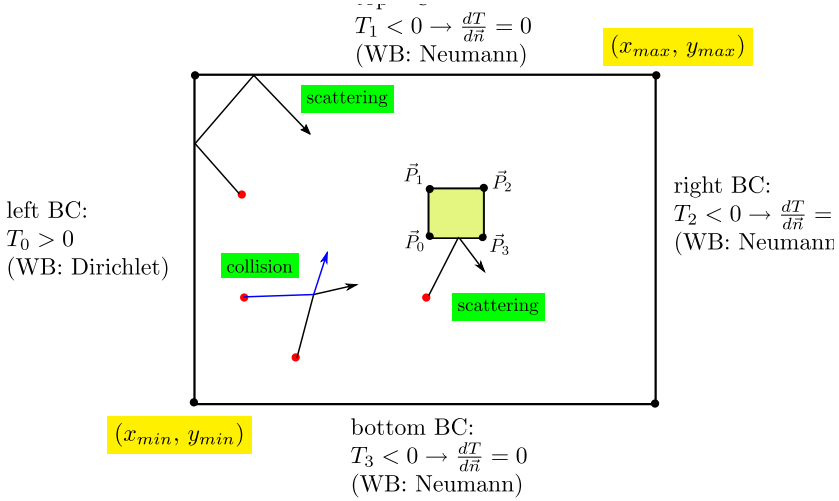
Monte Carlo: symulacja dynamiki gazu - część 1

15 czerwca 2023

1 Wstęp

1.1 Metoda bezpośredniej symulacji MC (Direct Simulation Monte Carlo)

Na zajęciach wykonamy symulację dynamiki gazu przy użyciu metody MC. Geometria układu pokazana jest na rysunku 1. W symulacji cząsteczki gazu mogą zderzać się ze sobą, z brzegiem układu oraz z brzegiem obiektu umieszczonego w środku. Na brzegu zewnętrznym możemy zadać warunki brzegowe: Dirichleta - wówczas cząstka padająca na brzeg jest zastępowana cząsteczką o energii kinetycznej losowanej z rozkładu Maxwella dla temperatury brzegu, warunek brzegowy Neumana - wówczas ze względu na zerowanie gradientu temperatury cząstki są odbijane (kąt padania = kąt odbicia). Symulację wykonamy przy pomocy procedur zawartych w klasie **DSMC_2D**.



Rysunek 1: Geometria układu w którym wykonujemy symulację gazu metodą DSMC. Cząstki gazu zamknięte są w obszarze prostokątnym, którego lewy dolny róg wyznacza punkt (x_{min}, y_{min}) a prawy górny to (x_{max}, y_{max}). Cząsteczki gazu rozpraszają się na sobie oraz na brzegach układu lub obiektu umieszczonego w środku (opcjonalnie). Na brzegach zewnętrznych możemy zadać warunki brzegowe Dirichleta lub Neumanna. Obiekt rozpraszający w środku definiujemy w postaci wielokąta podając jego wierzchołki w kolejności zgodnej z ruchem wskazówek zegara (zderzenia są wykrywane jako przecięcie trajektorii z odpowiednio zorientowaną krawędzią brzegu).

W obliczeniach tego typu zazwyczaj używa się cząstek gazu w ilości $N = 10^5 - 10^7$. Aby przyśpieszyć działanie programu należy rozważać ewentualne zderzenia cząstek tylko w lokalnym otoczeniu.

Dlatego cały obszar obliczeniowy dzieli się na małe komórki o wymiarach $\Delta x \times \Delta y$ i rozważa się zderzenia jedynie w danej kómórce o indeksie (i, j) oraz 8 ją okalających - cząstka będąca blisko brzegu swojej komórki w przedziale czasu $(t, t + \Delta t)$ może przejść do sąsiedniej, gdzie zderzy się inną cząstką. Taki zabieg znacząco podnosi wydajność algorytmu.
Elementy algorytmu metody DSMC

