PLOS ONE

Supporting Information:
A computational modelling framework to quantify the effects of passaging cell lines

Wang Jin · Catherine J Penington · Scott W McCue · Matthew J Simpson

July 13, 2017

School of Mathematical Sciences, Queensland University of Technology (QUT), Brisbane, Australia
E-mail: matthew.simpson@qut.edu.au

C++ code for cell culture growth

```cpp
#include <cmath>
#include <iostream>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <ctime>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <random>

using namespace std;

void migration1 (const int agent_row, const int agent_col,
                 const int rowN, const int colN, int * migPosition) {
    migPosition[0] = agent_row;
    migPosition[1] = agent_col;
    double P = ((double) rand() / (RAND_MAX)); //choose migration
        directions
    if ((P < 1.0/6.0) && (agent_row > 0) && (agent_col > 0)) {//Position 1
        migPosition[0] = migPosition[0] - 1;
        migPosition[1] = migPosition[1] - 1;}
    else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
        agent_col < colN-1)) {//Position 2
        migPosition[0] = migPosition[0] - 1;}
    else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-1)) {//
        Position 3
        migPosition[1] = migPosition[1] + 1;}
    else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
        agent_col < colN-1)) {//Position 4
        migPosition[0] = migPosition[0] + 1;}
    else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1) && (
        agent_col > 0)) {//Position 5
        migPosition[0] = migPosition[0] + 1;
        migPosition[1] = migPosition[1] - 1;}
    else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
        migPosition[1] = migPosition[1] - 1;}
}
```

```
36
37 void migration2 (const int agent_row, const int agent_col,
38                  const int rowN, const int colN, int * migPosition) {
39     migPosition[0] = agent_row;
40     migPosition[1] = agent_col;
41     double P = ((double) rand() / (RAND_MAX)); //choose migration
           directions
42     if ((P < 1.0/6.0) && (agent_row > 0)) {//Position 1
43         migPosition[0] = migPosition[0] - 1;}
44     else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
           agent_col < colN-1)) {//Position 2
45         migPosition[0] = migPosition[0] - 1;
46         migPosition[1] = migPosition[1] + 1;}
47     else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-2)) {//
           Position 3
48         migPosition[1] = migPosition[1] + 1;}
49     else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
           agent_col < colN-1)) {//Position 4
50         migPosition[0] = migPosition[0] + 1;
51         migPosition[1] = migPosition[1] + 1;}
52     else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1)) {//
           Position 5
53         migPosition[0] = migPosition[0] + 1;}
54     else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
55         migPosition[1] = migPosition[1] - 1;}
56 }
57
58 void proliferation1 (const int agent_row, const int agent_col,
59                      const int rowN, const int colN, int * proPosition) {
60     proPosition[0] = agent_row;
61     proPosition[1] = agent_col;
62     double P = ((double) rand() / (RAND_MAX)); //choose proliferation
           directions
63     if ((P < 1.0/6.0) && (agent_row > 0) && (agent_col > 0)) {//Position 1
64         proPosition[0] = proPosition[0] - 1;
65         proPosition[1] = proPosition[1] - 1;}
66     else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
           agent_col < colN-1)) {//Position 2
67         proPosition[0] = proPosition[0] - 1;}
68     else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-1)) {//
           Position 3
69         proPosition[1] = proPosition[1] + 1;}
```

```
70      else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
            agent_col < colN-1)) {//Position 4
71          proPosition[0] = proPosition[0] + 1;}
72      else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1) && (
            agent_col > 0)) {//Position 5
73          proPosition[0] = proPosition[0] + 1;
74          proPosition[1] = proPosition[1] - 1;}
75      else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
76          proPosition[1] = proPosition[1] - 1;}
77  }
78
79  void proliferation2 (const int agent_row, const int agent_col,
80                       const int rowN, const int colN, int * proPosition) {
81      proPosition[0] = agent_row;
82      proPosition[1] = agent_col;
83      double P = ((double) rand() / (RAND_MAX)); //choose proliferation
            directions
84      if ((P < 1.0/6.0) && (agent_row > 0)) {//Position 1
85          proPosition[0] = proPosition[0] - 1;}
86      else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
            agent_col < colN-1)) {//Position 2
87          proPosition[0] = proPosition[0] - 1;
88          proPosition[1] = proPosition[1] + 1;}
89      else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-2)) {//
            Position 3
90          proPosition[1] = proPosition[1] + 1;}
91      else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
            agent_col < colN-1)) {//Position 4
92          proPosition[0] = proPosition[0] + 1;
93          proPosition[1] = proPosition[1] + 1;}
94      else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1)) {//
            Position 5
95          proPosition[0] = proPosition[0] + 1;}
96      else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
97          proPosition[1] = proPosition[1] - 1;}
98  }
99
100 int CalTotal (const int rowN, const int colN) {
101     int Ntotal;
102     if (rowN % 2 == 0) {
103         Ntotal = rowN * colN - rowN/2;
104     } else {
```

```
105         Ntotal = rowN * colN - (rowN+1)/2;
106     }
107     return Ntotal;
108 }
109
110 int main(int argc, char **argv) {
111     srand(time(NULL)); //random seeds
112     random_device rd;
113     mt19937 generator(rd());
114     //set clock
115     clock_t TIME;
116     TIME = clock();
117
118     //Initialisation
119     int passageI, passageN, t, tau;
120     double Pm, Pp;
121     passageI = 0; //passage index
122     passageN = 30; //max passage number
123     tau = 1; //time step: 1 ~ 1/12h
124     t = 0; //current time
125     const int d = 24; //spacing: 1 ~ 24mum
126     enum {rowN = 3610, colN = 4168}; //full scale
127     //enum {rowN = 362, colN = 418}; //scale ~ 1/10
128     const int Ntotal = CalTotal(rowN,colN);
129     const int Nstart = Ntotal * 0.15; //seeding condition
130     const int Nend = Ntotal * 0.85; //85% confluent
131     static vector<vector<int> > domain(rowN, std::vector<int>(colN)); //
            simulation domain
132     static vector<vector<double> > domain_Pp(rowN, std::vector<double>(
