − lCo
      S_iRED1[end] = k+1 ~ current label to which we assign unit j
      pcomp = comp(S_i^{RED1}, COMP$_{t-1}$)
      // if(pcomp = 0) lgweight[k] = ln(0) ~ ma è commentato, quindi per ora lo controllo su un controllo che poi non uso

- unit j
- current label k_j
  ↓
simulate to enter cls and compute their cohesion
↓
cl1 ... clk_j ... clk
↓
get normalized probabilities (weights) and use the one for k_j to compute ph
↓
if we can, sample from ph the new γ_{it}

    \# unit j could create a new cluster (a singleton for now)
    s_{1o} = s_1[j]
    s_{2o} = //
    lCn = spatial_cohesion(...)
    lgweight[nclusRED] = ln(M_{DP}) + lCn
    S_iRED1[end] = nclusRED + 1
    pcomp = comp(S_i^{RED1}, COMP$_{t-1}$)
    // if(pcomp = 0) lgweight[nclusRED] = ln(0)
    exponentiate (with numerical safety) the weights and normalize them

    ph = $\dfrac{\alpha_t}{\alpha_t + (1-\alpha_t)\,\text{lgweight}[\text{cit}_{1}-1]}$ ~ to pass from the label to the C index ~ we take the weight corresponding to unit j label, the other k were needed just for the normalization

    if ($\gamma_{jt}=0$)
      COMP$_{t-1}$ = $\rho_{t-1}^{R_t(+j)}$ = S_i^{iter}[jj ≠ j : $\gamma_{jjt}=1$, t−1] ∪ S_i^{iter}[j, t−1]
      COMP$_{1t}$ = $\rho_t^{R_t(+j)}$ = S_i^{iter}[// , t] ∪ S_i^{iter}[j, t]
      ρ^{comp} = comp(COMP$_{t-1}$, COMP$_{1t}$)
      if (ρ^{comp} = 0) ph = 0
    $g_t \sim$ Ber(ph)
    $\gamma_{jt} = g_t$

problem tackled by the if

$S_i^{iter} = \begin{array}{c|ccccccc} & \ldots & t-1 & t & \ldots & T \\ 1 & & 1 & 1 & & \\ 2 & & 1 & 1 & & \\ 3 & & 2 & 2 & & \\ 4 & & 3 & 3 & & \\ j & & 4 & 3 & & \\ 5 & & 2 & 2 & & \\ n & & 1 & 2 & & \end{array}$ R_t

COMP$_{t-1}$ = [1 1 3 ④ 1]
relabeled = [1 1 2 3 1] ~ (≠ ...) not compatible

COMP$_{1t}$ = [1 1 3 ③ 2]
relabeled = [1 1 2 2 3] ~ indeed unit j moved, changed cluster, so γ_{it} has to stay again 0, we can't sample it to 1

INPUT: draws, burnin, thin
nsubjects (n)
ntime (T)
y, s1, s2 (data and spatial coords)
X (the covariate matrix)
M (Dirichlet mass parameter)
α (starting alpha)
model_priors = [μ_0, s_0, A_σ, A_τ, A_λ, b_{η_1}]
α_priors = [a_α, b_α]
time_specific_α
unit_specific_α
update_α
update_η1
update_φ1
sPPM (true if we model the spatial coords)
model_x (true if we model the covariates)
spatial_cohesion
covariates_similarity
cParams (vector for the spatial cohesion params: [μ_0, k_0, ν_0, Λ_0])
mh (vector for metropolis steps update for [σ^2, τ, λ, η_1, φ_1])
space_1 (messo in FALSE in R, se usare lo spazio solo a t=1 o sempre)
simple_model
σ^2_{y} (old) used if simple model is TRUE)