- **warunek początkowy**
  Ponieważ wyniki symulacji DSMC są rozwiązaniem równania transportowego, więc warunek początkowy będzie determinował zachowanie układu w początkowej fazie symulacji - w stanie nieustalonym. W stanie ustalonym oczekujemy równowagi termodynamicznej gazu czyli Maxwellowskiego rozkładu prędkości (pod warunkiem, że układ nie wymienia ciepła z otoczeniem - co oznacza narzucenie warunku Neumanna na każdej krawędzi brzegu zewnętrznego). Warunek początkowy w postaci rozkładu Maxwella w 2D - punktem wyjścia jest rozkład Boltzmanna (w dziedzinie energii)

$$f_E = \left(\frac{m}{2\pi k_B T}\right)^{d/2} e^{-\frac{E_{kin}}{2k_B T}}, \quad d = 1, 2, 3 \text{ - liczba wymiarów} \tag{1}$$

z warunkiem unormowania

$$\int_0^\infty f_E(E) dE = 1 \tag{2}$$

Ponieważ $E_{kin} = m(V_x^2 + V_y^2)/2$ więc

$$f_E(E) = \underbrace{\left(\frac{m}{2\pi k_B T}\right)^{1/2} e^{-\frac{mV_x^2}{2k_B T}}}_{f_{Ex}} \cdot \underbrace{\left(\frac{m}{2\pi k_B T}\right)^{1/2} e^{-\frac{mV_y^2}{2k_B T}}}_{f_{Ey}} = f_{Ex} \cdot f_{Ey} \tag{3}$$

i składowe prędkości w obu kierunkach możemy losować z rozkładu normalnego

$$V_x, V_y \sim \sigma_V \cdot N(0, 1), \qquad \sigma_V = \sqrt{\frac{k_B T}{m}} \tag{4}$$

Zazwyczaj interesuje nas maxwellowski rozkład prędkości cząstek tj. zależny tylko i wyłącznie od wartości prędkości, aby go uzyskać musimy dokonać transformacji zmiennych przechodząc do opisu we współrzędnych cylindrycznych

$$f_E(E)dE = f_1(V_x, V_y)dV_x dV_y = 2\pi f_2(V) V dV = f_V^{2D} dV \tag{5}$$

co daje rozkład

$$f_V^{2D} = \frac{mV}{k_B T} e^{-\frac{mV^2}{2k_B T}}, \qquad V = \sqrt{V_x^2 + V_y^2} \tag{6}$$

z warunkiem normalizacji

$$\int_0^\infty f_V^{2D} dV = 1 \tag{7}$$

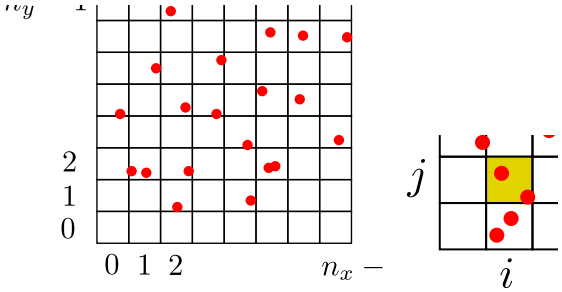
W układzie izolowanym to byłby oczekiwany rozkład w stanie ustalonym.

- **krok czasowy**
  Zakładamy, że w czasie Δt cząstka nie może przemieścić się o więcej niż wynosi szerokość/wysokość komórki definiującej jej lokalne otoczenie

$$\Delta t(t) \leqslant \frac{\min\{\Delta x, \Delta y\}}{V_{max}(t)}, \qquad V_{max} = \max\{V_1, V_2, \ldots, V_{n_{tot}}\} \tag{8}$$

  Cząstki zderzając się ze sobą zmieniają prędkość, zatem $V_{max}(t)$ i $\Delta t(t)$ należy wyznaczać w każdym kroku.

- **zderzenia dwóch cząstek**
  W oryginalnej wersji DSMC, cząstki są rozpraszane w komórkach w sposób losowy co oznacza, że na podstawie ich średniej prędkości kwadratowej oraz przekrojów czynnych na rozpraszanie określa się ile par cząstek ma się rozproszyć, a następnie rozprasza się je w układzie środka masy w losowym kierunku (tak aby pęd środka masy był zachowany). W programie, który użyjemy zderzenia są wykrywane a rozpraszane są tylko te cząstki, których trajektorie rzeczywście się przecinają, natomiast ich kierunki po zderzeniu są randomizowane w układzie środka masy. Takie podejście jest mniej wydajne, ale zwiększa dokładność i rozdzielczość przestrzenną symulacji.