            colN)); //Pp
133     //static vector<vector<int> > domain_passage(rowN, std::vector<int>(
            colN)); //passage
134     //vector<vector<int> > domain_age(rowN, std::vector<int>(colN)); //age
135     //static vector<vector<int> > domain_generation(rowN, std::vector<int
            >(colN)); //generation
136     Pm = 0.35; //migration probability
137     //default_random_engine generator;
138     normal_distribution<double> distribution(4e-3,1e-3); //Pp distribution
139     normal_distribution<double> eps1(2e-5,2e-5); //eps1 distribution
140     normal_distribution<double> eps2(1e-5,1e-6); //eps2 distribution
141     int passageEff = 0;
142     int ageEff = 0;
```

```
143     double Pd = 1.0;
144     double Pd2 = 0.0;
145     double avePp;
146 
147     ofstream file1;
148     ofstream file2;
149     //ofstream file3;
150     //ofstream file4;
151     //ofstream file5;
152     ofstream file6;
153     ofstream file7;
154     //ofstream file8;
155     //ofstream file9;
156     //ofstream file10;
157     ofstream file11;
158     file1.open("domain.csv");
159     file2.open("domain_Pp.csv");
160     //file3.open("domain_passage.csv");
161     //file4.open("domain_age.csv");
162     //file5.open("domain_generation.csv");
163     file6.open("domain2.csv");
164     file7.open("domain2_Pp.csv");
165     //file8.open("domain2_passage.csv");
166     //file9.open("domain2_age.csv");
167     //file10.open("domain2_generation.csv");
168     file11.open("td.csv");
169 
170     //Seeding
171     int agent_row, agent_col;
172     for (int i=Nstart; i>0; --i) {
173         agent_row = rand() % rowN;
174         agent_col = rand() % colN;
175         if (((rowN-agent_row) % 2 != 0) && (agent_col == colN-1)) {
176             i = i + 1;
177         } else if (domain[agent_row][agent_col] == 0) {
178             domain[agent_row][agent_col] = 1;
179             domain_Pp[agent_row][agent_col] = distribution(generator);
180             if (domain_Pp[agent_row][agent_col] < 0) {
181                 domain_Pp[agent_row][agent_col] = 0.0;
182             }
183 
184         } else {
```

```
            i = i + 1;
        }
    }

    //Count initial cells
    int agent_N = 0;
    for (int i=0; i<rowN; ++i) {
        for (int j=0; j<colN; ++j) {
            if (domain[i][j] == 1) {
                agent_N = agent_N + 1;
            }
        }
    }
    //cout << agent_N;

    //Write to files
    for (int i=0; i<rowN; ++i) {
        for (int j=0; j<colN; ++j) {
            file1 << domain[i][j];
            file2 << domain_Pp[i][j];
            //file3 << domain_passage[i][j];
            //file4 << domain_age[i][j];
            //file5 << domain_generation[i][j];
            if (j < colN-1) {
                file1 << ", ";
                file2 << ", ";
                //file3 << ", ";
                //file4 << ", ";
                //file5 << ", ";
            } else {
                file1 << "\n";
                file2 << "\n";
                //file3 << "\n";
                //file4 << "\n";
                //file5 << "\n";
            }
        }
    }

    t = t + tau;
    int migPosition[2], proPosition[2];
    double P;
```

```
227
228     while (passageI <= passageN) {//check passage number
229         cout << "passage number #" << passageI << ", ";
230         int N = agent_N;
231         cout << "N(0) = " << N << endl;
232
233         //check average Pp
234         avePp = 0;
235         for (int i=0; i<rowN; ++i) {
236             for (int j=0; j<colN; ++j) {
237                 if (domain[i][j] == 1) {
238                     avePp = avePp + domain_Pp[i][j];
239                 }
240             }
241         }
242         avePp = avePp/N;
243         cout << avePp << endl;
244         while (N < Nend) { //check cell population
245             //Migration
246             for (int i=0; i<N; ++i) {
247                 P = ((double) rand() / (RAND_MAX)); //generate a random
                        number
248                 if (P <= Pm) { //perform migration
249                     //select a random site
250                     int ifGhost = 0;
251                     agent_row = rand() % rowN;
252                     agent_col = rand() % colN;
253                     while ((domain[agent_row][agent_col] != 1) || (ifGhost
                            == 1)) { //select a random site, stop until
                            finding an agent
254                         agent_row = rand() % rowN;
255                         agent_col = rand() % colN;
256                     }
257                     //test ghost node
258                     if (((rowN-agent_row) % 2 != 0) && (agent_col == colN
                            -1)) {
259                         ifGhost = 1;
260                     }
261                     if (ifGhost == 0) {
262                         if ((rowN-agent_row) % 2 == 0) {
263                             migration1(agent_row, agent_col, rowN, colN,
                                    migPosition);
```

```
264                         if (domain[migPosition[0]][migPosition[1]] ==
                                0) {//if the new site is not taken
265                             domain[agent_row][agent_col] = 0; //delete
                                    the previous site
266                             domain[migPosition[0]][migPosition[1]] =
                                    1; //move to the new site
267                             /*if (domain_Pp[migPosition[0]][
                                    migPosition[1]]!= 0 && domain_passage[
                                    migPosition[0]][migPosition[1]]!= 0 &&
                                     domain_age[migPosition[0]][
                                    migPosition[1]]!= 0 &&
                                    domain_generation[migPosition[0]][
                                    migPosition[1]]!= 0) {
268                                 cout << "error mig 1" << endl;
269                             }*/
270                             domain_Pp[migPosition[0]][migPosition[1]]
                                    = domain_Pp[agent_row][agent_col]; //
                                    move the corresponding Pp
271                             //domain_passage[migPosition[0]][
                                    migPosition[1]] = domain_passage[
                                    agent_row][agent_col]; //move the
                                    corresponding passage index
272                             //domain_age[migPosition[0]][migPosition
                                    [1]] = domain_age[agent_row][agent_col
                                    ]; //move the corresponding age
273                             //domain_generation[migPosition[0]][
