```
#include
#include "nuts.h"
#include
#include
// #include "armadillo"
#include

/*Definitions */
#define NArgs 17
#define StrLen 40
#define ArgExt ".arg"
#define ResExt ".res"
#define StatExt ".sts"
#define MAXBUFSIZE ((int) 1e6)

//
//using namespace std;
//using namespace arma;
/*Global variables */
/*Seeds for the kiss generator (see alea) */

/*Files */
Generator g;
char DataFile[StrLen], StatFile[StrLen];
ofstream parameterFP,zFP, StatFP,testfile;
/*Prior hyperparameters */
int Kmax;
/*Sampler settings */
int NOut, SubSamp, NIt;

/*Fixed k move */
double PFixed;
/*Birth and death */
double PBirth, PDeath, PFixed_or_BD;
/*Split and merge */
double PSplit, PMerge;
int Curve_num, THREAD_NUM;
int Nm;
double w_eps,v0_eps,sigmav2_eps;

/* Function prototypes / struct STATS { string split_or_merge,acc_or_rej,simu_acc_or_simu_rej,delete_empty_component; double sm_prob,simu_prob; void initialize(){
split_or_merge = "NULL"; acc_or_rej = "NULL"; simu_acc_or_simu_rej = "NULL"; delete_empty_component = "NULL"; sm_prob = 0.0; simu_prob = 0.0; } void print(){
cout << split_or_merge <<" "<* argv, curve Data[] );
MatrixXd readMatrix(const char filename); void write_data(pq_point & theta, double logl,STATS & stats,VectorXi & z,int in); void RJMH_birth_or_death(curve Data[],
pq_point & m,double logl,VectorXi & z,STATS & stats);
void RJMH_split_or_merge(curve Data[], pq_point & m, double* logl,STATS & stats);
double Simulated_Annealing( double *logl, double *logl_old,pq_point & theta, pq_point & theta_old,int in,STATS & stats);

int main(int argc, char** argv) {
curve Data[MaxM];
pq_point theta(1);
double ran;
double logl;
int in;
VectorXi z;
STATS stats;
```

```
    stats.initialize();
    time_t t_start,t_end;
    ofstream TimeFP("time.txt");
    double DiffTime;
    t_start = time(NULL);

    read_parameters(argc, argv, Data);

    draw_initial_model(Data, theta, &logl);
```

```
// theta.print("theta:");
Gibbs_Sampling_z(Data, theta,z);
write_data(theta, logl, stats,z,0);
```

```
        t_end = time(NULL);
        DiffTime = difftime(t_end,t_start);
        TimeFP<<"0 "<<DiffTime<<endl;

        /* Main loop                                    */
        pq_point theta_old;
        double logl_old,ratio= 1.0;
        for (in=1; in<=NIt; in++) {
                cout<<in<<"th iter"<<endl;
                /* Fixed k move                                 */
                /*_____________update pi________________________________*/
                if( kiss(g)<0.6) {
                        RJMH_split_or_merge(Data, theta, &logl, stats);

                }
                else{
                        /*************sample pi*************************/
                        cout<<"gibbs sample z:"<<endl;
                        Gibbs_Sampling_z(Data,theta,z);
                        cout<<"gibbs sample pi"<<endl;
                        Gibbs_Sampling_pi(theta,z);
                        cout<<"nuts"<<endl;
                        /*____________________sample theta__________________*/
                        sample_nuts_cpp(Data,theta,z);
                        cout<<"after nuts, v:"<<endl;
                        cout<<theta.v<<endl;
                        cout<<"gibbs sample pi:"<<endl;
                        Gibbs_Sampling_pi(theta,z);
                        cout<<"gibbs sample z:"<<endl;
                        Gibbs_Sampling_z(Data,theta,z);
                        /*_________________change pi and K____________________________*/
                        cout<<"delete empty component:"<<endl;
                        RJMH_birth_or_death(Data, theta, &logl,z,stats);
                        //split_or_merge,acc_or_rej,simu_acc_or_simu_rej,delete_empty_component;
                        stats.print();
                        cout<<"after "<<stats.split_or_merge<<stats.acc_or_rej<<", v:"<<endl;
                        cout<<theta.v<<endl;
                }

                if(in>1) {
                        ratio = Simulated_Annealing(&logl,&logl_old,theta,theta_old,in,stats);
                }

                theta_old = theta;
                logl_old = logl;
                /* Note: The test below is true for in = 0 and in = NIt        */
                //if ((div(in, SubSamp)).rem == 0){
                // Gibbs_Sampling_z(Data,theta,z);
                write_data(theta, logl, stats,z, in);
                t_end = time(NULL);
                DiffTime = difftime(t_end,t_start);
                TimeFP<<in<<" "<<DiffTime<<" ";
                cout<<"Difftime:"<<DiffTime<<"log likelihood:"<<logl<<endl;
                if (in%10 == 0) TimeFP<<endl;

        }
        TimeFP.close();
        testfile.close();
        return(0);
```