Rysunek 2: Lewy rysunek pokazuje podział układu na niewielkie komórki zawierające $20 - 80$ cząstek gazu natomiast prawy rysunek pokazuje komórkę (i, j) w której cząstki mogą oddziaływać ze sobą oraz z cząstkami z sąsiednich ośmiu komórek.

2 Opis klasy DSMC_2D

Symulację wykonamy przy użyciu gotowych procedur zawartych w klasie **DSMC_2D**.

2.1 klasy prywatne

Klasa **DSMC_2D** zawiera klasę **PARTICLE**. Obiekt typu **PARTICLE** przechowuje aktualne informacje dotyczące pojedynczej cząstki. W programie dane cząstek przechowywane są w tablicy **par**[], której elementy są typu **PARTICLE**. Zmienne w klasie **PARTICLE**

nazwa zmiennej w klasie PARTICLE	opis
double rc	promień cząstki w [m]
double mc	masa cząstki w [kg]
double x0,y0	położenie początkowe cząstki (inicjalizacja) w [m]
double vx0,vy0	początkowy wektor prędkości cząstki (inicjalizacja) w [m/s]
double x,y	położenie cząstki w aktualnej chwili w [m]
double vx,vy	wektor prędkości cząstki w aktualnej chwili w [m/s]
double v	wartość prędkości cząstki w aktualnej chwili w [m/s]
int ix,iy	indeksy komórki obliczeniowej, w której aktualnie znajduje się cząstka
int indx	indeks globalny cząstki przydatny przy śledzeniu zmian
int nbound_col	całkowita liczba zderzeń cząstki z brzegiem
int ncol	liczba zderzeń z innymi cząsteczkami wykorzystywana np. do liczenia średniej drogi swobodnej
double path	długość drogi jaką przebyła cząstka od początku symulacji w [m]
double cv	współczynnik korelacji prędkości cząstki do wyznaczania współczynnika dyfuzji w $[m^2/s]$

2.2 zmienne w klasie DSMC_2D

int ntot	liczba cząstek gazu w układzie
int n_mix	liczba rodzajów cząstek gazu
int init_dist	rozkład początkowy gazu, 0-wczytywany z pliku ("pos_vel_start.dat"); 1-wszystkie cząstki mają identyczne prędkości (v=sqrt(2*kB*Temp/masa)), ale kierunki i położenia losowe; 2- rozkład Maxwella-Boltzmanna, położenia losowe; 3-identyczna energia kinetyczna, cząstki umieszczone losowo w komórce obliczeniowej (0,0); 4-rozkład Maxwella-Boltzmanna w komórce obliczeniowej (0,0)
int icol=0/1 (default=1)	0-brak zderzen (cząstki się nie widzą), 1-zderzenia są obsługiwane
int nthreads (default=1)	liczba rdzeni procesora używanych w obliczeniach (instrukcje Openmp)
double xmin,xmax,ymin,ymax	lewy dolny i prawy górny wierzchołek prostokąta tworzącego układ w [m]
int nx,ny	liczba komórek w kierunku x-owym i y-owym, na które dzielony jest układ
double delta_x, delta_y	szerokość i wysokość podstawowej komórki obliczeniowej na jakie jest podzielony cały układ w [m]
double temp	temperatura gazu w chwili początkowej w [K]
double dt	aktualna wartość kroku czasowego w [s], zmienia się w zależności od maksymalnej prędkości cząstek vmax w danej chwili, dt=min(delta_x,delta_y)/vmax
double time_sum	całkowity czas trwania ewolucji, wartość zmieniana na bieżąco w funckji **step()**, która wykonuje ewolucję układu w ciągu jednego kroku czasowego w [s]
double cv_coeff	aktualna wartość współczynnika autokorelacji prędkości w układzie dla cząstek które nie zderzyły się ze ściankami (brzegiem)
int nodes_out (default=4)	liczba wierzchołków obszaru zewnętrznego
int nodes	liczba wierzchołków obszaru wewnętrznego