                                    migPosition[1]] = domain_generation[
                                    agent_row][agent_col]; //move the
                                    corresponding generation
274                             domain_Pp[agent_row][agent_col] = 0.0; //
                                    delete the previous Pp
275                             //domain_passage[agent_row][agent_col] =
                                    0; //delete the previous passage index
276                             //domain_age[agent_row][agent_col] = 0; //
                                    delete the previous age
277                             //domain_generation[agent_row][agent_col]
                                    = 0; //delete the previous generation
278                             /*if (domain_Pp[agent_row][agent_col]!=0
                                    || domain_passage[agent_row][agent_col
                                    ]!=0 || domain_age[agent_row][
                                    agent_col]!=0 || domain_generation[
                                    agent_row][agent_col]!=0) {
```

```
279                                 cout << "error mig 1" << endl;
280                             }*/
281                         }
282                     } else {
283                         migration2(agent_row, agent_col, rowN, colN,
                                migPosition);
284                         if (domain[migPosition[0]][migPosition[1]] ==
                                0) {//if the new site is not taken
285                             domain[agent_row][agent_col] = 0; //delete
                                    the previous site
286                             domain[migPosition[0]][migPosition[1]] =
                                    1; //move to the new site
287                             /*if (domain_Pp[migPosition[0]][
                                    migPosition[1]]!= 0 && domain_passage[
                                    migPosition[0]][migPosition[1]]!= 0 &&
                                     domain_age[migPosition[0]][
                                    migPosition[1]]!= 0 &&
                                    domain_generation[migPosition[0]][
                                    migPosition[1]]!= 0) {
288                                 cout << "error mig 2" << endl;
289                             }*/
290                             domain_Pp[migPosition[0]][migPosition[1]]
                                    = domain_Pp[agent_row][agent_col]; //
                                    move the corresponding Pp
291                             //domain_passage[migPosition[0]][
                                    migPosition[1]] = domain_passage[
                                    agent_row][agent_col]; ////move the
                                    corresponding passage index
292                             //domain_age[migPosition[0]][migPosition
                                    [1]] = domain_age[agent_row][agent_col
                                    ]; //move the corresponding age
293                             //domain_generation[migPosition[0]][
                                    migPosition[1]] = domain_generation[
                                    agent_row][agent_col]; //move the
                                    corresponding generation
294                             domain_Pp[agent_row][agent_col] = 0.0; //
                                    delete the previous Pp
295                             //domain_passage[agent_row][agent_col] =
                                    0; //delete the previous passage index
296                             //domain_age[agent_row][agent_col] = 0; //
                                    delete the previous age
```

```
297                                 //domain_generation[agent_row][agent_col]
                                        = 0; //delete the previous age
298                                 /*if (domain_Pp[agent_row][agent_col]!=0
                                        || domain_passage[agent_row][agent_col
                                        ]!=0 || domain_age[agent_row][
                                        agent_col]!=0 || domain_generation[
                                        agent_row][agent_col]!=0) {
299                                     cout << "error mig 2" << endl;
300                                 }*/
301                             }
302                         }
303                     } else {
304                         i = i - 1;
305                     }
306
307                 }
308             }
309             //Proliferation
310             for (int i=0; i<N; ++i) {
311                 //select a random site
312                 int ifGhost = 0;
313                 agent_row = rand() % rowN;
314                 agent_col = rand() % colN;
315                 while ((domain[agent_row][agent_col] != 1) || (ifGhost ==
                        1)) { //select a random site, stop until finding an
                        agent
316                     agent_row = rand() % rowN;
317                     agent_col = rand() % colN;
318                 }
319                 //test ghost node
320                 if (((rowN-agent_row) % 2 != 0) && (agent_col == colN-1))
                        {
321                     ifGhost = 1;
322                 }
323                 if (ifGhost == 0)
324                 {
325                     Pp = domain_Pp[agent_row][agent_col];
326                     P = ((double) rand() / (RAND_MAX)); //generate random
                            number
327                     if (P <= Pp) {
328                         if ((rowN-agent_row) % 2 == 0) {
```

```
329                         proliferation1(agent_row, agent_col, rowN,
                                colN, proPosition);
330                         if (domain[proPosition[0]][proPosition[1]] ==
                                0) {//if the new site is not taken
331                             domain[proPosition[0]][proPosition[1]] =
                                    1; //proliferate at the new site
332                             P = ((double) rand() / (RAND_MAX)); //
                                    generate a random number
333                             if (ageEff && P<=Pd2) {//damage
334                                 domain_Pp[proPosition[0]][proPosition
                                        [1]] = Pp - max(eps2(generator),
                                        0.0);
335                                 domain_Pp[proPosition[0]][proPosition
                                        [1]] = max(domain_Pp[proPosition
                                        [0]][proPosition[1]], 0.0);
336                                 domain_Pp[agent_row][agent_col] =