ALGORITMO

# inizializzazione params
nout = $\left(\frac{\text{draws} - \text{burnin}}{\text{thin}}\right)$

ww=0 for saving MCMC iterates

$$S_w\text{-iter} = \begin{array}{c|cc} & 0 \dots & T \\ 1 \\ \vdots & \boxed{1} & \boxed{0} \\ n \end{array}$$

$$\gamma\text{-iter} = \begin{array}{c|c} & 0 \dots T \\ 1 \\ \vdots & 0 \\ n \end{array}$$

$$n_h = \begin{array}{c|ccc} & 0 & 1 \dots & T \\ 1 & n & n \dots & n \\ \vdots & 0 & & 0 \\ n \end{array}$$

unità: come label ora

nclus_iter = $\begin{pmatrix} 0 & 1 & \dots & T \\ 1 & 1 & \dots & 1 \end{pmatrix}$

ora tutti nello stesso cluster iniziale

altre cose inizializzate tutte a 0 (variabili ausiliarie per dopo)

estrazione delle mie
usando le variabili in input

# start effettivo dell'algoritmo
for (i=0, i<draws)
for (t=0, t<T)
update γ
update ρ
update μ
update σ^2
update θ
update τ^2
update η_1
update α
update φ_0
update φ_1
update λ^2

update ρ
ρTMP = copy(ρ_t) = copy(S_i^{iter}[,t])
for (j=0; j<nsubjects)
# we only update the partition relatively to the units than can move (we write $\gamma_{jt}=0$)
if ($\gamma_{jt}=0$) means: controllo con un for che sia effettivamente free
# we now simulate to remove the unit j to the cluster she was belonging to

well not simulate, remove (but only) we actually remove ... S_i^{iter}, the label don't set as -1 or null)

if (unit j ∈ a non-singleton cl)
easy correction
nh[that cl] -= 1
else (unit j ∈ a singleton cl)
less easy correction
relabel to set unit j's cl as the last one then again remove it as in the if
ρTMP = copy(ρ_t) = copy(S_i^{iter}[,t])

# we now simulate the unit to be assigned to one of the existing clusters...
for (h=0; h<nclus_iter[t])
ρTMP[j] = h
$comp_{1t} = \rho_{TMP}^{R_{t+1}} = \rho_t^{R_{t+1}}|label_j=h$
$comp_{t+1} = S_w^{iter}[jj: \gamma_{jj,t+1}=1, t+1] = \rho_{t+1}^{R_{t+1}}$

$$\Pr(c_{it}=h \mid -) \propto \begin{cases} N(Y_{it} \mid \mu^\star_{c_{it}=h,t}, \sigma^{2\star}_{c_{it}=h,t})\Pr(c_{it}=h)\mathrm{I}[\boxed{\rho_t^{h\mathfrak{R}_{t+1}}} = \boxed{\rho_{t+1}^{\mathfrak{R}_{t+1}}}] & \text{for } h=1,\dots,k_t^{-i}, \\ N(Y_{it} \mid \mu^\star_{new_h,t}, \sigma^{2\star}_{new_h,t})\Pr(c_{it}=h)\mathrm{I}[\rho_t^{h\mathfrak{R}_{t+1}} = \rho_{t+1}^{\mathfrak{R}_{t+1}}] & \text{for } h=k_t^{-i}+1, \end{cases} \tag{S.10}$$

ρcomp = compatibility($comp_{1t}$, $comp_{t+1}$)
if (ρcomp=0)
ph[h] = log(0)
else (ρcomp=1)

we think that there is a smarter way (just ... what the ... cluster h and then add 1 to cluster h and then subtract 1)

recompute nhTMP
compute nclusTMP
lpp=0
for (hh=0, hh<nclusTMP)
s_m = [s_{jj} : ρ_{TMP}[jj] = hh+1]
lCn = spatial_cohesion(s_m)
lXn = similarity_function(...)
lpp += log(M_{DP}) + log Γ(nhTMP[hh]) + lCn + lXn

here we finally enter the covariates update

if (t=0) first (c...) time
ph[h] = log($f_N(\mu^\star_{ht}, \sigma^{2\star}_{ht})(y_{jt})$) + lpp
else (t>0)
ph[h] = log($f_N(\mu^\star_{ht} + \eta_{1j}^{iter}\cdot y_{j,t-1}, \sigma^{2\star}_{ht}(1-\eta_{1j}^{iter\,2}))(y_{jt})$) + lpp

