INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)

Monte Carlo C++ Code for Modelling the Phase Interface in a 1-octanol/water/solute Solution Used to Find a Partition Coefficient

John H. Summerfield
Professor, Department of Chemical and Physical Sciences,
Missouri Southern State University, Joplin, Missouri, US State
Michael W. Manley
Research Associate, Department of Chemical and Physical Sciences,
Missouri Southern State University, Joplin, Missouri, US State

Abstract:
Typically a partition coefficient is measured using a 1-octanol/water solution. Unfortunately a microemulsion forms at the interface between the two liquids. This emulsion adds to the partition coefficient's error since the solute is in neither phase. A C++ code is given here to model this phase interface. The details of generating the graphics for the microemulsion are published here for the first time. Only free software is relied on so access should not be a problem.

Keywords: *Partition coefficient, microemulsion, Monte Carlo computer model, phase interface*

1. Introduction

Partition coefficients are relied on to calculate tissue bioconcentration of a toxin, soil sorption of a solute, and lethal concentration of a solute. The solution are typically stirred for 30 to 45 days. Even still the phases may not be at equilibrium. Instead a no surfactantmicro emulsionlayer is formed composed of octanol, water, and solute. Since the solute is in neither phase, error is introduced in the partition function calculation. [i] Due to the chaotic nature of this phase boundary it is not easy to analyze experimentally.

Micro emulsions of octanol in the water form. For substances with a K_{OW} greater than 10 this region can be ignored. However if the log K_{OW} is 6, if one of every 100,000 parts of octanol in the experiment exists in the water phase as part of a microemulsion, the experimenter will measure apparent log K_{OW} of 5, or a full order of magnitude low.[ii]

There are a few recent studies that report theoretical analysis based on percolation phenomenon, which occurs as the microstructure of the microemulsion changes. [iii-vi] Recently, the static and dynamic percolation models have been proposed for describing the percolation mechanism. The static percolation theory describes the percolation as a bicontinuous oil and water structure, while the dynamic percolation model describes the formation of percolation clusters by interaction between water globules. This work adopts the dynamic model. [vii, viii]

The program presented attempts to aid in the understanding of partition coefficients by illuminating the 1-octanol/water phase boundary. It is written in C++. This language is notorious for its difficulty with graphics. For this reason step-by-step instructions are included. Free software is used so heavy fees are not involved.

The phase boundary is unorganized. For this reason the computer code uses the Monte Carlo algorithm. Monte Carlo simulation is a type of simulation that relies on repeated random sampling and statistical analysis to compute the results. This method of simulation is very closely related to random experiments, experiments for which the specific result is not known in advance. In this context, Monte Carlo simulation can be considered as a methodical way of doing so-called *what-if* analysis. The program uses a 20 by 20 lattice. The probability of occupying a site is set. Each site is visited with that probability of being occupied. Neighboring sites are checked to see if a cluster of liquid has formed. Each cluster can be clicked on to start a new trial. [ix]

2. The Monte Carlo C++ Code

The computer code functions using the OpenGL graphic interface with the free GLUT toolkit. While the primary libraries being used are in most, if not all, standard C++ front ends and compilers, along with OpenGL being a standard now, the free GLUT library is not implemented by default. In order to make the percolation code function, a list of steps must be followed to add in the free GLUT library. While some sources note of being able to load the library into Microsoft Visual Studios and Dev-C++, the method that worked the best was using the 64-bit version of Eclipse-Neon (not tested on earlier versions).

In order to run the computer code four core programs must be downloaded: Eclipse Neon Installer, Java SE Development Kit (JDK) 8u102 for 64-bit Windows, TDM-GCC 64-bit Bundle, and Martin Payne's Window Binaries (MSVC and MinGW).
After all the applications are download the first program which will be installed is the TDM-GCC bundle:
Three options should come appear in the window: Create, Manage, and Remove. Click the Create button.

1. Click the second box labeled MinGw-w64/TDM64 (32-Bit and 64-bit) and click "Next".
2. After reading the licensing page click "Next".
3. Keep the installation directory as default and click "Next" again.
4. Click Source Forge Default as the Mirror and then click "Next" once more.
5. For this step ensure that the type of install is set to "TDM-GG Recommended, C/C++" then make sure all boxes are checked on the components list except for the option to install "gdb32." After everything is correct click "Install."
6. When installation is complete click "Next" and then "Finish" and the window should disappear.

