Introduction to Computational Physics
Exercise 2

Philipp Gloor

October 21, 2010

Abstract

Something doesnt work quite right with my programm and I'm not quite sure where it comes from.
Two problems: I dont get the distribution I should although when I run the algorithm on a lattice and check the lattice output it seems that my code numbers the clusters correctly - I've also double checked the mass addings and I couldnt see anything wrong (maybe I got blind to the problem meanwhile). Secondly I sometimes get an buffer overrun error in the array clustersize - that is when M[j] (which also is the index of the clusterSize array) exceeds its boundary - which, in my opinion, shouldnt happen.
I'm aware that my graph is faulty but I'll add it nonetheless for completeness :)

1 Code

1.1 Clustersize distribution

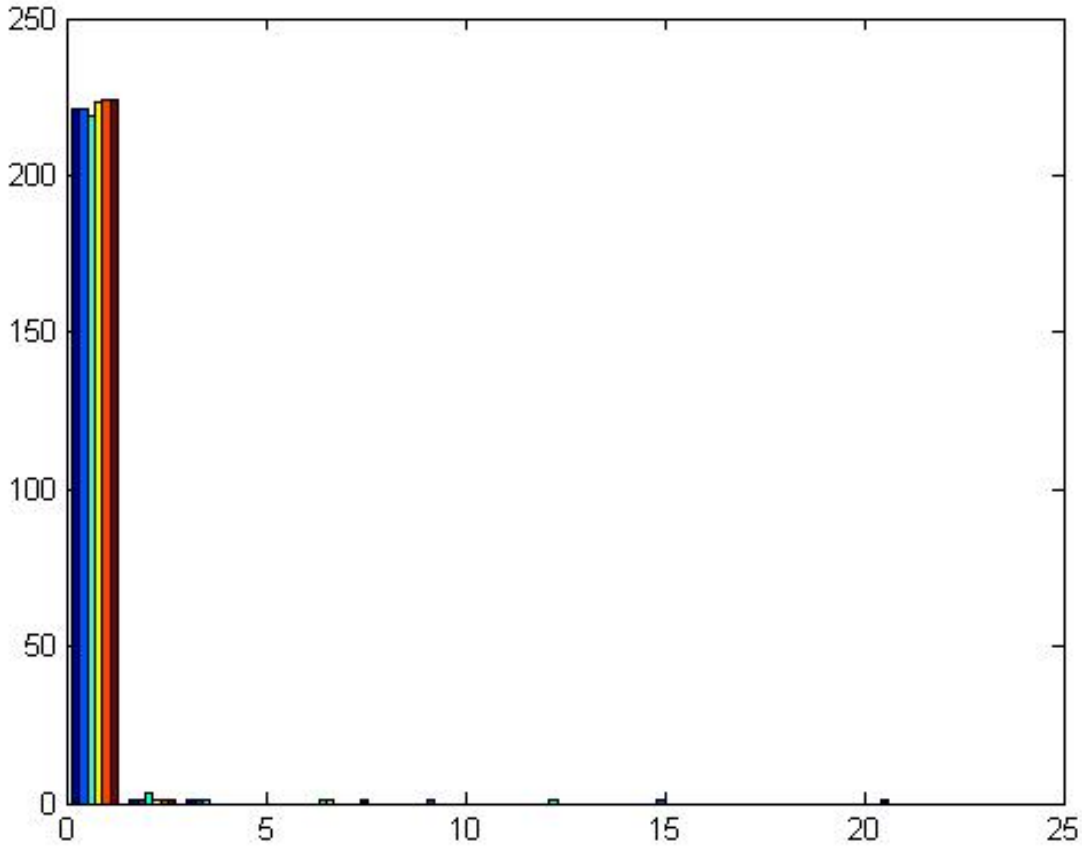
```
001 #include <stdlib.h>
002 #include "rng.h"
003 #include <stdlib.h>
004 #include <iostream>
005 #include <fstream>
006 #include <time.h>
007
008 using namespace std;
009
010 ////////////////////////////////////
011 //for debug purposes///////////////
012 bool toggle_lattice = true;
013 bool toggle_origin_lattice = false;
014 bool toggle_mass_list = false;
015 ////////////////////////////////////
016
017 ////////////////////////////////////
018 //////Mass and Cluster Array///////
019 int M[N*N] = {0};
020 int T[N*N] = {0};
021 ////////////////////////////////////
022
023
024
```

```
025 ////////////////////////////////
026 //////recursive loop for real k////
027 int get_real_k(int k_inp) {
028   if (M[k_inp] >= 0)
029     return k_inp;
030   else
031     return get_real_k(-M[k_inp]);
032 }
033 ////////////////////////////////
034
035
036 ////////////////////////////////
037 ////////////alogrithm//////////////
038 void hkalg(double P, int *clusterSize) {
039
040   //////local variables//////
041   int k = 2;
042   int kmax = 2;
043   srand(time(NULL));
044   rng myRNG; //thats the RNG from Exercise1
045   myRNG.set_values(313, 81916, rand()%20);
046   int lat[N*N];
047
048   //////reset mass, cluster array//////
049   for (int i = 0; i<N*N;i++) {
050     M[i]=0;
051     T[i]=0;
052   }
053
054
055   //////initialize new lattice//////
056   for (int i=0; i<N*N; i++) {
057     if(myRNG.get_rnd()<P) {
058       lat[i]=1; //tree
059     }
060     else {
061       lat[i]=0;  //empty
062     }
063   }
064
065
066   //////detecting first occupied location///////
067   int firstSiteLoc;
068   bool firstSitefound = false;
069   while(!firstSitefound) {
070     int i = 0;
071     if (lat[i]==1) {
072       firstSitefound = true;
073       firstSiteLoc = i;
074       T[i] = k;
075       M[k] = 1;
076     }
077     i++;
```

```
078   }
079
080
081   //////main loop//////////
082   for (int i = firstSiteLoc+1; i<N*N; i++) {
083
084     bool bottomRow = false;
085     bool leftColumn = false;
086     if (i%N == 0) leftColumn = true;
087     if (i-N < 0) bottomRow = true;
088