autoregressive part for later (>1) times

$$\Pr(c_{it}=h) = Pr(\rho_t^h) \propto M\Gamma(|S_{ht}^{-i} \cup \{i\}|)g(s_{ht}^{-i\star} \cup s_i|\nu_0)\prod_{j\neq h}^{k_t^{-i}} M\Gamma(|S_{jt}^{-i}|)g(s_{jt}^{-i\star}|\nu_0), \tag{S.11}$$

before the loop on k
nh
1 9 9 0
6 0 0 0
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

nclusTMP = 3

now inside the else with k=0
nh_tmp
2 6 1

now inside the else with k=1
nh_tmp
1 7 1

now inside the else with k=2
nh_tmp
1 6 2

$+ x_{t,h}^T \beta_t$

... plus the case for unit j to be assigned to a new cluster
ρTMP[j] = nclus_iter[t]+1
$comp_{1t} = \rho_{TMP}^{R_{t+1}} = \rho_t^{R_{t+1}}|label_j = nclus_t+1$
$comp_{t+1} = S_i^{iter}[jj: \gamma_{jj,t+1}=1, t+1] = \rho_{t+1}^{R_{t+1}}$
ρcomp = compatibility($comp_{1t}$, $comp_{t+1}$)
if (ρcomp=0)
ph[nclus_iter[t]] = log(0)
else (ρcomp=1)
μ_{draw} = rand($N(\theta_t^{iter}, \tau_t^{2\,iter})$)
σ_{draw} = rand($U(0, A_\sigma)$)

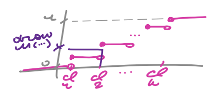
draw the new params for the new cluster

recompute cl numerosity nhTMP
recompute cl number nclusTMP
lpp=0
for (hh=0, hh<nclusTMP)
s_m = [s_{jj} : ρ_{TMP}[jj] = hh+1]
lCn = spatial_cohesion(s_m)
lXn = similarity_function(...)
lpp += log(M_{DP}) + log Γ(nhTMP[hh]) + lCn + lXn
if (t=0) first (c...) time
ph[h] = log($f_N(\mu_{draw}, \sigma^2_{draw})(y_{jt})$) + lpp
else (t>0)
ph[h] = log($f_N(\mu_{draw} + \eta_{1j}^{iter}\cdot y_{j,t-1}, \sigma^2_{draw}(1-\eta_{1j}^{iter\,2}))(y_{jt})$) + lpp

nel caso volessimo inserire un regressore β_t

$+ x_{t,h+1}^T \beta_t$

# now compute the probabilities
ph_TMP = copy(ph)
exponentiate them and normalize them
draw a U(0,1)
assign unit to the sampled cluster
now update all the related stuff
(S_i^{iter}, nh, nclus_iter[t], μ, σ^2)



$S_w^{TMP} = S_i^{iter}$[,t]
$S_i^{TMP_2}$, reorder, old_lab = relabel(S_w^{TMP})
S_i^{iter}[,t] = $S_i^{TMP_2}$
and for nh, params μ, σ^2 we need reorder and relabel variables

serve perché magari da 1 1 1 ... 1 dove consideriamo che è unità j=1 e quindi andava in un nuovo cl (2) e quindi ora avremmo 2 1 1 ... 1 che deve essere rilabellato in 1 2 2 ... 2

$$\gamma_{iter} = \begin{array}{c|c} & \dots\ t+1 \dots \\ j & 1 \\ 2 & 0 \\ 3 & 1 \\ 4 & 0 \\ 5 & 1 \\ 6 & 0 \end{array}$$

$$S_w^{iter} = \begin{array}{c|cc} & t & t+1 \\ j & 2 & 1 \\ 2 & 1 & 2 \\ 3 & 1 & 1 \\ 4 & 2 & 1 \\ 5 & 1 & 1 \\ 6 & 3 & 3 \end{array}$$

label k+1 = 2

$\rho_t^{hR_{t+1}}$ = [2 1 1]
$\rho_{t+1}^{R_{t+1}}$ = [1 1 1]

non compatibili, cioè dobbiamo verificare se con lo assegnare $c_{jt}=2$ perché non è compatibile con ρ_{t+1} che abbiamo già assegnato prima nel codice

