```c
/**************************************************************
 * 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 "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/units 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
* s1p = nsubject x 1 vector containing spatial coordinate one for prediction (currently not used)
* s2p = nsubject x 1 vector containing spatial coordinate two for prediction (currently not used)
*
* M = double - indicating value of M associated with cohesion (scale parameter of DP).
* alpha = double - prior probability of being pegged, starting value only if  is TRUE
* modelPrior = vector - containing values for prior distributions as follows
*       m0 - mean phi0, s20 - variance of phi0
*       A - upper bound of sigma*_{jt}
*       A0 - upper bound of tau
*   Al - upper bound of lam
*       a - alpha_t shape 1 parameter, b - alpha_t shape 2 parameter
*   be - scale parameter of eta.
*
* global_alpha = integer - logical indicating wether to make alpha time-specific or one global alpha.
* alpha_0 = integer - logical indicating whether  alpha = 0 or not.
* eta1_0 = integer - logical indicating whether  eta1 = 0 or not.
* phi1_0 = integer - logical indicating whether  phi1 = 0 or not.
* 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
* mh -
* verbose  - logical indicating if information should be printed to screen
*
* OUTPUT
* Si -
* mu -
* sig2 -
* eta1
* theta -
* tau2 -
* phi0 -
* phi1 -
* lam2 -
* gamma -
* alpha.out -
* like
* lpml -
* waic -
****************************************************************************************/



void mcmc_drpm_ar1(int *draws, int *burn, int *thin, int *nsubject, int *ntime,
                          double *y, double *s1, double *s2, double *M,
                          double *modelPriors,
                          int *global_alpha, int *alpha_0, int *eta1_0, int *phi1_0,
                          int *sPPM, int *SpatialCohesion, double *cParms, double *mh, int *verbose,
                          int *Si, double *mu, double *sig2, double *eta1, double *theta, double *tau2,
                          double *phi0, double *phi1, double *lam2, int *gamma, double *alpha_out,
                          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
        // p - prediction iterate

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

        int nout = (*draws - *burn)/(*thin);

    if(*verbose){
          Rprintf("nsubject = %d\n", *nsubject);
          Rprintf("ntime = %d\n", *ntime);
          Rprintf("nout = %d\n", nout);
        }

        // ===================================================================================
        //
        // 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, 0.0);
        double *theta_iter = R_VectorInit(ntime1, 0.0);
        double *tau2_iter = R_VectorInit(ntime1, 1.0);

        double phi0_iter = 0.0;
        double phi1_iter = 0.0;
        double lam2_iter = 1.0;

        double *alpha_iter = R_VectorInit(ntime1, 0.0);

        // ===================================================================================
        //
        // Memory vectors to hold MCMC iterates for cluster specific parameters
        //
        // ===================================================================================

        double *muh = R_VectorInit((*nsubject)*(ntime1), 0.0);
        double *sig2h = R_VectorInit((*nsubject)*(ntime1), 0.5);


        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;
                }
        }

        // 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;
        int nclus_redtmp=0, nh_redtmp[*nsubject], n_redtmp=0, nh_tmp[*nsubject];
        int nh_redtmp_no_zero[*nsubject], nh_red_no_zero[*nsubject],nh_tmp_no_zero[*nsubject];
        double lpp_full=0.0, lpp_red=0.0;

        int nh_red_1[*nsubject];
        int nclus_redtmp_1=0, nh_redtmp_1[*nsubject], n_redtmp_1=0, nh_tmp_1[*nsubject];
        int nh_redtmp_no_zero_1[*nsubject], nh_red_no_zero_1[*nsubject],nh_tmp_no_zero_1[*nsubject];
        double lpp_full_1=0.0, lpp_red_1=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_red_no_zero[j] = 0; nh_tmp_no_zero[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[(*nsubject)];
        int rho_tmp[*nsubject], Si_tmp[*nsubject], Si_tmp2[*nsubject];
        int oldLab[*nsubject], reorder[*nsubject];
        int iaux, Rindx1, Rindx2, n_tmp, nclus_tmp, n_tmp_1, nclus_tmp_1,rho_comp, indx;
        double auxm, auxs, mudraw, sigdraw, maxph, denph, cprobh, uu, lCo, lCn, lCn_1;
        double *ph = R_VectorInit(*nsubject, 0.0);
        double *phtmp = R_VectorInit(*nsubject, 0.0);
        double *probh = 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;

        // 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], predSi_iter[*nsubject];


        // ===================================================================================
        //
        // Prior parameter values
        //
        // ===================================================================================

        // upper bound for sig, tau, lam
        double A=modelPriors[2];
        double A0=modelPriors[3];
        double Al=modelPriors[4];

        // priors for phi0
        double m0 = modelPriors[0], s20 = modelPriors[1];

        // priors for alpha
        double a = modelPriors[5], b = modelPriors[6];

        //priors for eta1
        double b_eta1 = modelPriors[7];

    if(*verbose){
          Rprintf("Prior values: m0 = %.2f, s20 = %.2f\n \v Asig = %.2f, Atau = %.2f, Alam = %.2f,\n \t a = %.2f,
b = %.2f, b_eta1 = %.2f\n",
                   m0, s20, A, A0, Al, a, b, b_eta1);
    }
        // DP weight parameter
        double Mdp = *M;

//      Rprintf("Mdp = %f\n", Mdp);

        // 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];
//      Rprintf("k0 = %f\n", k0);
//      RprintVecAsMat("L0", L0, 2, 2);


//      RprintVecAsMat("mh", mh, 1, 5);
        // M-H step tunning parameter
        double csigSIG=mh[0], csigTAU=mh[1], csigLAM=mh[2], csigETA1=mh[3], csigPHI1=mh[4];

//      Rprintf("csigETA1 = %f\n", csigETA1);



        GetRNGstate();


        // ===================================================================================
        //
        // start of the mcmc algorithm;
        //
        // ===================================================================================

        for(i = 0; i < *draws; i++){

                if(*verbose){
                        if((i+1) % 5000 == 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++){

//                      Rprintf("t = %d\n", t);


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

                        // The complete partition does not change as gammas change
                        for(j=0;j<*nsubject;j++){
                                nh_red[j]=0; nh_redtmp[j]=0; nh_tmp[j] = 0;
                        }

                        for(k = 0; k < nclus_iter[t]; k++){
                                nh_tmp[k] = nh[k*(ntime1) + t];
                        }


                        // Note this value does not change even as gamma changes.  The only
                        // concern is that rho_t-1 and rho_t are compatible.