2.3 tablice w klasie DSMC_2D

double tempi[k]	warunek brzegowy dla temperatury: k=0-lewy brzeg, k=1-górny brzeg, k=2-prawy brzeg, k=3-dolny brzeg, tempi[k]<0 - warunek brzegowy Neumanna (odbicie), tempi[k]$=T>0$ - warunek Dirichleta w [K]
PARTICLE par[ntot]	tablica obiektów klasy **PARTICLE**
double edge_out[nodes_out][2]	tablica zawiera położenia wierzchołków wielokąta (prostokąta) stanowiącego obszar obliczeniowy, kolejność wierzchołków wpisana zgodnie z ruchem wskazówek zegara w [m]
double edge[nodes][2]	tablica zawiera położenia wierzchołków wielokąta stanowiącego obszar bariery wewnętrznej, kolejność wierzchołków wpisana zgodnie z ruchem wskazówek zegara w [m]
double temp_cell[nx][ny]	tablica zwiera rozkład temperatur w komórkach obliczeniowych
double dens_cell[nx][ny]	tablica zwiera rozkład gęstości w komórkach obliczeniowych
double vx_cell[nx][ny]	tablica zwiera rozkład średniej x-owej składowej wektora prędkości w komórkach obliczeniowych
double vy_cell[nx][ny]	tablica zwiera rozkład średniej y-owej składowej wektora prędkości w komórkach obliczeniowych
double press_x[nx]	tablica zwiera rozkład ciśnienia w komórkach obliczeniowych w kierunku x
double press_y[ny]	tablica zwiera rozkład ciśnienia w komórkach obliczeniowych w kierunku y
double velocity_x[nx]	tablica zwiera uśrednioną po kierunku y-owym składową x-ową wektora prędkości
double velocity_y[ny]	tablica zwiera uśrednioną po kierunku x-owym składową y-ową wektora prędkości

2.4 funkcje w klasie DSMC_2D

Lista przydatnych funkcji

void read (const char * nazwa_pliku)	funkcja wczytuje parametry symulacji z pliku wejściowego "nazwa_pliku"
void init()	funkcja inicjalizuje wartości zmiennych dla chwili startowej
void step()	funkcja wykonuje ewolucję układu dla czasu (t,t+dt), krok dt jest wyznaczany automatycznie
void evolution (double tmax, int iter_max)	funkcja prowadzi ewolucję układu dopóki spełniony jest jeden z warunków: (i) $t < tmax$ lub (ii) $it < iter\_max$
void write_position_velocity (const char * nazwa_pliku)	funkcja zapisuje do pliku "nazwa_pliku" aktualny wektor położenia-prędkości (x,y,vx,vy) dla każdej cząstki
void write_cell_bounds (const char * nazwa_pliku)	funkcja zapisuje do pliku położenia wierzchołków wszystkich komórek obliczeniowych
void hist_velocity_all (const char * nazwa_pliku,double mnoznik, int k)	funkcja wyznacza histogram wartości prędkości, liczba komórek jest równa **k**, maksymalny zakres histogramu to **vmax*mnoznik**, gdzie **vmax** to maksymalna prędkość cząsteczek w układzie (wyznaczana przez program), histogram zapisywany jest do pliku "nazwa_pliku"

2.5 Plik wejściowy z danymi

Funkcja **read(nazwa_pliku)** wczytuje dane z pliku co ułatwia wykonywanie symulacji, ponieważ nie wymaga rekompilacji programu. Struktura przykładowego pliku wejściowego (znaczenie wartości podane po prawej stronie)

```
1.0      2.       0.0     1.          // xmin,xmax,ymin,ymax
50      50                           // nx,ny
1.0E-23                              // stała Boltzmana
300.    -1.   -1.   -1.   -1.        // temp,tempi[0-3]
2                                    // init_dist=0,1,2,3,4


1                                    // n_mix - liczba typów cząstek
200000    40.0E-27     1.0E-6        // nc1,mc1,rc1 - liczba cząstek,
                                                    masa cząstki,
                                                    promień cząstki 1-typu

4                                    // nodes - wierzchołki bariery wewnętrznej

0.2 0.2                              // edge[0][1],edge[0][1]
0.2 0.4                              // edge[1][1],edge[1][1]
0.4 0.4                              // edge[2][1],edge[2][1]
```

```
0.4 0.2                          // edge[nodes-1][1],edge[nodes-1][1]
```

Uwaga:

- jeśli w układzie nie ma barier to należy ustawić **nodes=0** położenia wierzchołków **edge**[][] nie są wówczas wczytywane

2.6 Przykładowy kod sterujący, kompilacja, symulacja

Kompilacja kodu

```
g++ -O3 -fopenmp  prog*.cpp  -std=c++17 -lstdc++fs
```

Kod programu

```
using namespace std;
#include "dsmc_2d.cpp"

int main(){

        DSMC_2D ob;

        ob.read("i.dat");    //wczytujemy dane zpliku wejściowego
        ob.init();           //automatyczna inicjalizacja położeń i prędkości
        ob.write_position_velocity("rv.dat");  //zapis ustawień początkowych
        ob.nthreads=1;     //obliczenia na jednym rdzeniu
        ob.icol=1;         //cząstki zderzają się
        ob.evolution(0.0,20000); //wykonujemy 20 tysięcy kroków (tmax - nieznany)
        ob.hist_velocity_all("hist2.dat",5.0,50);  //zapis histogramu
                                                    prędkosci do pliku
        ob.write_position_velocity("rv.dat");   //zapis położeń
                                               i prędkości końcowych do pliku

        return 0;
}
```

3 Zadania do wykonania

W symulacji przyjąć parametry: $x_{min} = 0$, $y_{min} = 0$, $x_{max} = 1$, $y_{max} = 1$, $n_y = n_y = 50$, $k_B = 1.38 \cdot 10^{-23}$, $temp = 300$, warunki brzegowe Neumanna (odbijające), init_dist - to będziemy zmieniać, liczba cząstek $n\_mix = 1$, $n_1 = 10^5$ (lub więcej jeśli program będzie szybko działał - zależy od komputera), masa cząsteczki $m_1 = 40{\cdot}10^{-27}$ kg, promień cząsteczki $r_1 = 10^{-6}$ m, nodes=0 (brak obiektu w środku). Uwaga: co 10 iteracji procedura **evolution()** zapisuje rozkłady: gęstości (n), ciśnienia (p), temperatury (t) i prędkości (v) do plików **nptv_nr_iteracji.dat** w katalogu **wyniki** - to powinno ułatwić analizę wyników. Gdyby ktoś chciał zapisywać wyniki z inną częstością należy zmienić odpowiedni warunek w funkcji **evolution()**.

1. Ustawić zmienną **init_dist=1**, wówczas cząsteczki będą miały identyczne energie kinetyczne i prędkości. Wykonać symulację, narysować rozkład końcowy prędkości (w stanie ustalonym) oraz 2 rozkłady w stanie nieustalonym. Symulację wykonać dla dwóch promieni cząsteczek $r_1 =$

$10^{-5}; 10^{-6}$ m - dla większych cząsteczek zauważymy szybsze dochodzenie do stanu równowagi. Rozkład końcowy porównać z rozkładem teoretycznym.

2. Powtórzyć symulacje z zadania 1 dla **init_dist=3** - wszystkie cząstki są początkowo umieszczone w jednej komórce (0,0) - uwaga: na początku program będzie działał baaaaardzo wolno, bo wszystkie cząstki są umieszczone blisko siebie. Dla kilku pierwszych iteracjach, wykonanie jednego kroku może trwać 1-2 minuty, po około 10-15 iteracjach czas wykonania pojedynczego kroku maleje do około 1 sekundy. Narysować rozkład cząstek w kilku pierwszych iteracjach oraz ich rozkład końcowy.

3. Ustawić parametr **init_dist=2** (rozkład Maxwella w całym obszarze), przyjąć temperaturę początkową $temp = 300$, na lewym brzegu ustawić temperaturę 1000 K, na pozostałych krawędziach przyjąć warunek Neumanna. Sporządzić wykresy rozkładu temperatury i ciśnienia wzdłuż kierunku x-owego w kilku wybranych chwilach czasu. Narysować rozkład prędkości w chwili startowej i końcowej na jednym rysunku. W stanie ustalonym spodziewamy się braku gradientu temperatury.

4. Powtórzyć symulacje z zadania 3, ale dla warunku Dirichleta na prawym brzegu ustalając tam temperaturę o wartości 300 K. W stanie ustalonym gradient temperatury powienien być identyczny (+ fluktuacje) w całym obszarze.