- loop on units j
- having $\gamma_{jt}=0$

remove unit j and therefore update nh and params

simulate the unit to be assigned to an existing cl or to a new cl

cl1 cl2 ... clh clh+1 (new)

if assignment is feasible compute their weight

ph[1] ph[2] ph[h] ph[h+1]

- exponentiate and normalize them
- then sample from them

finally relabel and fix params after the label switching

INPUT: draws, burn, thin
nsubjects (n)
ntime (T)
y, s_1, s_2 (data and spatial coords)
X (the covariate matrix)
M (Dirichlet mass parameter)
α (starting alpha)
model_priors = $[m_0, s_0, A_\sigma, A_\tau, A_\lambda, b_{\eta_1}]$
α_priors = $[a_\alpha, b_\alpha]$
time_specific_α
unit_specific_α
update_α
update_η_1
update_φ_1
sPPM (true if we model the spatial coords)
modelX (true if we model the covariates)
spatial_cohesion
covariates_similarity
cParams (vector for the spatial cohesion params: $[\mu_0, k_0, \nu_0, \Lambda_0]$)

mh (vector for metropolis steps update for $[\sigma^2, \tau, \lambda, \eta_1, \varphi_1]$)

space_1 (messo in FALSE in R, era per usare lo spazio solo a $t=1$ o sempre)

simple_model
$\theta_{\tau 2}$ (old) used if simple model is TRUE)

ALGORITMO

initialize some params

nout = $\left(\frac{\text{draws} - \text{burnin}}{\text{thin}}\right)$

ii = 0 (for saving MCMC iterates)

$S_i\text{\_iter} = \begin{array}{c} \\ 1 \\ \vdots \\ n \end{array}\begin{array}{cc} 0 \cdots & T \\ \left(\boxed{1}\right. & \left.\boxed{0}\right) \end{array}$

$\gamma\text{\_iter} = \begin{array}{c} 1 \\ \vdots \\ n \end{array}\begin{pmatrix} 0 & \cdots & T \\ & 0 & \end{pmatrix}$

$n_h = \begin{array}{c} 1 \\ \vdots \\ n \end{array}\begin{pmatrix} \overset{0}{n} & \overset{1}{n} & \cdots & \overset{T}{n} \\ 0 & & & 0 \end{pmatrix}$

unità come label sta

nclus_iter = $\begin{pmatrix} \overset{0}{1} & \overset{1}{1} & \cdots & \overset{T}{1} \end{pmatrix}$ sono tutti nello stesso cluster inizialmente

altre cose inizializzate tutte a 0 (variabili ausiliarie per dopo)

estrazione delle info usando le variabili in input

# start effettivo dell'algoritmo
for ($i=0$, $i<$draws)
  for ($t=0$, $t<T$)
    update γ
    update ρ
    update $\mu^\star_h$
    update $\sigma^{2\star}_h$
    update θ
    update τ^2
  update η_1
  update α
  update φ_0
  update φ_1
  update λ^2

però controllando col codice vengono in effetto calcolati solo così più o non computazionalmente! solo per le iterate dopo la prima

MCMC iterate $i=1$:

$t=0$

$\gamma_{it} = \begin{array}{c} 1 \\ \\ n \end{array}\begin{pmatrix} \overset{t=0}{0} & 0 & \cdots & \overset{T}{0} \\ \vdots & & & \\ 0 & 0 & & 0 \end{pmatrix}$ then $S_i^{\text{iter}} = \begin{array}{c} 1 \\ \\ n \end{array}\begin{pmatrix} \overset{t=0}{E} & \cdots & \overset{T}{} \\ C & & \\ C & & \end{pmatrix}$

no checks

check wrt $\gamma_{it} = 1$ at $t=1$

ma inizialmente lì era tutto zero

$t=1$

$\gamma_{it} = \begin{array}{c} 1 \\ \\ n \end{array}\begin{pmatrix} \overset{0}{0} & \overset{t=1}{E} & 0 & \cdots & \overset{T}{0} \\ \vdots & C & & & \\ 0 & C & 0 & & 0 \end{pmatrix}$ then $S_i^{\text{iter}} = \begin{array}{c} 1 \\ \\ n \end{array}\begin{pmatrix} \overset{0}{E} & \overset{t=1}{\text{NEW}} & \cdots & \overset{T}{} \\ C & E & & \\ C & C & & \\ & C & & \end{pmatrix}$

