Symulacje Monte Carlo: Ciecz Lennarda – Jonesa

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Kod programu

(Napisany w języku C++)

// Input format:

// ./lj2d L N MCS T [U\_minstep] [U\_jump] [start\_dist] [delta]

#include <iostream>

#include <iomanip>

#include <assert.h>

#include <random>

#include <math.h>

#include <string>

#include <time.h>

#include <forward\_list>

using namespace std;

typedef unsigned int uint;

// GLOBAL CONSTANTS (to be read from parameters) \_\_\_

double L;

double delta;

double T;

uint N;

uint MCS;

double start\_dist;

// GLOBAL ARRAYS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

double\* x;

double\* y;

double\* U;

// RANDOM NUMBER GENERATOR

// (returns random double x = [0.0, 1.0)) \_\_\_\_\_\_\_\_\_\_

uint seed = time(nullptr);

default\_random\_engine eng(seed);

uniform\_real\_distribution<double> dist(0., 1.);

double rndReal()

{

    return dist(eng);

}

// PERIODIC BOUNDARY CONDITIONS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

double pbc(double q)

{

   double pbc = fmod(q,L);

   if(pbc < 0) pbc += L;

   return pbc;

}

// CALCULATION OF ENERGY OF AN ATOM

// (after being moved by [dx,dy]) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

double energy(uint i, double dx = 0.0, double dy = 0.0)

{

    double Ui = 0;

    for(int j = 0; j < N; ++j)

    {

        if(i != j)

        {

            double dxij = pbc(x[i] + dx) - x[j];

            if(dxij > L/2) dxij = L - dxij;

            if(dxij < -L/2) dxij = L + dxij;

            double dyij = pbc(y[i] + dy) - y[j];

            if(dyij > L/2) dyij = L - dyij;

            if(dyij < -L/2) dyij = L + dyij;

            double r2ij = pow(dxij,2) + pow(dyij,2);

            Ui += 4.0\*(pow(r2ij,-6) - pow(r2ij,-3));

        }

    }

    return Ui;

}

// CALCULATION OF VARIANCE \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

double variance(forward\_list<double>\* list)

{

    double sum = 0;

    uint count = 0;

    for(auto j = list->begin(); j != list->end(); ++j)

    {

        sum+=\*j;

        ++count;

    }

    double avg = sum/count;

    double var = 0;

    for (auto j = list->begin(); j!= list->end(); ++j)

    {

        var += pow(\*j-avg,2);

    }

    var /= count;

    return var;

}

// SINGLE MONTE CARLO STEP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void mcStep()

{

    for(uint i = 0; i < N; i++)

    {

        double dx = delta\*(rndReal()-0.5);

        double dy = delta\*(rndReal()-0.5);

        double u\_new = energy(i,dx,dy);

        double dU = u\_new - U[i];

        if(dU < 0  ||  rndReal() < exp(-dU/T))

        {

            U[i] = u\_new;

            x[i] = pbc(x[i]+dx);

            y[i] = pbc(y[i]+dy);

        }

    }

    return;

}

int main(int argc, char\*\* argv)

{

    assert(argc > 4);

    // READ INPUT PARAMETERS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    L = stod(argv[1]);

    N = stoi(argv[2]);

    MCS = stoi(argv[3]);

    T = stod(argv[4]);

    uint U\_minstep =(argc > 5) ? stoi(argv[5]) : 0;

    uint U\_jump = (argc > 6) ?  stoi(argv[6]) : 1;

    start\_dist = (argc > 7) ? stod(argv[7]) : 0.3;

    delta = (argc > 8) ? stod(argv[8]) : 0.1;

    assert(L>0);

    assert(N>0);

    assert(MCS>=0);

    assert(T>0);

    double ro = (double)N / L / L;

    cout <<"# L = "<< L << "  N = " << N << "  MCS =" << MCS

    << "  ro\* = " << ro << "  T\* = "<< T << endl

    << "# Seed = " << seed << endl;

    // GENERATE START CONFIGURATION \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    x = new double[N];

    y = new double[N];

    U = new double[N];

    forward\_list<double> U\_step;

    uint rows = sqrt(N);

    uint i = 0;

    for (uint j = 0; j < rows; j++)

    {

        for (uint k = 0; k < N/rows +1; k++)

        {

            if (i < N)

            {

                x[i] = start\_dist\*j;

                y[i] = start\_dist\*k;

                i++;

            }

        }

    }

    for (uint i = 0; i < N; i++)

    {

        U[i] = energy(i);

    }

    // RUN THE SIMULATION \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    for(uint step=0; step<MCS; step++)

    {

        mcStep();

        if (step > U\_minstep)

        {

            if ((step-U\_minstep) % U\_jump == 0)

            {

                double U\_total = 0;

                for(uint i=0;i<N;i++) U\_total += U[i];

                U\_step.push\_front(U\_total);

            }

        }

    }

    double c = 1/(N\*T\*T) \* variance(&U\_step);

    // PRINT RESULTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    cout << "# x\t\t  y\t\t  U" << endl;

    cout << "# c = " << c <<endl;

    for (uint i = 0; i < N; i++)

    {

        cout << fixed << x[i]<< "\t"  << y[i] << "\t" << scientific << U[i] << endl;

    }

    delete x;

    delete y;

    delete U;

    return 0;

}