                        lpp_full = partition_prob_crp(nh_tmp, nclus_iter[t], Mdp, *nsubject, 1);

//                      Rprintf("lpp_full = %f\n", lpp_full);



                        ////////////////////////////////////////////////////////////////////////////////
                        // find the reduced partition information
                        // i.e., number of units clustered, number of clusters, size of clusters;
                        ////////////////////////////////////////////////////////////////////////////////
                        n_red = 0; n_redtmp=0;
                        for(j = 0; j < *nsubject; j++){
                                if(gamma_iter[j*(ntime1) + t] == 1){
                                        nh_red[Si_iter[j*(ntime1) + t]-1] = nh_red[Si_iter[j*(ntime1) + t]-1]+1;
                                        n_red=n_red+1;

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

                                        n_redtmp=n_redtmp+1;
                                }

                        }



                        nclus_red = 0, nclus_redtmp=0;
                        for(j = 0; j < *nsubject; j++){
                                if(nh_red[j] > 0) nclus_red = nclus_red + 1;
                                if(nh_redtmp[j] > 0) nclus_redtmp = nclus_redtmp + 1;
                        }
                        ////////////////////////////////////////////////////////////////////////////////

//                      Rprintf("nclus_red = %d\n", nclus_red);
//                      Rprintf("n_red = %d\n \n \n \n", n_red);


                        remove_zero(nh_red, *nsubject, nh_red_no_zero);

//                      RprintIVecAsMat("nh_red_no_zero", nh_red_no_zero, 1, *nsubject);

//                      RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("Si_iter", Si_iter, *nsubject, ntime1);


                        for(j = 0; j < *nsubject; j++){
//                              Rprintf("t = %d\n", t);
//                              Rprintf("j = %d\n", j);


//                              RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                              Rprintf("gamma_iter[j*(ntime1) + t] = %d\n", gamma_iter[j*(ntime1) + t]);

                                // at time period one, all gammas are zero (none are ‘‘pegged’’)
                                if(t == 0){
                                        gamma_iter[j*(ntime1) + t] = 0;
                                } else {

                                        // this may need to be updated depending on if the value of gamma changes

//                                      RprintIVecAsMat("nh_red_no_zero", nh_red_no_zero, 1, *nsubject);

                                        lpp_red = partition_prob_crp(nh_red_no_zero, nclus_red, Mdp, n_red, 1);


//                                      Rprintf("lpp_full = %f\n", lpp_full);
//                                      Rprintf("lpp_red = %f\n", lpp_red);

                                        // If gamma is 1 at current MCMC iterate, then there are no
                                        // concerns about partitions being incompatible as gamma changes
                                        // from 1 to 0.


                                        if(gamma_iter[j*(ntime1) + t] == 1){

//                                              Rprintf("Starting at gamma=1 and staying at gamma=1 \n");
                                                // if gamma remains 1, then no changes
                                                ph[1] = lpp_full - lpp_red + log(alpha_iter[t]);
//                                              Rprintf("ph[1] = %f\n", ph[1]);


                                                // if gamme moves from 1 to 0 need to remove one unit from
                                                // rho_t.R

                                                nh_redtmp[Si_iter[j*ntime1+t]-1] = nh_red[Si_iter[j*ntime1+t]-1] -
 1      ;
                                                n_redtmp = n_redtmp - 1;

                                                remove_zero(nh_redtmp, *nsubject, nh_redtmp_no_zero);

//                                              RprintIVecAsMat("nh_redtmp", nh_redtmp, 1, *nsubject);

//                                              RprintIVecAsMat("nh_redtmp_no_zero", nh_redtmp, 1, *nsubject);

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

//                                              Rprintf("Starting at gamma=1 and moving to gamma=0 \n");

//                                              RprintIVecAsMat("nh_red", nh_red, 1, *nsubject);
//                                              RprintIVecAsMat("nh_redtmp", nh_redtmp, 1, *nsubject);

//                                              Rprintf("lpp_full = %f\n", lpp_full);
                                                lpp_red = partition_prob_crp(nh_redtmp_no_zero, nclus_redtmp, Mdp,
 n_redtmp, 1);

//                                              Rprintf("llp_red = %f\n", lpp_red);

                                                ph[0] = lpp_full - lpp_red + log(1-alpha_iter[t]);


                                        }




                                        // 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){

//                                              Rprintf("lpp_red = %f\n", lpp_red);
                                                // if gamma remains zero.  nothing changes to evaluate
//                                              Rprintf("Starting at zero and staying at zero \n");

//                                              Rprintf("lpp_full = %f\n", lpp_full);
//                                              Rprintf("lpp_red = %f\n", lpp_red);
//                                              Rprintf("log(1-alpha_iter[t]) = %f\n", log(1-alpha_iter[t]));

                                                ph[0] = lpp_full - lpp_red + log(1-alpha_iter[t]);

//                                              Rprintf("ph[0] = %f\n", ph[0]);
                                                // to move from gamma=0 to gamma=1 need to make sure that partitio
ns

                                                // remain compatible, if compatible .

//                                              RprintIVecAsMat("Si_iter", Si_iter, *nsubject, ntime1);

                                                // to move from gamma=0 to gamma=1 need to add unit to rho_t.R

//                                              Rprintf("Starting at zero and moving to one \n");

                                                nh_redtmp[Si_iter[j*ntime1 + t] - 1] = nh_red[Si_iter[j*ntime1+t]-
1] + 1;

                                                n_redtmp = n_red + 1;
                                                nclus_redtmp=0;
                                                for(jj = 0; jj < *nsubject; jj++){
                                                        if(nh_redtmp[jj] > 0) nclus_redtmp = nclus_redtmp + 1;
                                                }

//                                              RprintIVecAsMat("nh_redtmp", nh_redtmp, 1, *nsubject);
//                                              Rprintf("n_redtmp = %d\n", n_redtmp);
//                                              Rprintf("nclus_redtmp = %d\n", nclus_redtmp);
```

```
                                                // 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 w
hen
                                                        // 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;
                                                        }
                                                }

//                                              Rprintf("Rindx1 = %d\n", Rindx1);
//                                              RprintIVecAsMat("comptm1", comptm1, 1, *nsubject);
//                                              RprintIVecAsMat("comp1t", comp1t, 1, *nsubject);

                                                rho_comp = compatibility(comptm1, comp1t, Rindx1);

//                                              Rprintf("rho_comp = %d\n", rho_comp);
                                                if(rho_comp==1){

                                                        remove_zero(nh_redtmp, *nsubject, nh_redtmp_no_zero);

//                                                      RprintIVecAsMat("nh_redtmp_no_zero", nh_redtmp_no_zero, 1,
 *nsubject);


                                                        lpp_red = partition_prob_crp(nh_redtmp_no_zero, nclus_redt

mp, Mdp, n_redtmp, 1);

//                                                      Rprintf("lpp_full = %f\n", lpp_full);
//                                                      Rprintf("lpp_red = %f\n", lpp_red);
//                                                      Rprintf("log(alpha_iter[t]) = %f\n", log(alpha_iter[t]));

                                                        ph[1] = lpp_full - lpp_red + log(alpha_iter[t]);

                                                } else {

                                                        ph[1] = log(0); // partitions are not compatible

                                                }

//                                              Rprintf("ph[0] = %f\n", ph[0]);


//                                              RprintVecAsMat("ph", ph, 1, 2);



                                        }


//                                      RprintVecAsMat("ph = ", ph, 1, 2);

                                        maxph = ph[0]; if(maxph < ph[1]) maxph=ph[1];
//                                      Rprintf("maxph = %f\n", maxph);

                                        ph[0] = exp(ph[0] - maxph); ph[1] = exp(ph[1] - maxph);

//                                      RprintVecAsMat("ph = ", ph, 1, 2);

                                        probh[1] = ph[1]/(ph[0] + ph[1]);

//                                      RprintVecAsMat("probh = ", probh, 1, 2);

//                                      Rprintf("probh[1] = %f\n", probh[1]);

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

//                                      Rprintf("gt = %d\n", gt);
```

```
                                        if(gt != gamma_iter[j*(ntime1) + t]){

                                                gamma_iter[j*(ntime1) + t] = gt;
                                                n_red = n_redtmp;
                                                nclus_red = nclus_redtmp;
                                                nh_red[Si_iter[j*(ntime1)+t]-1] = nh_redtmp[Si_iter[j*(ntime1)+t]-
1];

                                        } else {

                                                nh_redtmp[Si_iter[j*(ntime1)+t]-1] = nh_red[Si_iter[j*(ntime1)+t]-
1];

                                                nclus_redtmp = nclus_red;
                                                n_redtmp = n_red;
                                        }

                                }

                                remove_zero(nh_red, *nsubject, nh_red_no_zero);


//                              Rprintf("gamma_iter[j*(ntime1) + t] = %d\n", gamma_iter[j*(ntime1) + t]);
//                              Rprintf("nclus_red = %d\n", nclus_red);
//                              Rprintf("n_red = %d\n", n_red);
//                              RprintIVecAsMat("nh_red = ", nh_red, 1, *nsubject);
//                              RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                              RprintIVecAsMat("Si_iter", Si_iter, *nsubject, ntime1);

                        }


//                      RprintIVecAsMat("Si_iter", Si_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);


//                      Rprintf("Begin updating partition for time %d\n", t+1);


                        ////////////////////////////////////////////////////////////////////////////////
                        //
                        // update partition
                        //
                        ////////////////////////////////////////////////////////////////////////////////
                        // The cluster probabilities depend on four partition probabilities
                        //
                        // rho_t
                        // rho_t.R
                        // rho_t+1
                        // rho_t+1.R
                        //
                        // 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];

                        }

//                      RprintIVecAsMat("rho_tmp", rho_tmp, 1, *nsubject);

                        // 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++){
//                              Rprintf("t ======= %d\n", t);
//                              Rprintf("j ======= %d\n", j);

                                // Only need to update partition relative to units that are not pegged
                                if(gamma_iter[j*(ntime1) + t] == 0){

                                        if(nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] > 1){

                                                // Observation belongs to a non-singleton ...