check wrt $\gamma_{it} = 0$ (bene) at $t=0$ $\to t-1$

check wrt $\gamma_{it} = 1$ at $t=2$ $\to t+1$

ancora zero inizialmente

⋮

MCMC iterate 2:

now also the checks for γ start to be meaningful

INPUT:

- draws, burnin, thin
- nsubjects (n)
- ntime (T)
- y, s1, s2 (data and spatial coords)
- X (the covariate matrix)
- M (Dirichlet mass parameter)
- α (starting alpha)
- model_priors = $[m_0, s_0, A_\sigma, A_\tau, A_\lambda, b_{\eta_1}]$
- α_priors = $[a_\alpha, b_\alpha]$
- time_specific_α
- unit_specific_α
- update_α
- update_η_1
- update_φ_1
- sPPM (true if we model the spatial coords)
- modelX (true if we model the covariates)
- spatial_cohesion
- covariates_similarity
- cParams (vector for the spatial cohesion params: $[\mu_0, k_0, \nu_0, L_0]$)
- mh (vector for metropolis steps update for $[\sigma^2, \tau, \lambda, \eta_1, \varphi_1]$)
- space_1 (messo su FALSE in R, era per usare lo spazio solo a t=1 o sempre)
- simple_model
- $\theta_{\tau 2}$ (old) used if simple model is TRUE)

$$Y_{it} \sim N\left(\mu^*_{c_{it}} + \eta_{1i} Y_{i,t-1} + x_{it}^T \beta_t,\ \sigma^2_{c_{it}}(1-\eta_{1i}^2)\right)$$

$$Y_{i1} \sim N\left(\mu^*_{c_{i1}} + x_{i1}^T \beta_1,\ \sigma^2_{c_{i1}}\right)$$

usiamo solo covariate numeriche?

$$\beta_t \sim N(b_0, s^2_\beta Id)$$

where to them update in the MCMC?

$$\sigma^{*2}_{ht} \sim invGamma(\alpha_\sigma, \beta_\sigma)$$

$$\mu^*_{ht} \sim N(\theta_t, \tau^2_t)$$

$$\xi_i = logit\left(\tfrac{1}{2}(\eta_{1i}+1)\right) \sim Laplace(a, b)$$

$$\tau^2_t \sim invGamma(\alpha_\tau, \beta_\tau)$$

$$\theta_t \sim N\left((1-\varphi_1)\varphi_0 + \varphi_1 \theta_{t-1},\ \lambda^2(1-\varphi_1^2)\right)$$

$$\theta_1 \sim N(\varphi_0, \lambda^2)$$

$$\varphi_0 \sim N(m_0, s_0^2) \qquad \varphi_1 \sim U(-1, 1) \qquad \lambda^2 \sim invGamma(\alpha_\lambda, \beta_\lambda)$$

ALGORITMO

\# inizializzazione params

nout = $\left(\dfrac{\text{draws} - \text{burnin}}{\text{thin}}\right)$

ii = 0 (for saving MCMC iterates)

$$\text{Si\_iter} = \begin{array}{c|cccc} & 0 & \cdots & & T \\ \hline 1 & & & & \\ \vdots & & \boxed{1} & & \boxed{0} \\ n & & & & \end{array}$$

$$\gamma\text{\_iter} = \begin{array}{c|ccc} & 0 & \cdots & T \\ \hline 1 & & & \\ \vdots & & 0 & \\ n & & & \end{array}$$

$$\text{nh} = \begin{array}{c|cccc} & \overset{0}{n} & \overset{1}{n} & \cdots & \overset{T}{n} \\ \hline 1 & & & & \\ \vdots & 0 & & & 0 \\ n & & & & \end{array}$$

intesi come label ora

$$\text{nclus\_iter} = \begin{array}{cccc} 0 & 1 & \cdots & T \\ (1 & 1 & \cdots & 1) \end{array}$$

sono tutti nello stesso cluster inizialmente

altre cose inizializzate tutte a 0 (variabili ausiliarie per dopo)