089     if(lat[i]) {
090
091       //check if bottom and left side are empty - if its the bottomrow, bottom site
is considered empty
092       if (((lat[i-1]==0) || leftColumn) && ((lat[i-N]==0) || bottomRow)) {
093         k += 1;
094         M[k] = 1;
095         T[i] = k;
096         kmax = k;
097       }
098
099       //check if both are occupied but with different cluster indexes
100       else if (((lat[i-1] && !leftColumn) && (lat[i-N] && !bottomRow)) &&
get_real_k(T[i-1]) != get_real_k(T[i-N])) {
101         if (get_real_k(T[i-1]) < get_real_k(T[i-N])) { //choose the smaller
clusternumber to use e.g. k1
102           T[i] = get_real_k(T[i-1]); //assign parent cluster number
103           M[T[i]] += M[get_real_k(M[T[i-N]])] + 1; //assign the proper masses m(k1)
= m(k1) + m(k2) + 1
104           M[T[i-N]] = -T[i-1]; //link k2 to k1
105         }
106         else {
107           T[i] = get_real_k(T[i-N]); //case if k2<k1
108           M[T[i]] += M[get_real_k(M[T[i-1]])] + 1;
109           M[T[i-1]] = -T[i-N];
110         }
111
112       }
113       //check if one or both of them has value (aka is occupied) and if both are
occupied make sure they have the same cluster number
114       else if (((lat[i-1] && !leftColumn) || (lat[i-N] && !bottomRow)) ||
(((lat[i-1] && !leftColumn) && (lat[i-N] && !bottomRow)) && get_real_k(T[i-1]) ==
get_real_k(T[i-N]))) {
115         if (lat[i-1] && !leftColumn) { //also make sure you dont connect clusters
within rows in case only the 2nd statement of the previous if is true
116           T[i] = get_real_k(T[i-1]);
117           M[get_real_k(T[i-1])] += 1;
118         }
119         else if (!bottomRow) {
120           T[i] = get_real_k(T[i-N]);;
121           M[get_real_k(T[i-N])] += 1;
122         }
```

```
123       }
124
125
126     }
127     else
128       T[i] = 0;
129   }
130
131
132   //////if M[j] has mass m, count +1 at clusterSize[m]//////
133   for (int j=2; j<=kmax; j++){
134     if(M[j] >=0){
135       clusterSize[M[j]] += 1;
136     }
137   }
138
139
140   //////for debug purpose only//////
141   if(toggle_mass_list){
142     ofstream myFile2;
143     myFile2.open("Masses.txt");
144     for (int i=0; i<N*N;i++)
145       myFile2 << M[i] << " " << endl;
146   }
147
148   if (toggle_lattice){
149     ofstream myFile3;
150     myFile3.open("lattice.txt");
151     for (int n = N-1; n >= 0; n--){
152       myFile3 << endl;
153       for (int m = n*N; m < (n*N + (N-1)); m++)
154         myFile3 << T[m] << "\t";
155     }
156   }
157
158
159   if (toggle_origin_lattice){
160     ofstream myFile4;
161     myFile4.open("Origin_latt.txt");
162     for (int n = N-1; n >= 0; n--){
163       myFile4 << endl;
164       for (int m = n*N; m < (n*N + (N-1)); m++)
165         myFile4 << lat[m] << "\t";
166     }
167   }
168 }
169
170
171
```

1.2 Plot



Diff colors indicate diff occupation probabilities where blue is the lowest and red the highest - but apparently something's wrong.