After complete installation of the TDM-GCC setup the next step is to install the Java SE Development Kit (JDK). When started the installation wizard begins analyzing the processes needed to install the JDK kit and will provide a prompt to click next when ready.

1. Click next from the starting screen and it will take you to the optional features screen.
2. Keep the default features selected and click next to begin preparation for the JDK package.
3. When installation is complete designate the destination folder for the JDK package and click next.
4. Once installation is complete click "Close" and the window should disappear.

With the Java SE Development Kit (JDK) a base program allows Eclipse-Neon to be installed. When the Eclipse Installer is opened a prompt is revealed asking which integrated development environment (IDE) to install with Eclipse-Neon.

1. Select "Eclipse IDE for C/C++ Developers" and a new prompt should pull up for the IDE.
2. Change the Installation Folder to C:\cpp-neon
3. When ready click "Install." Read the Eclipse Foundation Software User Agreement and click "Accept Now" if you agree to the Terms of Service.
4. When the installation is finished, close out of the window and click "No" to prevent the application from launching.

With all of the programs installed onto the system the last step is to properly provide Eclipse-Neon with the designated free GLUT libraries. The compressed file downloaded earlier, Martin Payne's Window Binaries (MSVC and MinGW), contains the designated free GLUT files under the name freeglut-MinGW-3.0.0-1.mp and will need to be unzipped. After unzipping the files are ready to be moved into the designated areas.

1. Open the new unzipped file, which should be labeled as free-glut.MinGW-3.0.0-1.mp and proceed into the "include" folder inside and then into the "GL" folder inside of that.
2. The files "freeglut.h", "freeglut_ext.h", freeglut_std.h", and "glut.h", in the GL folder, should now be copied and pasted into:
   C:\TDM-GCC-64\x86_64-w64-mingw32\include\GL
3. Navigate back to the freeglut-MinGW-3.0.0-1.mp folder. Proceed into the "lib" folder and then into the x64 folder inside of that one. The files "libfreeglut.a" and "libfreeglut_static" should be copy and pasted into:
   C:\TDM-GCC-64\x86_64-w64-mingw32\lib
4. Navigate back to the freeglut-MinGW-3.0.0-1.mp folder. Proceed into the "bin" folder and then into the "x64" Folder inside of that one. The file "freeglut.dll" should be copy and pasted into:

C:\Windows\System32

Now that the libraries are installed the final step is getting eclipse to recognize the libraries themselves so that the Percolation code can be run. After all of the preceding steps are complete Eclipse-Cpp-Neon can be opened. After closing the "Welcome" screen Eclipse is ready to be used.

1. On the top toolbar select Window → Preferences.
2. On the left side of the newly opened prompt expand "C/C++" and click onto "New C/C++ Project."
3. On the right side of the prompt, under "Toolchains" click "MinGW GCC" option and then click "Make Toolchain(s) Preferred." When finished click "OK" and the prompt should now disappear.
4. On the Toolbar at the top click File → New → C++ Project.
5. Name the project and ensure the "MinGW GCC" option is selected under "Toolchains" and then click "Next >."
6. Ensure that the Debug and Release configurations are checked and click "Finish."
7. In the Project Explorer, right click on the newly created project and navigate to New → Source File.
8. Give a designated name to the Source File (ensuring that it ends with *.cpp) and click "Finish."
9. Starting on 8, place the designated code into the program area.
10. In the Project Explorer, right click on the project again and click "Properties."
11. On the left pane of the new prompt expand "C/C++ Build" and select Settings.
12. On the left side of the right pane expand "MinGW C++ Linker" and click on "Libraries."

13. The uppermost right of the right pane should now say Libraries (-I). Click on the "Add New Library" key, which is a green plus sign on a document. Type in "opengl32" and click "OK." The process needs to be repeated to add "glu32" and "freeglut."
14. After all three files are added, click "OK" at the bottom of the prompt.
15. On the top toolbar navigate to Project → Build Project
16. In the Project Explorer, expand the "Binaries" tab and then click the one file inside of it.
17. Go to the lower-most toolbar at the top of the screen and click the Run button (green play button).
18. The computer code should now run.[10]

Microemulsion Code:

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

#ifdef __APPLE__
#  include <GLUT/glut.h>
#else
#  include <GL/glut.h>
#endif

int N = 20;                  // number of sites in x and y
double p = 0.55;             // site occupation probability
bool** occupied;             // NxN array to store the state of the lattice

int clusters;                // number of clusters
int** label;                 // NxN array of cluster labels for each site
int currentLabel;            // label of current cluster
int spanningLabel;           // label of spanning cluster

void addNewNeighbor(int i, int j) {
   if (occupied[i][j] &&       // site is occupied, and
      label[i][j] == 0)       // not yet labeled
   {
      label[i][j] = currentLabel;
      if (i < N-1)
         addNewNeighbor(i+1, j);   // east
      if (i > 0)
         addNewNeighbor(i-1, j);   // west
      if (j < N-1)
         addNewNeighbor(i, j+1);   // north
      if (j > 0)
         addNewNeighbor(i, j-1);   // south
   }
}

void newSample() {

   // visit each site and occupy it with probability p
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         occupied[i][j] = (rand() / (RAND_MAX + 1.0)) < p;
         label[i][j] = 0;    // not yet labeled
      }
   }

   // find and label all clusters of occupied sites
```

```
    clusters = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (occupied[i][j] &&   // site is occupied, and
                label[i][j] == 0)   // not yet labeled
            {
                currentLabel = ++clusters;  // assign a new label
                addNewNeighbor(i, j);       // add to new cluster
            }
        }
    }

    // check each cluster for percolation
    spanningLabel = 0;
    for (int cluster = 1; cluster <= clusters; ++cluster) {

        // check west boundary sites
        bool west = false;
        for (int j = 0; j < N; j++)
            if (label[0][j] == cluster)   { west = true; break; }

        // check east boundary sites
        bool east = false;
        for (int j = 0; j < N; j++)
            if (label[N-1][j] == cluster) { east = true; break; }

        // check south boundary sites
        bool south = false;
        for (int i = 0; i < N; i++)
            if (label[i][0] == cluster)   { south = true; break; }

        // check north boundary sites
        bool north = false;
        for (int i = 0; i < N; i++)
            if (label[i][N-1] == cluster) { north = true; break; }

        // check whether cluster touches all boundaries
        if (west && east && south && north) {
            spanningLabel = cluster;
            break;
        }
    }
}

void initialize() {
    // allocate memory for arrays
    occupied = new bool* [N];
    label = new int* [N];
    for (int i = 0; i < N; i++) {
        occupied[i] = new bool [N];
        label[i] = new int [N];
    }
    newSample();
}

void step() {
    newSample();
    glutPostRedisplay();
}
```

```
int mouseLabel;         // label of site under mouse pointer

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++) {
      if (occupied[i][j]) {
        if (label[i][j] == spanningLabel)
           glColor3ub(255, 0, 0);
        else if (label[i][j] == mouseLabel)
           glColor3ub(0, 0, 255);
        else
           glColor3ub(0, 255, 0);
      } else continue;
      glPushMatrix();
      glTranslated(i + 0.5, j + 0.5, 0);
      glBegin(GL_POLYGON);
         const double pi = 4*atan(1.0);
         const int circlePoints = 12;
         const double r = 0.5;
         for (int k = 0; k < circlePoints; k++) {
            double angle = 2*pi*k/double(circlePoints);
            glVertex2d(r*cos(angle), r*sin(angle));
         }
      glEnd();
      glPopMatrix();
   }
   glutSwapBuffers();
}

void reshape(int w, int h) {
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   double aspect = w / double(h);
   if (aspect > 1) {
      double dx = (aspect - 1) * N / 2.0;
      glOrtho(-dx, N + dx, 0, N, -1, 1);
   } else {
      double dy = (1/aspect - 1) * N / 2.0;
      glOrtho(0, N, -dy, N + dy, -1, 1);
   }
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

bool running;           // is the animation running?

void mouse(int button, int state, int x, int y) {
   switch (button) {
   case GLUT_LEFT_BUTTON:
      if (state == GLUT_DOWN && !running)
         step();
      break;
   case GLUT_RIGHT_BUTTON:
      if (state == GLUT_DOWN) {
         if (running)
            glutIdleFunc(NULL);
```