}

MatrixXd readMatrix(const char *filename)

{

int cols = 0, rows = 0;

double buff[MAXBUFSIZE];

```
        // Read numbers from file into buffer.
        ifstream infile;
        infile.open(filename);
        while (!infile.eof())
        {
                string line;
                getline(infile, line);

                int temp_cols = 0;
                stringstream stream(line);
                while(!stream.eof())
                        stream >> buff[cols*rows+temp_cols++];

                if (temp_cols == 0)
                        continue;

                if (cols == 0)
                        cols = temp_cols;

                rows++;
        }
```

```
infile.close();
// cout<<c/ols<<endl;
rows--;

// Populate matrix with numbers.
MatrixXd result(rows,cols);
for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
                result(i,j) = buff[ cols*i+j ];

return result;
```

};

// /* Read parameters (including seeds for the random generator) and data / //

void read_parameters(int argc, char* argv, curve Data[]) {

testfile.open("generator.res");

char argfile[StrLen];

```
strcpy(StatFile, argv[1]);
strcpy(DataFile, argv[1]);
strcpy(argfile, argv[1]);

strcat(argfile, ArgExt);
ifstream argfp(argfile);
argfp>>g.i>>g.j>>g.k>>NOut>>THREAD_NUM>>SubSamp>>Kmax>>PFixed>>PBirth>>PDeath>>PSplit;
argfp.close();
/* Compute frequently used quantities                    */
NIt = NOut*SubSamp;
PMerge = 1.0 - (PFixed + PBirth + PDeath + PSplit);
PFixed_or_BD = PFixed + PBirth + PDeath;

/* Read data                                */
strcat(DataFile,".dat");
MatrixXd AAA =  readMatrix(DataFile);


Curve_num = AAA.rows()/2;
Nm = AAA.cols();

for(int m=0; m<Curve_num; m++) {
        Data[m].X = AAA.row(2 * m).segment(0, Nm - 1).transpose();
        Data[m].Y = AAA.row(2 * m + 1).segment(0, Nm - 1).transpose();
}
// cout<<Data[0].X<<endl;
// cout<<Data[0].Y<<endl;
w_eps = 5e-6;
v0_eps = 0.5;
sigmav2_eps = 0.005;
printf( "Data has %d curves. Running %d x %d iterations of the sampler...\n", Curve_num, NOut, SubSamp);
cout<<THREAD_NUM<<endl;
```

}

// /* Write iterations on disk / /*/

void write_data(pq_point & theta, double logl, STATS & stats,VectorXi & z,int in) {

```
if (in == 0) {
        /* Initial value, we need to open the files                */
        /* Open statfile                            */
        strcat(StatFile, StatExt);
        StatFP.open(StatFile);

        parameterFP.open("parameter.res");
        zFP.open("z.res");
}

else{
        StatFP<<in<<" "<<theta.w.size()<<" "<<logl<<" ";
        stats.write_file(StatFP);
        // <<stats.split_or_merge
        // <<" "<<stats.acc_or_rej<<" "<<stats.sm_prob<<" "<<
        // stats.simu_acc_or_simu_rej<<" "<<stats.simu_prob<<endl;
        // ResFP<<in<<endl;
        theta.write_file(parameterFP);
        zFP<<z.transpose()<<endl;
}
if(in == NOut) {
        StatFP.close();
        parameterFP.close();
        zFP.close();
}
```

}

void RJMH_birth_or_death(curve Data[], pq_point & m,double* logl,VectorXi & z,STATS & stats) {
int K=m.w.size();
VectorXi label_num=VectorXi::Zero(K);
int k;

```
        for(int i = 0; i< Curve_num; i++) {k = z(i); label_num(k)+=1;}
        VectorXi empty_component(K);
        int j=0, empty_component_num=0;
        for(int k=0; k<K; k++) {
                if(label_num(k)==0)
                {empty_component(j)=k; j++;}
        }
        empty_component_num = j;
        for(int i=0; i<empty_component_num; i++) {
                m.deleteP_seq(empty_component(i)-i);
        }

        m.pi = m.pi/m.pi.sum();

        if(empty_component_num>0) {
                stats.delete_empty_component = "delete";
        }
        else stats.delete_empty_component = "reserve";
        *logl = log_likelihood2(Data,m);
```