\# estrazione delle info

usando le variabili in input

\# start effettivo dell'algoritmo

```
for (i=0, i<draws)
    for (t=0, t<T)
        update γ ✓
        update ρ ✓
        update μ*_h ✓ slides
        update σ²*_h ○ ess) non
        update θ ✓ slides
        update τ² ⊘ ess) non
    update η1 metropolis
    update α ✓ slides
    update φ0 ✓ slides
    update φ1 metropolis
    update λ² ⊘ ess) non
β (inoltre a fine loop)
```

vedo nuovi livelli di calcoli / le σ² che aspettano e carlo... dove stanno nella struttura delle Sit

anche per loro aspetto per capire se volevano combinare qualcosa o no

INPUT: draws, burnin, thin
nsubjects (n)
ntime (T)
y, s1, s2 (data and spatial coords)
X (the covariate matrix)
M (Dirichlet mass parameter)
α (starting alpha)
model_priors = [m0, s0, Aσ, Aτ, Aλ, bη1]
α_priors = [aα, bα]
time_specific_α
unit_specific_α
update_α
update_η1
update_φ1
sPPM (true if we model the spatial coords)
modelX (true if we model the covariates)
spatial_cohesion
covariates_similarity
cParms (vector for the spatial cohesion params: [μ0, k0, ν0, L0])
mh (vector for metropolis steps update for [σ², τ, λ, η1, φ1])
space_1 (messo in FALSE in R, era per usare lo spazio solo a t=1 o sempre)
simple_model
$\theta_{\tau 2}$ (old) used if simple model is TRUE)

$$Y_{it} \sim N\left(\mu^*_{c_{it}} + \eta_{1i} Y_{i,t-1} + x_{it}^T \beta_t,\ \sigma^{2}_{c_{it}}(1-\eta_{1i}^2)\right)$$
$$Y_{i1} \sim N\left(\mu^*_{c_{i1}} + x_{i1}^T \beta_1,\ \sigma^{2}_{c_{i1}}\right)$$

usiamo solo covariate numeriche?

$$\beta_t \sim N(b_0, \sigma^2_\beta Id)$$

where to update them in the MCMC?

$$\sigma^{*2}_{jt} \sim invGamma(\alpha_\sigma, \beta_\sigma)$$
$$\mu^*_{jt} \sim N(\theta_t, \tau^2_t)$$
$$\xi_i = logit\left(\tfrac{1}{2}(\eta_{1i}+1)\right) \sim Laplace(a,b)$$
$$\tau^2_t \sim invGamma(\alpha_\tau, \beta_\tau)$$
$$\theta_t \sim N\left((1-\varphi_1)\varphi_0 + \varphi_1\theta_{t-1},\ \lambda^2(1-\varphi_1^2)\right)$$
$$\theta_1 \sim N(\varphi_0, \lambda^2)$$
$$\varphi_0 \sim N(m_0, s_0^2) \qquad \varphi_1 \sim U(-1,1) \qquad \lambda^2 \sim invGamma(\alpha_\lambda, \beta_\lambda)$$

ALGORITMO

\# inizializzare some params
nout = (draws − burnin) / thin
ii = 0 (for saving MCMC iterates)

$$Si\_iter = \begin{array}{c|cc} & 0 \cdots & T \\ 1 \\ \vdots & \boxed{1} & \boxed{0} \\ n \end{array}$$

$$\gamma\_iter = \begin{array}{c|c} & 0 \cdots T \\ 1 \\ \vdots & 0 \\ n \end{array}$$

$$nh = \begin{array}{c|c} & 0\ 1 \cdots T \\ 1 \\ \vdots & 0 \quad\quad 0 \\ n \end{array}$$

iniziali come label era

nclus_iter = (1 1 ... 1) (indices 0 1 ... T) — sono tutti nello stesso cluster iniziale

altre cose inizializzate tutte a 0 (variabili ausiliarie per dopo)