```
            else
               glutIdleFunc(step);
            running = !running;
         }
         break;
      default:
         break;
      }
}

void hover(int x, int y) {
   int w = glutGet(GLUT_WINDOW_WIDTH);
   int h = glutGet(GLUT_WINDOW_HEIGHT);
   double x0, y0, d;
   if (w < h) {
      x0 = 0;
      y0 = (h - w) / 2;
      d = w;
   } else {
      x0 = (w - h) / 2;
      y0 = 0;
      d = h;
   }
   double dd = d / (N - 1);
   int i = int(N * (x - x0) / d);
   int j = int(N * (d - y + y0) / d);
   if (i >= 0 && i < N && j >= 0 && j < N) {
      mouseLabel = label[i][j];
      glutPostRedisplay();
   }
}

int main(int argc, char *argv[]) {
   glutInit(&argc, argv);
   if (argc > 1) {
      istringstream is(argv[1]);
      is >> p;
   }
   initialize();
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(400, 400);
   glutInitWindowPosition(100, 100);
   ostringstream os;
   os << "Percolation in 2D: p = " << p << ends;
   glutCreateWindow(os.str().c_str());
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glShadeModel(GL_FLAT);
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutPassiveMotionFunc(hover);
   glutMainLoop();
}[11, 12]
```

3. Result and Discussion

In Figure 1 the green dots represent water droplets. The red dots are octanol being emulsified. The program enables one to click on any region of the output and restart the simulation using the same probability. That is, the site probability is unchanged.

At 0.1 probability there isn't any microemulsion formation. At 0.55, the mixture is still equilibrating. This can be seen by clicking on the 0.55 output. A microemulsion sometimes forms. This is shown in Figure 2 for 0.55 and 0.66.

Figure 1 illustrates the formation of microemulsions for different probabilities.

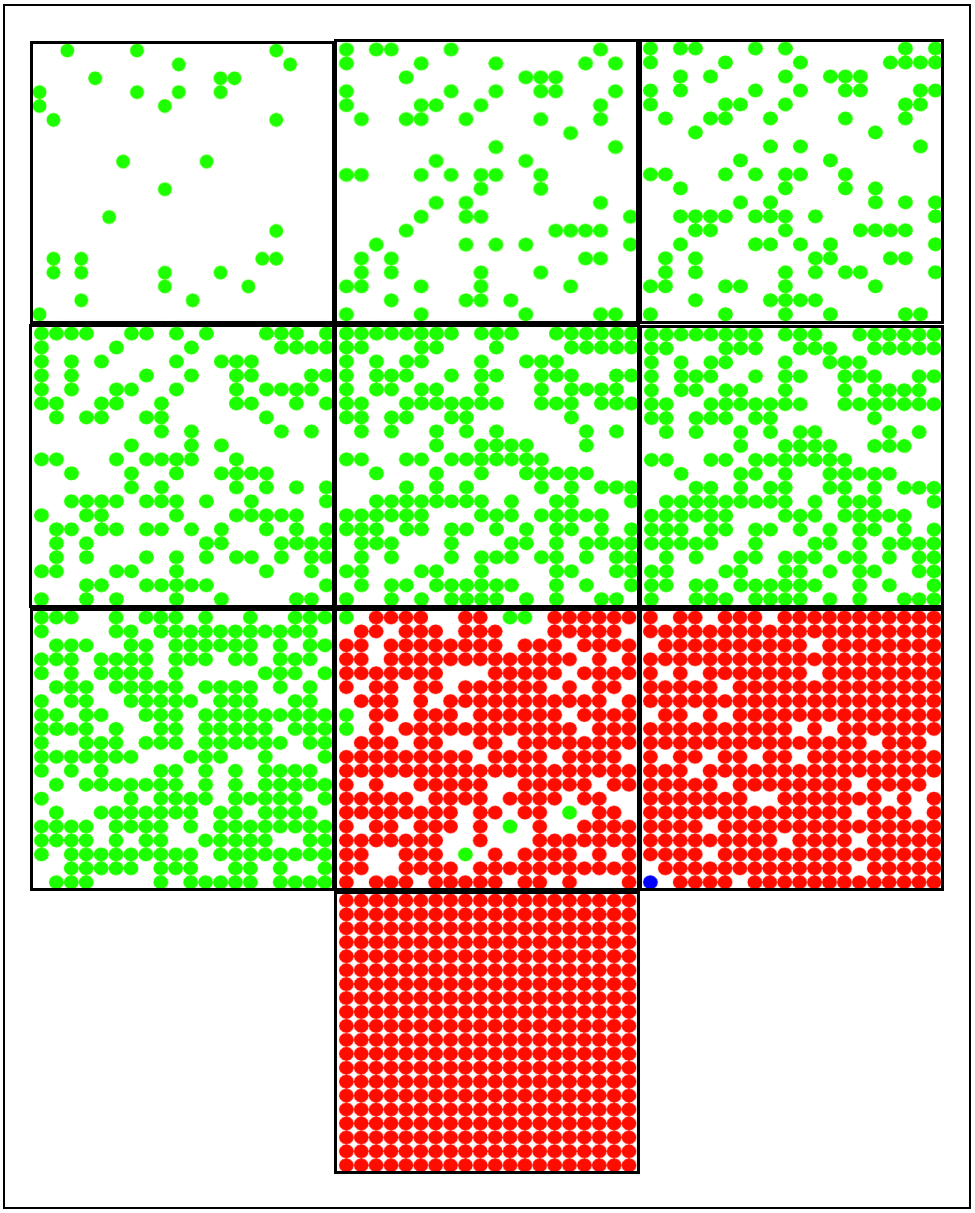

Figure 1.Starting in the upper left and moving left to right, probability of 0.1, 0.2, 0.3, 0.4, 0.55, 0.6, 0.7, 0.8, 0.9, 1.0.

Figure 2 shows two results when the algorythimis restarted at a set probability.