                                                nh[(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t] = nh[(Si_iter[j*(ntim
e1) + t]-1)*(ntime1) + t] - 1;

                                        }else{

                                                // Observation is a member of a singleton cluster ...
                                                iaux = Si_iter[j*(ntime1) + t];
//                                              Rprintf("iaux = %d\n", iaux);
                                                if(iaux < nclus_iter[t]){

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


                                                        // 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 par
ameters
                                                        // so that the newly labeled subjects from previous step r
etain
                                                        // their correct cluster specific parameters
                                                        auxs = sig2h[(iaux-1)*ntime1 + t];
                                                        sig2h[(iaux-1)*ntime1 + t] = sig2h[(nclus_iter[t]-1)*(ntim
e1)+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 t
he 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];

                                        }

//                                      RprintIVecAsMat("Si_iter", Si_iter, *nsubject, ntime1);
//                                      RprintIVecAsMat("nh ", nh,  *nsubject, ntime1);
//                                      RprintIVecAsMat("rho_tmp", rho_tmp, 1, *nsubject);

//                                      Rprintf("nclus_iter[t] = %d\n", nclus_iter[t]);



                                        for(k = 0; k < nclus_iter[t]; k++){
//                                              Rprintf("k === %d\n\n", k);

                                                // Beginning of spatial part
                                                lCo = lCn = 0.0;
                                                if(*sPPM==1){
                                                        indx = 0;
                                                        for(jj = 0; jj < *nsubject; jj++){

                                                                if(Si_iter[jj*(ntime1) + t] == k+1 & j != jj){

//                                                                      Rprintf("indx = %d\n", indx);
```

```
                                                                        s1o[indx] = s1[jj];
                                                                        s2o[indx] = s2[jj];

                                                                        s1n[indx] = s1[jj];
                                                                        s2n[indx] = s2[jj];

                                                                        indx = indx+1;
                                                                }
                                                                if(j == jj){
                                                                        s1n[nh[k*(ntime1) + t]] = s1[jj];
                                                                        s2n[nh[k*(ntime1) + t]] = s2[jj];

                                                                }


                                                        }
//                                                      RprintVecAsMat("s1o", s1o, 1, nh[k*(ntime1) + t]);
//                                                      RprintVecAsMat("s2o", s2o, 1, nh[k*(ntime1) + t]);
//                                                      RprintVecAsMat("s1n", s1n, 1, nh[k*(ntime1) + t]+1);
//                                                      RprintVecAsMat("s2n", s2n, 1, nh[k*(ntime1) + t]+1);
//                                                      Rprintf("Cohesion = %d\n", *SpatialCohesion);

                                                        lCo = Cohesion3_4(s1o, s2o, mu0, k0, v0, L0, nh[k*(ntime1)
 + t], *SpatialCohesion, 1);

                                                        lCn = Cohesion3_4(s1n, s2n, mu0, k0, v0, L0, nh[k*(ntime1)
 + t]+1,*SpatialCohesion, 1);
                                                }
                                                // End of spatial part

//                                              Rprintf("lCo = %f\n", lCo);
//                                              Rprintf("lCn = %f\n", lCn);
                                                rho_tmp[j] = k+1;

//                                              RprintIVecAsMat("rho_tmp", rho_tmp, 1, *nsubject);
//                                              RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                                              RprintIVecAsMat("Si_iter", Si_iter, *nsubject, ntime1);


                                                // 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;
                                                        }
                                                }


//                                              Rprintf("Rindx2 = %d\n", Rindx2);
//                                              RprintIVecAsMat("comp2t", comp2t, 1, *nsubject);
//                                              RprintIVecAsMat("comptp1", comptp1, 1, *nsubject);

                                                // check for compatibility
                                                rho_comp = compatibility(comp2t, comptp1, Rindx2);

//                                              Rprintf("rho_comp = %d\n", rho_comp);


                                                if(rho_comp != 1){

                                                        ph[k] = log(0); // Not compatible

                                                } else {
                                                        // Need to compute Pr(rhot), Pr(rhot.R), Pr(rhot+1), Pr(rh
ot+1.R)


                                                        for(jj = 0; jj < *nsubject; jj++){
                                                                nh_tmp[jj] = 0;
                                                                nh_redtmp[jj] = 0;
                                                                nh_tmp_1[jj] = 0;
                                                                nh_redtmp_1[jj] = 0;
                                                        }
                                                        n_tmp = 0;
                                                        n_redtmp = 0;
                                                        n_tmp_1 = 0;
                                                        n_redtmp_1 = 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;
                                                                if(gamma_iter[jj*ntime1 + t] == 1){
                                                                        nh_redtmp[rho_tmp[jj]-1] = nh_redtmp[rho_t
mp[jj]-1]+1;
```

```
                                                                        n_redtmp = n_redtmp+1;
                                                                }

                                                                nh_tmp_1[Si_iter[jj*ntime1 + (t+1)]-1] = nh_tmp_1[
Si_iter[jj*ntime1 + (t+1)]-1]+1;

                                                                n_tmp_1=n_tmp_1+1;
                                                                if(gamma_iter[jj*ntime1 + t+1] == 1){
                                                                        nh_redtmp_1[Si_iter[jj*ntime1 + (t+1)]-1]
= nh_redtmp_1[Si_iter[jj*ntime1 + (t+1)]-1]+1;

                                                                        n_redtmp_1 = n_redtmp_1+1;
                                                                }

                                                        }

//                                                      RprintIVecAsMat("nh_tmp", nh_tmp, 1, *nsubject);
//                                                      RprintIVecAsMat("nh_redtmp", nh_redtmp, 1, *nsubject);

//                                                      Rprintf("nsubject = %d\n", *nsubject);
//                                                      Rprintf("n_redtmp = %d\n", n_redtmp);

                                                        remove_zero(nh_tmp, *nsubject, nh_tmp_no_zero);
                                                        remove_zero(nh_redtmp, *nsubject, nh_redtmp_no_zero);

                                                        remove_zero(nh_tmp_1, *nsubject, nh_tmp_no_zero_1);
                                                        remove_zero(nh_redtmp_1, *nsubject, nh_redtmp_no_zero_1);

//                                                      RprintIVecAsMat("nh_tmp_no_zero",nh_tmp_no_zero, 1, *nsubj
ect);
//                                                      RprintIVecAsMat("nh_redtmp_no_zero", nh_redtmp_no_zero, 1,
 *nsubject);


                                                        nclus_redtmp=0;
                                                        nclus_tmp=0;
                                                        nclus_redtmp_1=0;
                                                        nclus_tmp_1=0;
                                                        for(jj = 0; jj < *nsubject; jj++){
