

Manual

Langevin 2D

Table of Contents

1. INTRODUCTION AND SYSTEM DESCRIPTION	2
1.1 THE SYSTEM UNDER CONSIDERATION	2
1.2 FILE STRUCTURE AND LOCATION OF THE INPUT PARAMETERS	3
2. RUNNING THE SIMULATION	8
2.1 QUICK START – RUNNING A SIMPLE SIMULATION	8
2.2 MOLECULE INFORMATION AND FILE CONTENT	9
2.2.1 <i>Additional functionality of the program</i>	10
3. OUTPUT FILES	11
3.1 OUTPUT FILE STRUCTURE – EXAMPLE	12

1. Introduction and System Description

The purpose of this manual is to give a quick introduction to the system under consideration, the general features of the algorithm used to solve the problem and a fast guide to run the simulations. Further details on the functions, methods and other documentation can be found within the *lang2D.cpp* file. It is assumed that the user has enough C++ knowledge to run and modify a simple code, as needed.

1.1 The system under consideration

Consider a system consisting of a liquid of infinite volume where molecules of type *A* and *B* are diffusing around the fluid undergoing Brownian motion. The molecules are made out of circles and diffusing in a two-dimensional constrained environment. The environment in which the molecules move is a two-dimensional channel (that will be referred to as the *pore*) of infinite length (*x*-axis) and finite width *W* (*y*-axis).

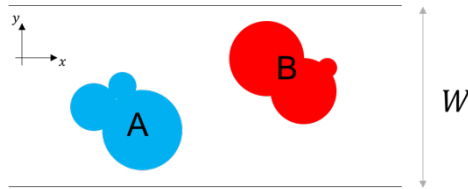


Figure 1: Model of the molecules diffusing in the constrained environment. The molecules can translate and rotate such that they do not intersect with each other and/or the pore walls.

The molecules move in the over-damped regime, $\frac{d^2\vec{x}}{dt^2} = 0$, and the molecules obey steric constraints, i.e., the molecules can translate within the channel along the *x* and *y* axis, and rotate in the *xy* plane, with the only constraint that they cannot intersect each other or intersect the pore walls. As the molecules diffuse inside the pore, if there is enough space, the molecules will go past each other. The goal of this program is to determine how many times the molecules pass each other and how many times the molecules separate/fail to pass each other.

1.2 File structure and location of the input parameters

The file starts with a brief description of what the program does and some basic information:

```
/*  
  
Title: Langevin simulation in two dimensions.  
  
Author: Andres Garcia.  
  
Description: Langevin simulation to calculate the passing propensity of two polyatomic molecules made of  
circles.  
  
Date: March 2 / 2016.  
  
*/
```

Following the basic information is the “Modules and Imports” section. In this section, the specific libraries to be imported are defined; these contain the functionality for the needed operations:

```
//-----  
//  MODULES AND IMPORTS  
//-----  
  
#include <algorithm> // Has the functions for Min and Max (see documentation)  
#include <array> // Contains the library for manipulating arrays  
#include <cmath> // Trigonometric, exponential, logarithmic functions, etc.  
#include <fstream> // To write the info in a file  
#include <iostream> // Basic input/output file  
#include <random> // Includes the random number generators (we want the Mersenne Twister, mt19937_64)  
#include <sstream> // Creates a special input/output stream  
#include <string> // To be able to manipulate strings  
#include <vector> // The proper package with the functions to manipulate vectors  
#include <time.h> // Contains time functions, to seed the generator  
  
using namespace std;
```

Next, the pseudo random number generator is defined. In this case we use the Mersenne Twister that is seeded by a pure random number generator, or by an unsigned integer seed. The generator is not the same as the distribution used, thus, the distribution is also defined. The distribution is defined to be a uniform real distribution that generates long double numbers in the range (0.0L, 1.0L); where the suffix “L” indicates that the number should be interpreted as a long double number:

```
//-----  
//  RANDOM NUMBER GENERATOR(S)  
//-----  
  
random_device rd; // Get a random seed from a generic random device  
unsigned int seed = rd(); // Get a random seed  
mt19937_64 generator(seed); // Get the number generator (the 64 bit Mersenne Twister)  
normal_distribution<long double> rand_gauss(0.0L, 1.0L); // Define the Gaussian distribution  
uniform_real_distribution<long double> rand_real(0.0L, 1.0L); // A uniform real valued distribution
```