                                        domain_Pp[proPosition[0]][
                                        proPosition[1]];
337                                 if (domain_Pp[agent_row][agent_col]<0
                                        || domain_Pp[agent_row][agent_col
                                        ]!=domain_Pp[proPosition[0]][
                                        proPosition[1]]) {
338                                     cout << "error" << endl;
339                                 }
340                             } else {//no damage
341                                 domain_Pp[proPosition[0]][proPosition
                                        [1]] = Pp; //adopt Pp
342                             }
343                             //domain_passage[proPosition[0]][
                                    proPosition[1]] = passageI; //passage
                                    number when it was born
344                             //domain_age[proPosition[0]][proPosition
                                    [1]] = -1; //initiate age = -1 so that
                                     when updating cell age it equals 0
345                             //domain_generation[proPosition[0]][
                                    proPosition[1]] = domain_generation[
                                    agent_row][agent_col] + 1; //add
                                    genertion
346                             //domain_generation[agent_row][agent_col]
                                    = domain_generation[agent_row][
                                    agent_col] + 1; //add generation
```

```
347                             agent_N = agent_N + 1; //update agent
                                    population
348                         }
349                     } else {
350                         proliferation2(agent_row, agent_col, rowN,
                                colN, proPosition);
351                         if (domain[proPosition[0]][proPosition[1]] ==
                                0) {//if the new site is not taken
352                             domain[proPosition[0]][proPosition[1]] =
                                    1; //proliferate at the new site
353                             P = ((double) rand() / (RAND_MAX)); //
                                    generate a random number
354                             if (ageEff && P<=Pd2) {//damage
355                                 domain_Pp[proPosition[0]][proPosition
                                        [1]] = Pp - max(eps2(generator),
                                        0.0);
356                                 domain_Pp[proPosition[0]][proPosition
                                        [1]] = max(domain_Pp[proPosition
                                        [0]][proPosition[1]], 0.0);
357                                 domain_Pp[agent_row][agent_col] =
                                        domain_Pp[proPosition[0]][
                                        proPosition[1]];
358                                 if (domain_Pp[agent_row][agent_col]<0
                                        || domain_Pp[agent_row][agent_col
                                        ]!=domain_Pp[proPosition[0]][
                                        proPosition[1]]) {
359                                     cout << "error" << endl;
360                                 }
361                             } else {//no damage
362                                 domain_Pp[proPosition[0]][proPosition
                                        [1]] = Pp; //adopt Pp
363                             }
364                             //domain_passage[proPosition[0]][
                                    proPosition[1]] = passageI; //passage
                                    number when it was born
365                             //domain_age[proPosition[0]][proPosition
                                    [1]] = -1; //initiate age = -1 so that
                                     when updating cell age it equals 0
366                             //domain_generation[proPosition[0]][
                                    proPosition[1]] = domain_generation[
                                    agent_row][agent_col] + 1; //add
                                    generation
```

```
367                             //domain_generation[agent_row][agent_col]
                                    = domain_generation[agent_row][
                                    agent_col] + 1; //add generation
368                             /*if (passageI!=0)
369                             {
370                                 cout << "Pp = " << Pp << endl;
371                             }*/
372                             agent_N = agent_N + 1; //update agent
                                    population
373                         }
374                     }
375                 }
376             } else {
377                 i = i - 1;
378             }
379         }
380
381         //update cell age
382         /*for (int i=0; i<rowN; ++i) {
383             for (int j=0; j<colN; ++j) {
384                 if (domain[i][j]==1) { //check if there is a cell
385                     domain_age[i][j] = domain_age[i][j] + tau;
386                 }
387             }
388         }*/
389
390         N = agent_N; //update cell number
391         //cout << "N(" << t << ") = " << N << endl;
392
393         //Write to files
394         if (N >= Nend) {
395             for (int i=0; i<rowN; ++i) {
396                 for (int j=0; j<colN; ++j) {
397                     file6 << domain[i][j];
398                     file7 << domain_Pp[i][j];
399                     //file8 << domain_passage[i][j];
400                     //file9 << domain_age[i][j];
401                     //file10 << domain_generation[i][j];
402                     if (j < colN-1) {
403                         file6 << ",␣";
404                         file7 << ",␣";
405                         //file8 << ", ";
```

```
406                             //file9 << ", ";
407                             //file10 << ", ";
408                         } else {
409                             file6 << "\n";
410                             file7 << "\n";
411                             //file8 << "\n";
412                             //file9 << "\n";
413                             //file10 << "\n";
414                         }
415                     }
416                 }
417             }
418 
419             t = t + tau;
420 
421         }
422         //cout << " DONE N = " << agent_N << endl;
423         cout << "DONE N(" << (t-tau) << ") = " << N << endl;
424         file11 << (t-tau);
425         if (passageI != passageN + 1)
426         {