                                                                if(nh_redtmp[jj] > 0) nclus_redtmp = nclus_redtmp
+ 1;
                                                                if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
                                                                if(nh_redtmp_1[jj] > 0) nclus_redtmp_1 = nclus_red
tmp_1 + 1;
                                                                if(nh_tmp_1[jj] > 0) nclus_tmp_1 = nclus_tmp_1 + 1
;

                                                        }

//                                                      Rprintf("nclus_tmp = %d\n", nclus_tmp);
//                                                      Rprintf("nclus_redtmp = %d\n", nclus_redtmp);

                                                        lpp_full = partition_prob_crp(nh_tmp_no_zero, nclus_tmp, M
dp, *nsubject, 1);
                                                        lpp_red = partition_prob_crp(nh_redtmp_no_zero, nclus_redt
mp, Mdp, n_redtmp, 1);


                                                        lpp_full_1 = partition_prob_crp(nh_tmp_no_zero_1, nclus_tm
p_1, Mdp, *nsubject, 1);
                                                        lpp_red_1 = partition_prob_crp(nh_redtmp_no_zero_1, nclus_
redtmp_1, Mdp, n_redtmp_1, 1);


//                                                      Rprintf("lpp_full = %f\n", lpp_full);
//                                                      Rprintf("lpp_red = %f\n", lpp_red);

//                                                      Rprintf("lpp_full_1 = %f\n", lpp_full_1);
//                                                      Rprintf("lpp_red_1 = %f\n", lpp_red_1);


//                                                      Rprintf("lpp_full = %f\n", lpp_full);
//                                                      Rprintf("lpp_red = %f\n", lpp_red);


//                                                      Rprintf("muh[k*(ntime1) + t] = %f\n", muh[k*(ntime1) + t])
;
//                                                      Rprintf("sigh[k*(ntime1) + t] = %f\n", sqrt(sig2h[k*(ntime
1) + t]));
//                                                      Rprintf("y[j*(*ntime) + t] = %f\n", y[j*(*ntime) + t]);
//                                                      Rprintf("nh[k] = %d\n", nh_tmp[k]);
//                                                      Rprintf("dnorm(y[j*(*ntime) + t], muh[k*(ntime1) + t], sqr
t(sig2h[k*(ntime1) + t]), 1) = %f\n", dnorm(y[j*(*ntime) + t], muh[k*(ntime1) + t], sqrt(sig2h[k*(ntime1) + t]), 1
));

                                                        if(t==0){

                                                                ph[k] = dnorm(y[j*(*ntime) + t],
                                                                               muh[k*(ntime1) + t],
                                                                               sqrt(sig2h[k*(ntime1) + t]), 1) +
                                                                        lpp_full - lpp_red +
                                                                        lpp_full_1 - lpp_red_1 +
                                                                        lCn - lCo; // Spatial part of cohesion fun
ction;

                                                        }
                                                        if(t > 0){
                                                                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_full - lpp_red +
                                                                        lpp_full_1 - lpp_red_1 +
                                                                        lCn - lCo; // Spatial part of cohesion fun
ction;

                                                        }

                                                        // use this to test if MCMC draws from prior are correct
//                                                      ph[k] =  lpp_full - lpp_red + lpp_full_1 - lpp_red_1;


//                                                      Rprintf("ph[k] = %f\n", ph[k]);

                                                }

                                        }

//                                      RprintVecAsMat("ph = ", ph, 1, nclus_iter[t] );


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

//                                      Rprintf("nclus_iter[t] = %d\n", nclus_iter[k]);

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

//                                      RprintIVecAsMat("rho_tmp", rho_tmp, 1, *nsubject);
//                                      RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);

                                        // 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;
                                                }
                                        }

//                                      Rprintf("Rindx2 = %d\n", Rindx2);
//                                      RprintIVecAsMat("comp2t", comp2t, 1, *nsubject);
//                                      RprintIVecAsMat("comptp1", comptp1, 1, *nsubject);


                                        // check for compatibility
                                        rho_comp = compatibility(comp2t, comptp1, Rindx2);


//                                      Rprintf("rho_comp = %d\n", rho_comp);

                                        if(rho_comp != 1){
                                                ph[nclus_iter[t]] = log(0); // going to own cluster is not compati
ble;
                                        } else {

//                                              RprintIVecAsMat("nh ", nh,  *nsubject, ntime1);

//                                              Rprintf("mu_iter[t] = %f\n", theta_iter[t]);
//                                              Rprintf("sqrt(tau2_iter[t]) = %f\n", sqrt(tau2_iter[t]));

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

//                                              Rprintf("mudraw = %f\n", mudraw);
//                                              Rprintf("sigdraw = %f\n", sigdraw);
 //                             Rprintf("y[j*(*ntime) + t] = %f\n", y[j*(*ntime) + t]);

//                                              RprintIVecAsMat("nh_tmp", nh_tmp, 1, nclus_iter[t]);
```

```
						for(jj = 0; jj < *nsubject; jj++){
							nh_tmp[jj] = 0;
							nh_redtmp[jj] = 0;
							nh_tmp_1[jj] = 0;
							nh_redtmp_1[jj] = 0;
						}
						n_tmp = 0;
						n_redtmp = 0;
						n_tmp_1 = 0;
						n_redtmp_1 = 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;
							if(gamma_iter[jj*ntime1 + t] == 1){
								nh_redtmp[rho_tmp[jj]-1] = nh_redtmp[rho_tmp[jj]-1
]+1;
								n_redtmp = n_redtmp+1;
							}

							nh_tmp_1[Si_iter[jj*ntime1 + (t+1)]-1] = nh_tmp_1[Si_iter[
jj*ntime1 + (t+1)]-1]+1;
							n_tmp_1=n_tmp_1+1;
							if(gamma_iter[jj*ntime1 + t+1] == 1){
								nh_redtmp_1[Si_iter[jj*ntime1 + (t+1)]-1] = nh_red
tmp_1[Si_iter[jj*ntime1 + (t+1)]-1]+1;
								n_redtmp_1 = n_redtmp_1+1;
							}

						}

//						RprintIVecAsMat("nh_tmp", nh_tmp, 1, *nsubject);
//						RprintIVecAsMat("nh_redtmp", nh_redtmp, 1, *nsubject);

						remove_zero(nh_tmp, *nsubject, nh_tmp_no_zero);
						remove_zero(nh_redtmp, *nsubject, nh_redtmp_no_zero);

						remove_zero(nh_tmp_1, *nsubject, nh_tmp_no_zero_1);
						remove_zero(nh_redtmp_1, *nsubject, nh_redtmp_no_zero_1);

//						RprintIVecAsMat("nh_tmp_no_zero",nh_tmp_no_zero, 1, *nsubject);
//						RprintIVecAsMat("nh_redtmp_no_zero", nh_redtmp_no_zero, 1, *nsubje
ct);

						nclus_redtmp=0;
						nclus_tmp=0;
						nclus_redtmp_1=0;
						nclus_tmp_1=0;
						for(jj = 0; jj < *nsubject; jj++){
							if(nh_redtmp[jj] > 0) nclus_redtmp = nclus_redtmp + 1;
							if(nh_tmp[jj] > 0) nclus_tmp = nclus_tmp + 1;
							if(nh_redtmp_1[jj] > 0) nclus_redtmp_1 = nclus_redtmp_1 +
1;
							if(nh_tmp_1[jj] > 0) nclus_tmp_1 = nclus_tmp_1 + 1;

						}

//						Rprintf("nclus_tmp = %d\n", nclus_tmp);
//						Rprintf("nclus_redtmp = %d\n", nclus_redtmp);

						lpp_full = partition_prob_crp(nh_tmp_no_zero, nclus_tmp, Mdp, *nsu
bject, 1);
						lpp_red = partition_prob_crp(nh_redtmp_no_zero, nclus_redtmp, Mdp,
 n_redtmp, 1);

						lpp_full_1 = partition_prob_crp(nh_tmp_no_zero_1, nclus_tmp_1, Mdp
, *nsubject, 1);
						lpp_red_1 = partition_prob_crp(nh_redtmp_no_zero_1, nclus_redtmp_1
, Mdp, n_redtmp_1, 1);

//						RprintIVecAsMat("nh_tmp", nh_tmp, 1, nclus_iter[t]+1);
//						Rprintf("nclus_iter = %d\n", nclus_iter[t]+1);

						lpp_full = partition_prob_crp(nh_tmp_no_zero, nclus_tmp, Mdp, *nsu
bject, 1);
						lpp_red = partition_prob_crp(nh_redtmp_no_zero, nclus_redtmp, Mdp,
 n_redtmp, 1);

						lpp_full_1 = partition_prob_crp(nh_tmp_no_zero_1, nclus_tmp_1, Mdp
, *nsubject, 1);
						lpp_red_1 = partition_prob_crp(nh_redtmp_no_zero_1, nclus_redtmp_1
, Mdp, n_redtmp_1, 1);

//						Rprintf("lpp_full = %f\n", lpp_full);
//						Rprintf("lpp_red = %f\n", lpp_red);
```

```
//                                              Rprintf("lpp_full_1 = %f\n", lpp_full_1);
//                                              Rprintf("lpp_red_1 = %f\n", lpp_red_1);

//                                              Rprintf("dnorm(y[j*(*ntime) + t], mudraw, sigdraw, 1) = %f\n", dno
rm(y[j*(*ntime) + t], mudraw, sigdraw, 1));


                                                // spatial part
                                                lCn_1 = 0.0;