The section after is “File Names and Save Options”. This program outputs files with the final statistics, so that they can be read and analyzed using an appropriate graphical interface. Thus, the names of the output files are defined here for ease of manipulation:

```
//-----  
//  FILE NAMES AND SAVE OPTIONS  
//-----  
  
// File name to save  
const string FL_NAME_BASE = "lang2D";  
const string FILE_SAVE_CONF = FL_NAME_BASE + "-mlConfig.csv";  
const string FILE_SAVE_NAME = FL_NAME_BASE + "-2Dresults.csv";
```

The names of the files refer to:

- FL_NAME_BASE: The pre-name of the file used for customized file name output.
- FILE_SAVE_CONF: The name of the file that contains the configuration of the molecules within the pore.
- FILE_SAVE_NAME: The file that contains the final statistics.

The section that follows is “Constants”, that is the most important section since it is the section where the parameters have to be input for the program to run properly. It will be discussed in a section of its own, thus, for the moment, the section after this one is the one that is going to be described.

The section following is the “General Parameters” section. In this section, the storage and statistics variables are defined for the general program:

```
//-----  
//  GENERAL PARAMETERS  
//-----  
  
// Auxiliary counters  
int cntT;  
int cntW;  
  
// Pass/fail parameters  
int fails;  
int passes;  
  
// The save counter  
int saveCnt;  
bool firstTime;  
  
// If a simulation is to be loaded  
bool loadSim;  
  
// Percentage tracker  
long double perc;  
  
// The maximum phase space possible for a molecule to move  
long double TOLLIM1;  
long double TOLLIM2;  
  
// The maximum angle that the molecules can rotate  
long double MAXANGL1;  
long double MAXANGL2;  
  
// The success or fail condition  
long double failPass;  
long double successPass;  
  
// The time step to be used  
long double TIMESTEP;
```

The next two sections contain the variables where the information of the molecules is stored:

```
//-----  
//  PARAMETERS FOR MOLECULE 1  
//-----  
  
const int N1 = 1; // Number of atoms for molecule 1  
  
string mol1Name; // The name of molecule 1  
  
long double boundR1; // Bounding radius for molecule 1  
long double Xcm1[2]; // Array that stores the position of the center of mass of molecule 1  
long double X1[N1][2]; // Array that stores the position vectors of the atoms in molecule 1  
long double Xor1[2][2]; // Array that stores the orientation vectors of molecule 1  
long double radii1[N1]; // Radii of the atoms in molecule 1  
  
// Difusion tensor of molecule 1, D1 = (D_1_x, D_1_y, D_1_theta)  
long double D1[3];  
  
long double Xcm1i[2]; // Array that stores the position of the center of mass of molecule 1 initially  
long double X1i[N1][2]; // Array that stores the position vectors of the atoms in molecule 1 initially  
long double Xor1i[2][2]; // Array that stores the orientation vectors of molecule 1 initially  
  
long double Xcm1T[2]; // Array that stores the position of the center of mass of molecule 1 temporarily  
long double X1T[N1][2]; // Array that stores the position vectors of the atoms in molecule 1 temporarily  
long double Xor1T[2][2]; // Array that stores the orientation vectors of molecule 1 temporarily  
  
//-----  
//  PARAMETERS FOR MOLECULE 2  
//-----  
  
const int N2 = 3; // Number of atoms for molecule 2  
  
string mol2Name; // The name of molecule 2  
  
long double boundR2; // Bounding radius for molecule 2  
long double Xcm2[2]; // Array that stores the position of the center of mass of molecule 2  
long double X2[N2][2]; // Array that stores the position vectors of the atoms in molecule 2  
long double Xor2[2][2]; // Array that stores the orientation vectors of molecule 2  
long double radii2[N2]; // Radii of the atoms in molecule 2  
  
// Difusion tensor of molecule 2, D2 = (D_2_x, D_2_y, D_2_theta)  
long double D2[3];  
  
long double Xcm2i[2]; // Array that stores the position of the center of mass of molecule 2 initially  
long double X2i[N2][2]; // Array that stores the position vectors of the atoms in molecule 2 initially  
long double Xor2i[2][2]; // Array that stores the orientation vectors of molecule 2 initially  
  
long double Xcm2T[2]; // Array that stores the position of the center of mass of molecule 2 temporarily  
long double X2T[N2][2]; // Array that stores the position vectors of the atoms in molecule 2 temporarily  
long double Xor2T[2][2]; // Array that stores the orientation vectors of molecule 2 temporarily
```

