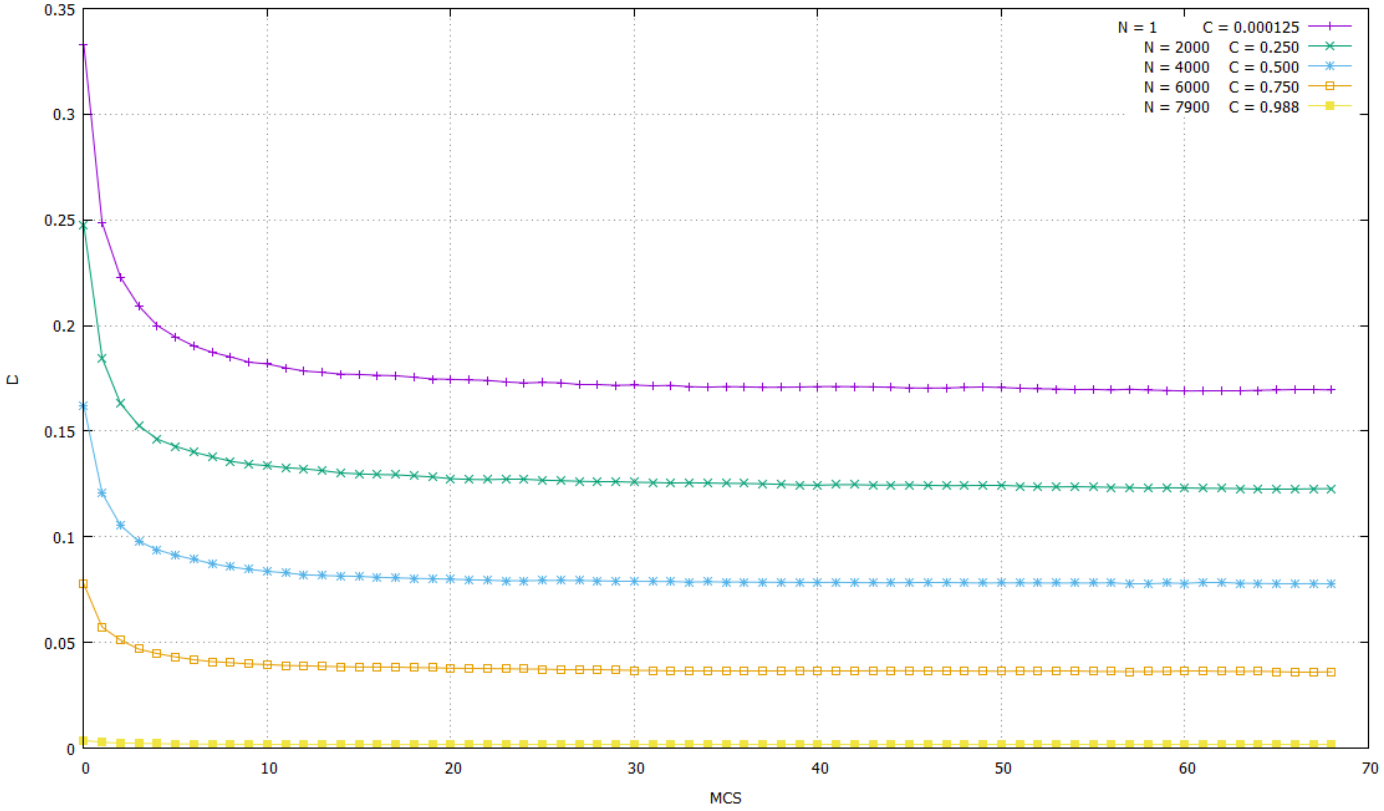
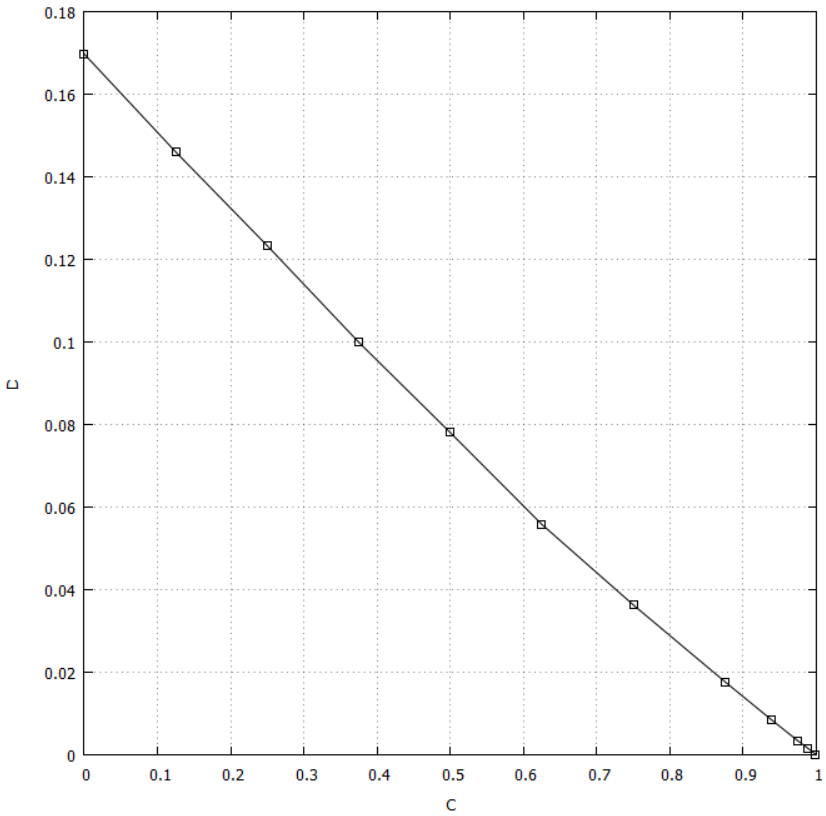
Sebastian Jezierski, 236780 26.03.2020