//                                              Rprintf("lCn_1 = %f\n", lCn_1);
//                                              Rprintf("sPPM = %d\n", *sPPM);
                                                if(*sPPM==1){
                                                        s1o[0] = s1[j];
                                                        s2o[0] = s2[j];
                                                        lCn_1 = Cohesion3_4(s1o, s2o, mu0, k0, v0, L0, 1, *Spatial
Cohesion, 1);
                                                }

//                                              Rprintf("lCn_1 = %f\n", lCn_1);

                                                if(t==0){
                                                        ph[nclus_iter[t]] = dnorm(y[j*(*ntime) + t], mudraw, sigdr
aw, 1) +
                                                                                                           lpp_full -
 lpp_red +
                                                                                                           lpp_full_1
 - lpp_red_1 +
                                                                                                           lCn_1; //t
his is spatial part
                                                }
                                                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_i
ter[j]), 1) +
                                                                                                           lpp_full -
 lpp_red +
                                                                                                           lpp_full_1
 - lpp_red_1 +
                                                                                                           lCn_1; //t
his is spatial part

                                                }

//                                              ph[nclus_iter[t]] =  lpp_full - lpp_red + lpp_full_1 - lpp_red_1;


//                                              Rprintf("ph[nclus_iter[t]] = %f\n", ph[nclus_iter[t]]);

                                        }

//                                      RprintVecAsMat("ph = ", ph, 1, nclus_iter[t] + 1);
//                                      RprintIVecAsMat("rhotmp = ", rho_tmp, 1, *nsubject);

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


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

//                                      RprintVecAsMat("phtmp ", phtmp, 1, nclus_iter[t]+1);


                                        maxph = phtmp[nclus_iter[t]];

//                                      Rprintf("maxph = %f\n", maxph);

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

                                                ph[k] = exp(ph[k] - maxph);
//                                              ph[k] = pow(exp(ph[k] - maxph), (1 - exp(-0.0001*(i+1))));
                                                denph = denph + ph[k];

                                        }

//                                      RprintVecAsMat("ph", ph, 1, nclus_iter[t]+1);

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

                                                probh[k] = ph[k]/denph;
                                        }
//                                      Rprintf("denph = %f\n", denph);

//                                      RprintVecAsMat("probh", probh, 1, nclus_iter[t]+1);

                                        uu = runif(0.0,1.0);
//                                      Rprintf("uu = %f\n", uu);

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

                                                cprobh = cprobh + probh[k];

                                                if (uu < cprobh){

                                                        iaux = k+1;
                                                        break;
                                                }
                                        }


//                                      Rprintf("iaux = %d\n \n \n", iaux);

                                        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;

                                        }

//                                      Rprintf("Si_iter[j*(ntime1) + t] = %d\n", Si_iter[j*(ntime1) + t]);
//                                      RprintVecAsMat("muh", muh, *nsubject, ntime1);
//                                      RprintVecAsMat("sig2h", sig2h, *nsubject, ntime1);
//                                      RprintIVecAsMat("Si_iter ", Si_iter, *nsubject, ntime1);
//                                      RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                                      RprintIVecAsMat("nh ", nh, *nsubject, ntime1);
//                                      RprintIVecAsMat("nclus_iter", nclus_iter, 1, ntime1);

                                }


                        }
//                      RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("Si_iter ", Si_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("nh ", nh, *nsubject, ntime1);
//                      RprintIVecAsMat("nclus_iter", nclus_iter, 1, ntime1);



                        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);

//                      RprintIVecAsMat("gamma_iter", gamma_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("Si_tmp2", Si_tmp2, 1, *nsubject);
//                      RprintIVecAsMat("reorder", reorder, 1, *nsubject);
//                      RprintIVecAsMat("oldLab", oldLab, 1, nclus_iter[t]);


//                      RprintIVecAsMat("Si_iter ", Si_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("nh ", nh, *nsubject, ntime1);
//                      RprintIVecAsMat("nclus_iter", nclus_iter, 1, ntime1);


//                      RprintVecAsMat("muh", muh, *nsubject, ntime1);
//                      RprintVecAsMat("sig2h", sig2h, *nsubject, ntime1);

                        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)];
                        }


//                      RprintIVecAsMat("Si_iter ", Si_iter, *nsubject, ntime1);
//                      RprintIVecAsMat("nh ", nh, *nsubject, ntime1);
//                      RprintIVecAsMat("nclus_iter", nclus_iter, 1, ntime1);

//                      RprintVecAsMat("muh", muh, *nsubject, ntime1);
//                      RprintVecAsMat("sig2h", sig2h, *nsubject, ntime1);

//                      for(k = 0; k < nclus_iter[t]; k++) sig2h[k*(ntime1)+t] = 1.0;




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

//                              Rprintf("k = %d\n", k);

                                /////////////////////////////////////////
                                //                                                                          //
                                // udate muh                                                        //
                                //                                                                          //
                                /////////////////////////////////////////

//                              Rprintf("sumy = %f\n", sumy);
//                              Rprintf("nh[k*(ntime1)+t] = %d\n",  nh[k*(ntime1)+t]);
//                              Rprintf("sig2h[k*(ntime1) + t] = %f\n",  sig2h[k*(ntime1) + t]);
//                              Rprintf("tau2_iter[t] = %f\n",  tau2_iter[t]);
//                              Rprintf("theta_iter[t] = %f\n",  theta_iter[t]);

                                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_i
ter[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]);

                                }
//                              Rprintf("sume2 = %f\n", sume2);
//                              Rprintf("sumy = %f\n", sumy);
//                              Rprintf("mstar = %f\n", mstar);
//                              Rprintf("sqrt(s2star) = %f\n", sqrt(s2star));
```

```
                                muh[k*(ntime1) + t] = rnorm(mstar, sqrt(s2star));

//                              muh[k] = 0.0;
//                              Rprintf("muh[k*(ntime1) + t] = %f\n", muh[k*(ntime1) + t]);

//                              RprintVecAsMat("muh", muh, *nsubject, ntime1);


                                /////////////////////////////////////////
                                //                                                                          //
                                // udate sig2h                                            //
                                //                                                                          //
                                /////////////////////////////////////////
                                osig = sqrt(sig2h[k*(ntime1) + t]);
                                nsig = rnorm(osig,csigSIG);

                                if(nsig > 0.0 & nsig < A){

                                        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);
                                                        }
                                                }
                                        }

//                                      Rprintf("ms = %f\n", ms);
//                                      Rprintf("osig = %f\n", osig);
//                                      Rprintf("nsig = %f\n", nsig);