}

// /* Implements the RJ MH move based on split/merge proposal / //
void RJMH_split_or_merge(curve Data[], pq_point & m, double logl, STATS & stats) {
double prop_ratio, add_logratio,ratio;
double logl_new;
pq_point m_new;
int k, k1, k2,K=m.w.size();
double split, accept;
VectorXi z_new;
double secondary_moment;
/* Take care of case 1, 2, M-1 and M components (this could be done / / more simply) */
if (K == 1) {
split = 1;

```
                prop_ratio = PMerge/(1.0-PFixed_or_BD);
        }
        else if (K == Kmax) {
                split = 0;
                prop_ratio = PSplit/(1.0-PFixed_or_BD);
        }
        else {
                if (kiss(g) < PSplit/(1.0-PFixed_or_BD)) {
                        /* Split                                */
                        split = 1;
                        if (K == (Kmax-1))
                                prop_ratio = (1.0-PFixed_or_BD)/PSplit;
                        else
                                prop_ratio = PMerge/PSplit;
                }
                else {
                        /* Merge                                */
                        split = 0;
                        if (K == 2)
                                prop_ratio = (1.0-PFixed_or_BD)/PMerge;
                        else
                                prop_ratio = PSplit/PMerge;
                }
        }


        if(split) {
                cout<<"split:"<<endl;
                stats.split_or_merge = "split";
                m_new = m;
                k = (int)floor((double)K *kiss(g));
                if(k==K) k--;
                /* Proposes a split move and returns log-likelihood            */

                logl_new = prop_split(Data, m_new, k,&k1,&k2);
                secondary_moment = calc_secondary_moment((Data[0]),m,m_new,k,k1,k2);
                add_logratio = compute_log_split_ratio(m, m_new, k, k1,k2);
        }
        else {
                cout<<"merge"<<endl;
                /* Draw two distinct indices using modulo m->k addition        */
                stats.split_or_merge = "merge";
                k1 = (int)floor((double)(K-1) * kiss(g));
                if(k1==K-1) k1--;
                k2 = 1+k1;
                logl_new = prop_merge(Data, m, m_new, &k,k1, k2);
                add_logratio =-compute_log_split_ratio(m_new, m, k, k1, k2);
                // secondary_moment = calc_secondary_moment((Data[0]),m_new,m,k,k1,k2);
        }

        double x;
        x = kiss(g);
        ratio = exp((logl_new-(*logl))+add_logratio);
        ratio *= prop_ratio;
        if(ratio>1) ratio =1.0;
        if(ratio>0) {}
        else
                ratio = 0.0;
        // if (secondary_moment>0.01) ratio= -1.0;
        /* Accept/reject                            */
        if (x<ratio) {
                accept = 1;
                /* Modify the parameters and log likelihood                */
                m=m_new;
                *logl = logl_new;
                stats.acc_or_rej = "accept";
                // cout<<"secondary_moment "<<secondary_moment<<endl;
        }
        else{
                accept = 0;
                stats.acc_or_rej= "reject";
        }

        stats.sm_prob = ratio;
```

}

double Simulated_Annealing(double *logl, double *logl_old,pq_point & theta, pq_point & theta_old,int in,STATS & stats){
int K = theta.w.size();
int old_K = theta_old.w.size();

```
        double add_logratio = compute_log_prior_ratio(theta,theta_old);

        double criation = ( 4.0 *K)*log((double)(Curve_num*Nm));
        double criation_old = ( 4.0 *old_K)*log((double)(Curve_num*Nm));

        double Ta = 1.0, Tf = 1e-5;
        double Tb = (Ta - Tf)/(double)NIt;
        double x = kiss(g);
        double ratio = (((*logl)-(*logl_old))+add_logratio -criation/2.0+criation_old/2.0);

        // testfile<<in<<"th iter: Curve num="<<Curve_num<<" Nm="<<Nm<<" criation="<<criation
        // <<" criation_old="<<criation_old<<endl;
        // testfile<<"add_logratio="<<add_logratio<<" logl="<<*logl<<" logl_old="<<*logl_old<<endl;
        double temp = Ta-Tb*in;
        //tempureture from 1 to 1e-5
        ratio = ratio;
        ratio = exp(ratio);
        if(ratio>1.0) ratio =1.0;
        if(ratio>0.0) {}
        else
                ratio = 0.0;
        stats.simu_prob = ratio;
        if (x<ratio) {
                stats.simu_acc_or_simu_rej = "accept";
        }
        else{
                theta = theta_old;
                *logl = *logl_old;
                stats.simu_acc_or_simu_rej = "reject";
        }
        return ratio;
```

}