\# estrazione delle info usando le variabili in input

\# start effettivo dell'algoritmo

for (i=0, i<draws)
 for (t=0, t<T)
  update γ ✓
  update ρ ✓
  update μ^*_c ✓ slides
  update σ^{2*}_c O (ex) now
  update θ ✓ slides
  update τ^2 O (ex) now
 update η_1 metropolis
 update α ✓ slides
 update φ_0 ✓ slides
 update φ_1 metropolis
 update λ^2 O (ex) now
 β (inside a for loop)

vedo numeri ... di calcoli / la σ² ... separatamente ... della struttura delle Sit

anche per loro aspettiamo per capire se vogliamo cambiare qualcosa o no

UPDATING β_t

In the case that for now we include all the info of

$$Y_{it} \sim N(\mu^*_{c_{it}} + \eta_{1i} Y_{i,t-1} + x_{it}^T \beta_t, \sigma^{2*}_{c_{it}}(1-\eta_{1i}^2))$$
$$Y_{i1} \sim N(\mu^*_{c_{i1}} + x_{i1}^T \beta_1, \sigma^{2*}_{c_{i1}})$$
$$\beta_t \sim N(b_0, s^2 Id) \quad \text{p-dim vector}$$

- $\boxed{\text{for } t=1:}$

$$f(\beta_t \mid -) \propto f(\beta_t) \cdot f(Y_t) = f(\beta_1) \cdot f(Y_1) = f(\beta_1) \cdot \prod_{i=1}^{n} f(Y_{i,t=1}) =$$
$$= \left[\frac{1}{\sqrt{(2\pi)^p |s^2 Id|}} e^{-\frac{1}{2}(\beta_t - b_0)^T \frac{1}{s^2}(\beta_t - b_0)}\right] \cdot \left[\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^{2*}_{c_{i1}}}} e^{-\frac{1}{2\sigma^{2*}_{c_{i1}}}(Y_{i1} - \mu^*_{c_{i1}} - x_{i1}^T\beta_1)^2}\right] \propto$$
$$\propto \left[\frac{1}{\sqrt{s^{2p}}} e^{-\frac{1}{2s^2}[\beta_t^T\beta_t + b_0^T b_0 - 2\beta_t^T b_0]}\right] \cdot \left[e^{-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{1}{\sigma^{2*}_{c_{i1}}}(Y_{i1} - \mu^*_{c_{i1}} - x_{i1}^T\beta_1)^2\right)}\right] =$$
$$= \left[e^{-\frac{1}{2s^2}(\beta_t^T\beta_t - 2\beta_t^T b_0)}\right] \cdot \left[e^{-\frac{1}{2}\sum \frac{1}{\sigma^{2*}_{c_{i1}}}((Y_{i1} - \mu^*_{c_{i1}}) - x_{i1}^T\beta_1)^2}\right]$$
$$-\frac{1}{2}\left[\left(\sum \frac{(Y_{i1} - \mu^*_{c_{i1}})^2}{\sigma^{2*}_{c_{i1}}}\right) - 2\left(\sum \frac{(Y_{i1} - \mu^*_{c_{i1}}) x_{i1} \cdot \beta_1}{\sigma^{2*}_{c_{i1}}}\right) + \left(\sum \frac{(x_{i1} \cdot \beta_1)^2}{\sigma^{2*}_{c_{i1}}}\right)\right]$$
$$= e^{-\frac{1}{2}\left[\frac{\beta_t^T\beta_t}{s^2} - \frac{2\beta_t \cdot b_0}{s^2}\right] - \frac{1}{2}\left[\left(\sum\frac{(x_{i1}\cdot\beta_t)^2}{\sigma^{2*}_{c_{i1}}}\right) - 2\left(\sum\frac{(\ldots) x_{i1}\cdot\beta_t}{\sigma^{2*}_{c_{i1}}}\right)\right]} =$$
$$= e^{-\frac{1}{2}\left[\ldots - 2\left(\frac{b_0}{s^2} + \sum\frac{(Y_{i1} - \mu^*_{c_{i1}}) x_{i1}}{\sigma^{2*}_{c_{i1}}}\right)\cdot\beta_1\right]} =$$
$$\beta_t^T\left(\frac{1}{s^2} Id\right)\beta_t + \sum\frac{(x_{i1}^T\beta_t)^2}{\sigma^{2*}_{c_{i1}}} = \beta_t^T\left(\sum\frac{x_{i1}x_{i1}^T}{\sigma^{2*}_{c_{i1}}}\right)\beta_t \quad \text{because}$$
$$\left((a^Tb)^2 = \left(\sum a_i b_i\right)^2 \qquad b^T(a a^T) b = b^T[a(a^Tb)] = (b^Ta)(a^Tb) = (a^Tb)(a^Tb) = (\ldots)^2\right) \quad \text{commutative property } b \cdot a = a \cdot b$$
$$= e^{-\frac{1}{2}\left[\beta_t^T \underbrace{\left(\frac{1}{s^2}Id + \sum_i \frac{x_{i1}x_{i1}^T}{\sigma^{2*}_{c_{i1}}}\right)}_{A}\beta_t - 2\underbrace{\left(\frac{b_0}{s^2} + \sum\frac{(Y_{i1} - \mu^*_{c_{i1}}) x_{i1}}{\sigma^{2*}_{c_{i1}}}\right)}_{b^{UP}}\cdot\beta_t\right]}$$
$$= e^{-\frac{1}{2}\left[\beta_t^T A \beta_t - 2 b^{UP}\cdot\beta_t\right]} \propto e^{-\frac{1}{2}\left[(\beta_t - b^{UP})^T A (\beta_t - b^{UP})\right]}$$
$$\propto \boxed{N_p(b^{UP}, A^{-1})}$$

```
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 compatability
            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;
                        }

                      }
//
                      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==0){
//                                 muh[k*(ntime1) + t],
//                                 sqrt(sig2h[k*(ntime1) + t]), 1));
                  ph[k] = dnorm(y[j*(*ntime) + t],
                               muh[k*(ntime1) + 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])),
1));
                  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 compatability
            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] = sig2_tmp[(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++){
        ////////////////////////////////////////
        //
        // udate muh
        //
        ////////////////////////////////////////
```