Symulacje Monte Carlo: Dyfuzja 3D



Parametry:

- rozmiar pudełka $L = 20$
- liczba kroków $MCS = 70$
- liczba niezależnych symulacji zależna od liczby atomów ($L_p = 20000/N$)



Obliczone współczynniki dyfuzji:

N	L_p	C	D
1	20000	0.000125	0.169726
1000	20	0.125	0.146013
2000	10	0.250	0.123302
3000	6	0.375	0.099928
4000	5	0.500	0.078053
5000	4	0.625	0.055812
6000	3	0.750	0.036308
7000	2	0.875	0.017721
7500	2	0.938	0.008584
7800	2	0.975	0.003358
7900	2	0.988	0.001656
7990	2	0.999	0.000156

Kod programu
(Napisany w języku Python, z użyciem biblioteki NumPy)

```
import numpy as np
rnd = np.random

# configuration -----------------------------------------------------
box_size = 20
n_atoms = 1000
mc_steps = 70
independent_simulations = 20000//n_atoms

assert n_atoms < box_size ** 3

# initialization ----------------------------------------------------
print("#    L = {}\n"
      "#    N = {}\n"
      "#    C = {}\n"
      "#    {} MCS, {} independent simulations"
      .format(box_size, n_atoms, n_atoms*box_size**(-3), mc_steps, independent_simulations))


# possible moves of an atom in a step
moves = [[1, 0, 0], [-1, 0, 0], [0, 1, 0],
         [0, -1, 0], [0, 0, 1], [0, 0, -1]]

results_avg = np.zeros(mc_steps)    # R2[MCS] averaged over atoms and ind. simul.

for sim_no in range(independent_simulations):

    results = np.zeros_like(results_avg)    # R2[MCS] averaged over atoms

    # occupied places
    occupied = np.zeros((box_size, box_size, box_size), dtype=bool)

    # positions array
    atoms = np.zeros((n_atoms, 3), dtype=int)
    # [x1,y1,z1]
    # [x2,x2,z2]
    #    ...
    # [xn,yn,zn]

    # inserting atoms in random positions
    for i in range(n_atoms):
        r = rnd.randint(box_size, size=3)  # r = [x,y,z] (random position)
        while occupied[r[0], r[1], r[2]]:
            r = rnd.randint(box_size, size=3)

        atoms[i] = r
        occupied[r[0], r[1], r[2]] = True

    start_pos = atoms.copy()  # save a copy of starting positions
```

```
# simulation ----------------------------------------------------------
    for i in range(mc_steps):

        # Monte Carlo step ------------------------------------------------
        for r in atoms:

            dr = moves[rnd.choice(6)]   # random direction (vector from moves[])
            nb = (r + dr) % box_size    # coordinates of the neighbor (in PBC)
            rb = r % box_size           # coordinates of the atom (in PBC)

            if not occupied[nb[0], nb[1], nb[2]]:
                occupied[nb[0], nb[1], nb[2]] = True
                occupied[rb[0], rb[1], rb[2]] = False
                r += dr

        # Calculating distance travelled
        DeltaR2 = np.linalg.norm(atoms - start_pos, axis=1) ** 2
        results[i] = np.average(DeltaR2)    # average over atoms

    results_avg += results/independent_simulations  # average over independent simulations

# Calculating and printing the diffusion coefficient ---------------------
D = [results_avg[t] / (6*t) for t in range(1,mc_steps)]

for item in D:
    print(item)
```