                                        llo = llo + dunif(osig, 0.0, A, 1);
                                        lln = lln + dunif(nsig, 0.0, A, 1);


//                                      Rprintf("llo = %f\n", llo);
//                                      Rprintf("lln = %f\n", lln);

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

//                                      Rprintf("llr = %f\n", llr);
//                                      Rprintf("log(uu) = %f\n", log(uu));

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

//                                      sig2h[k*(ntime1) + t] = 1.0;

                                }

//                              Rprintf("sig2h[k*(ntime1) + t] = %f\n", sig2h[k*(ntime1) + t]);

//                              RprintVecAsMat("sig2h", sig2h, *nsubject, ntime1);



                        }


//                      RprintVecAsMat("muh", muh, *nsubject, ntime1);
//                      RprintVecAsMat("sig2h", sig2h, *nsubject, ntime1);
```

```
                        ///////////////////////////////////////////////////////////////////////////////
                        //
                                                                //
                        // update theta (mean of mh)
                        //
                        //
                                                                //
                        ///////////////////////////////////////////////////////////////////////////////
                        summu = 0.0;
                        for(k = 0; k < nclus_iter[t]; k++){
                                summu = summu + muh[k*(ntime1) + t];
//                              Rprintf("nh[k*(ntime1)+t] = %d\n",  nh[k*(ntime1)+t]);
                        }
//                      Rprintf("summu = %f\n", summu);
//                      Rprintf("nclus_iter[t] = %d\n",  nclus_iter[t]);

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

                        if(t==0){
//                              Rprintf("t = %d\n", t);

                                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))
);

//                              Rprintf("mstar = %f\n", mstar);
//                              Rprintf("sqrt(s2star) = %f\n", sqrt(s2star));

                        } else if(t==(*ntime-1)){

//                              Rprintf("t = %d\n", t);
                                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]
));
//                              Rprintf("mstar = %f\n", mstar);
//                              Rprintf("sqrt(s2star) = %f\n", sqrt(s2star));

                        } 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)));

                        }

//                      Rprintf("mstar = %f\n", mstar);
//                      Rprintf("sqrt(s2star) = %f\n", sqrt(s2star));

                        theta_iter[t] = rnorm(mstar, sqrt(s2star));

//                      Rprintf("theta_iter = %f\n", theta_iter[t]);


                        ///////////////////////////////////////////////////////////////////////////////
                        //
                                                                //
                        // 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);
                                }

//                              Rprintf("ms = %f\n", ms);
//                              Rprintf("osig = %f\n", osig);
//                              Rprintf("nsig = %f\n", nsig);
                                llo = llo + dunif(ot, 0.0, A0, 1);
                                lln = lln + dunif(nt, 0.0, A0, 1);


//                              Rprintf("llo = %f\n", llo);
//                              Rprintf("lln = %f\n", lln);

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

//                              Rprintf("llr = %f\n", llr);
//                              Rprintf("log(uu) = %f\n", log(uu));

                                if(log(uu) < llr){

                                        tau2_iter[t] = nt*nt;

//                                      tau2_iter[t] = 5*5;
                                }


                        }


//                      Rprintf("tau2_iter = %f\n", tau2_iter[t]);


                }


//              RprintIVecAsMat("Si_iter ", Si_iter, *nsubject, ntime1);

                ////////////////////////////////////////////////////////////////////////////////
                //
                                                //
                // update eta1 (temporal correlation parameter at likelihood)                          //
                //
                                                //
                ////////////////////////////////////////////////////////////////////////////////
                if(*eta1_0==0){
                        for(j = 0; j < *nsubject; j++){
//                              Rprintf("j = %d\n", j);

//                              Rprintf("eta1_iter = %f\n", eta1_iter[j]);

                                e1o = eta1_iter[j];
                                e1n = rnorm(e1o, csigETA1);

//                              Rprintf("e1o = %f\n", e1o);
//                              Rprintf("e1n = %f\n", e1n);

                                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*(nt
ime1) + t]-1)*(ntime1) + t] +
                                                                            e1o*y[j*(*ntime)+t-1],
                                                                        sqrt(sig2h[(Si_iter[j*(ntime1) + t]-1)*(nt
ime1) + t]*
                                                                                   (1-e1o*e1o)), 1);

                                                lln = lln + dnorm(y[j*(*ntime)+t],
                                                                                                muh[(Si_iter[j*(nt
ime1) + t]-1)*(ntime1) + t] +
                                                                            e1n*y[j*(*ntime)+t-1],
                                                                            sqrt(sig2h[(Si_iter[j*(ntime1) + t]-1)
*(ntime1) + t]*
                                                                                     (1-e1n*e1n)), 1);

                                        }

//                                      Rprintf("llo = %f\n", llo);
//                                      Rprintf("lln = %f\n", lln);

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

//                                      Rprintf("logito = %f\n", logito);
//                                      Rprintf("logitn = %f\n", logitn);

//                                      Rprintf("(1/b_eta1)*fabs(logito - 0.0) = %f\n", (1/b_eta1)*fabs(logito - 0
.0));
//                                      Rprintf("(1/b_eta1)*fabs(logitn - 0.0) = %f\n", (1/b_eta1)*fabs(logitn - 0
.0));

//                                      Rprintf("fabs(logito - 0.0) = %f\n", fabs(logito - 0.0));
//                                      Rprintf("fabs(logitn - 0.0) = %f\n", fabs(logitn - 0.0));

                                        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);

//                                      Rprintf("llo = %f\n", llo);
//                                      Rprintf("lln = %f\n", lln);

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


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

//                                      Rprintf("eta1_iter = %f\n", eta1_iter[j]);
                                }
                        }
                }

//              RprintVecAsMat("eta1", eta1_iter, 1, *nsubject);

                //////////////////////////////////////////////////////////////////////////////
                //
                                                //
                // update alpha
                                //
                //
                                                //
                //////////////////////////////////////////////////////////////////////////////
                if(*alpha_0 == 0){
                        if(*global_alpha == 1){
                                sumg = 0;
                                for(j = 0; j < *nsubject; j++){
                                        for(t = 1; t < *ntime; t++){

                                                sumg = sumg + gamma_iter[j*ntime1 + t];

                                        }


                                }

//                              Rprintf("sumg = %d\n", sumg);
                                astar = (double) sumg + a;
                                bstar = (double) ((*nsubject)*(*ntime-1) - sumg) + b;

//                              Rprintf("astar = %f\n", astar);
//                              Rprintf("bstar = %f\n", bstar);

                                alpha_tmp = rbeta(astar, bstar);

                                for(t=0;t<*ntime;t++){alpha_iter[t] = alpha_tmp;}
//                              Rprintf("alpha_iter = %f\n", alpha_iter);

                        } else {

                                for(t = 0; t < *ntime; t++){
                                        sumg = 0;
                                        for(j = 0; j < *nsubject; j++){
                                                sumg = sumg + gamma_iter[j*ntime1 + t];
                                        }