The variables N1 and N2 should be the same as the number of atoms defined in the files that have the molecule's information. The documentation in the code is self-explanatory.

There are a couple of sections where other parameters and constants that do not need to be modified are defined:

```
//-----  
//  FILE OPERATION CONSTANTS  
//-----  
  
//Constants for the different operations  
const int OPEN_FILE = 0;  
const int WRITE_FILE = 1;  
const int NEW_LINE_FILE = 2;  
const int OPEN_FILE_OVER = 3;  
  
//-----  
//  PORE PARAMETERS  
//-----  
  
long double poreTop; // Denotes the coordinate at the top of the pore  
long double poreBot; // Denotes the coordinate at the bottom of the pore  
long double poreWidth; // Denotes the width of the pore  
  
//-----  
//  RESULTS PARAMETERS  
//-----  
  
// Get the size for the array of results  
vector <vector <long double> > timesAndWidths; // The array that keeps track of the examined times
```

This program has a ***save and load*** features in case anything goes wrong or the simulation has to be stopped and run from the last save point. The variables in the “Save and Load File Constants and Variables” section accomplishes this:

```
//-----  
//  SAVE AND LOAD FILE CONSTANTS AND VARIABLES  
//-----  
  
//Name of the file where to save the simulation state  
const string STATE_FILE = "state.csv";  
const string GEN_STATE_FILE = "genState.txt";  
  
//If the simulation should be saved/loaded  
const bool LOAD_FILE = false;  
const bool SAVE_FILE = true;
```

The variables are defined such that:

- `const string STATE_FILE`: The name of the file where the state of the simulation is stored. Two files are created to account for file corruption; e.g., if the simulation is stopped in the middle writing one of the files. The extension must be that of a comma

separated file (.csv). This file does **not** save the seed or the state of the random number generator.

- `const string GEN_STATE_FILE`: The name of the file where the random number generator of the simulation is saved. Two files are created to account for file corruption; e.g., if the simulation is stopped in the middle writing one of the files. The extension must be that of a text file (.txt).
- `const bool LOAD_FILE`: The boolean variable that denotes if a simulation file should be loaded. True means that a simulation should be loaded from a file. False means that a simulation should **not** be loaded from a file.
- `const bool SAVE_FILE`: The boolean variable that denotes if the simulation should be saved. True means that a simulation should be saved to a file periodically. False means that a simulation should **not** be saved to a file.

2. Running the Simulation

2.1 Quick start – Running a simple simulation

To simply run the simulation, under the section “Constants”:

```
//-----  
//  CONSTANTS  
//-----
```

The following parameters have to be given as input:

- The counter of that determines how many trials should be made to determine the passing propensity, for this case, use 1000:

```
// Number of simulations over which to average  
const int MAXCNTR = 1000;
```

- The number of trials after which to save the state of the simulation, use 10 for this case:

```
// Number of trials after which to save  
const int SAVEAFT = 100;
```


- The name of the files where the information of the molecules is stored. They must have the file extension as shown. In our case, the examples are that of a monomer and a trimer. More details on how the file must be formatted is given in XYZ:

```
// Strings that contain the file names of the molecules to load
const string ST_M1 = "monomer.csv";
const string ST_M2 = "trimer.csv";
```

- This program has the feature that multiple time steps and pore diameters can be examined in a single run; sequentially, not in parallel. Thus, multiple times to be examined along with the pore diameters can be specified. In our case, we shall examine two time steps, and for each time step three pore widths, for a total of six sets of parameters:

```
// Time and diameter vectors
vector<long double> timesExam = {0.01L, 0.01L };
vector<long double> diamsExam = {4.4L, 4.8L, 5.2L,}
```

To run the program, run with the usual code with the usual compiler and with the flag:

```
-std=c++11 or -std=c++0x
```

or equivalent.