427             file11 << ", ";
428         }
429 
430         //check average Pp
431         avePp = 0;
432         for (int i=0; i<rowN; ++i) {
433                 for (int j=0; j<colN; ++j) {
434                     if (domain[i][j] == 1) {
435                         avePp = avePp + domain_Pp[i][j];
436                     }
437                 }
438         }
439         avePp = avePp/N;
440         cout << avePp << endl;
441 
442         //Subculturing
443         passageI = passageI + 1; //update passage number
444         t = 0; //reset time
445         if (passageI != passageN + 1) {//check if subculture
446             agent_N = 0; //delete current cell number
447             double * Pp_temp;
```

```
448             //int * passage_temp;
449             //int * generation_temp;
450             Pp_temp = new double [Nstart];
451             //passage_temp = new int [Nstart];
452             //generation_temp = new int[Nstart];
453             //Select cells
454             for (int i=Nstart; i>0; --i) {
455                 agent_row = rand() % rowN;
456                 agent_col = rand() % colN;
457                 if (((rowN-agent_row) % 2 != 0) && (agent_col == colN-1))
                        {
458                     i = i + 1;
459                 } else if (domain[agent_row][agent_col] == 1) {
460                                         Pp_temp[Nstart-i] = domain_Pp[
                                            agent_row][agent_col];
461                                         //passage_temp[Nstart-i] =
                                            domain_passage[agent_row][
                                            agent_col];
462                                         //age_temp[Nstart-i] = domain_age[
                                            agent_row][agent_col];
463                                         //generation_temp[Nstart-i] =
                                            domain_generation[agent_row][
                                            agent_col];
464                 } else {
465                     i = i + 1;
466                 }
467             }
468
469             //Delete cells
470             for (int i=0; i<rowN; ++i) {
471                 for (int j=0; j<colN; ++j) {
472                     if (domain[i][j] == 1) {
473                         domain[i][j] = 0;
474                         domain_Pp[i][j] = 0.0;
475                         //domain_passage[i][j] = 0;
476                         //domain_age[i][j] = 0;
477                         //domain_generation[i][j] = 0;
478                     }
479                 }
480             }
481
482                         //Shuffle & delete temp values
```

```
483                         for (int i=Nstart; i>0; --i) {
484                                 agent_row = rand() % rowN;
485                                 agent_col = rand() % colN;
486                                 if (((rowN-agent_row) % 2 != 0) && (
                                        agent_col == colN-1)) {
487                                     i = i + 1;
488                                 } else if (domain[agent_row][agent_col] ==
                                        0) {
489                                     domain[agent_row][agent_col] = 1;
490                     P = ((double) rand() / (RAND_MAX)); //generate a
                            random number
491                     if (passageEff && P<=Pd) {//damage
492                         domain_Pp[agent_row][agent_col] = Pp_temp[Nstart-i
                                ] - max(eps1(generator), 0.0);
493                         //cout << domain_Pp[agent_row][agent_col] << endl;
494                         domain_Pp[agent_row][agent_col] = max(domain_Pp[
                                agent_row][agent_col], 0.0);
495                         if (domain_Pp[agent_row][agent_col] < 0) {
496                             cout << "error" << endl;
497                         }
498                     } else {//no damage
499                         domain_Pp[agent_row][agent_col] = Pp_temp[Nstart-i
                                ];
500                     }
501                     //domain_passage[agent_row][agent_col] = passage_temp[
                            Nstart-i];
502                     //domain_age[agent_row][agent_col] = age_temp[Nstart-i
                            ];
503                     //domain_generation[agent_row][agent_col] =
                            generation_temp[Nstart-i];
504
505                                 agent_N = agent_N + 1;
506                                 } else {
507                                     i = i + 1;
508                                 }
509                         }
510
511             //Delete arrays
512             delete[] Pp_temp;
513             //delete[] passage_temp;
514             //delete[] generation_temp;
515
```

```
516             //Write to files
517             for (int i=0; i<rowN; ++i) {
518                 for (int j=0; j<colN; ++j) {
519                     file1 << domain[i][j];
520                     file2 << domain_Pp[i][j];
521                     //file3 << domain_passage[i][j];
522                     //file4 << domain_age[i][j];
523                     //file5 << domain_generation[i][j];
524                     if (j < colN-1) {
525                         file1 << ", ";
526                         file2 << ", ";
527                         //file3 << ", ";
528                         //file4 << ", ";
529                         //file5 << ", ";
530                     } else {
531                         file1 << "\n";
532                         file2 << "\n";
533                         //file3 << "\n";
534                         //file4 << "\n";
535                         //file5 << "\n";
536                     }
537                 }
538             }
539         }
540     }
541
542
543     file1.close();
544     file2.close();
545     //file3.close();
546     //file4.close();
547     //file5.close();
548     file6.close();
549     file7.close();
550     //file8.close();
551     //file9.close();
552     //file10.close();
553     file11.close();
554
555     TIME = clock() - TIME;
556     cout << "It takes "<< (float)TIME/CLOCKS_PER_SEC <<" second(s)."<<
        endl;
```

```
557     cin.get();
558     return 0;
559 }
```

C++ code for scratch assays

```
#include <cmath>
#include <iostream>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <ctime>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <random>

using namespace std;

void migration1 (const int agent_row, const int agent_col,
                 const int rowN, const int colN, int * migPosition) {
    migPosition[0] = agent_row;
    migPosition[1] = agent_col;