//                                      Rprintf("sumg = %d\n", sumg);

                                        astar = (double) sumg + a;
                                        bstar = (double) ((*nsubject) - sumg) + b;

//                                      Rprintf("astar = %f\n", astar);
//                                      Rprintf("bstar = %f\n", bstar);


                                        alpha_iter[t] = rbeta(astar, bstar);

                                }

                        }

                        alpha_iter[0] = 0.0;

                }
```

```
//              RprintVecAsMat("alpha_iter", alpha_iter, 1, *ntime);


                //////////////////////////////////////////////////////////////////////////////
                //
                                                //
                // update phi0
                                //
                //
                                                //
                //////////////////////////////////////////////////////////////////////////////
                phi1sq = phi1_iter*phi1_iter;
                one_phisq = (1-phi1_iter)*(1-phi1_iter);
                lam2tmp = lam2_iter*(1.0 - phi1sq);
//              Rprintf("lam2tmp = %f\n", lam2tmp);
                sumt = 0.0;
                for(t=1; t<*ntime; t++){
//                      Rprintf("t = %d\n", t);
//                      Rprintf("theta_iter[t] = %f\n", theta_iter[t]);
//                      Rprintf("theta_iter[t-1] = %f\n", theta_iter[t-1]);
//                      Rprintf("phi1_iter = %f\n", phi1_iter);
//                      Rprintf("(theta_iter[t] - phi1_iter*theta_iter[t-1]) = %f\n",(theta_iter[t] - phi1_iter*th
eta_iter[t-1]));

                        sumt = sumt + (theta_iter[t] - phi1_iter*theta_iter[t-1]);
//                      Rprintf("sumt = %f\n", sumt);
                }

                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);

//              Rprintf("sumt = %f\n", sumt);
//              Rprintf("mstar = %f\n", mstar);
//              Rprintf("s2star = %f\n", s2star);

//              Rprintf("m0 = %f\n", m0);
//              Rprintf("s20 = %f\n", s20);

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

//              Rprintf("phi0_iter = %f\n", phi0_iter);

                //////////////////////////////////////////////////////////////////////////////
                //
                                                //
                // update phi1
                                //
                //
                                                //
                //////////////////////////////////////////////////////////////////////////////

                if(*phi1_0==0){
                        op1 = phi1_iter;
                        np1 = rnorm(op1, csigPHI1);

//                      Rprintf("op1 = %f\n", op1);
//                      Rprintf("np1 = %f\n", np1);

                        if(np1 > -1 & np1 < 1){
                                llo = 0.0, lln = 0.0;
                                for(t=1; t < *ntime; t++){
//                                      Rprintf("t = %d\n", t);
//                                      Rprintf("theta_iter[t] = %f\n", theta_iter[t]);
//                                      Rprintf("theta_iter[t-1] = %f\n", theta_iter[t-1]);
//                                      Rprintf("lam2_iter = %f\n", lam2_iter);
//                                      Rprintf("lam2_iter*(1.0 - op1*op1) = %f\n", lam2_iter*(1.0 - op1*op1));
//                                      Rprintf("lam2_iter*(1.0 - np1*np1) = %f\n", lam2_iter*(1.0 - np1*np1));
//                                      Rprintf("phi0_iter + op1*theta_iter[t-1] = %f\n", phi0_iter + op1*theta_it
er[t-1]);
//                                      Rprintf("phi0_iter + np1*theta_iter[t-1] = %f\n", phi0_iter + np1*theta_it
er[t-1]);


                                        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);

//                                      Rprintf("llo = %f\n", llo);
//                                      Rprintf("lln = %f\n", lln);
                                }
//                              Rprintf("llo = %f\n", llo);
//                              Rprintf("lln = %f\n", lln);
                                llo = llo + dunif(op1, -1,1, 1);
                                lln = lln + dunif(np1, -1,1, 1);
                                llr = lln - llo;
//                              Rprintf("llr = %f\n", llr);
                                if(llr > log(runif(0,1))) phi1_iter = np1;
                        }
                }
//              Rprintf("phi1_iter = %f\n", phi1_iter);

                //////////////////////////////////////////////////////////////////////////////
                //
                                                //
                // update lam2
                                //
                //
                                                //
                //////////////////////////////////////////////////////////////////////////////
//              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) + al;
//              bstar = 0.5*ssq + 1/bl;

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

                // 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-p
hi1sq),1);
                                lln = lln + dnorm(theta_iter[t],
                                                  phi0_iter*(1-phi1_iter) + phi1_iter*theta_iter[t-1], nl*sqrt(1-p
hi1sq),1);
                        }
                        llo = llo + dnorm(theta_iter[0], phi0_iter, ol, 1) + dunif(ol, 0.0, Al, 1);
                        lln = lln + dnorm(theta_iter[0], phi0_iter, nl, 1) + dunif(nl, 0.0, Al, 1);

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

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




//              Rprintf("lam2_iter = %f\n", lam2_iter);

                //////////////////////////////////////////////////////////////////////////////
                //
                // predict partition for new time period
                //
                // THIS HAS YET TO BE FINISHED
                //
                //////////////////////////////////////////////////////////////////////////////
/*
                for(p = 0; p < *npred; p++){

                        for(j=0; j<*nsubject; j++){
                                nh_pred[j] = 0;
                                predSi_iter[j*(*npred) + p] = 0;
                        }
//                      RprintIVecAsMat("nh_pred", nh_pred, 1, *nsubject);

                        if(*alpha_0 == 1){
                                n_red = 0;
                                for(j=0;j<*nsubject;j++){

                                        gpred[j] = rbinom(1,0.0);

                                        if(gpred[j] == 1){
                                                nh_pred[Si_iter[j*(ntime1)+(*ntime)-1] - 1] = nh_pred[Si_iter[j*(n
time1)+(*ntime)-1] - 1] + 1;
```

```
                                                n_red = n_red + 1;

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

                        }


                        if(*alpha_0 == 0){
                                if(*global_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_i
ter[j*(ntime1)+(*ntime)-1] - 1] + 1;
                                                        n_red = n_red + 1;

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

                                }else {

                                }
                        }

//                      RprintIVecAsMat("predSi_iter", predSi_iter, *npred, *nsubject);
//                      RprintIVecAsMat("gpred", gpred, 1, *nsubject);
//                      RprintIVecAsMat("nh_pred", nh_pred, 1, *nsubject);
//                      Rprintf("n_red = %d\n", n_red);

                        remove_zero(nh_pred, *nsubject, nh_tmp_no_zero);
//                      RprintIVecAsMat("nh_tmp_no_zero", nh_tmp_no_zero, 1, *nsubject);

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

//                      Rprintf("nclus_tmp = %d\n", nclus_tmp);

                        for(j=0;j<*nsubject;j++){
//                              Rprintf("j = %d\n", 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);

//                                      RprintVecAsMat("probh = ", probh, 1, nclus_tmp+1);

                                        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;
                                                }
                                        }

//                                      Rprintf("iaux = %d\n", iaux);
                                        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;

//                                      RprintIVecAsMat("predSi_iter", predSi_iter, *npred, *nsubject);
//                                      RprintIVecAsMat("nh_pred", nh_pred, 1, *nsubject);
//                                      Rprintf("nclus_tmp = %d\n", nclus_tmp);
//                                      Rprintf("n_red = %d\n", n_red);
                                }
                        }
                }
*/
                ////////////////////////////////////////////////////////////////////////////////////////////
                //
                // evaluating likelihood that will be used to calculate LPML and WAIC?