variable $y^* \in \mathbb{R}^{nt}$, but to save/store the result we assign a certain y_{c_i} to each observation i. Then there will be obs which they have same y if they $\in$ to the same cluster

$$S_i = \underbrace{\begin{pmatrix} 1 \\ 2 \\ 2 \\ 4 \\ | \end{pmatrix}}_{t=4} \Rightarrow y = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_2 \\ y_4 \\ | \end{pmatrix}}_{t=4}$$

```
        if(t==0){
          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 > 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*(*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));
//        muh[k] = 0.0;
        ////////////////////////////////////////
        //                      //
        // udate sig2h                //
        //                      //
        ////////////////////////////////////////
```

in this one and in mu) others the update step is through Metropolis (not & Hastings) method, where:

- we have a symmetric proposal (that's why is only Metropolis) centered at the old value

$\sigma_{new} \sim N(\sigma_{old}, -)$

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

          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)*(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(muh[k*(ntime1) + t], theta_iter[t],
ot,1);
          lln = lln + dnorm(muh[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 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++){

            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 != 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                                 //
    //                                       //

//////////////////////////////////////////////////////////////////////
    phi1sq = phi1_iter*phi1_iter;
    one_phisq = (1-phi1_iter)*(1-phi1_iter);
    lam2tmp = lam2_iter*(1.0 - phi1sq);
    sumt = 0.0;
    for(t=1; 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=1; 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, 1);
        lln = lln + dunif(np1, -1,1, 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=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];
            }
```

```
      }

    }else {

    }
  }
  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 = 0; 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 == 0){
            like_iter[j*(*ntime)+t] = dnorm(y[j*(*ntime)+t], mudraw,
sigdraw, 1);
            fitted_iter[j*(*ntime)+t] = mudraw;

          }
          if(t > 0){
            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 = 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(mnlike[j*(*ntime)+t]));
    }
  }
  waic[0] = -2*elppdWAIC;
  PutRNGstate();

}
```