    double P = ((double) rand() / (RAND_MAX)); //choose migration
        directions
    if ((P < 1.0/6.0) && (agent_row > 0) && (agent_col > 0)) {//Position 1
        migPosition[0] = migPosition[0] - 1;
        migPosition[1] = migPosition[1] - 1;}
    else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
        agent_col < colN-1)) {//Position 2
        migPosition[0] = migPosition[0] - 1;}
    else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-1)) {//
        Position 3
        migPosition[1] = migPosition[1] + 1;}
    else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
        agent_col < colN-1)) {//Position 4
        migPosition[0] = migPosition[0] + 1;}
    else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1) && (
        agent_col > 0)) {//Position 5
        migPosition[0] = migPosition[0] + 1;
        migPosition[1] = migPosition[1] - 1;}
    else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
        migPosition[1] = migPosition[1] - 1;}
}
```

```
36
37 void migration2 (const int agent_row, const int agent_col,
38                  const int rowN, const int colN, int * migPosition) {
39     migPosition[0] = agent_row;
40     migPosition[1] = agent_col;
41     double P = ((double) rand() / (RAND_MAX)); //choose migration
           directions
42     if ((P < 1.0/6.0) && (agent_row > 0)) {//Position 1
43         migPosition[0] = migPosition[0] - 1;}
44     else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
           agent_col < colN-1)) {//Position 2
45         migPosition[0] = migPosition[0] - 1;
46         migPosition[1] = migPosition[1] + 1;}
47     else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-2)) {//
           Position 3
48         migPosition[1] = migPosition[1] + 1;}
49     else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
           agent_col < colN-1)) {//Position 4
50         migPosition[0] = migPosition[0] + 1;
51         migPosition[1] = migPosition[1] + 1;}
52     else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1)) {//
           Position 5
53         migPosition[0] = migPosition[0] + 1;}
54     else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
55         migPosition[1] = migPosition[1] - 1;}
56 }
57
58 void proliferation1 (const int agent_row, const int agent_col,
59                      const int rowN, const int colN, int * proPosition) {
60     proPosition[0] = agent_row;
61     proPosition[1] = agent_col;
62     double P = ((double) rand() / (RAND_MAX)); //choose migration
           directions
63     if ((P < 1.0/6.0) && (agent_row > 0) && (agent_col > 0)) {//Position 1
64         proPosition[0] = proPosition[0] - 1;
65         proPosition[1] = proPosition[1] - 1;}
66     else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
           agent_col < colN-1)) {//Position 2
67         proPosition[0] = proPosition[0] - 1;}
68     else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-1)) {//
           Position 3
69         proPosition[1] = proPosition[1] + 1;}
```

```
70      else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
            agent_col < colN-1)) {//Position 4
71          proPosition[0] = proPosition[0] + 1;}
72      else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1) && (
            agent_col > 0)) {//Position 5
73          proPosition[0] = proPosition[0] + 1;
74          proPosition[1] = proPosition[1] - 1;}
75      else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
76          proPosition[1] = proPosition[1] - 1;}
77  }
78
79  void proliferation2 (const int agent_row, const int agent_col,
80                       const int rowN, const int colN, int * proPosition) {
81      proPosition[0] = agent_row;
82      proPosition[1] = agent_col;
83      double P = ((double) rand() / (RAND_MAX)); //choose migration
            directions
84      if ((P < 1.0/6.0) && (agent_row > 0)) {//Position 1
85          proPosition[0] = proPosition[0] - 1;}
86      else if ((P >= 1.0/6.0) && (P <1.0/3.0) && (agent_row > 0) && (
            agent_col < colN-1)) {//Position 2
87          proPosition[0] = proPosition[0] - 1;
88          proPosition[1] = proPosition[1] + 1;}
89      else if ((P >= 1.0/3.0) && (P < 1.0/2.0) && (agent_col < colN-2)) {//
            Position 3
90          proPosition[1] = proPosition[1] + 1;}
91      else if ((P >= 1.0/2.0) && (P < 2.0/3.0) && (agent_row < rowN-1) && (
            agent_col < colN-1)) {//Position 4
92          proPosition[0] = proPosition[0] + 1;
93          proPosition[1] = proPosition[1] + 1;}
94      else if ((P >= 2.0/3.0) && (P < 5.0/6.0) && (agent_row < rowN-1)) {//
            Position 5
95          proPosition[0] = proPosition[0] + 1;}
96      else if ((P >= 5.0/6.0) && (agent_col > 0)) {//Position 6
97          proPosition[1] = proPosition[1] - 1;}
98  }
99
100 int CalTotal (const int rowN, const int colN) {
101     int Ntotal;
102     if (rowN % 2 == 0) {
103         Ntotal = rowN * colN - rowN/2;
104     } else {
```

```
105         Ntotal = rowN * colN - (rowN+1)/2;
106     }
107     return Ntotal;
108 }
109
110 void importdata (const int N, double * dataPp) {
111         ifstream ifs ("Pp.csv");
112         char dummy;
113         for (int i = 0; i < N; ++i){
114                 ifs >> dataPp[i];
115                 if (i < (N - 1))
116             ifs >> dummy;
117         }
118 }
119
120 int main(int argc, char **argv) {
121     srand(time(NULL)); //random seeds
122     random_device rd;
123     mt19937 generator(rd());
124     //set clock
125     clock_t TIME;
126     TIME = clock();
127
128         //Initialisation
129     int simuNum, sampleNum, t, T, tau, agent_row, agent_col, migPosition
        [2], proPosition[2];
130     double Pm, Pp, P, avePp;
131         simuNum = 1; //simulation index
132     sampleNum = 2; //total simulations
133     tau = 1; //time step: 1 ~ 1/12h
134     t = 0; //current time
135     T = 864; //end time
136     Pm = 0.35;
137     const int d = 24; //spacing: 1 ~ 24mum
138     enum {rowN = 68, colN = 80}; //1400mum by 1900 mum
139     enum {sCol1 = 29, sCol2 = 51}; //scratch width
140     const int Ntotal = CalTotal(rowN,colN);
141     const int Nstart = Ntotal * 0.3; //seeding condition
142     cout << "Ntotal = " << Ntotal << endl;
143     cout << "Nseed = " << Nstart << endl;
144     static vector<vector<int> > domain(rowN, std::vector<int>(colN)); //
        simulation domain
```

```
145     static vector<vector<double> > domain_Pp(rowN, std::vector<double>(
            colN)); //Pp
146 
147     ofstream file1;
148     ofstream file2;
149     ofstream file3;
150     ofstream file4;
151     ofstream file5;
152     file1.open("domain0.csv");
153     file2.open("domain24.csv");
154     file3.open("domain48.csv");
155         file4.open("domain72.csv");