                // (see page 81 Christensen Hansen and Johnson)
                //
                ////////////////////////////////////////////////////////////////////////////////////////////
                if((i > (*burn-1)) & ((i+1)  % (*thin) == 0)){

                        like0=0;
                        for(j = 0; j < *nsubject; j++){
//                              Rprintf("j = %d\n", j);
                                for(t = 0; t < *ntime; t++){
//                                      Rprintf("t = %d\n", t);

//                                      Rprintf("Si_iter[j*(ntime1) + t] = %d\n", Si_iter[j*(ntime1) + t]);
//                                      Rprintf("(Si_iter[j*(ntime1) + t]-1)*(ntime1) + t = %d\n", (Si_iter[j*(nti
me1) + t]-1)*(ntime1) + t);

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

//                                      Rprintf("mudraw = %f\n", mudraw);
//                                      Rprintf("sigdraw = %f\n", sigdraw);
//                                      Rprintf("eta1_iter[j] = %f\n", eta1_iter[j]);
//                                      Rprintf("y[j*(*ntime)+t] = %f\n", y[j*(*ntime)+t]);
//                                      Rprintf("y[j*(*ntime)+t-1] = %f\n",y[j*(*ntime)+t-1]);

                                        if(t == 0){
 //                                       Rprintf("mudraw = %f\n", mudraw);
//                                        Rprintf("sigdraw = %f\n", sigdraw);
//                                        Rprintf("y[j*(*ntime)+t] = %f\n", y[j*(*ntime)+t]);

                                          like_iter[j*(*ntime)+t] = dnorm(y[j*(*ntime)+t], mudraw, sigdraw, 1);
                      fitted_iter[j*(*ntime)+t] = mudraw;
//                                        Rprintf("like_iter = %f\n", like_iter[j*(*ntime)+t]);
                                        }
                                        if(t > 0){

//                                              Rprintf("mudraw + eta1_iter[j]*y[j*(*ntime)+t-1] = %f\n", mudraw +
 eta1_iter[j]*y[j*(*ntime)+t-1]);
                                                like_iter[j*(*ntime)+t] = dnorm(y[j*(*ntime)+t],
                                                                                                         mudraw + e
ta1_iter[j]*y[j*(*ntime)+t-1],
                                                                                                         sigdraw*sq
rt(1-eta1_iter[j]*eta1_iter[j]), 1);
                        fitted_iter[j*(*ntime)+t] = mudraw + eta1_iter[j]*y[j*(*ntime)+t-1];

                                        }

//                                      Rprintf("like_iter = %f\n", like_iter[j*(*ntime)+t]);

                                        // 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;


//                                      Rprintf("mnlike = %f\n", mnlike[j*(*ntime)+t]);
//                                      Rprintf("mnllike = %f\n", mnllike[j*(*ntime)+t]);

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

                                        CPO[j*(*ntime)+t] = CPO[j*(*ntime)+t] + (1/(double) nout)*(1/exp(like_iter
[j*(*ntime)+t]));

                                }
                        }
//                      Rprintf("like0 = %d\n", like0);
//                      if(like0==1) nout_0 = nout_0 + 1;
//                      if(i == (*draws-1)) Rprintf("xb = %f\n", xb);
//                      Rprintf("nout_0 = %d\n", nout_0);
//                      if(like0==0){
//                              Rprintf("nout - nout_0 = %d\n", nout - nout_0);
//                              for(j = 0; j < *nsubject; j++){
//                                      for(t = 0; t < *ntime; t++){
//                                              Rprintf("like_iter[j*(*ntime)+t] = %f\n", like_iter[j*(*ntime)+t])
;
//                                              Rprintf("exp(like_iter[j*(*ntime)+t]) = %f\n", exp(like_iter[j*(*n
time)+t]));
//                                              CPO[j*(*ntime)+t] = CPO[j*(*ntime)+t] + (1/exp(like_iter[j*(*ntime
)+t]));

//                                              Rprintf("CPO = %f\n", CPO[j*(*ntime)+t]);

//                                      }
//                              }
//                      }
//            RprintVecAsMat("llike", like_iter, *nsubject, *ntime);
                }



                ///////////////////////////////////////////////////////////////////////////////////
                //
                                                        //
                // Save MCMC iterates
                                //
                //
                                                        //
                ///////////////////////////////////////////////////////////////////////////////////
                if((i > (*burn-1)) & ((i+1) % *thin == 0)){


//                      Rprintf("ii = %d\n", ii);
//                      RprintVecAsMat("theta", theta_iter, 1, *ntime);

                        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++){
//                                      Rprintf("(Si_iter[j]-1)*(ntime1) + t = %d\n", (Si_iter[j*(ntime1) + t]-1)*
(ntime1) + t);

                                        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)*(n
time1) + 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;
//                      Rprintf("ii = %d\n", ii);

                }




        }



        lpml_iter=0.0;
        for(t = 0; t < *ntime; t++){
//              Rprintf("t = %d\n", t);
                for(j = 0; j < *nsubject; j++){
//                      Rprintf("j = %d\n", j);
//                      Rprintf("CPO = %f\n", CPO[j*(*ntime)+t]);

//                      lpml_iter = lpml_iter - log((1/(double) nout-nout_0)*CPO[j*(*ntime)+t]);
                        lpml_iter = lpml_iter - log(CPO[j*(*ntime)+t]);

                }
        }
//      Rprintf("nout_0 = %d\n", nout_0);
        lpml[0] = lpml_iter;
//      Rprintf("lpml_iter = %f\n", lpml_iter);


        elppdWAIC = 0.0;
//      RprintVecAsMat("mnlike",mnlike, 1,(*nsubject)*(ntime1));
//      RprintVecAsMat("mnllike", mnllike, 1, (*nsubject)*(ntime1));

        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();




}
```