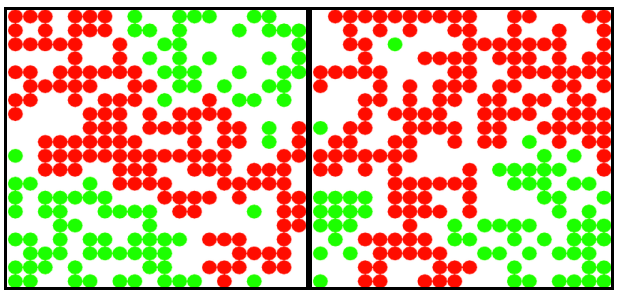

Figure 2: From the left, the restarted code at probability 0.55 and 0.60.

The restarted code shows emulsion formation. The oil is surrounded by water droplets. Dentrite, finger-like penetration takes place as microemulsion begins. At 0.40 and below, emulsion formation does not occur on restart.
By visualizing micoemulsion formation as a randonized process we are better able to understand it.

4. References

i. L. Ropel, L.S. Belvèze, S. Aki et al., (2005) 1-octanol-water Partition Coefficients of Imidazolium-based Ionic Liquids, Green Chem., 2005, 83-90. 10.1039/B410891D
ii. Tolls, J, Bodo, K., et al., (2003) Slow-stirring method for determining the n-octanol/water partition coefficient for highly hydrophobic chemicals, Environ. Toxicol. Chem., 22, 1051-1057.
iii. Metha, S.K., Kaar, G., Bhasin K.K., (2007), Analysis of Tweenbased microemulsion in the presence of TB drug rifampicinColloid Surf B., 8, 95-104.
iv. Peng, N., Hou, W.G., (2008)A Novel Surfactant-free Microemulsion System: N,N-Dimethyl Formamide/Furaldehyde/H_2O J. Chem. 26, 1335-1338
v. Metha, S.K., Kaar, G., Bhasin K.K., (2008), Incorporation of antitubercular drug isoniazid in pharmaceutically accepted microemulsion: effect on microstructure and physical parameters, Pharmaceut. Res., 25, 227-236.
vi. EL-Hefnawy, M., (2012)Water in Olive Oil Surfactantless Micro emulsions as Medium for CdS Nanoparticles Synthesis Mod. Appl. Sci. 6, 101-105
vii. Lagourette, B., Peyrelasse, J., et al. (1979) Percolative Conduction in Microemulsion Type Systems, Nature, 6, 60-62.
viii. Zech, O. Thomaier, S., et al. (2009), Microemulsion with an ionic liquid surfactant and room temperature and room temperature ionic liquids as polar pseudo phase, J. Phys. Chem. B, 113, 465-473.
ix. Jain, S. (1992) Monte Carlo Simulations of Disordered Systems. World Scientific, New York.
x. https://www.youtube.com/watch?v=qFlJXMpxAO4
xi. http://journals.aps.org/prl/abstract/10.1103/PhysRevLett.85.4104
xii. http://www.physics.buffalo.edu/phy411-506/topic2/