156         file5.open("domain96.csv");
157     //Import data
158     int size_data = 12790529; //size of import data
159     double * dataPp;
160     dataPp = new double [size_data];
161     importdata(size_data, dataPp);
162 
163     //Seeding
164     for (int i = Nstart; i > 0; --i) {
165         agent_row = rand() % rowN;
166         agent_col = rand() % colN;
167         if (((rowN-agent_row) % 2 != 0) && (agent_col == colN-1)) {
168             i = i + 1;
169         } else if (domain[agent_row][agent_col] == 0) {
170                 int randPp = rand() % size_data;
171             domain[agent_row][agent_col] = 1;
172             domain_Pp[agent_row][agent_col] = dataPp[randPp];
173         } else {
174             i = i + 1;
175         }
176     }
177     delete [] dataPp;
178 
179     //Scratch
180     for (int i=0; i<rowN; ++i) {
181         if ((rowN-agent_row) % 2 == 0) {
182             for (int j=sCol1; j<=sCol2; ++j) {
183                 domain[i][j] = 0;
184                 domain_Pp[i][j] = 0.0;
185             }
```

```
186         } else {
187             for (int j=sCol1; j<=sCol2-1; ++j) {
188                 domain[i][j] = 0;
189                 domain_Pp[i][j] = 0.0;
190             }
191         }
192     }
193 
194     //Count initial cells
195     int agent_N = 0;
196     avePp = 0.0;
197     for (int i=0; i<rowN; ++i) {
198         for (int j=0; j<colN; ++j) {
199             if (domain[i][j] == 1) {
200                 agent_N = agent_N + 1;
201                 avePp = avePp + domain_Pp[i][j];
202             }
203         }
204     }
205     avePp = avePp / (double)agent_N;
206 
207     //Write to files
208     for (int i=0; i<rowN; ++i) {
209         for (int j=0; j<colN; ++j) {
210             file1 << domain[i][j];
211             //file2 << domain_Pp[i][j];
212             if (j < colN-1) {
213                 file1 << ", ";
214                 //file2 << ", ";
215             } else {
216                 file1 << "\n";
217                 //file2 << "\n";
218             }
219         }
220     }
221 
222     t = t + tau;
223     int timestop = T / t;
224 
225     while (simuNum <= sampleNum) {
226         cout << "SimuNum " << simuNum << ", ";
```

```
227         cout << "N(0) = " << agent_N << ", Average Pp = " << avePp << endl
                ;
228         for (int tindex=0; tindex<timestop; ++tindex) { //check time
229             int N = agent_N;
230             //Migration
231             for (int i=0; i<N; ++i) {
232             P = ((double) rand() / (RAND_MAX)); //generate a random
                    number
233             if (P <= Pm) { //perform migration
234                 //select a random site
235                 int ifGhost = 0;
236                 agent_row = rand() % rowN;
237                 agent_col = rand() % colN;
238                 while ((domain[agent_row][agent_col] != 1) || (ifGhost
                        == 1)) { //select a random site, stop until
                        finding an agent
239                     agent_row = rand() % rowN;
240                     agent_col = rand() % colN;
241                 }
242                 //test ghost node
243                 if (((rowN-agent_row) % 2 != 0) && (agent_col == colN
                        -1)) {
244                     ifGhost = 1;
245                 }
246                 if (ifGhost == 0) {
247                     if ((rowN-agent_row) % 2 == 0) {
248                         migration1(agent_row, agent_col, rowN, colN,
                                migPosition);
249                         if (domain[migPosition[0]][migPosition[1]] ==
                                0) {//if the new site is not taken
250                             domain[agent_row][agent_col] = 0; //delete
                                    the previous site
251                             domain[migPosition[0]][migPosition[1]] =
                                    1; //move to the new site
252                             domain_Pp[migPosition[0]][migPosition[1]]
                                    = domain_Pp[agent_row][agent_col]; //
                                    move the corresponding Pp
253                             domain_Pp[agent_row][agent_col] = 0.0; //
                                    delete the previous Pp
254                         }
255                     } else {
```

```
256                             migration2(agent_row, agent_col, rowN, colN,
                                    migPosition);
257                             if (domain[migPosition[0]][migPosition[1]] ==
                                    0) {//if the new site is not taken
258                                 domain[agent_row][agent_col] = 0; //delete
                                        the previous site
259                                 domain[migPosition[0]][migPosition[1]] =
                                        1; //move to the new site
260                                 domain_Pp[migPosition[0]][migPosition[1]]
                                        = domain_Pp[agent_row][agent_col]; //
                                        move the corresponding Pp
261                                 domain_Pp[agent_row][agent_col] = 0.0; //
                                        delete the previous Pp
262                             }
263                         }
264                     } else {
265                         i = i - 1;
266                     }
267                 }
268             }
269                 //Proliferation
270                 for (int i=0; i<N; ++i) {
271                 //select a random site
272                 int ifGhost = 0;
273                 agent_row = rand() % rowN;
274                 agent_col = rand() % colN;
275                 while ((domain[agent_row][agent_col] != 1) || (ifGhost ==
                        1)) { //select a random site, stop until finding an
                        agent
276                     agent_row = rand() % rowN;
277                     agent_col = rand() % colN;
278                 }
279                 //test ghost node
280                 if (((rowN-agent_row) % 2 != 0) && (agent_col == colN-1))
                        {
281                     ifGhost = 1;
282                 }
283                 if (ifGhost == 0) {
284                     Pp = domain_Pp[agent_row][agent_col];
285                     P = ((double) rand() / (RAND_MAX)); //generate a
                            random number
286                     if (P <= Pp) {
```

```
287                         if ((rowN-agent_row) % 2 == 0) {
288                             proliferation1(agent_row, agent_col, rowN,
                                    colN, proPosition);
289                             if (domain[proPosition[0]][proPosition[1]] ==