2.2 Molecule information and file content

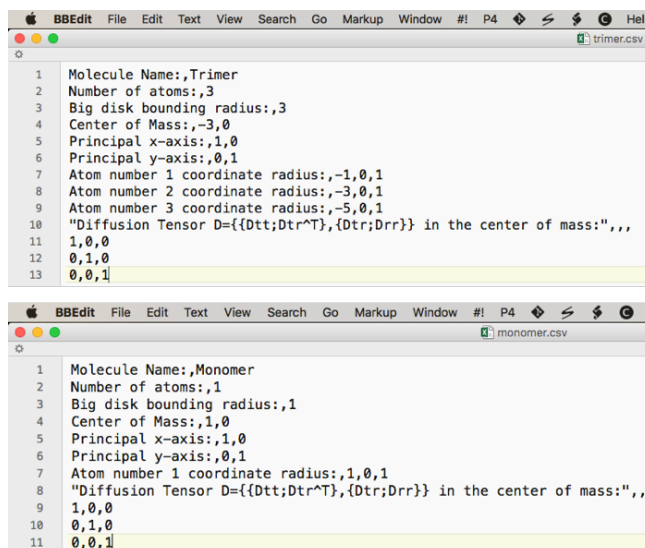
For the program to work, the information file of the molecules must be provided in a very specific format. Below is an example of the file as viewed with MS Excel and with Notepad:

A	B	C	D
Molecule Name:	Monomer		
1	Molecule Name:	Monomer	
2	Number of atoms:	1	
3	Big sphere bounding radius:	1	
4	Center of Mass:	1	0
5	Principal x-axis:	1	0
6	Principal y-axis:	0	1
7	Atom number 1 coordinate radius:	1	0
8	Atom number 2 coordinate radius:	0	1
9	Atom number 3 coordinate radius:	1	0
10	Diffusion Tensor D=([Dtt;Dtr*T],[Dtr;Drr]) in the center of mass:		
11		1	0
12		0	1
13		0	0
14		0	1

A	B	C	D
Molecule Name:	Trimer		
1	Molecule Name:	Trimer	
2	Number of atoms:	3	
3	Big sphere bounding radius:	3	
4	Center of Mass:	-3	0
5	Principal x-axis:	1	0
6	Principal y-axis:	0	1
7	Atom number 1 coordinate radius:	-1	0
8	Atom number 2 coordinate radius:	-3	0
9	Atom number 3 coordinate radius:	-5	0
10	Diffusion Tensor D=([Dtt;Dtr*T],[Dtr;Drr]) in the center of mass:		
11		1	0
12		0	1
13		0	0
14		0	1

Figure 2: The format in which the files should be organized. Notice that for the monomer, corresponding to **molecule 1**, the center of mass is positioned in the **right side** of the pore (positive x coordinate); the trimer, corresponding to **molecule 2**, the center of mass is set to the **left side** of the pore (negative x coordinate).

If the files are opened in a text editor such as BBEdit, the files **must** have the following structure:



```

1 Molecule Name:,Trimer
2 Number of atoms:,3
3 Big disk bounding radius:,3
4 Center of Mass:,-3,0
5 Principal x-axis:,1,0
6 Principal y-axis:,0,1
7 Atom number 1 coordinate radius:,-1,0,1
8 Atom number 2 coordinate radius:,-3,0,1
9 Atom number 3 coordinate radius:,-5,0,1
10 "Diffusion Tensor D={{Dtt;Dtr^T},{Dtr;Drr}} in the center of mass:",,,
11 1,0,0
12 0,1,0
13 0,0,1

1 Molecule Name:,Monomer
2 Number of atoms:,1
3 Big disk bounding radius:,1
4 Center of Mass:,1,0
5 Principal x-axis:,1,0
6 Principal y-axis:,0,1
7 Atom number 1 coordinate radius:,1,0,1
8 "Diffusion Tensor D={{Dtt;Dtr^T},{Dtr;Drr}} in the center of mass:",,,
9 1,0,0
10 0,1,0
11 0,0,1