                                    0) {//if the new site is not taken
290                                 domain[proPosition[0]][proPosition[1]] =
                                        1; //proliferate at the new site
291                                 domain_Pp[proPosition[0]][proPosition[1]]
                                        = Pp; //adopt Pp
292                                 agent_N = agent_N + 1; //update agent
                                        population
293                             }
294                         } else {
295                             proliferation2(agent_row, agent_col, rowN,
                                    colN, proPosition);
296                             if (domain[proPosition[0]][proPosition[1]] ==
                                    0) {//if the new site is not taken
297                                 domain[proPosition[0]][proPosition[1]] =
                                        1; //proliferate at the new site
298                                 domain_Pp[proPosition[0]][proPosition[1]]
                                        = Pp; //adopt Pp
299                                 agent_N = agent_N + 1; //update agent
                                        population
300                             }
301                         }
302                     }
303                 } else {
304                     i = i - 1;
305                 }
306             }
307                 //Write to files
308             if (t == 288) { //24h
309                 for (int i=0; i<rowN; ++i) {
310                     for (int j=0; j<colN; ++j) {
311                         file2 << domain[i][j];
312                         if (j < colN-1) {
313                             file2 << ", ";
314                         } else {
315                             file2 << "\n";
316                         }
317                     }
318                 }
```

```
319             }
320             if (t == 576) { //48h
321                 for (int i=0; i<rowN; ++i) {
322                     for (int j=0; j<colN; ++j) {
323                         file3 << domain[i][j];
324                         if (j < colN-1) {
325                             file3 << ", ";
326                         } else {
327                             file3 << "\n";
328                         }
329                     }
330                 }
331             }
332             if (t == 864) { //72h
333                 for (int i=0; i<rowN; ++i) {
334                     for (int j=0; j<colN; ++j) {
335                         file4 << domain[i][j];
336                         if (j < colN-1) {
337                             file4 << ", ";
338                         } else {
339                             file4 << "\n";
340                         }
341                     }
342                 }
343             }
344             /*if (t == 1152) { //96h
345                 for (int i=0; i<rowN; ++i) {
346                     for (int j=0; j<colN; ++j) {
347                         file5 << domain[i][j];
348                         if (j < colN-1) {
349                             file5 << ", ";
350                         } else {
351                             file5 << "\n";
352                         }
353                     }
354                 }
355             }*/
356                 t = t + tau; //forward time
357         }
358 
359         cout << "DONE N = " << agent_N << endl;
360         simuNum = simuNum + 1; //update the simulation index
```

```
361         t = 0; //reset time
362
363         if (simuNum != sampleNum+1) {
364                 for (int i=0; i<rowN; ++i) {
365                 for (int j=0; j<colN; ++j) {
366                     domain[i][j] = 0;
367                     domain_Pp[i][j] = 0.0;
368                 }
369             }
370             //Import data
371             double * dataPp;
372                 dataPp = new double [size_data];
373                 importdata(size_data, dataPp);
374             //Re-seed cells
375                 for (int i = Nstart; i > 0; --i) {
376                         agent_row = rand() % rowN;
377                         agent_col = rand() % colN;
378                         if (((rowN-agent_row) % 2 != 0) && (agent_col ==
                                colN-1)) {
379                         i = i + 1;
380                         } else if (domain[agent_row][agent_col] == 0) {
381                                 int randPp = rand() % size_data;
382                         domain[agent_row][agent_col] = 1;
383                         domain_Pp[agent_row][agent_col] = dataPp[randPp];
384                         } else {
385                         i = i + 1;
386                         }
387                 }
388                 delete [] dataPp;
389             //Scratch
390             for (int i=0; i<rowN; ++i) {
391                         if ((rowN-agent_row) % 2 == 0) {
392                         for (int j=sCol1; j<=sCol2; ++j) {
393                                 domain[i][j] = 0;
394                                 domain_Pp[i][j] = 0.0;
395                         }
396                         } else {
397                         for (int j=sCol1; j<=sCol2-1; ++j) {
398                                 domain[i][j] = 0;
399                                 domain_Pp[i][j] = 0.0;
400                         }
401                         }
```

```
402                 }
403             //Count cells
404             agent_N = 0;
405             avePp = 0.0;
406                 for (int i=0; i<rowN; ++i) {
407                         for (int j=0; j<colN; ++j) {
408                         if (domain[i][j] == 1) {
409                                 agent_N = agent_N + 1;
410                         avePp = avePp + domain_Pp[i][j];
411                         }
412                         }
413                 }
414             avePp = avePp / (double)agent_N;
415             //Write to files
416                 for (int i=0; i<rowN; ++i) {
417                         for (int j=0; j<colN; ++j) {
418                         file1 << domain[i][j];
419                         if (j < colN-1) {
420                                 file1 << ", ";
421                         } else {
422                                 file1 << "\n";
423                         }
424                         }
425                 }
426             t = t + tau;
427         }
428     }
429
430     file1.close();
431     file2.close();
432     file3.close();
433     file4.close();
434     file5.close();
435     TIME = clock() - TIME;
436     cout << "It takes "<< (float)TIME/CLOCKS_PER_SEC <<" second(s)."<<
        endl;
437     cin.get();
438     return 0;
439 }
```