```

Figure 3: The files in a non-spreadsheet program. Notice that the lines must end with the newline character. If the file is **not** in this format, the program will fail to run.

The structure and the meaning of the lines is self-explanatory. The only quantity that requires an explanation is the “Big disk bounding radius:” quantity. The “Big disk bounding radius” is defined as the disk with the minimum radius that encloses the molecule, centered at the **center of mass** of the molecule:

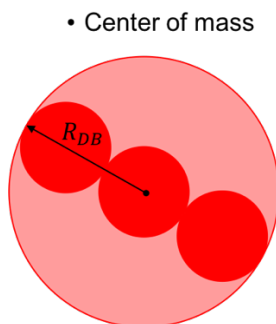


Figure 4: The “Bounding disk radius” is defined as the disk with the minimum radius that encloses the molecule, centered at the center of mass. A linear trimer (dark red circles) with the center of mass at the geometrical center is shown. The bounding disk radius, R_{DB} , is half the length of the longest axis of the molecule (light red big circle).

2.2.1 Additional functionality of the program

There is the option of printing the configuration of the pore molecules using the function:

`void write_mol_config(int).`

This function receives an integer parameter defined in the “Files Names and Save Options”.

```
//-----  
//  FILE NAMES AND SAVE OPTIONS  
//-----
```

Before using the function to print the state of the pore, the function should be called to open and generate the file where the information will be stored. The way to call the function is:

```
write_mol_config(OPEN_FL).
```

To write the pore configuration, use:

```
write_mol_config(WRIT_FL).
```

3. Output Files

When the program is done running, output files will be generated with the statistics. The files that are generated by default are the files that contain the single, double, triple, quartet and quintet statistics.

The files are generated by the function *write_info_to_file()*, that contains the instructions. Since the computing precision of the spreadsheet programs and other programming languages can be less than the `long double` used in the program, there is a function to generate customized output:

```
write_info_to_file_custm().
```

The statistics for a specific run can be obtained from the accumulated statistics variables:

```
long double failPass;  
long double successPass;
```

In the documentation within the program file, it is specified how the statistics are defined.

3.1 Output file structure – Example

The output file is a comma separated valued file. If the files are opened in MS Excel 2016, the files will display the information of the parameters of the system and the information of the passing propensity of the molecules after a certain number of trials, see Figure 5.

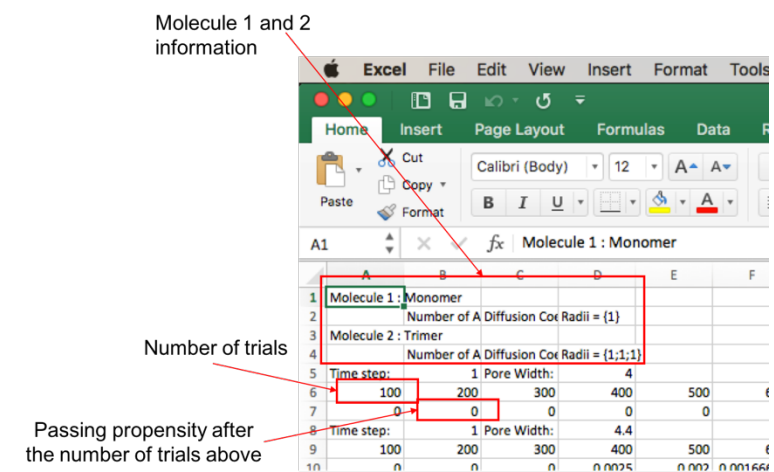


Figure 5: The output file with the statistics. Some of the entries are self-explanatory.